

4 Consensus and Reliable Broadcasts (cont.)

4.5 Broadcast Problems

Our system model includes only point-to-point links for communication. If a server wants to broadcast a message to all others, the server may crash during the operation and it is possible that some servers receive a message but others don't. The purpose of *reliable broadcast* and its extensions is to prevent that. When additional ordering requirements are imposed (partial orders such as FIFO and causal or total order), the problem becomes harder to solve. This section is based on [HT93].

4.5.1 Reliable Broadcast

Reliable broadcast (RBC) requires that all correct servers deliver the same set of messages, and that this set includes all messages broadcast by correct servers but no spurious messages. The sender associated to a particular message is a distinguished server and its identity is assumed to be known. Formally, RBC is characterized by two events $r\text{-broadcast}(m)$, executed by the sender to “r-broadcast” the message m , and $r\text{-deliver}(m)$, executed by all servers when they “r-deliver” m .

When multiple messages are broadcast, one may imagine that the servers run multiple *instances* of a broadcast primitive. Every instance is associated with a unique identifier that is also added to all messages generated by the protocol; since the sender is known, this identifier may also include the identity of the sender.

Definition 4.13 (Reliable Broadcast). A protocol for *reliable broadcast*¹ satisfies:

Validity: If a correct server $r\text{-broadcasts}$ a message m , then it eventually $r\text{-delivers}$ m .

Agreement: If a server $r\text{-delivers}$ a message m , then all correct servers eventually $r\text{-deliver}$ m .

Integrity: Every server delivers at most one message m , and only if m was previously broadcast by the associated sender.

Thus if the sender is faulty, either all servers deliver a message or none.

Algorithm 4.14 (Reliable Broadcast). We consider the implementation of a single instance (a protocol for broadcasting multiple messages is obtained in a straightforward way by aggregating as many instances as there are messages). Let P_s denote the sender of the broadcast instance; server P_i executes the following steps:

¹This actually defines *uniform* reliable broadcast; all other broadcasts in this section are also uniform.

```

upon r-broadcast( $m$ ):           // sender  $P_s$  only
    send the message ( $\text{send}, m$ ) to itself
upon receiving message ( $\text{send}, m$ ):
    if message  $m$  has not been r-delivered yet then
        send the message ( $\text{send}, m$ ) to all
        r-deliver( $m$ )

```

Although our network model assumes reliable point-to-point links between all servers, the algorithm works even if every pair of correct servers is connected only via a path consisting entirely of correct servers (in which case the statement “send to all” means “send to all directly connected servers”). The following theorem is immediate.

Theorem 4.15. *Algorithm 4.14 implements reliable broadcast for $n > t$.*

4.5.2 FIFO Broadcast

When multiple messages are reliably broadcast concurrently, RBC does not guarantee anything about the order in which the messages are delivered. One of the simplest orderings is provided by FIFO broadcast, which guarantees that messages from the same sender are delivered in the same sequence as they were broadcast by the sender; this does not affect messages from different senders.

A protocol for *FIFO broadcast* is a protocol for reliable broadcast defined in terms of two events *f-broadcast* and *f-deliver* that also satisfies:

FIFO Order: If a server *f-broadcasts* a message m before it *f-broadcasts* a message m' , then no server *f-delivers* m' unless it has previously *f-delivered* m .

Algorithm 4.16 (FIFO Broadcast from Reliable Broadcast). Given an implementation of reliable broadcast, server P_i executes the following steps:

initialization:

```

 $\mathcal{M} \leftarrow []$            // set of received but not f-delivered messages
 $s \leftarrow 0$              //  $P_i$ 's sequence number
 $n_j \leftarrow 0$    ( $\forall j \in [1, n]$ ) // next sequence number to be f-delivered from  $P_j$ 

```

```

upon f-broadcast( $m$ ):           // sender  $P_s$  only
    r-broadcast the message ( $s, m$ )
     $s \leftarrow s + 1$ 

```

```

upon r-delivering ( $s', m'$ ) with sender  $P_j$ :
     $\mathcal{M} \leftarrow \mathcal{M} \cup \{(j, s', m')\}$ 
    while  $\exists (j, t, m) \in \mathcal{M}$  such that  $t = n_j$  do
        f-deliver( $m$ )
         $n_j \leftarrow n_j + 1$ 

```

Theorem 4.17. *Given a protocol for reliable broadcast, Algorithm 4.16 implements FIFO broadcast.*

4.5.3 Causal Broadcast

The causal precedence relation is an important concept in distributed computing. An event e *causally precedes* f , written $e \rightarrow f$, whenever the same server executes e before f , or when e is the event of sending a message and f the event of receiving it, or if there is an event g such that $e \rightarrow g$ and $g \rightarrow f$. Causal order is a specialization of FIFO order.

A protocol for *causal broadcast* is a protocol for reliable broadcast defined in terms of two events c -broadcast and c -deliver that also satisfies:

Causal Order: The c -broadcast of a message m causally precedes the c -broadcast of a message m' , then no server c -delivers m' unless it has previously c -delivered m .

Algorithm 4.18 (Causal Broadcast from FIFO Broadcast). Given an implementation of FIFO broadcast, server P_i executes the following steps:

initialization:

$M \leftarrow \emptyset$ // list of recently c -delivered messages

upon c -broadcast(m): // sender P_s only

f -broadcast the message $(M||m)$, where $||$ means to append an element m to a list M

$M \leftarrow \perp$

upon f -delivering ($[m_1, m_2, \dots, m_l]$):

for $k = 1, \dots, l$ **do**

if m_k has not been c -delivered yet **then**

c -deliver(m_k)

$M \leftarrow M||m_k$

Theorem 4.19. Given an implementation of FIFO broadcast, Algorithm 4.18 implements causal broadcast.

4.5.4 Atomic Broadcast

FIFO and causal orders are partial orders. In particular, causal order does not impose anything for two causally *unrelated* messages and it is possible that the servers deliver the messages in different orders. Many applications do not allow such behavior because they must maintain a consistent state at all servers; these applications require that the same state updates are executed by all servers and that every server executes them in the same order. Such a total order is provided by atomic broadcast.

A protocol for *atomic broadcast* is a protocol for reliable broadcast defined in terms of two events a -broadcast and a -deliver that also satisfies:

Total Order: If two servers P_i and P_j both a -deliver messages m and m' , then P_i a -delivers m before m' if and only if P_j a -delivers m before m' .

Note that *total order* does not imply *FIFO* or *causal order*; hence, FIFO and causal broadcasts are orthogonal to atomic broadcast, and it is possible to consider also FIFO atomic and causal atomic broadcasts.

Implementing the total order property is considerably more difficult than the other orderings considered before. In fact, atomic broadcast is as powerful as consensus and hence impossible in asynchronous networks using deterministic protocols.

Theorem 4.20. *Given a protocol for atomic broadcast, there is a protocol for consensus that does not involve any additional messages.*

Proof sketch. To propose a value v , a server uses the atomic broadcast protocol and *a-broadcasts* v ; then every server waits for the *a-delivery* of the *first* message v' and *decides* for v' . The *agreement* and *total order* properties of atomic broadcast imply *agreement* of consensus. \square

A convenient way to implement atomic broadcast is to use a consensus primitive. The atomic broadcast algorithm below proceeds in global rounds; it uses one instance of consensus in every round to agree on a set of messages, which are then delivered in a fixed order at the end of the round.

Algorithm 4.21 (Atomic Broadcast from Consensus and Reliable Broadcast [CT96]). Given an implementation of consensus and reliable broadcast, server P_i executes the following steps:

initialization:

$\mathcal{R} \leftarrow \emptyset$ // set of *r-delivered* messages
 $\mathcal{A} \leftarrow \emptyset$ // set of *a-delivered* messages
 $r \leftarrow 0$ // round number

upon *a-broadcast*(m):

r-broadcast(m)

upon *r-deliver*(m):

$\mathcal{R} \leftarrow \mathcal{R} \cup \{m\}$

repeat forever: // concurrently with the above statements

if $\mathcal{R} \setminus \mathcal{A} \neq \emptyset$ **then**

propose($\mathcal{R} \setminus \mathcal{A}$) in consensus r

wait for *decide*(\mathcal{S}) of consensus r

a-deliver all messages in $\mathcal{S} \setminus \mathcal{A}$ in some deterministic order

$\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{S}$

$r \leftarrow r + 1$

Theorem 4.22. *Given protocols for consensus and for reliable broadcast, Algorithm 4.21 implements atomic broadcast.*

Proof sketch. *Validity* follows from the *validity* of reliable broadcast and from the *validity* and *agreement* of consensus (if a correct server *a-broadcasts* a message m , it is eventually contained in the set \mathcal{R} of every correct server) combined with the *integrty* of consensus (eventually, every set proposed in consensus contains m).

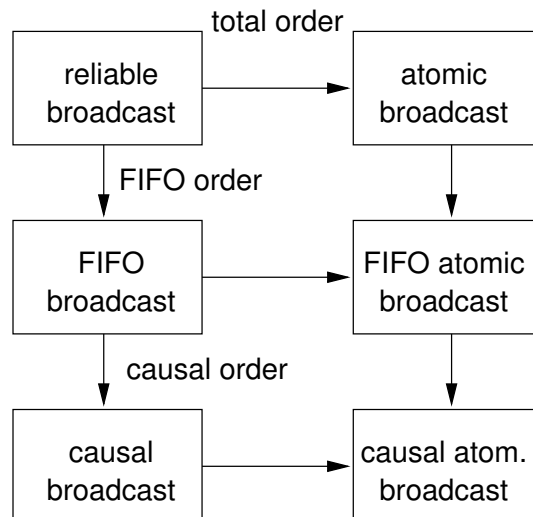
Agreement and total order are based on the following two facts. Let $\mathcal{B}_r(i)$ denote the set $\mathcal{S} \setminus \mathcal{A}$ of server P_i in round r of the algorithm and suppose P_i and P_j are correct. Then:

- If P_i executes *propose* for consensus r , then P_j eventually executes *propose* for consensus r .
- If P_i *a-delivers* all messages in $\mathcal{B}_r(i)$, then P_j eventually *a-delivers* all messages in $\mathcal{B}_r(j)$; moreover, $\mathcal{B}_r(i) = \mathcal{B}_r(j)$ for all $r \geq 0$.

□

Corollary 4.23. *Atomic broadcast and consensus are equivalent in asynchronous distributed systems with reliable point-to-point links and crash failures.*

4.5.5 Summary



Relations among the broadcast primitives [HT93].

References

- [CT96] T. D. Chandra and S. Toueg, *Unreliable failure detectors for reliable distributed systems*, Journal of the ACM **43** (1996), no. 2, 225–267.
- [HT93] V. Hadzilacos and S. Toueg, *Fault-tolerant broadcasts and related problems*, Distributed Systems (S. J. Mullender, ed.), ACM Press & Addison-Wesley, New York, 1993, Expanded version appears as Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca NY, 1994.