Abstract

We show how to upgrade a Reliable Broadcast (RB) primitive to Atomic Reliable Broadcast (ARB) by leveraging a synchronous DenyList (DL) object. In a purely asynchronous message-passing model with crashes, ARB is impossible without additional power. The DL supplies this power by enabling round closing and agreement on a set of "+winners" for each round. We present the algorithm, its safety arguments, and discuss liveness and complexity under the assumed synchrony of DL.

Keywords Atomic broadcast, total order broadcast, reliable broadcast, consensus, synchrony, shared object, linearizability.

1 Introduction

Atomic Reliable Broadcast (ARB)—a.k.a. total order broadcast—ensures that all processes deliver the same sequence of messages. In asynchronous message-passing systems with crashes, implementing ARB is impossible without additional assumptions, as it enables consensus. We assume a synchronous DenyList (DL) object and demonstrate how to combine DL with an asynchronous RB to realize ARB.

2 Model

We consider a static set of n processes with known identities, communicating by reliable point-to-point channels, in a complete graph. Messages are uniquely identifiable.

Synchrony. The network is asynchronous. Processes may crash; at most f crashes occur.

Communication. Processes can exchange through a Reliable Broadcast (RB) primitive (defined below) which's invoked with the functions $\mathsf{RB\text{-}cast}(m)$ and $\mathsf{RB\text{-}received}(m)$. There exists a shared object called DenyList (DL) (defined below) that is interfaced with the functions $\mathsf{APPEND}(x)$, $\mathsf{PROVE}(x)$ and $\mathsf{READ}()$.

Notation. Let Π be the finite set of process identifiers and let $n \triangleq |\Pi|$. Two authorization subsets are $\Pi_M \subseteq \Pi$ (processes allowed to issue APPEND) and $\Pi_V \subseteq \Pi$ (processes allowed to issue PROVE). Indices $i, j \in \Pi$ refer to processes, and p_i denotes the process with identifier i. Let $\mathcal{M}$ denote the universe of uniquely identifiable messages, with $m \in \mathcal{M}$. Let $\mathcal{R} \subseteq \mathbb{N}$ be the set of round identifiers; we write $r \in \mathcal{R}$ for a round. We use the precedence relation $\prec$ for the DL linearization: $x \prec y$ means that operation x appears strictly before y in the linearized history of DL. For any finite set $A \subseteq \mathcal{M}$, $\mathsf{ordered}(A)$ returns a deterministic total order over A (e.g., lexicographic order on $(\mathit{senderId}, \mathit{messageId})$ or on message hashes). For any round $r \in \mathcal{R}$, define $\mathsf{Winners}_r \triangleq \{\, j \in \Pi \mid (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}(r) \,\}$, i.e., the set of processes whose $\mathsf{PROVE}(r)$ appears before the first $\mathsf{APPEND}(r)$ in the DL linearization. We denoted by $\mathsf{PROVE}^{(j)}(r)$ or $\mathsf{APPEND}^{(j)}(r)$ the operation $\mathsf{PROVE}(r)$ or $\mathsf{APPEND}(r)$ invoked by process j.

3 Primitives

3.1 Reliable Broadcast (RB)

RB provides the following properties in the model.

- **Integrity**: Every message received was previously sent. $\forall p_i : \mathsf{RB\text{-}received}_i(m) \Rightarrow \exists p_j : \mathsf{RB\text{-}cast}_j(m)$.
- **No-duplicates**: No message is received more than once at any process.
- **Validity**: If a correct process broadcasts m, every correct process eventually receives m.

3.2 DenyList (DL)

We assume a synchronous DenyList (DL) object with the following properties.

The DenyList object type supports three operations: APPEND, PROVE, and READ. These operations appear as if executed in a sequence Seq such that:

- **Termination.** A PROVE, an APPEND, or a READ operation invoked by a correct process always returns.
- **APPEND Validity.** The invocation of $\mathsf{APPEND}(x)$ by a process p is valid if:
  - $p \in \Pi_M \subseteq \Pi$; **and**
  - $x \in S$, where S is a predefined set.

  Otherwise, the operation is invalid.
- **PROVE Validity.** If the invocation of a $op = \mathsf{PROVE}(x)$ by a correct process p is not valid, then:
  - $p \notin \Pi_V \subseteq \Pi$; **or**
  - A valid $\mathsf{APPEND}(x)$ appears before op in Seq.

  Otherwise, the operation is valid.
- **PROVE Anti-Flickering.** If the invocation of a operation $op = \mathsf{PROVE}(x)$ by a correct process $p \in \Pi_V$ is invalid, then any $\mathsf{PROVE}(x)$ operation that appears after op in Seq is invalid.
- **READ Validity.** The invocation of $op = \mathsf{READ}()$ by a process $p \in \pi_V$ returns the list of valid invocations of PROVE that appears before op in Seq along with the names of the processes that invoked each operation.
- **Anonymity.** Let us assume the process p invokes a $\mathsf{PROVE}(v)$ operation. If the process p' invokes a $\mathsf{READ}()$ operation, then p' cannot learn the value v unless p leaks additional information.

4 Target Abstraction: Atomic Reliable Broadcast (ARB)

Processes export $\mathsf{AB\text{-}broadcast}(m)$ and $\mathsf{AB\text{-}deliver}(m)$. ARB requires total order:

$$\forall m_1, m_2,\ \forall p_i, p_j : \quad \mathsf{AB\text{-}deliver}_i(m_1) < \mathsf{AB\text{-}deliver}_i(m_2) \Rightarrow \mathsf{AB\text{-}deliver}_j(m_1) < \mathsf{AB\text{-}deliver}_j(m_2),$$

plus Integrity/No-duplicates/Validity (inherited from RB and the construction).

5 Algorithm

Definition 1 (Closed round)**.** Given a DL linearization H, a round $r \in \mathcal{R}$ is *closed* in H iff H contains an operation APPEND(r). Equivalently, there exists a time after which every PROVE(r) is invalid in H.

5.1 Variables

Each process p_i maintains:

received $\leftarrow \emptyset$	▷ Messages received via RB but not yet delivered
delivered $\leftarrow \emptyset$	▷ Messages already delivered
prop$[r][j] \leftarrow \bot, \ \forall r, j$	▷ Proposal from process j for round r
current $\leftarrow 0$	

DenyList. The DL is initialized empty. We assume $\Pi_M = \Pi_V = \Pi$ (all processes can invoke APPENDand PROVE).

5.2 Handlers and Procedures

Algorithm A RB handler (at process p_i)

```
function RBReceived(S, r, j)
    received ← received ∪ {S}
    prop[r][j] ← S                          ▷ Record sender j's proposal S for round r
end function
```

Algorithm B AB-broadcast(m) (at process p_i)

```
function ABbroadcast(m)
    P ← READ()                              ▷ Fetch latest DL state to learn recent PROVE operations
    r_max ← max({r' : ∃j, (j, PROVE(r')) ∈ P})    ▷ Pick current open round
    S ← (received \ delivered) ∪ {m}        ▷ Propose all pending messages plus the new m

    for each r ∈ {r_max, r_max + 1, ···} do
        RB-cast(S, r, i); PROVE(r); APPEND(r);
        P ← READ()                          ▷ Refresh local view of DL
        if (((i, prove(r)) ∈ P ∨ (∃j, r' : (j, prove(r')) ∈ P ∧ m ∈ prop[r'][j]))) then
            break                           ▷ Exit loop once m is included in some closed round
        end if
    end for
end function
```

Algorithm C AB-deliver() at process p_i

```
function ABDELIVER
    r ← current
    P ← READ()
    if ∀j : (j, prove(r)) ∉ P then
        return ⊥
    end if
    APPEND(r); P ← READ()
    W_r ← {j : (j, prove(r)) ∈ P}
    if ∃j ∈ W_r, prop[r][j] = ⊥ then
        return ⊥
    end if
    M_r ← ⋃_{j∈W_r} prop[r][j]
    m ← ordered(M_r \ delivered)[0]        ▷ Set m as the smaller message not already delivered
    delivered ← delivered ∪ {m}
    if M_r \ delivered = ∅ then           ▷ Check if all messages from round r have been delivered
        current ← current + 1
    end if
    return m
end function
```

6 Correctness

Lemma 1 (Stable round closure)**.** *If a round r is closed, then there exists a linearization point t_0 of APPEND(r) in the DL, and from that point on, no PROVE(r) can be valid. Once closed, a round never becomes open again.*

Proof. By Definition 1, some APPEND(r) occurs in the linearization H.
H is a total order of operations, the set of APPEND(r) operations is totally ordered, and hence there exists a smallest APPEND(r) in H. We denote this operation $\mathsf{APPEND}^{(\star)}(r)$ and t_0 its linearization point.
By the validity property of DL, a PROVE(r) is valid iff $\mathsf{PROVE}(r) \prec \mathsf{APPEND}^{(\star)}(r)$. Thus, after t_0, no PROVE(r) can be valid.
H is a immutable grow-only history, and hence once closed, a round never becomes open again.
Hence there exists a linearization point t_0 of APPEND(r) in the DL, and from that point on, no PROVE(r) can be valid and the closure is stable. □

Definition 2 (First APPEND)**.** Given a DL linearization H, for any closed round $r \in \mathcal{R}$, we denote by $\mathsf{APPEND}^{(\star)}(r)$ the earliest APPEND(r) in H.

Lemma 2 (Across rounds)**.** *If there exists a r such that r is closed, $\forall r'$ such that $r' < r$, r' is also closed.*

Proof. Base. For a closed round $k = 0$, the set $\{k' \in \mathcal{R},\ k' < k\}$ is empty, hence the lemma is true.
Induction. Assume the lemma is true for round $k \geq 0$, we prove it for round $k + 1$.

Assume $k + 1$ is closed and let $\mathsf{APPEND}^{(\star)}(k + 1)$ be the earliest APPEND$(k + 1)$ in the DL linearization H. By Lemma 1, after an APPEND(k) is in H, any later PROVE(k) is rejected also, a process's program order is preserved in H.

There are two possibilities for where $\mathsf{APPEND}^{(\star)}(k + 1)$ is invoked.

- **Case (B6) :** Some process $p^\star$ executes the loop (lines B5–B11) and invokes $\mathsf{APPEND}^{(\star)}(k+1)$ at line B6. Immediately before line B6, line B5 sets $r \leftarrow r+1$, so the previous loop iteration (if any) targeted k. We consider two sub-cases.

  – *(i) $p^\star$ is not in its first loop iteration.* In the previous iteration, $p^\star$ executed $\mathsf{PROVE}^{(\star)}(k)$ at B6, followed (in program order) by $\mathsf{APPEND}^{(\star)}(k)$. If round k wasn't closed when $p^\star$ execute $\mathsf{PROVE}^{(\star)}(k)$ at B9, then the condition at B8 would be true hence the tuple $(p^\star, \mathsf{prove}(k))$ should be visible in P which implies that $p^\star$ should leave the loop at round k, contradicting the assumption that $p^\star$ is now executing another iteration. Since the tuple is not visible, the $\mathsf{PROVE}^{(\star)}(k)$ was rejected by the DL which implies by definition an $\mathsf{APPEND}(k)$ already exists in H. Thus in this case k is closed.

  – *(ii) $p^\star$ is in its first loop iteration.* To compute the value r_{max}, $p^\star$ must have observed one or many $(\_, \mathsf{prove}(k+1))$ in P at B2/B3, issued by some processes (possibly different from $p^\star$). Let's call p_1 the issuer of the first $\mathsf{PROVE}^{(1)}(k+1)$ in the linearization H.
  When p_1 executed $P \leftarrow \mathsf{READ}()$ at B2 and compute r_{max} at B3, he observed no tuple $(\_, \mathsf{prove}(k+1))$ in P because he's the issuer of the first one. So when p_1 executed the loop (B5–B11), he run it for the round k, didn't seen any $(1, \mathsf{prove}(k))$ in P at B8, and then executed the first $\mathsf{PROVE}^{(1)}(k+1)$ at B6 in a second iteration.
  If round k wasn't closed when p_1 execute $\mathsf{PROVE}^{(1)}(k)$ at B6, then the condition at B8 should be true which implies that p_1 sould leave the loop at round k, contradicting the assumption that p_1 is now executing $\mathsf{PROVE}^{(1)}(r+1)$. In this case k is closed.

- **Case (C8) :** Some process invokes $\mathsf{APPEND}(k+1)$ at C8. Line C8 is guarded by the presence of $\mathsf{PROVE}(\mathit{next})$ in P with $\mathit{next} = k+1$ (C5). Moreover, the local pointer next grow by increment of 1 and only advances after finishing the current round (C17), so if a process can reach $\mathit{next} = k+1$ it implies that he has completed round k, which includes closing k at C8 when $\mathsf{PROVE}(k)$ is observed. Hence $\mathsf{APPEND}^\star(k+1)$ implies a prior $\mathsf{APPEND}(k)$ in H, so k is closed.

In all cases, $k+1$ closed implie k closed. By induction on k, if the lemme is true for a closed k then it is true for a closed $k+1$. Therefore, the lemma is true for all closed rounds r. ☐

Definition 3 (Winner Invariant)**.** For any closed round r, define

$$\mathsf{Winners}_r \triangleq \{j : \mathsf{PROVE}^{(j)}(r) \prec \mathsf{APPEND}^\star(r)\}$$

as the unique set of winners of round r.

Lemma 3 (Invariant view of closure)**.** *For any closed round r, all correct processes eventually observe the same set of valid tuples $(\_, \mathsf{prove}(r))$ in their DL view.*

Proof. Let's take a closed round r. By Definition 2, there exists a unique earliest $\mathsf{APPEND}(r)$ in the DL linearization, denoted $\mathsf{APPEND}^\star(r)$.

Consider any correct process p that invokes $\mathsf{READ}()$ after $\mathsf{APPEND}^\star(r)$ in the DL linearization. Since $\mathsf{APPEND}^\star(r)$ invalidates all subsequent $\mathsf{PROVE}(r)$, the set of valid tuples $(\_, \mathsf{prove}(r))$ observed by any correct process after $\mathsf{APPEND}^\star(r)$ is fixed and identical across all correct processes.

Therefore, for any closed round r, all correct processes eventually observe the same set of valid tuples $(\_, \mathsf{prove}(r))$ in their DL view. ☐

Lemma 4 (Well-defined winners)**.** *For any correct process and round r, if the process computes W_r at line C9, then :*

- $\mathsf{Winners}_r$ *is defined;*
- *the computed W_r is exactly* $\mathsf{Winners}_r$*.*

Proof. Let take a correct process p_i that reach line C9 to compute W_r.
By program order, p_i must have executed $\mathsf{APPEND}^{(i)}(r)$ at C8 before, which implies by Definition 1 that round r is closed. So by Definition 3, $\mathsf{Winners}_r$ is defined.
By Lemma 3, all correct processes eventually observe the same set of valid tuples $(\_, \mathsf{prove}(r))$ in their DL view. Hence, when p_i executes the $\mathsf{READ}()$ at C8 after the $\mathsf{APPEND}^{(i)}(r)$, it observes a set P that includes all valid tuples $(\_, \mathsf{prove}(r))$ such that

$$W_r = \{j : (j, \mathsf{prove}(r)) \in P\} = \{j : \mathsf{PROVE}^{(j)}(r) \prec \mathsf{APPEND}^{\star}(r)\} = \mathsf{Winners}_r$$

□

Lemma 5 (No APPEND without PROVE)**.** *If some process invokes $\mathsf{APPEND}(r)$, then at least a process must have previously invoked $\mathsf{PROVE}(r)$.*

Proof. Consider the round r such that some process invokes $\mathsf{APPEND}(r)$. There are two possible cases

- **Case (B6) :** There exists a process $p^\star$ who's the issuer of the earliest $\mathsf{APPEND}^{(\star)}(r)$ in the DL linearization H. By program order, $p^\star$ must have previously invoked $\mathsf{PROVE}^{(\star)}(r)$ before invoking $\mathsf{APPEND}^{(\star)}(r)$ at B6. In this case, there is at least one $\mathsf{PROVE}(r)$ valid in H issued by a correct process before $\mathsf{APPEND}^{(\star)}(r)$.
- **Case (C8) :** There exist a process $p^\star$ invokes $\mathsf{APPEND}^{(\star)}(r)$ at C8. Line C8 is guarded by the condition at C5, which ensures that p observed some $(\_, \mathsf{prove}(r))$ in P. In this case, there is at least one $\mathsf{PROVE}(r)$ valid in H issued by some process before $\mathsf{APPEND}^{(\star)}(r)$.

In both cases, if some process invokes $\mathsf{APPEND}(r)$, then some process must have previously invoked $\mathsf{PROVE}(r)$. □

Lemma 6 (No empty winners)**.** *Let r be a round, if* $\mathsf{Winners}_r$ *is defined, then* $\mathsf{Winners}_r \neq \emptyset$*.*

Proof. If $\mathsf{Winners}_r$ is defined, then by Definition 3, round r is closed. By Definition 1, some $\mathsf{APPEND}(r)$ occurs in the DL linearization.
By Lemma 5, at least a process must have invoked a valid $\mathsf{PROVE}(r)$ before $\mathsf{APPEND}^{(\star)}(r)$. Hence, there exists at least one j such that $\{j : \mathsf{PROVE}^{(j)}(r) \prec \mathsf{APPEND}^{\star}(r)\}$, so $\mathsf{Winners}_r \neq \emptyset$. □

Lemma 7 (Winners must propose)**.** *For any closed round r, $\forall j \in \mathsf{Winners}_r$, process j must have invoked a $\mathsf{RB\text{-}cast}(S^{(j)}, r, j)$*

Proof. Fix a closed round r. By Definition 3, for any $j \in \mathsf{Winners}_r$, there exist a valid $\mathsf{PROVE}^{(j)}(r)$ such that $\mathsf{PROVE}^{(j)}(r) \prec \mathsf{APPEND}^{\star}(r)$ in the DL linearization. By program order, if j invoked a valid $\mathsf{PROVE}^{(j)}(r)$ at line B6 he must have invoked $\mathsf{RB\text{-}cast}(S^{(j)}, r, j)$ directly before. □

Definition 4 (Messages invariant)**.** For any closed round r and any correct process p_i such that $\nexists j \in \mathsf{Winners}_r : prop^{[i)}[r][j] = \bot$, define

$$\mathsf{Messages}_r \triangleq \bigcup_{j \in \mathsf{Winners}_r} \mathsf{prop}^{(i)}[r][j]$$

as the unique set of messages proposed by the winners of round r.

Lemma 8 (Non-empty winners proposal)**.** *For any closed round r, $\forall j \in \mathsf{Winners}_r$, for any correct process p_i, eventually $\mathsf{prop}^{(i)}[r][j] \neq \bot$.*

Proof. Fix a closed round r. By Definition 3, for any $j \in \mathsf{Winners}_r$, there exist a valid $\mathsf{PROVE}^{(j)}(r)$ such that $\mathsf{PROVE}^{(j)}(r) \prec \mathsf{APPEND}^{\star}(r)$ in the DL linearization. By Lemma 7, j must have invoked $\mathsf{RB\text{-}cast}(S^{(j)}, r, j)$.

Let take a process p_i, by RB *Validity*, every correct process eventually receives j's RB message for round r, which sets $\mathsf{prop}[r][j]$ to a non-$\bot$ value at line A3. ☐

Lemma 9 (Eventual proposal closure)**.** *If a correct process p_i define M_r at line C13, then for every $j \in \mathsf{Winners}_r$, $\mathsf{prop}^{(i)}[r][j] \neq \bot$.*

Proof. Let take a correct process p_i that computes M_r at line C13. By Lemma 4, p_i computes the unique winner set $\mathsf{Winners}_r$.

By Lemma 6, $\mathsf{Winners}_r \neq \emptyset$. The instruction at line C13 where p_i computes M_r is guarded by the condition at C10, which ensures that p_i has received all RB messages from every winner $j \in \mathsf{Winners}_r$. Hence, when p_i computes $M_r = \bigcup_{j \in \mathsf{Winners}_r} \mathsf{prop}^{(i)}[r][j]$, we have $\mathsf{prop}^{(i)}[r][j] \neq \bot$ for all $j \in \mathsf{Winners}_r$. ☐

Lemma 10 (Unique proposal per sender per round)**.** *For any round r and any process p_i, p_i invokes at most one $\mathsf{RB\text{-}cast}(S, r, i)$.*

Proof. By program order, any process p_i invokes $\mathsf{RB\text{-}cast}(S, r, i)$ at line B6 must be in the loop B5–B11. No matter the number of iterations of the loop, line B5 always uses the current value of r which is incremented by 1 at each turn. Hence, in any execution, any process p_i invokes $\mathsf{RB\text{-}cast}(S, r, j)$ at most once for any round r. ☐

Lemma 11 (Proposal convergence)**.** *For any round r, for any correct processes p_i that define M_r at line C13, we have*

$$M_r^{(i)} = \mathsf{Messages}_r$$

Proof. Let take a correct process p_i that define M_r at line C13. That implies that p_i has defined W_r at line C9. It implies that, by Lemma 4, r is closed and $W_r = \mathsf{Winners}_r$.
By Lemma 9, for every $j \in \mathsf{Winners}_r$, $\mathsf{prop}^{(i)}[r][j] \neq \bot$. By Lemma 10, each winner j invokes at most one $\mathsf{RB\text{-}cast}(S^{(j)}, r, j)$, so $\mathsf{prop}^{(i)}[r][j] = S^{(j)}$ is uniquely defined. Hence, when p_i computes

$$M_r^{(i)} = \bigcup_{j \in \mathsf{Winners}_r} \mathsf{prop}^{(i)}[r][j] = \bigcup_{j \in \mathsf{Winners}_r} S^{(j)} = \mathsf{Messages}_r.$$

☐

Lemma 12 (Inclusion)**.** *If some correct process invokes $\mathsf{AB\text{-}broadcast}(m)$, then there exist a round r and a process $j \in \mathsf{Winners}_r$ such that p_j invokes*

$$\mathsf{RB\text{-}cast}(S, r, j) \quad \textit{with} \quad m \in S.$$

Proof. Fix a correct process p_i that invokes $\mathsf{AB\text{-}broadcast}(m)$ and eventually exits the loop (B5–B11) at some round r. There are two possible cases.

- **Case 1:** p_i exits the loop because $(i, \mathsf{prove}(r)) \in P$. In this case, by Definition 3, p_i is a winner in round r. By program order, p_i must have invoked $\mathsf{RB\text{-}cast}(S, r, i)$ at B6 before invoking $\mathsf{PROVE}^{(i)}(r)$ at B7. By line B4, $m \in S$. Hence there exist a closed round r and a correct process $j = i \in \mathsf{Winners}_r$ such that j invokes $\mathsf{RB\text{-}cast}(S, r, j)$ with $m \in S$.

- **Case 2:** p_i exits the loop because $\exists j, r' : (j, \mathsf{prove}(r')) \in P \wedge m \in \mathsf{prop}[r'][j]$. In this case, by Lemma 7 and Lemma 10 j must have invoked a unique $\mathsf{RB\text{-}cast}(S, r', j)$. Which set $\mathsf{prop}^{(i)}[r'][j] = S$ with $m \in S$.

In both cases, if some correct process invokes $\mathsf{AB\text{-}broadcast}(m)$, then there exist a round r and a correct process $j \in \mathsf{Winners}_r$ such that j invokes

$$\mathsf{RB\text{-}cast}(S, r, j) \quad \text{with} \quad m \in S.$$

□

Lemma 13 (Broadcast Termination)**.** *If a correct process invokes AB-broadcast(m), then he eventually exit the function and returns.*

Proof. Let a correct process p_i that invokes $\mathsf{AB\text{-}broadcast}(m)$. The lemma is true if $\exists r_1$ such that $r_1 \geq r_{max}$ and if $(i, \mathsf{prove}(r_1)) \in P$; or if $\exists r_2$ such that $r_2 \geq r_{max}$ and if $\exists j : (j, \mathsf{prove}(r_2)) \in P \wedge m \in \mathsf{prop}[r_2][j]$ (like guarded at B8).

Let admit that there exists no round r_1 such that p_i invokes a valid $\mathsf{PROVE}(r_1)$. In this case p_i invokes infinitely many $\mathsf{RB\text{-}cast}(S, \_, i)$ at B6 with $m \in S$ (line B4).
The assumption that no $\mathsf{PROVE}(r_1)$ invoked by p is valid implies by DL *Validity* that for every round $r' \geq r_{max}$, there exists at least one $\mathsf{APPEND}(r')$ in the DL linearization, hence by Lemma 6 for every possible round r' there at least a winner. Because there is an infinite number of rounds, and a finite number of processes, there exists at least a correct process p_k that invokes infinitely many valid $\mathsf{PROVE}(r')$ and by extension infinitely many $\mathsf{AB\text{-}broadcast}(\_)$. By RB *Validity*, p_k eventually receives p_i 's RB messages. Let call t_0 the time when p_k receives p_i 's RB message.
At t_0, p_k execute Algorithm A and do $\mathsf{received} \leftarrow \mathsf{received} \cup \{S\}$ with $m \in S$ (line A2). For the first invocation of $\mathsf{AB\text{-}broadcast}(\_)$ by p_k after the execution of Algorithm A, p_k must include m in his proposal S at line B4 (because m is pending in j's $\mathsf{received} \setminus \mathsf{delivered}$ set). There exists a minimum round r_2 such that $p_k \in \mathsf{Winners}_{r_2}$ after t_0. By Lemma 8, eventually $\mathsf{prop}^{(i)}[r_2][k] \neq \bot$. By Lemma 10, $\mathsf{prop}^{(i)}[r_2][k]$ is uniquely defined as the set S proposed by p_k at B6, which by Lemma 12 includes m. Hence eventually $m \in \mathsf{prop}^{(i)}[r_2][k]$ and $k \in \mathsf{Winners}_{r_2}$.

So if p_i is a winner he cover the condition $(i, \mathsf{prove}(r_1)) \in P$. And we show that if the first condition is never satisfied, the second one will eventually be satisfied. Hence either the first or the second condition will eventually be satisfied, and p_i eventually exits the loop and returns from $\mathsf{AB\text{-}broadcast}(m)$. □

Lemma 14 (Validity)**.** *If a correct process p invokes AB-broadcast(m), then every correct process that invokes a infinitely often times AB-deliver$()$ eventually delivers m.*

Proof. Let p_i a correct process that invokes $\mathsf{AB\text{-}broadcast}(m)$ and p_q a correct process that infinitely invokes $\mathsf{AB\text{-}deliver}()$. By Lemma 12, there exist a closed round r and a correct process $j \in \mathsf{Winners}_r$ such that p_j invokes

$$\mathsf{RB\text{-}cast}(S, r, j) \quad \text{with} \quad m \in S.$$

By Lemma 9, when p_q computes M_r at line C13, $\mathsf{prop}[r][j]$ is non-$\bot$ because $j \in \mathsf{Winners}_r$. By Lemma 10, p_j invokes at most one $\mathsf{RB\text{-}cast}(S, r, j)$, so $\mathsf{prop}[r][j]$ is uniquely defined. Hence, when p_q computes

$$M_r = \bigcup_{k \in \mathsf{Winners}_r} \mathsf{prop}[r][k],$$

we have $m \in \mathsf{prop}[r][j] = S$, so $m \in M_r$. By Lemma 11, M_r is invariant so each computation of M_r by any correct process that defines it includes m. At each invocation of $\mathsf{AB\text{-}deliver}()$ which deliver m', m' is add to $\mathsf{delivered}$ until $M_r \subseteq \mathsf{delivered}$. Once this append we're assured that there exist an invocation of $\mathsf{AB\text{-}deliver}()$ which return m. Hence m is well delivered. □

Lemma 15 (No duplication)**.** *No correct process delivers the same message more than once.*

Proof. Let consider two invokations of $\mathsf{AB\text{-}deliver}()$ made by the same correct process which returns m. Let call these two invocations respectively $\mathsf{AB\text{-}deliver}^{(A)}()$ and $\mathsf{AB\text{-}deliver}^{(B)}()$.

When $\mathsf{AB\text{-}deliver}^{(A)}()$ occur, by program order and because it reach line C19 to return m, the process must have add m to $\mathsf{delivered}$. Hence when $\mathsf{AB\text{-}deliver}^{(B)}()$ reach line C14 to extract the next message to deliver, it can't be m because $m \notin (M_r \setminus \{..., m, ...\})$. So a $\mathsf{AB\text{-}deliver}^{(B)}()$ which deliver m can't occur. □

Lemma 16 (Total order)**.** *For any two messages m_1 and m_2 delivered by correct processes, if a correct process p_i delivers m_1 before m_2, then any correct process p_j that delivers both m_1 and m_2 delivers m_1 before m_2.*

Proof. Consider any correct process that delivers both m_1 and m_2. By Lemma 14, there exist closed rounds r'_1 and r'_2 and correct processes $k_1 \in \mathsf{Winners}_{r'_1}$ and $k_2 \in \mathsf{Winners}_{r'_2}$ such that

$$\mathsf{RB\text{-}cast}(S_1, r'_1, k_1) \quad \text{with} \quad m_1 \in S_1,$$

$$\mathsf{RB\text{-}cast}(S_2, r'_2, k_2) \quad \text{with} \quad m_2 \in S_2.$$

Let consider three cases :

- **Case 1:** $r_1 < r_2$. By program order, any correct process must have waited to append in $\mathsf{delivered}$ every messages in M_{r_1} (which contains m_1) to increment $\mathsf{current}$ and eventually set $\mathsf{current} = r_2$ to compute M_{r_2} and then invoke the $\mathsf{AB\text{-}deliver}()$ that returns m_2. Hence, for any correct process that delivers both m_1 and m_2, it delivers m_1 before m_2.

- **Case 2:** $r_1 = r_2$. By Lemma 11, any correct process that computes M_{r_1} at line C13 computes the same set of messages $\mathsf{Messages}_{r_1}$. By line C14 the messages are pull in a deterministic order defined by $\mathsf{ordered}(\_)$. Hence, for any correct process that delivers both m_1 and m_2, it delivers m_1 and m_2 in the deterministic order defined by $\mathsf{ordered}(\_)$.

In all possible cases, any correct process that delivers both m_1 and m_2 delivers m_1 and m_2 in the same order. □

Lemma 17 (Fifo Order)**.** *For any two messages m_1 and m_2 broadcast by the same correct process p_i, if p_i invokes $\mathsf{AB\text{-}broadcast}(m_1)$ before $\mathsf{AB\text{-}broadcast}(m_2)$, then any correct process p_j that delivers both m_1 and m_2 delivers m_1 before m_2.*

Proof. Let take two messages m_1 and m_2 broadcast by the same correct process p_i, with p_i invoking $\mathsf{AB\text{-}broadcast}(m_1)$ before $\mathsf{AB\text{-}broadcast}(m_2)$. By Lemma 14, there exist closed rounds r_1 and r_2 and correct processes $k_1 \in \mathsf{Winners}_{r_1}$ and $k_2 \in \mathsf{Winners}_{r_2}$ such that

$$\mathsf{RB\text{-}cast}(S_1, r_1, k_1) \quad \text{with} \quad m_1 \in S_1,$$

$$\mathsf{RB\text{-}cast}(S_2, r_2, k_2) \quad \text{with} \quad m_2 \in S_2.$$

By program order, p_i must have invoked $\mathsf{RB\text{-}cast}(S_1, r_1, i)$ before $\mathsf{RB\text{-}cast}(S_2, r_2, i)$. By Lemma 10, any process invokes at most one $\mathsf{RB\text{-}cast}(S, r, i)$ per round, hence $r_1 < r_2$. By Lemma 16, any correct process that delivers both m_1 and m_2 delivers them in a deterministic order.

In all possible cases, any correct process that delivers both m_1 and m_2 delivers m_1 before m_2. □

Theorem 18 (FIFO-ARB)**.** *Under the assumed* DL *synchrony and* RB *reliability, the algorithm implements FIFO Atomic Reliable Broadcast.*

Proof. We show that the algorithm satisfies the properties of FIFO Atomic Reliable Broadcast under the assumed DL synchrony and RB reliability.

First, by Lemma 13, if a correct process invokes $\mathsf{AB\text{-}broadcast}(m)$, then it eventually returns from this invocation. Moreover, Lemma 14 states that if a correct process invokes $\mathsf{AB\text{-}broadcast}(m)$, then every correct process that invokes AB-deliver() infinitely often eventually delivers m. This gives the usual Validity property of ARB.

Concerning Integrity and No-duplicates, the construction only ever delivers messages that have been obtained from the underlying RB primitive. By the Integrity property of RB, every such message was previously RB-castby some process, so no spurious messages are delivered. In addition, Lemma 15 states that no correct process delivers the same message more than once. Together, these arguments yield the Integrity and No-duplicates properties required by ARB.

For the ordering guarantees, Lemma 16 shows that for any two messages m_1 and m_2 delivered by correct processes, every correct process that delivers both m_1 and m_2 delivers them in the same order. Hence all correct processes share a common total order on delivered messages. Furthermore, Lemma 17 states that for any two messages m_1 and m_2 broadcast by the same correct process, any correct process that delivers both messages delivers m_1 before m_2 whenever m_1 was broadcast before m_2. Thus the global total order extends the per-sender FIFO order of AB-broadcast.

All the above lemmas are proved under the assumptions that DL satisfies the required synchrony properties and that the underlying primitive is a Reliable Broadcast (RB) with Integrity, No-duplicates and Validity. Therefore, under these assumptions, the algorithm satisfies Validity, Integrity/No-duplicates, total order and per-sender FIFO order, and hence implements FIFO Atomic Reliable Broadcast, as claimed. □

7 Reciprocity

So far, we assumed the existence of a synchronous DenyList (DL) object and showed how to upgrade a Reliable Broadcast (RB) primitive into FIFO Atomic Reliable Broadcast (ARB). We now briefly argue that, conversely, an ARB primitive is strong enough to implement a synchronous DL object (ignoring the anonymity property).

DenyList as a deterministic state machine. Without anonymity, the DL specification defines a deterministic abstract object: given a sequence Seq of operations $\mathsf{APPEND}(x)$, $\mathsf{PROVE}(x)$, and READ(), the resulting sequence of return values and the evolution of the abstract state (set of appended elements, history of operations) are uniquely determined.

State machine replication over ARB. Assume a system that exports a FIFO-ARB primitive with the guarantees that if a correct process invokes AB-broadcast(m), then every correct process eventually AB-deliver(m) and the invocation eventually returns. Following the classical *state machine replication* approach such as described in Schneider [1], we can implement a fault-tolerant service by ensuring the following properties:

> **Agreement.** Every nonfaulty state machine replica receives every request.
> **Order.** Every nonfaulty state machine replica processes the requests it receives in the same relative order.

Which are cover by our FIFO-ARB specification.

Correctness.

Theorem 19 (From ARB to synchronous DL)**.** *In an asynchronous message-passing system with crash failures, assume a FIFO Atomic Reliable Broadcast primitive with Integrity, No-duplicates, Validity, and the liveness of AB-broadcast. Then, ignoring anonymity, there exists an implementation of a synchronous DenyList object that satisfies the Termination, Validity, and Anti-flickering properties.*

Proof. Because the DL object is deterministic, all correct processes see the same sequence of operations and compute the same sequence of states and return values. We obtain:

- **Termination.** The liveness of ARB ensures that each AB-broadcast invocation by a correct process eventually returns, and the corresponding operation is eventually delivered and applied at all correct processes. Thus every APPEND, PROVE, and READ operation invoked by a correct process eventually returns.
- **APPEND/PROVE/READ Validity.** The local code that forms AB-broadcastrequests can achieve the same preconditions as in the abstract DL specification (e.g., $p \in \Pi_M$, $x \in S$ for APPEND(x)). Once an operation is delivered, its effect and return value are exactly those of the sequential DL specification applied in the common order.
- **PROVE Anti-Flickering.** In the sequential DL specification, once an element x has been appended, all subsequent PROVE(x) are invalid forever. Since all replicas apply operations in the same order, this property holds in every execution of the replicated implementation: after the first linearization point of APPEND(x), no later PROVE(x) can return “valid” at any correct process.

Formally, we can describe the DL object with the state machine approach for crash-fault, asynchronous message-passing systems with a total order broadcast layer [1]. □

References

[1] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.