Implementing Reliable Broadcast in Anonymous Distributed Systems with Fair Lossy Channels

Jian Tang
Distributed Systems Laboratory (LSD)
Universidad Politécnica de Madrid
28031, Madrid, Spain
Email: tjapply@gmail.com

Mikel Larrea
Universidad del País Vasco
20018, San Sebastián, Spain
Email: mikel.larrea@ehu.es

Sergio Arévalo
Universidad Politécnica de Madrid
28031 Madrid, Spain
Email: sergio.arevalo@eui.upm.es

Ernesto Jiménez
Universidad Politécnica de Madrid
28031 Madrid, Spain
Email: ernes@eui.upm.es

1. Introduction

Reliable Broadcast is used to disseminate messages among a set of processes with *RB_broadcast()* and *RB_deliver()* operations, which was introduced in [1]. Reliable Broadcast is a broadcast service that requires all non-crashed processes deliver the same set of messages. This service has been extensively studied in non-anonymous systems, usually assuming that communication channels are reliable (if a process p sends a message m to a process q, and q is correct, then q eventually receives m) [2]. However, most of real channels 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 [3]). While some work has been done to construct reliable channels over unreliable channels in non-anonymous systems [3], [4], the study of reliable broadcast in anonymous distributed systems with unreliable communication channels still has a big gap that needs to be filled up.

2. System Model and Definitions

We consider anonymous asynchronous systems in which processes have no identifiers and communicate with each other via a completely connected network with fair lossy channels. 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 $receive_i(m)$.

Processes Processes are anonymous, asynchronous and execute the same algorithm. The system is composed of a set of n anonymous processes, denoted by $\Pi = \{p_i\}(i = 1, ..., n)$. We consider that i is the index of each process in 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, which is also used for notation.

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.

Communication channels Each pair of processes is connected by bidirectional communication channels. Processes communicate by sending and receiving messages through these channels. We assume that channels neither duplicate nor create messages, but may loose messages. Note that in our anonymous system, when a process receives a message, it cannot determine who is the sender of this message.

Communication channels are *fair lossy*. Formally, a channel between two processes p and q is fair lossy if it satisfies the following properties [5]:

- 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.

Reliable Broadcast Formally, Reliable Broadcast is defined by two primitives, namely $RB\_broadcast(m)$ and $RB\_deliver(m)$, satisfying the following 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.
- *Integrity*: For any message m, every correct process delivers m at most once, and only if m was previously broadcast by a process.

Research 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, and the University of the Basque Country UPV/EHU, under grant UFI11/45.

3. The Algorithm

A reliable broadcast algorithm for anonymous asynchronous systems with fair lossy communication channels is proposed here. The algorithm makes every message $RB\_broadcast$ by processes in the system unique. To do so, it relies on a random number generation function $random_i()$ at each process p_i. This function is used to assign a unique number to each $RB\_broadcast$ message as a label, denoted by tag. We assume that no random function can generate the same random number to two different messages. Note that this label assignment to messages does not violate the anonymity of the system, since labels do not correspond to process identifiers (the label does not allow determining the sender of a message).

Algorithm 1 Reliable Broadcast in anonymous asynchronous systems with fair lossy communication channels (code of process p_i)

```
1  Initialization
2     sets $MSG_i$, $RB\_DELIVERED_i$ empty
3     activate Task 1
4  When $RB\_broadcast_i(m)$ 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 $(m, tag)$ 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
```

Description of the algorithm:

In Algorithm 1, each process p_i manages a random function $random_i()$ and two local sets: MSG_i, that records all messages that p_i has received; and $RB\_DELIVERED_i$, that records all messages that p_i has reliably delivered. Remember that index i is used just for description, no process knows which process is p_i, even itself.

The algorithm runs as follows:

At the beginning, p_i initializes its sets MSG_i and $RB\_DELIVERED_i$ to empty and activates Task 1 (Lines 1-3). When p_i calls $RB\_broadcast_i(m)$ (Line 4), its random function generates a random number as a tag for m firstly (Line 5). Then, p_i inserts the pair (m, tag) into its set MSG_i (Line 6) in order to be broadcast as (MSG, m, tag) periodically by Task 1 (Lines 15-19).

When $receive_i(MSG, m, tag)$ is executed (Line 7), p_i first records the pair (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 reliably delivered (by checking if the pair (m, tag) exists in $RB\_DELIVERED_i$ or not). If not, p_i records the pair (m, tag) into $RB\_DELIVERED_i$ and reliably delivers m by executing $RB\_deliver_i(m)$ (Lines 11-14).

Discussion:

Observe that in this algorithm, due to the assumption of fair lossy channels, processes have to broadcast periodically by Task 1 all messages that have been received (and reliably delivered), in order to guarantee that all correct processes deliver the same set of messages. This non-quiescent broadcast can be considered as the cost of obtaining reliability over the fair lossy channels in anonymous systems.

It turns out that this weakness also indicates the direction in order to transform Algorithm 1 into a quiescent one, i.e., once a message m has been delivered by all correct processes, it is not needed to broadcast it anymore, which means that m should be removed from the MSG set of each correct process. So, the quiescence problem consists in guaranteeing that a message has been delivered by all correct processes. In order to achieve this, we propose to combine two mechanisms: (1) each process broadcasts an acknowledgment message after it has delivered a message m, and collects acknowledgment messages on m from all processes (including itself); (2) a failure detector is needed to obtain information of all correct processes.

4. Conclusion

Reliable Broadcast is a basic communication primitive in distributed systems, which allows processes communicating consistently and reliably. This agreement problem has been extensively investigated in distributed systems where all processes have different identifiers. In this paper, we studied this primitive in asynchronous anonymous systems by overcoming two adversaries: process anonymity and message loss.

References

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

[2] A. Schiper. Failure detection vs group membership in fault-tolerant distributed systems: hidden trade-offs. Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, pp. 1-15, Springer-Verlag, London, 2002.

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

[4] 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): 1267–1297, 1994.

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