Revisiting Reliable Broadcast (extended abstract)

Svend Frolund, Fernando Pedone
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-192
August 2nd, 2001*

E-mail: {frolund, pedone} @hpl.hp.com

Reliable broadcast is a fundamental problem in fault-tolerant, distributed computing. The literature contains different implementations of reliable broadcast providing different guarantees and capturing different trade-off in terms of cost. We introduce a generic algorithm template that allows us to express these existing algorithms within a unified framework. We can obtain the various guarantees and trade-off by plugging specific algorithm fragments into our generic template. In addition to expressing existing solutions, we also introduce a novel algorithm that efficiently implements uniform reliable broadcast.

* Internal Accession Date Only Approved for External Publication

Revisiting Reliable Broadcast

(extended abstract)

Svend Frølund Fernando Pedone
Hewlett-Packard Laboratories
Palo Alto, CA 94304, USA
phone: +1 650 857 5121
fax: + 1 650 852 3732
{frolund, pedone}@hpl.hp.com

Abstract

Reliable broadcast is a fundamental problem in fault-tolerant, distributed computing. The literature contains different implementations of reliable broadcast providing different guarantees and capturing different trade-off in terms of cost. We introduce a generic algorithm template that allows us to express these existing algorithms within a unified framework. We can obtain the various guarantees and trade-off by plugging specific algorithm fragments into our generic template. In addition to expressing existing solutions, we also introduce a novel algorithm that efficiently implements uniform reliable broadcast.

1 Introduction

Reliable broadcast is a fundamental building block for fault-tolerant systems. In general, a broadcast algorithm seeks to disseminate information (i.e., messages) to a group of processes in a distributed system. A *reliable* broadcast algorithm provides certain dissemination guarantees in the presence of process failures. For example, one variant of reliable broadcast ensures that the set of non-faulty processes always deliver the same set of messages. We can use a reliable broadcast algorithm as a building block in many

contexts. For example, a transaction coordinator can use reliable broadcast to distribute transaction outcome information (abort or commit) to the participants in a distributed transaction.

There are many different ways to implement reliable broadcast, and when implementing it, one faces the traditional trade-off in distributed computing, such as latency versus total number of messages, and performance in "good" runs without failures versus performance in failure-handling situations. There is no single implementation of reliable broadcast that we can use for all applications. Furthermore, there is no single specification of reliable broadcast that matches the need of all applications. Different specifications are more or less stringent in terms of *agreement* , a property that captures the the set of messages that all non-faulty processes must deliver. For example, non-uniform agreement requires all non-faulty processes to deliver the same set of messages; in addition, uniform agreement requires non-faulty processes to also deliver messages that faulty processes delivered before they failed. Of course, we can always use uniform agreement in place of non-uniform agreement (because one implies the other), but there is an intimate connection between the strength of the agreement property and the cost of providing it: always using uniform agreement would incur an unnecessary cost for applications that only need non-uniform agreement.

We have defined a generic template for reliable broadcast algorithms. With this template, the variations in agreement, and the various trade-off in cost, are particular implementations of two well-defined algorithm fragments. We can obtain different agreement properties, and enforce different cost trade-off, simply by replacing the implementation of these two fragments—the overall algorithm structure remains the same. One fragment is concerned with message propagation, and implements the way in which a process relays the messages that it receives to other processes. The goal of propagation is to ensure appropriate diffusion of messages in a fault-tolerant manner (e.g., even if the original sender of a message fails). Another fragment is concerned with the condition for message delivery, and ensures that messages are not delivered too soon. For example, a process may have to ensure that other processes have received a message before it can deliver the message.

Using the generic template for reliable broadcast, we discuss four different implementations, all sharing the same basic structure. These resulting four algorithms capture three traditional implementations of reliable broadcast as well as a novel implementation.

Our novel algorithm implements uniform reliable broadcast in an efficient manner. We compare the cost of the four algorithms in terms of latency and total number of messages, considering executions with and without process failures.

The remainder of the paper is structured as follows. Section 2 presents the system model and defines reliable broadcast. Section 3 discusses four reliable broadcast implementations. Section 4 evaluates the implementations, and Section 5 concludes the paper.

2 System Model and Problem Definition

We consider a system with n processes, $p_1, \ldots, p_n$. Processes can fail by crashing; if a process fails, it permanently stops executing its algorithm—we do not consider Byzantine failures nor do we consider recovery after a crash. A process that does not fail is *correct* ; otherwise it is *faulty* . Processes communicate through message passing. A process can invoke the `send` primitive to send a message, and it can invoke the `receive` primitive to access messages sent to it. We assume that the underlying communication channels are reliable: if a correct process p sends a message to a correct process q, then q will eventually receive the message. The system is asynchronous: there is no bound on the time it takes for a process to execute a step in its algorithm, and there is no bound on the delivery time for messages. In some of the algorithms, we assume that processes have access to a failure-detection module [CT96]. In those algorithms, we assume an eventually strong failure detector (a failure detector in the class $\diamond S$). That is, failure detectors satisfy the following properties [CT96]: (strong completeness) there is a time after which any faulty process is permanently suspected by every correct process, and (weak accuracy) there is a time after which a correct process is never suspected by any correct process.

We define reliable broadcast in terms of two events: `broadcast` and `deliver`. The `broadcast` event denotes the initiation, by a single process, of a broadcast operation (the dissemination of a message to all processes in the system). The `deliver` event denotes the end of a broadcast operation at a particular process (either the initiator of the broadcast or a "listener" process). We denote $sender(m)$ the process that broadcasts message m. Every message carries the identification of its sender.

Reliable broadcast guarantees three properties, integrity, validity, and agreement, defined over the broadcast and deliver events [HT93]. Validity (V) and integrity (I) are defined as follows:

- (V). If a correct process broadcasts a message m, then it eventually delivers m.
- (I). For any message m, every process delivers m at most once, and only if m was previously broadcast by $sender(m)$.

We define two agreement properties (A1–A2), which, when combined with validity and integrity, will lead to a different flavor of reliable broadcast:

- (A1). if a correct process delivers m, then all correct processes deliver m.
- (A2). if a process delivers m, then all correct processes deliver m.

The property A1 is called non-uniform reliable broadcast, and property A2 is called uniform reliable broadcast. Notice that uniform reliable broadcast is stronger than non-uniform reliable broadcast: an implementation of A2 also satisfies A1. In the following section we present a generic algorithm template to implement uniform and non-uniform reliable broadcast with different trade-off in terms of cost.

3 Implementing Reliable Broadcast

In this section, we present implementations of uniform and non-uniform reliable broadcast. We first introduce a generic template for reliable broadcast algorithms, and then show how to derive uniform and non-uniform algorithms as particular implementations of fragments that plug into the generic template.

Reliable broadcast algorithms have to address two concerns: (i) how to propagate broadcast messages to processes, and (ii) when a process can safely deliver a message. The first concern has a direct impact on the cost of the algorithm, as we discuss in the next section; the second concern has to avoid situations when a process delivers a message "too early," that is, before enough processes know about the message.

Figure 1 shows our generic algorithm template. Whenever a process wants to broadcast a message m, it sends m to all processes. Upon receiving m for the first time, a process p executes a propagation procedure, which is an algorithm fragment that propagates broadcast messages (concern (i) above). A message m can be delivered by p as soon as the deliverable(m) predicate holds at p. The deliverable predicate is an algorithm fragment that ensures safe delivery of messages (concern (ii) above). Notice that deliverable(m) is a local predicate at p.

```
To broadcast m:
  send m to all

when receive m for the first time
   propagate(m)

when ( deliverable(m) = true ) ∧ ( m ∉ deliveredSet )
  deliver(m)
  deliveredSet ← deliveredSet ∪ { m }
```

Figure 1: A generic template for reliable broadcast algorithms

```
To execute propagate(m):

  if sender(m) ≠ p_i then send m to all
```

Figure 2: Message propagation with flooding

We now show how uniform and non-uniform reliable broadcast can be implemented within our framework as particular implementations of the propagation procedure and deliverable predicate. For both uniform and non-uniform reliable broadcast, we discuss flood-based and failure-detection-based versions of the algorithms. These distinctions are related to how propagation is implemented.

3.1 Flooding Algorithms

With flooding algorithms, whenever a process receives a message for the first time, it sends the message to all other processes. While this mechanism may result in unnecessary network traffic, flooding algorithms do not depend on failure detection, and so, can be solved in purely "asynchronous systems." All algorithms presented in this section implement propagation as described in Figure 2. The deliverable predicate has different implementations for uniform and non-uniform reliable broadcast.

Non-Uniform Reliable Broadcast. The non-uniform reliable broadcast algorithm composed of the propagate procedure in Figure 2 and the deliverable predicate in Figure 3 was originally presented in [CT96]. Since only correct processes have to agree on the

```
Predicate deliverable(m) is defined as follows:

  deliverable(m) ≡ true
```

Figure 3: Delivery condition for non-uniform reliable broadcast

```
Predicate deliverable(m) is defined as follows:

  deliverable(m) ≡ [for f+1 process p_j : received m from p_j]
```

Figure 4: Delivery condition for uniform reliable broadcast

messages delivered, as soon as a process receives a message for the first time, it can deliver it. The idea is that if a correct process delivers a message m, and since it sends m to all, m will be received and delivered by all other correct processes as well.

Uniform Reliable Broadcast. The following algorithm, composed of the flooding propagation in Figure 2 and the deliverable predicate presented in Figure 4, assumes a majority of correct processes, i.e., $f < n/2$. Notice that since a process can only deliver a message after it knows that the message has been received by $f + 1$ processes, if half of the processes could fail, no message would be possibly delivered (i.e., the remaining processes, $n/2$, do not satisfy the $f + 1$ constraint), and so the assumption that $f < n/2$.

The algorithm presented in [ATD99] is the closest to our algorithm for uniform reliable broadcast based on flooding. The algorithm in [ATD99], however, uses a stronger failure detector than we use to solve uniform reliable broadcast even when $f \geq n/2$.[1] We believe we could use a similar mechanism to the one in [ATD99], and a stronger failure detector, to redefine our deliverable predicate and overcome the $f < n/2$ constraint. Due to space limitation we do not address this issue in this paper.

3.2 Failure Detection-Based Algorithms

Failure detection-based algorithms improve on flooding algorithms by being more judicious on when to propagate messages to the other processes. The criterion used to propagate a message is related to failure suspicions. Therefore, even though, in the absence of failures

[1] Furthermore, [ATD99] makes weaker assumptions about the communication links than we do in this paper.

```
To execute propagate(m):
  start task diffuse(m)

  task diffuse(m):
    when suspect sender(m)
      send m to all
```

Figure 5: Message diffusion for non-uniform reliable broadcast

and false suspicions, the algorithms presented in this section perform better than flooding algorithms, they cannot be used in asynchronous systems.

Non-Uniform Reliable Broadcast. With the propagation procedure in Figure 5, each process p starts a task diffuse(m) that constantly monitors $sender(m)$. Process p only sends message m to the other processes if p suspects $sender(m)$; a process can deliver a message as soon as it receives the message for the first time. That is, we can use the deliverable predicate defined in Figure 3, which was also used for the flooding version of reliable broadcast.

The resulting algorithm was originally proposed in [GS96]. Agreement is guaranteed as follows. If $sender(m)$ is correct, its initial message with m is received by all correct processes, which will deliver m. If $sender(m)$ is not correct but some correct process p delivers m, then p eventually suspects $sender(m)$ and relays m to all processes—since p is correct, all correct processes receive and deliver m.

Uniform Reliable Broadcast. This algorithm assumes $f < n/2$. We show the propagation part of the algorithm in Figure 6. A subset of processes of size $f + 1$ is responsible for propagating m initially. If a process in the initial subset of processes fails and cannot retransmit m, it is eventually suspected by the remaining ones which will then retransmit m. A process can only deliver m when it receives m from $f + 1$ processes, which ensures that when a process delivers m, at least one correct process has received m. Thus, the algorithm uses the deliverable predicate in Figure 4.

In Figure 6, we "hard-wire" the subset of processes that propagate messages. Hard-wiring the set simplifies the presentation. However, the set could be chosen by $sender(m)$ and shipped with m; $sender(m)$ could include, for instance, processes it does not suspect.

```
To execute propagate(m):

  if sender(m) ≠ p_i and p_i in { p_1, p_2, ..., p_{f+1} } then send m to all
  start task diffuse(m)

  task diffuse(m):
    when suspect some p_j in { p_1, p_2, ..., p_{f+1} } for the first time
      send m to all
```

Figure 6: Message diffusion for uniform reliable broadcast

Notice that the deliverable predicates for the non-uniform and the uniform algorithms based on failure detection are similar to the ones defined for the flooding-based algorithms; this spells out the fact that in our framework, the deliverable predicate is not related to the way messages are propagated, but to the agreement property of reliable broadcast.

Message Propagation		
	Flooding	**Failure detection**
RB	Figure 2	Figure 5
URB		Figure 6

Message Delivery		
	Flooding	**Failure detection**
RB	Figure 3	
URB	Figure 4	

Figure 7: The fragments used in the various algorithms

To summarize the construction of reliable broadcast algorithms from fragments, Figure 7 shows which fragement belongs to which algorithm(s).

4 Algorithm Evaluation

We evaluated the performance of the reliable broadcast algorithms discussed in this paper considering the number of messages exchanged by the processes and the latency to deliver a message. Latency is expressed in terms of δ, the maximum network message delay. For all algorithms, we consider executions without failures and, for failure detection-based algorithms, we also consider an execution where one process fails. We do not count the messages that processes send to themselves (i.e., loopback mechanism) and always consider the cost to deliver messages for the least favorable processes (e.g., in the failure detection-based algorithms, we assume that $sender(m)$ is not in $\{p_1, p_2, ..., p_{f+1}\}$). Table

1 presents our results (RB stands for Reliable Broadcast and URB for Uniform Reliable Broadcast).

Problem	**Flooding**	**Failure detection**	
	(no failures)	**(no failures)**	**(1 failure)**
RB	$n(n-1)$ δ	$n-1$ δ	$n(n-1)$ 3δ
URB	$n(n-1)$ 2δ	$(n-1)(f+2)$ 2δ	$n(n-1)$ 4δ

Table 1: Cost of broadcasting a message

With the flood-based propagation procedure, all n process sends a message to all but the process it received the message from, which amounts to $n(n-1)$ messages. In the reliable broadcast implementation, a process can deliver a message right after receiving it for the first time, and so, with latency δ. In the uniform reliable broadcast implementation, a process has to wait until it knows that $f+1$ processes have received the initial message, and so, the latency is 2δ.

With the failure detection-based procedure, and in the absence of failures, only the initial message is necessary for reliable broadcast. Without counting the message $sender(m)$ sends to itself, this amount to $n-1$ messages and latency of δ. For uniform reliable broadcast, however, processes also have to make sure that enough processes (i.e., $f+1$) received the initial message, which amounts to $n-1+(f+1)(n-1) = (n-1)(f+2)$ messages and latency of 2δ. In the presence of failures and suspicions, processes executing a failure detection-based algorithm end up sending messages to all processes, but only after suspecting some processes. Thus, in addition to the latency as in the failure-free case, processes first suspect a faulty process and then relay the message. Assuming that it takes δ to suspect a faulty process, this takes 3δ for reliable broadcast and 4δ for uniform reliable broadcast.

The results from Table 1 show that in terms of number of messages, failure detection-based algorithms are better than flood-based algorithms in the absence of failures, and as good as flood-based algorithms in the presence of failures. Failure detection-based algorithms, however, do not improve the latency of reliable broadcast algorithms. Actually, latency can get much worse in the presence of failures.

5 Discussion

This paper revisits in a systematic way implementations of reliable broadcast and uniform reliable broadcast. We recall three known implementations and propose a novel one. All algorithms are presented in the paper as variations of a propagation procedure, used by processes to rely messages, and a deliverable predicate, used by processes to know when to safely deliver messages. Simple assessment considering the number of messages and the latency to deliver messages showed an interesting tradeoff between flooding- and failure detection-based algorithms.

An open question for further research is whether stronger assumptions about the model (e.g., failure detectors) would allow for more efficient algorithms, in the presence and absence of failures. Further research should also address the impact of weaker communication links on the performance of the algorithms.

References

[ATD99] M. K. Aguilera, S. Toueg, and B. Deianov. On the weakest failure detector for uniform reliable broadcast. In *Proceedings of the International Symposium on Distributed Computing (DISC'99)*, September 1999.

[CT96] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[GS96] R. Guerraoui and A. Schiper. Consensus service: A modular approach for building fault-tolerant agreement protocols in distributed systems. In *Proceedings of the 26th International Symposium on Fault-Tolerant Computing (FTCS-26)*, pages 168–177, Sendai, Japan, June 1996.

[HT93] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In *Distributed Systems*, chapter 5. Addison-Wesley, 2nd edition, 1993.