? ? ?

JOLY Amaury
Encadrants : GODARD Emmanuel, TRAVERS Corentin

Aix-Marseille Université, Scille

15 mai 2025

1 Introduction

1.1 Model

1.1.1 Model Properties

The system consists of n asynchronous processes communicating via reliable point-to-point message passing.
Each process has a unique, unforgeable identifier and knows the identifiers of all other processes.
Up to $f < n$ processes may crash (fail-stop).
The network is reliable : if a correct process sends a message to another correct process, it is eventually delivered.
Messages are uniquely identifiable : two messages sent by distinct processes or at different rounds are distinguishable
2 messages sent by the same processus in two differents rounds are differents

Property 1 (Message Uniqueness) *If two messages are sent by different processes, or by the same process in different rounds, then the messages are distinct.*
Formally :

$$\forall p_1, p_2,\ \forall r_1, r_2,\ \forall m_1, m_2,\ \left(\begin{array}{l} send(p_1, r_1, m_1) \land send(p_2, r_2, m_2) \\ \land\ (p_1 \neq p_2 \lor r_1 \neq r_2) \end{array} \right) \Rightarrow m_1 \neq m_2$$

1.1.2 Reliable Broadcast Properties

Property 2 *Integrity Every message received was previously sent.*
Formally :
$\forall p_i : bc\text{-}recv_i(m) \Rightarrow \exists p_j : bc\text{-}send_j(m)$

Property 3 *No Duplicates No message is received more than once at any single processor.*
Formally :
$\forall m, \forall p_i : bc\text{-}recv_i(m)$ *occurs at most once*

Property 4 *Validity All messages broadcast by a correct process are eventually received by all non faulty processors.*
Formally :
$\forall m, \forall p_i : correct(p_i) \land bc\text{-}send_i(m) => \forall p_j : correct(p_j) \Rightarrow bc\text{-}recv_j(m)$

1.1.3 AtomicBroadcast Properties

Property 5 *AB Totally ordered* $\forall m_1, m_2, \forall p_i, p_j\ :\ ab\text{-}recv_{p_i}(m_1) < ab\text{-}recv_{p_i}(m_2) \Rightarrow ab\text{-}recv_{p_j}(m_1) < ab\text{-}recv_{p_j}(m_2)$

1.1.4 DenyList Properties

Let Π_M be the set of processes authorized to issue `APPEND` operations, and Π_V the set of processes authorized to issue `PROVE` operations.
Let S be the set of valid values that may be appended. Let `Seq` be the linearization of operations recorded in the DenyList.

Property 6 *APPEND Validity An operation `APPEND`(x) is valid iff : the issuing process $p \in \Pi_M$, and the value $x \in S$*

Property 7 *PROVE Validity An operation* $\texttt{PROVE}(x)$ *is valid iff : the issuing process* $p \in \Pi_V$*, and there exists no* $\texttt{APPEND}(x)$ *that appears earlier in* $\texttt{Seq}$*.*

Property 8 *PROGRESS If an APPEND(x) is invoked by a correct process, then all correct processes will eventually be unable to PROVE(x).*

Property 9 *READ Validity READ() return a list of tuples who is a random permutation of all valids PROVE() associated to the identity of the emiter process.*

1.2 Algorithms

We consider a set of processes communicating asynchronously over reliable point-to-point channels. Each process maintains the following shared variables :

— **received** : the set of messages received (but not yet delivered).
— **delivered** : the set of messages that have been received, ordered, and delivered.
— **prop**[r][j] : the proposal set of process j at round r. It contains the set of messages that process j claims to have received but not yet delivered at round r, concatenated with its newly broadcast message.
— **proves** : the current content of the `DenyList` registry, accessible via the operation `READ()`. It returns a list of tuples $(j, \texttt{PROVE}(r))$, each indicating that process j has issued a valid `PROVE` for round r.
— **winner**^{r} : the set of processes that have issued a valid `PROVE` operation for round r.
— **RB-cast** : a reliable broadcast primitive that satisfies the properties defined in Section 1.1.2.
— **APPEND**(r), **PROVE**(r) : operations that respectively insert (APPEND) and attest (PROVE) the participation of a process in round r in the DenyList registry.
— **READ()** : retrieves the current local view of valid operations (APPENDs and PROVEs) from the DenyList.
— **ordered**(S) : returns a deterministic total order over a set S of messages (e.g., via hash or lexicographic order).

1.3 proof

Theorem 1 (Integrity) *If a message* m *is delivered by any process, then it was previously broadcast by some process via the* $\texttt{AB-broadcast}$ *primitive.*

Proof 1 *Let* j *be a process such that AB-deliver*$_j(m)$ *occurs.*

$$
\begin{array}{ll}
\quad \textit{AB-deliver}_j(m) & \textit{(line 24)} \\
\Rightarrow \exists r_0 : m \in \texttt{ordered}(M^{r_0}) & \textit{(line 22)} \\
\Rightarrow \exists j_0 : j_0 \in \textit{winner}^{r_0} \wedge m \in \textit{prop}[r_0][j_0] & \textit{(line 21)} \\
\Rightarrow \exists m_0, S_0 : \textit{RB-received}_{j_0}(m_0, S_0, r_0, j_0) \wedge m \in S_0 & \textit{(line 2)} \\
\Rightarrow S_0 = (\textit{received}_{j_0} \setminus \textit{delivered}_{j_0}) \cup \{m_1\} & \textit{(line 5)} \\
\Rightarrow \textbf{\textit{if}}\ m_1 = m : \exists\, \textit{AB-broadcast}_{j_0}(m) \quad \square & \\
\quad \textbf{\textit{else if}}\ m_1 \neq m : & \\
\qquad m \in \textit{received}_{j_0} \setminus \textit{delivered}_{j_0} \Rightarrow m \in \textit{received}_{j_0} \wedge m \notin \textit{delivered}_{j_0} & \\
\qquad \exists j_1, S_1, r_1 : \textit{RB-received}_{j_1}(m, S_1, r_1, j_1) & \textit{(line 1)} \\
\qquad \Rightarrow \exists\, \textit{AB-broadcast}_{j_1}(m) \quad \square & \textit{(line 5)}
\end{array}
$$

Theorem 2 (No Duplication) *No message is delivered more than once by any process.*

RB-received(m, S, r_0, j_0)

```
received ← received ∪ {m}
prop[r_0][j_0] ← S
```

AB-broadcast(m, j_0)

```
proves ← READ()
r_0 ← max{r : ∃j, (j, PROVE(r)) ∈ proves} + 1
RB-cast(m, (received \ delivered) ∪ {m}, r_0, j_0)
PROVE(r_0)
APPEND(r_0)
repeat
    proves ← READ()
    r_1 ← max{r : ∃j, (j, PROVE(r)) ∈ proves} − 1
    winner^{r_1} ← {j : (j, PROVE(r_1)) ∈ proves}
    wait ∀j ∈ winner^{r_1}, prop[r_1][j] ≠ ⊥
until ∀r_2, ∃j_2 ∈ winner^{r_2}, m ∈ prop[r_2][j_2]
```

AB-listen

```
while true do
    proves ← READ()
    r_1 ← max{r : ∃j, (j, PROVE(r)) ∈ proves} − 1
    for r_2 ∈ [r_0, ..., r_1] do
        APPEND(r_2)
        proves ← READ()
        winner^{r_2} ← {j : (i, PROVE(r_2)) ∈ proves}
        wait ∀j ∈ winner^{r_2}, prop[r_2][j] ≠ ⊥
        M^{r_2} ← ⋃_{j ∈ winner^{r_2}} prop[r_2][j]
        for all m ∈ ordered(M^{r_2}) do
            delivered ← delivered ∪ {m}
            AB-deliver(m)
```

Proof 2 *Let j be a process such that both AB-deliver$_j(m_0)$ and AB-deliver$_j(m_1)$ occur, with $m_0 = m_1$.*

$$
\begin{aligned}
& AB\text{-}deliver_j(m_0) \wedge AB\text{-}deliver_j(m_1) && \textit{(line 24)} \\
\Rightarrow\ & m_0, m_1 \in delivered_j && \textit{(line 23)} \\
\Rightarrow\ & \exists r_0, r_1 : m_0 \in M^{r_0} \wedge m_1 \in M^{r_1} && \textit{(line 22)} \\
\Rightarrow\ & \exists j_0, j_1 : m_0 \in prop[r_0][j_0] \wedge m_1 \in prop[r_1][j_1] \\
& \quad \wedge\ j_0 \in winner^{r_0},\ j_1 \in winner^{r_1} && \textit{(line 21)}
\end{aligned}
$$

We now distinguish two cases :

Case 1 : $r_0 = r_1$ *:*

- *If $j_0 \neq j_1$: this contradicts message uniqueness, since two different processes would include the same message in round r_0.*
- *If $j_0 = j_1$:*

$$
\begin{aligned}
& \Rightarrow |(j_0, \mathtt{PROVE}(r_0)) \in proves| \geq 2 && \textit{(line 19)} \\
& \Rightarrow \mathtt{PROVE}_{j_0}(r_0) \textit{ occurs 2 times} && \textit{(line 6)} \\
& \Rightarrow \mathtt{AB\text{-}Broadcast}_{j_0}(m_0) \textit{ were invoked two times} \\
& \Rightarrow (max\{r : \exists j, (j, \mathtt{PROVE}(r)) \in proves\} + 1) && \textit{(line 4)} \\
& \quad \textit{returned the same value in two differents invokations of } \mathtt{AB\text{-}Broadcast} \\
& \quad \textbf{\textit{But }} \mathtt{PROVE}(r_0) \Rightarrow \mathtt{max}\{r : \exists j, (j, \mathtt{PROVE}(r)) \in proves\} + 1 > r_0 \\
& \quad \textit{It's impossible for a single process to submit two messages in the same round}
\end{aligned}
$$

Case 2 : $r_0 \neq r_1$ *:*

- *If $j_0 \neq j_1$: again, message uniqueness prohibits two different processes from broadcasting the same message in different rounds.*
- *If $j_0 = j_1$: message uniqueness also prohibits the same process from broadcasting the same message in two different rounds.*

In all cases, we reach a contradiction. Therefore, it is impossible for a process to deliver the same message more than once. □

1.3.1 Broadcast Validity

$\exists j_0, m_0 \quad AB\_broadcast_{j_0}(m_0) \Rightarrow \forall j_1 \quad AB\_received_{j_1}(m_0)$

Proof :

$$\exists j_0, m_0 \quad AB\_broadcast_{j_0}(m_0)$$
$$\forall j_1, \exists r_1 \quad RB\_deliver_{j_1}^{r_1}(m_0)$$
$$\exists receieved : m_0 \in receieved_{j_1}$$
$$\exists r_0 : RB\_deliver(PROP, r_0, m_0) \qquad LOOP$$
$$\exists prop : \text{prop}[r_0][j_0] = m_0$$
$$\text{if} \quad \nexists(j_0, PROVE(r_0)) \in \text{proves}$$
$$\quad r_0 += 1$$
$$\quad \text{jump to LOOP}$$
$$\text{else}$$
$$\quad \exists \text{winner}, \text{winner}^{r_0} \ni j_0$$
$$\quad \exists M^{r_0} \ni (\text{prop}[r_0][j_0] = m_0)$$
$$\quad \forall j_1, \quad AB\_deliver_{j_1}(m_0)$$

AB receive width

$$\exists j_0, m_0 \quad AB\_deliver_{j_0}(m_0) \Rightarrow \forall j_1 \ AB\_deliver_{j_1}$$

Proof :

$$\forall j_0, m_0 \ AB\_deliver_{j_0}(m_0) \Rightarrow \exists j_1 \text{ correct }, AB\_broadcast(m_0)$$
$$\exists j_0, m_0 \quad AB\_broadcast_{j_0}(m_0) \Rightarrow \forall j_1, \ AB\_deliver_{j_1}(m_0)$$