Distributed Algorithms
Reliable Broadcast

Alberto Montresor
University of Trento, Italy

2016/04/26

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
Contents

1 Introduction

2 Broadcast specifications and protocols
   - Best-Effort Broadcast
   - Reliable Broadcast
   - Uniform Reliable Broadcast

3 Message ordering
   - Introduction
   - Specification
   - A modular approach
   - Algorithms and proofs
   - Atomic Broadcast
Introduction

Efficient techniques are required, capable of supporting consistent behavior between system components in spite of failures.

Examples

- **Reliable Broadcast/Multicast protocols:**
  Ensure reliable message delivery to all participants

- **Agreement protocols:**
  Ensure all participants to have a consistent system view

- **Commit protocols:**
  Implement atomic behavior in transactional types of systems
Broadcast
Broadcast Protocol Layering
Basic assumptions (1)

- System is asynchronous
  - No bounds on messages and process execution delays

- Processes fail by crashing
  - stop executing actions after the crash
  - We do not consider Byzantine failures

- Correct/Faulty
  - A process that does not fail in a run is correct in that run
  - Otherwise, the process is faulty
Basic assumptions (2)

We will consider two failure models for communication:

**No Failures**

- **Validity**: If $p$ sends a message to $q$, and $q$ is correct, then $q$ will eventually receive $m$

- **Integrity**: No message is delivered to a process more than once, and only if it has been sent previously

**Perfect Channels**

- **Validity**: If $p$ sends a message to $q$, and $p,q$ are correct, then $q$ will eventually receive $m$

- **Integrity**: No message is delivered to a process more than once, and only if it has been sent previously
What kind of underlying network?

- **Complete graph**
  - Every process can communicate with every other process
  - A routing substrate realizes this abstraction

- **Point-to-point**
  - Every process can communicate with a subset of processes (its neighbors)
  - Routing is not implemented at the send/receive level (we may implement it at the level of our protocols)
Different flavors of broadcast

- Reliability
  - Best-effort
  - Reliable
  - Uniform Reliable

- Ordering
  - FIFO
  - Casual
  - Atomic
  - FIFO Atomic
  - Causal Atomic

- Time bounds
  - Timed Reliable

- Primitives
  - R-Broadcast
  - F-Broadcast
  - C-Broadcast
  - ...
Best-effort broadcast – Specification

**Definition (BEB1 – Validity)**

If $p$ and $q$ are correct, then every message B-broadcast by $p$ is eventually delivered by $q$.

**Definition (BEB2 – Uniform Integrity)**

$m$ is delivered by a process at most once, and only if it was previously broadcast.
Best-effort broadcast – Algorithm

Best-effort broadcast protocol executed by $p$

upon $\text{B-broadcast}(m)$ do
  foreach $q \in \Pi$ do
    send $m$ to $q$

upon $\text{receive}(m)$ do
  $\text{B-deliver}(m)$

Notation – Send to all

foreach $q \in \Pi$ do
  send $m$ to $q$

is equivalent to

send $m$ to $\Pi$
Best-effort broadcast – Proof

We can show that the protocol works with *Perfect Channels*:

- **BEB1 - Validity**: By the Validity property of Perfect Channels and the very facts that
  1. the sender sends the message to all
  2. every correct process that receives a message B-delivers it

- **BEB2 – Uniform Integrity**: By the Integrity property of Perfect Channels

Clearly, it will work also with No Failures
Best-effort broadcast – Example
Best-effort broadcast – Example
Best-effort broadcast – Problem

What happens if the sender fails?

- Even in the absence of communication failures:
  - if the sender crashes before being able to send the message to all
  - some processes will not deliver the message

What we do?

- First we revise the specification of broadcast
- Then we implement the new specification
Reliable Broadcast – Specification

Definition (RB1 – Validity)
If a correct process broadcasts $m$, then it eventually delivers $m$

Definition (RB2 – Uniform Integrity)
$m$ is delivered by a process at most once, and only if it was previously broadcast

Definition (RB3 – Agreement)
If a correct process delivers $m$, then all correct processes eventually deliver $m$
Reliable Broadcast – Scenario 1

Does this execution satisfy the RB specification?
Obviously yes – the fact that process $p_1$ does not deliver $m$ is not a problem, because Validity only requires correct processes to deliver their own messages.
Reliable Broadcast – Scenario 2

Does this execution satisfy the RB specification?
Obviously yes – the fact that no process delivers $m$ is not a problem, because process $p_1$ is faulty and Validity does not apply; and nobody delivers $m$, so Agreement does not apply.
Reliable Broadcast – Scenario 3

Does this execution satisfy the RB specification?
Obviously no – Agreement is not satisfied.
Reliable Broadcast – Algorithm v.1

Reliable broadcast protocol executed by $p$

upon initialization do

  Set $delivered \leftarrow \emptyset$ % Messages already delivered

upon $R$-broadcast($m$) do

  send $m$ to $\Pi - \{p\}$
  R-deliver($m$)
  $delivered \leftarrow delivered \cup \{m\}$

upon receive($m$) from $q$ do

  if not $m \in delivered$ then
    send $m$ to $\Pi - \{p, q\}$
    R-deliver($m$)
    $delivered \leftarrow delivered \cup \{m\}$
Reliable Broadcast – Scenario 4

Does this execution satisfy the RB specification?

Graphical representation of the scenario with processes and messages.
Yes, because before R-delivering the message, $p_2$ forwards it to all other processes.
Reliable Broadcast – Proof

Algorithm v.1 implements Reliable Broadcast.

- **RB1 – Validity:** If a correct process broadcasts m, then it eventually delivers m
  By the code implementing R-broadcast.

- **RB2 – Agreement:** If a correct process delivers m, then all correct processes eventually deliver m
  Before R-delivering m, a correct process p forwards m to all processes. By Validity of Perfect Channels and the fact that p is correct, all correct processes will eventually receive m and R-deliver it.

- **RB3 – Integrity:** m is delivered by a process at most once, and only if it was previously broadcast
  By the Integrity of Perfect Channels and the use of variable delivered
Reliable Broadcast – Scenario 5

Does this execution satisfy the RB specification?

\[ p_1 \]
\[ m_1 \]
\[ p_2 \]
\[ m_1 \]
\[ p_3 \]
\[ m_1 \]
Yes, because the fact that $m_1$ has been delivered by $p_1$ and $p_2$, which are not correct, does not imply that $m_1$ has to be delivered by $p_3$ as well. Yet, this situation is not desirable, because two processes deliver something and another one does not.
Uniform Reliable Broadcast – Specification

**Definition (URB1 – Validity)**
If a correct process broadcasts $m$, then it eventually delivers $m$

**Definition (URB2 – Uniform Agreement)**
If a correct process delivers $m$, then all correct processes eventually deliver $m$

**Definition (URB3 — Uniform Integrity)**
$m$ is delivered by a process at most once, and only if it was previously broadcast
**Uniform Reliable Broadcast – Proof**

Algorithm v.1 implements Uniform Reliable Broadcast...
... but under different assumptions!

- **URB1, URB2**: As RB1, RB2

- **RB3 – Uniform Agreement**: If a process delivers \( m \), then all correct processes eventually deliver \( m \)
  - Before R-delivering \( m \), a process forwards \( m \) to all processes.
  - By Validity of Perfect Channels, all correct processes will eventually receive \( m \) and R-deliver it.
  - In the absence of communication failures, all correct processes will eventually receive \( m \) and R-deliver it.
Message ordering

Problem

- Given the asynchronous nature of distributed systems, messages may be delivered in any order.
- Some services, such as replication, need messages to be delivered in a consistent manner, otherwise replicas may diverge.

Solution

We describe a collection of ordering policies and we show how to implement them in a modular way.
Message ordering

Definition (FIFO Order)

If a process $p$ broadcasts a message $m$ before it broadcasts a message $m'$, the no correct process delivers $m'$ unless it has previously delivered $m$

$$\text{broadcast}_p(m) \rightarrow \text{broadcast}_p(m') \Rightarrow \text{deliver}_q(m) \rightarrow \text{deliver}_q(m')$$
**Definition (Causal Order)**

If the broadcast of a message $m$ *happens-before* the broadcast of a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$

$$\text{broadcast}_p(m) \rightarrow \text{broadcast}_q(m') \Rightarrow \text{deliver}_r(m) \rightarrow \text{deliver}_r(m')$$

Is this causal? No!
Definition (Causal Order)

If the broadcast of a message $m$ happens-before the broadcast of a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$

$$\text{broadcast}_p(m) \rightarrow \text{broadcast}_q(m') \Rightarrow \text{deliver}_r(m) \rightarrow \text{deliver}_r(m')$$

Is this causal? Yes!
Definition (Causal Order)

If the broadcast of a message $m$ happens-before the broadcast of a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$

$$broadcast_p(m) \rightarrow broadcast_q(m') \Rightarrow deliver_r(m) \rightarrow deliver_r(m')$$

Is this causal? Yes!
Message ordering

Problem

Causal Broadcast does not impose any order on messages not causally related

Example

- Consider a replicated database with two copies of a bank account
- Initially, \( account = 1000\$ \)
- A user deposits 150\$ triggering a broadcast of \( m_1 = \{ \text{add 150\$ to } account \} \)
- At the same time the bank initiates a broadcast of \( m_2 = \{ \text{add 2\% interest to } account \} \)
- Causal Broadcast allows two processes to deliver these updates in different order, creating inconsistency
**Definition (Total Order)**

If correct processes $p$ and $q$ both deliver messages $m, m'$, then $p$ delivers $m$ before $m'$ if and only if $q$ delivers $m$ before $m'$

$$\text{deliver}_p(m) \rightarrow \text{deliver}_p(m') \Rightarrow \text{deliver}_q(m) \rightarrow \text{deliver}_q(m')$$

Is this totally ordered? No!
Definition (Total Order)

If correct processes $p$ and $q$ both deliver messages $m, m'$, then $p$ delivers $m$ before $m'$ if and only if $q$ delivers $m$ before $m'$

\[
\text{deliver}_p(m) \rightarrow \text{deliver}_p(m') \Rightarrow \text{deliver}_q(m) \rightarrow \text{deliver}_q(m')
\]

Is this totally ordered? Yes!
Uniform Versions

**Definition (Uniform FIFO Order)**

If a process $p$ broadcasts a message $m$ before it broadcast a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$

$$broadcast_p(m) \rightarrow broadcast_p(m') \Rightarrow deliver_q(m) \rightarrow deliver_q(m')$$

**Definition (Uniform Causal Order)**

If the broadcast of a message $m$ happens-before the broadcast of a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$

$$broadcast_p(m) \rightarrow broadcast_q(m') \Rightarrow deliver_r(m) \rightarrow deliver_r(m')$$

**Definition (Uniform Total Order)**

If correct processes $p$ and $q$ both deliver messages $m, m'$, then $p$ delivers $m$ before $m'$ if and only if $q$ delivers $m$ before $m'$

$$deliver_p(m) \rightarrow deliver_p(m') \Rightarrow deliver_q(m) \rightarrow deliver_q(m')$$
A modular approach to Broadcast

Reliable Broadcast

FIFO Broadcast

Causal Broadcast

Atomic Broadcast

FIFO Atomic Broadcast

Causal Atomic Broadcast

FIFO Order

Total Order

FIFO Order

Total Order

FIFO Order

Total Order

FIFO Order

Total Order

FIFO Order

Total Order

FIFO Order

Total Order
A modular approach to Broadcast

- Uniform Reliable Broadcast
  - Uniform FIFO Order
  - Uniform FIFO Broadcast
  - Uniform Causal Order
  - Uniform Causal Broadcast

- Uniform Atomic Broadcast
  - Uniform FIFO Order
  - Uniform FIFO Atomic Broadcast
  - Uniform Causal Order
  - Uniform Causal Atomic Broadcast

- Uniform Total Order
  - Uniform Total Order
  - Uniform Total Order
**Transformation**

**Informal definition**

A broadcast transformation is an algorithm that takes a weaker broadcast algorithm and transform it into a stronger version.

**Definition (Transformation)**

A transformation from problem $A$ to problem $B$ is an algorithm $T_{A \rightarrow B}$ that converts any algorithm $A$ that solves problem $A$ into an algorithm $B$ that solves problem $B$.

**Definition (Preservation)**

A transformation $T_{A \rightarrow B}$ preserves property $P$ if it converts any algorithm for $A$ into an algorithm that solves problem $B$ and also satisfies $P$. 
Transformation

- Properties of weakest RB must be preserved
  - **Uniform Integrity**: preserved in all transformations
    - No message is created
    - Messages are tagged to avoid re-delivery
  - **Validity, (Uniform) Agreement**:
    - To be proved case by case

- To add Total Order:
  - We cannot start from a simple reliable broadcast
  - We need stronger assumptions
**Definition (Blocking transformation)**

A transformation of one broadcast algorithm into another is **blocking** if the resulting broadcast algorithm has a run in which a process delays the delivery of a message for a later time.

**Example**

FIFO Order
FIFO Order – Algorithm

FIFO Order Transformation executed by process \( p \)

**upon** initialization **do**
- **Set** \( \text{buffer} \leftarrow \emptyset \)
- **integer**[] \( \text{next} \leftarrow \text{new integer}[1 \ldots |\Pi|] \)
- **foreach** \( q \in \Pi \text{ do } \text{next}[q] \leftarrow 1 \)

**upon** F-broadcast(\( m \)) **do**
- R-broadcast(\( m \))

**upon** R-deliver(\( m \)) **do**
- \( \text{buffer} \leftarrow \text{buffer} \cup \{m\} \)
- **while** \( \exists m' \in \text{buffer} : \text{sender}(m') = \text{sender}(m) \text{ and } \text{seqn}(m') = \text{next}[q] \text{ do } \)
  - F-deliver(\( m' \))
  - \( \text{next}[q] \leftarrow \text{next}[q] + 1 \)
  - \( \text{buffer} \leftarrow \text{buffer} - \{m'\} \)
FIFO Order – Proof

**Theorem**

For any process $p$, if $\text{next}_p[q] = k$ then $p$ has F-delivered the first $k - 1$ messages F-broadcast by $q$

**Theorem**

Suppose a correct process $p$ R-delivers a message $m$ from $q$ and F-delivers all the messages that $q$ F-broadcast before $m$. Then $p$ also F-delivers $m$

- Validity, (Uniform) Agreement, (Uniform) Total Order are preserved
- Uniform FIFO Order is satisfied
- The transformation is blocking
Claim 2 – Proof If $m'$ is the last message delivered from $\text{sender}(m)$, let $k$ be equal to $\text{seqn}(m')$. By claim 1, $\text{next}_p[q] = k + 1$ and by definition, $\text{seqn}(m) = k + 1$. Thus $m$ will be delivered as the next message.

Validity – Proof

- To F-broadcast $m$, a correct process $p$ R-broadcasts it.
- It will eventually R-deliver $m$ (Validity of RB)
- Suppose $m$ is the first message $p$ R-delivers, but not F-delivers For claim 2, it will eventually F-deliver $m$. Absurd.

Uniform FIFO Order – Proof

- Suppose a process $p$ F-delivers a message $m$ F-broadcast by $q$.
- Let $\text{seqn}(m) = k$.
- By the algorithm, just before F-delivering $m$, $\text{next}_p[q] = k$.
- By Claim 1, $p$ has already F-delivered all the $k - 1$ messages that $q$ F-broadcast before $m$, as wanted.
Causal Order - Algorithm

Two transformations:

- Both based on FIFO Reliable Broadcast
  - One is non-blocking
    - Each message is tagged with “recent history”
    - When a message is F-delivered, all the causal messages that have been F-delivered are locally delivered
    - Does this recall anything?

- One blocking
  - Based on vector clocks
Causal Order - Algorithm 1

Causal Order Transformation executed by process \( p \)

**upon initialization do**

- Set \( \text{delivered} \leftarrow \emptyset \) \% Messages already C-delivered
- \( \text{SEQUENCE } \text{recent} \leftarrow \langle \rangle \) \% Messages C-delivered since last C-broadcast

**upon C-broadcast(\( m \)) do**

- \( \text{F-broadcast}(\text{recent}||m) \)
- \( \text{recent} \leftarrow \langle \rangle \)

**upon F-deliver(\( \langle m_1, \ldots, m_k \rangle \)) do**

- for \( i \leftarrow 1 \) to \( k \) do
  - if not \( m_i \notin \text{delivered} \) then
    - \( \text{delivered} \leftarrow \text{delivered} \cup \{ m_i \} \)
    - \( \text{recent} \leftarrow \text{recent}||m_i \)
    - \( \text{C-deliver}(m_i) \)
Causal Order - Algorithm 1
Causal Order – Proof

- Validity, (Uniform) Agreement, (Uniform) Total Order are preserved
- Uniform Causal Order is satisfied
- The transformation is non-blocking
Causal Order - Algorithm 2

Causal Order Transformation executed by process $p$

upon initialization do
\[
\text{Set} \quad buffer \leftarrow \emptyset \quad \% \quad \text{Messages to be delivered}
\]
\[
\text{integer}[] \quad VC \leftarrow \{0, \ldots, 0\} \quad \% \quad \text{Vector clock}
\]

upon C-broadcast($m$) do
\[
\text{F-broadcast} (\langle m, VC \rangle)
\]

upon F-deliver($\langle m, TS \rangle$) do
\[
buffer \leftarrow buffer \cup \{\langle m, TS \rangle\}
\]
\[
\text{while} \quad \exists \langle m', TS' \rangle \in buffer : VC[\text{sender}(m')] = TS'[\text{sender}(m')] - 1 \land \\
\forall s \neq \text{sender}(m') : VC[s] \geq TS'[s] \text{ do}
\]
\[
\text{C-deliver}(m')
\]
\[
\text{update} \quad VC
\]
\[
buffer \leftarrow buffer - \{m\}
\]
Causal Order - Algorithm 2

$[1,0,0]$  $m_1$  $p_1$
$[1,2,0]$  $m_2$

$[1,2,0]$  $m_1$  $m_2$  $p_2$

$[1,0,0]$  $m_1$  $m_2$  $p_3$
Causal Order – Proof

- Validity, (Uniform) Agreement, (Uniform) Total Order are preserved
- Uniform Causal Order is satisfied
- The transformation is blocking
A modular approach to Broadcast

(Uniform) Reliable Broadcast

(Uniform) FIFO Broadcast

(Uniform) Causal Broadcast

(Uniform) Atomic Broadcast

(Uniform) FIFO Order

(Uniform) Total Order

(Uniform) Causal Order
Atomic Broadcast

There are three approaches:

1. We add synchronous assumptions to our system

2. We show that the Atomic Broadcast problem is equivalent to the Consensus problem
   - There is an algorithm $T_{\text{Consensus} \rightarrow \text{AtomicBroadcast}}$
   - There is an algorithm $T_{\text{AtomicBroadcast} \rightarrow \text{Consensus}}$

3. Through a coordinator (actual implementation, see later in group communication)
Timed Reliable Broadcast

Definition ((Uniform) Real-Time $\Delta$-Timeliness)
There is a known constant $\Delta$ such that if a message $m$ is broadcast at real-time $t$, then no correct (any) process delivers $m$ after real-time $t + \Delta$

Definition ((Uniform) Local-Time $\Delta$-Timeliness)
There is a known constant $\Delta$ such that no correct (any) process delivers $m$ after local time $TS(m) + \Delta$ on $p$’s clock, where $TS(m)$ is the timestamp obtained by the local clock of the sender

Note
(Uniform) Real-Time $\Delta$-Timeliness $\Rightarrow$
(Uniform) Local-Time $\Delta$-Timeliness
Atomic Broadcast, Algorithm 1

Total Order Transformation executed by process \( p \)

upon A-broadcast(\( m \)) do

| T-broadcast(\( m \))

upon T-deliver(\( m \)) do

| schedule A-deliver(\( m \)) at time \( TS(m) + \Delta \)
In the (Uniform) Consensus problem, the processes propose values and need to decide (agree) on one of these values.

**Definition (Uniform Validity)**
Any value decided is a value proposed

**Definition ((Uniform) Agreement)**
No two correct (any) processes decide differently

**Definition (Termination)**
Every correct process eventually decides

**Definition (Uniform Integrity)**
Every process decides at most once
From Atomic Broadcast to Consensus

Transformation executed by process $p$

upon initialization do
    boolean decided ← false

upon propose($v$) do
    A-broadcast($v$)

upon A-deliver($v$) do
    if not decided then
        decided ← true
        decide($u$)
From Consensus to Atomic Broadcast

Transformation executed by process $p$

**upon** initialization **do**

- **Set** unordered $\leftarrow \emptyset$ \hfill \% Messages to be ordered
- **Set** delivered $\leftarrow \emptyset$ \hfill \% Messages already delivered
- **boolean** wait $\leftarrow \text{false}$ \hfill \% \text{true} when Consensus is running
- **integer** $s \leftarrow 1$ \hfill \% Consensus protocol identifier

**upon** A-broadcast$(m)$ **do**

- R-broadcast$(m)$

**upon** R-deliver$(m)$ **do**

- **if** not $m \in$ delivered **then**
  - unordered $\leftarrow$ unordered $\cup \{m\}$
Transformation executed by process $p$

\[
\text{upon } \text{decide}_s(S) \text{ do }
\]
\[
\quad \text{unordered} \leftarrow \text{unordered} - S
\]
\[
\quad \text{foreach } m \in S \text{ do }
\]
\[
\quad \quad \text{A-deliver}(m) \quad \% \quad \text{In some deterministic order}
\]
\[
\quad \text{delivered} \leftarrow \text{delivered} \cup S
\]
\[
\quad s \leftarrow s + 1
\]
\[
\quad \text{wait} \leftarrow \text{false}
\]

\[
\text{upon } \text{unordered} \neq \emptyset \text{ and not } \text{wait} \text{ do }
\]
\[
\quad \text{wait} \leftarrow \text{true}
\]
\[
\quad \text{propose}_s(\text{unordered})
\]
Conclusions

Summary
Consensus and total order broadcast are equivalent problems in an asynchronous system with crashes and Perfect Channels

- Consensus can be obtained from total order broadcast
- Total order broadcast can be obtained from Consensus

Problem
But in this way, we have moved the problem from Atomic Broadcast to Consensus.
Next step: can we solve Consensus?
