A Lightweight Solution to Uniform Atomic Broadcast for Asynchronous Systems *

E. Anceaume
IRISA/CNRS
Campus Universitaire de Beaulieu, 35042 Rennes cedex, France

Abstract

Chandra and Toueg proposed in [5] a new approach to overcome the impossibility of reaching deterministically Consensus — and by corollary Atomic Broadcast — in asynchronous systems subject to crash failures. They augment the asynchronous system with a possibly Unreliable Failure Detector which provides some information about the operational state of processes. In this paper, we present an extension of the Consensus problem that we call Uniform Prefix Agreement. This extension enables all the processes to propose a flow of messages during an execution — instead of one as in the Consensus problem — and uses all these proposed messages to compose its decision value. Prefix Agreement is based on an Unreliable Failure Detectors. We use repeated executions of Prefix Agreement to build an efficient and lightweight Uniform Atomic Broadcast algorithm. This paper describes the Uniform Prefix Agreement and Uniform Atomic Broadcast algorithms.

1 Introduction

Atomic Broadcast is a powerful communication paradigm for fault-tolerant computing. It allows processes to reliably broadcast messages, so that they agree on the set of messages they deliver and the order of message deliveries. Applications based on this paradigm include Isis [3] and its new generation Horus [14], Transis [15], Totem [16], Psync [24], Newtop [13], Amoeba [22], Delta-4 [25], HAS [7], and FAA [10]. In 1985, Fisher, Lynch and Paterson proved that Consensus can not be solved deterministically in an asynchronous system subject to even a single crash failure [12].

Chandra and Toueg proposed in 1993 [5] a deterministic solution to Consensus, and by corollary to Atomic Broadcast, by augmenting the asynchronous model with an unreliable failure detector. An unreliable failure detector is a distributed "oracle" that gives (possibly incorrect) hints about which processes may have crashed so far. Their solution to Atomic Broadcast relies on Consensus: each correct process proposes to Consensus a single set of messages to be atomically delivered and *waits until* Consensus returns the decision value containing one of the proposed set of messages. This decision is committed once a majority of processes has agreed. In asynchronous systems, computing the decision can be very long, since it can take an arbitrarily long time before reaching the majority. Delivering atomically a great number of messages is thus a long process since numerous set of messages have to be decided by Consensus and for each, waiting for a majority is required.

The work presented in this paper is an algorithm that optimizes the time required to atomically deliver messages. We propose an extension of the Consensus problem and use it to build an efficient and lightweight Uniform Atomic Broadcast algorithm for asynchronous systems. Our extension, called *Uniform Prefix Agreement* [2], is based on the work of Chandra and Toueg and uses the eventual strong failure detector $\diamond\mathcal{S}$ [5]. Uniform Prefix Agreement enhances the performance of Consensus by handling flows of messages and flows from different processes that are not likely to overlap in time: processes having flows of messages to propose while waiting for the decision value may continue to propose them even if the other processes do not have more messages to propose, avoiding thus, idle period of time. This feature is quite relevant in asynchronous systems for which computation time and transmission time may reveal to be completely arbitrary from one process to another one. Our extension enables to decide on a large number of messages with a time and message cost equal to the one needed in [5] to decide on one message. The decision value needs four phases and $3(n-1)$ messages plus a reliable broadcast to be decided, assuming that neither failures nor failure suspicions occur. This behavior makes our extension adapted to bursty and high traffic load. Another important consequence of the efficiency of Uniform Prefix Agreement is that it offers to our Uniform Atomic Broadcast algorithm a very low message *delivery time*, i.e., the duration between the moment a message is submitted to Atomic Broadcast and the moment the message is delivered to every process. Indeed, our Uniform Atomic Broadcast algorithm uses repeated and possibly concurrent (but not independent) executions of Uniform Prefix Agreement and successively deliver the decided sequences of messages. Thus no time and message overhead are exhibited with regard to Uniform Prefix Agreement which makes our solution to Uniform Atomic Broadcast quite efficient, modular and lightweight.

Finally, it is important to stress that a process may be silent for a while by not acknowledging messages, however it will never be prevented from atomically broadcasting messages. And as long as it will not have been really crashed, it will atomically deliver the same sequence of messages as all the correct processes do. However, if a process really crashes, then it remains in the failure status forever.

The remainder of the paper is organized as follows: We present in Section 2 the model of computation. In Section 3, we briefly give the specification of Uniform Reliable Broadcast, Uniform Atomic Broadcast and Uniform Prefix Agreement. In Section 4 and Section 5, we describe the Uniform Atomic Broadcast and Uniform Prefix Agreement algorithms. In section 6, we discuss related work, and conclude in section 7.

2 Model

We consider asynchronous distributed systems in which there is no bound on message delay, clock drift, or the time to execute a step. Our model of asynchronous computation follows the one proposed in [12]. The system consists of a set of n processes. Every pair of processes is connected by a reliable communication channel. Processes are subject to crash failures [21]. If a process ever crashes, we say that it is *faulty*, otherwise it is *correct*. We require a majority of processes to be correct. The asynchronous system is augmented with any eventual strong failure detector $\diamond\mathcal{S}$, a class of failure detectors that satisfy the eventual weak accuracy and the strong completeness properties. $\diamond\mathcal{S}$ is the class of the weakest failure detectors necessary to solve Atomic Broadcast [5].

3 Specifications

In this section, specifications of Uniform Atomic Broadcast and Uniform Prefix Agreement are given. Our solution to Uniform Atomic Broadcast and Uniform Prefix Agreement relying on the services provided by Uniform Reliable Broadcast, the weakest fault tolerant communication primitive, we also give the specification of this primitive. Note that we require *Uniformity* [23, 17, 1]. The reason is that Uniformity cares about faulty processes, i.e., places restrictions on the messages delivered by faulty processes. Since we are dealing with failures, such restrictions are desirable, and thus enforced in our specification.

3.1 Uniform Reliable Broadcast

Uniform Reliable Broadcast [18] is a communication primitive requiring that all correct processes deliver the same set of messages which includes only messages broadcast by processes and no spurious messages. Uniform Reliable Broadcast is defined in terms of two primitives, *broadcast(m)* and *deliver(m)* where m is a message drawn from a set $\mathcal{M}$ of possible messages. Uniform Reliable Broadcast satisfies the following three properties:

- *Validity:* If a correct process *broadcasts* a message m, then it eventually *delivers* m.
- *Uniform Agreement:* If a process *delivers* a message m, then all correct processes eventually *deliver* m.
- *Uniform Integrity:* For any message m, each process *delivers* m at most once, and only if m was previously *broadcast* by the sender of m.

Validity plus *Uniform Agreement* ensure that a message broadcast by a correct process is delivered by all correct processes.

3.2 Uniform Atomic Broadcast

Uniform Atomic Broadcast [9, 18, 1] is a communication primitive requiring that all correct processes deliver the same sequence of messages in the same order, and that this sequence of messages include only messages broadcast by processes and no spurious messages. Uniform Atomic Broadcast is a Uniform Reliable Broadcast that satisfies the additional property of Uniform Total Order:

- *Uniform Total Order:* If any two processes p and q both *deliver* two messages m and m' then p *delivers(m)* before it *delivers(m')* if and only if q *delivers(m)* before it *delivers(m')*.

3.3 Uniform Prefix Agreement

Before defining Uniform Prefix Agreement [2], let us first recall the definition of the Consensus Problem as given in [5][1]: each process that executes Consensus proposes one initial value v, and may irrevocably decide on one value based on the initial values. Chandra and Toueg define the Consensus problem in terms of two primitives, *propose(v)* and *decide(v)*. The Consensus problem satisfies the following three properties:

- *Termination:* Every correct process eventually *decides* exactly one value.
- *Uniform Integrity:* If a process *decides* v, then v was *proposed* by some process.
- *Uniform Agreement:* If a process *decides* v, then all correct processes eventually *decide* v.

We extend Consensus by allowing processes to propose not only one but several values[2] during the execution of Uniform Prefix Agreement avoiding thus idle periods. Because of this flow of proposed values during the execution of Uniform Prefix Agreement, and due to the asynchrony of the system, processes may decide on different sequences of values. However, Uniform Prefix Agreement guarantees that all the decision values are related by the *prefix property* defined as follows:

Definition 1 (prefix property) *Let $\mathcal{A}$ and $\mathcal{B}$ be any two sequences. $\mathcal{A}$ and $\mathcal{B}$ meet the prefix property if and only if either $\mathcal{A}$ is a prefix of $\mathcal{B}$ or $\mathcal{B}$ is a prefix of $\mathcal{A}$.*

In Uniform Prefix Agreement all correct processes propose several sets of messages and each one eventually decides on a prefix of a same ordered set of messages, each message composing this decided ordered set having been proposed by at least one correct process. As Consensus, Uniform Prefix Agreement is defined in terms of two primitives, *propose* and *decide*. When some process executes *propose*(μ,ν), we say that it *proposes* μ assuming ν, with μ (resp. ν) a set (resp. an ordered set) of message identifiers. When some process executes *decide*(μ,ν), we say that it *decides* μ assuming ν, with μ and ν two ordered set of messages. A process is not restricted to *propose* only one set of messages during an execution of Uniform Prefix Agreement. Rather, if while waiting for the decision value, it has new sets of messages to *propose*, it *proposes* them to the current execution of Uniform Prefix Agreement.

Let us describe more precisely the role of ν. ν is the sequence, i.e., the ordered set, of message identifiers that may not have been entirely decided by all correct processes during an execution of Uniform Prefix Agreement. ν enforces the Uniform Agreement property by making the *decide* primitive to return the largest decision value that some process may have decided. Returning this value ν will allow each process to propose ν to the next execution of Uniform Prefix Agreement. ν guarantees that the Uniform Agreement of Consensus holds every two successive executions of Uniform Prefix Agreement. The Uniform Prefix Agreement is then completely defined as follows:

- *Termination:* Every correct process eventually *decides* exactly one value.
- *Uniform Integrity:* If a process *decides* μ assuming ν, then each message in μ was *proposed* by some processes, and $\nu = \text{Prefix}(\mu)$.
- *Uniform Prefix Agreement:* If a process *decides* μ assuming ν, then each correct process eventually *decides* μ' assuming ν' with μ and μ' (resp. ν and ν') satisfying the prefix property.

The following requirement holds whenever successive executions[3] of Uniform Prefix Agreement are invoked:

- *Uniform Agreement:* For any execution $k > 1$ of Uniform Prefix Agreement, and for any process p (whether correct or faulty). If p *decides* μ_p^k, then all correct processes q eventually *decide* μ_q^{k-1} and $\mu_q^{k-1} = \text{Prefix}(\odot_{j=k-1}^{k}\mu_p^j)$ (the definition of the concatenation operator $\odot$ is given in Figure 3).

In the following, for the sake of brevity, we refer to Uniform Atomic Broadcast, Uniform Reliable Broadcast and Uniform Prefix Agreement, simply as Atomic Broadcast, Reliable Broadcast and Prefix Agreement. Since Atomic Broadcast and Reliable Broadcast use the same primitives *broadcast* and *deliver*, we disambiguate them by preceding these primitives by "A" if they refer to Atomic Broadcast and by "R" if they refer to Reliable Broadcast.

4 Atomic Broadcast Algorithm

The algorithm we propose to solve Atomic Broadcast relies on Reliable Broadcast as well as repeated and concurrent (but not independent) executions of Prefix Agreement. We do not present an algorithm solving Reliable Broadcast since this communication primitive can be solved for asynchronous systems [18] and for several types of failures due to the absence of

[1] This version of Consensus is slightly different from the ones described in [12], but since our extension is based on it, we think more profitable to present Chandra and Toueg's version.

[2] Without loss of generality, these values we consider are messages.

[3] Processes disambiguate between successive executions of Uniform Prefix Agreement by tagging all the messages pertaining to the k^{th} execution with the number k, $k \geq 1$. The *propose* and *decide* primitives corresponding to the k^{th} invocation of Prefix Agreement by process p are then denoted by *propose*(μ_p^k,ν_p^{k-1},k) and *decide*(μ_p^k,ν_p^k,k). In the same way, variables pertaining to the k^{th} execution of Prefix Agreement are superscripted with k.

any *Termination* requirement. The Atomic Broadcast algorithm is modular: it uses the services provided by Reliable Broadcast as well as those provided by Prefix Agreement. This modularity makes the Atomic Broadcast algorithm small and easy to understand, and its proofs, also modular, are "safer". That is, when Atomic Broadcast algorithm invokes the Reliable Broadcast or Prefix Agreement primitives as a "black box", it can only relies on the specification of that "box", so its proofs cannot erroneously rely on a property that is only true for a *particular implementation* of that "box". Taking advantage of this modularity, we have chosen to first describe Atomic Broadcast since it is at the core of our paper, and then to present and prove in Section 5 our solution to Prefix Agreement.

4.1 Algorithm

The Atomic Broadcast algorithm is described in Figure 1 as three concurrent and indivisible tasks. Task 1 is in charge of *R.broadcasting* the messages submitted to Atomic Broadcast. In Task 2, processes store the messages they have *R.delivered* into a local buffer of messages, denoted *RnotA_msgs*. If, while waiting for the *decide* primitive, some process *R.delivers* a new message m, then it *proposes id(m)* to the current execution of Prefix Agreement. *id(m)* is defined as the tuple *(sender(m),seq(m))*, which makes every message unique. Task 3 uses the services provided by Prefix Agreement. Processes *propose* to Prefix Agreement the set of messages identifiers contained in *RnotA_msgs*. Upon receipt of a decision value, a process *A.delivers* each message contained its decision value.

We now show that Validity, Uniform Agreement, Uniform Integrity and Uniform Total order of Atomic Broadcast hold. To prove the correctness of our Atomic Broadcast algorithm, we assume that our Prefix Agreement algorithm is correct, that is, the Termination, the Uniform Integrity, the Uniform Prefix Agreement and the Uniform Agreement of Prefix Agreement hold. Section 5 presents the Prefix Agreement algorithm and provides the proofs of its correctness. By lack of space, some proofs of our Lemmata are omitted. The reader is invited to read them in [2].

Lemma 1 For all processes p (whether correct or faulty) and any message m, if $m \in RnotA\_msgs_p$, then for all correct processes q, eventually $m \in RnotA\_msgs_q$.

Proof: If $m \in RnotA\_msgs_p$ then p *R.delivered* m (in Task 2). By Uniform Agreement of Reliable Broadcast q eventually *R.delivers* m, and inserts m into $RnotA\_msgs_q$.

Every process p executes the following:

Initialization:

$RnotA\_msgs_p$: set of messages *R.delivered* but not yet *A.delivered* by p, initially $\emptyset$.
$RnotA\_ids_p$: identifiers of messages in $RnotA\_msgs_p$, initially $\emptyset$.
k: execution number of Prefix Agreement, initially 0.
seq_p^k: k^{th} sequence of identifiers of messages *A.delivered* by p, initially $\emptyset$.
μ_p^k: k^{th} sequence of message identifiers *decided* by p, initially $\emptyset$.
ν_p^k: array of two fields:
- $\nu_p^k[estimate]$: latest estimate of the decision value acknowledged in Prefix Agreement(k), initially $\emptyset$.
- $\nu_p^k[ts]$ is the round number in which $\nu_p^k[estimate]$ was adopted, initially 0.

$state_p$: state of process p, initially NOT PROPOSED.

```
{Task 1}
Upon receipt of A.broadcast(m)
  R.broadcast(m)
{Task 2}
Upon receipt of R.deliver(m)
  RnotA_msgs_p := RnotA_msgs_p ∪ {m}
  if (state_p = PROPOSED) then
    propose({id(m)},∅,k)
{Task 3}
do forever
  RnotA_ids_p := {id(m) | m ∈ RnotA_msgs_p}
  if ((RnotA_ids_p ≠ ∅) ∨ (μ_p^k ≠ ∅)) then
    k := k + 1
    propose(RnotA_ids_p, ν_p^{k-1}, k)
    state_p := PROPOSED
    wait for {decide(μ_p^k, ν_p^k, k)}
    state_p := NOT PROPOSED
    if (μ_p^k \ μ_p^{k-1} ≠ ∅) then
      seq_p^k := μ_p^k \ μ_p^{k-1} (see footnote †)
      ν_p^k[estimate] := ν_p^k[estimate] \ μ_p^{k-1} (see footnote ‡)
      repeat
        id := First(seq_p^k)
        wait for {m ∈ RnotA_msgs_p | id = id(m)}
        RnotA_msgs_p := RnotA_msgs_p − {m}
        seq_p^k := seq_p^k − {id}
        A.deliver(m)
      until (seq_p^k = ∅)
    else // empty
  else //empty
```

† Process p builds seq_p^k with its k^{th} decision value from which it has removed the overlapping subsequence corresponding to its $(k-1)^{th}$ decision value. Removing an overlapping sequence from another sequence is denoted with the minus operator "\". By definition, let $\mathcal{A}$ and $\mathcal{B}$ be any two sequences. Let $\langle e_{i_j}, ..., e_{i_l} \rangle$ be the maximal sequence such that Prefix($\mathcal{A}$) = Suffix($\mathcal{B}$) = $\langle e_{i_j}, ..., e_{i_l} \rangle$. Then, the sequence $\mathcal{A} \setminus \mathcal{B}$ is equal to the sequence $\mathcal{A}$ from which has been removed the sequence $\langle e_{i_j}, ..., e_{i_l} \rangle$. For example if $\mathcal{B} = \langle e_1, e_2, e_3 \rangle$ and $\mathcal{A} = \langle e_3, e_4 \rangle$, then $\mathcal{A} \setminus \mathcal{B} = \langle e_4 \rangle$.

‡ Process p updates $\nu_p^k[estimate]$ by removing from it a prefix corresponding to a subsequence of its $(k-1)^{th}$ decision value. This prefix is the maximal prefix that can be removed from $\nu_p^k[estimate]$ without violating the Uniform Total Order property of Atomic Broadcast. Indeed, by the Uniform Prefix Agreement property of Prefix Agreement, some process q may have *decided* in Prefix Agreement(k) a prefix of p k^{th} decision value, thus it is necessary to provide in Prefix Agreement($k+1$) this updated $\nu_p^k[estimate]$.

Figure 1: **Using Prefix Agreement and Reliable Broadcast to solve Atomic Broadcast**

Lemma 2 For all processes p (whether correct or faulty) and all $k > 1$, p does not execute *propose*(∗,∗,k) [4] unless it has previously executed *decide*(μ_p^{k-1},∗,k-1), removed from $RnotA\_msgs_p$ all the messages whose identifiers are in seq_p^{k-1}, and *A.delivered* them.

Proof: Suppose process p executes *propose(∗,∗,k)*. From Task 3, p previously executed *decide(∗,∗,k-1)*, built seq_p^{k-1}, and later exited from the repeat-until loop, satisfying the condition "$seq_p^{k-1} = \phi$". Thus p must have previously removed from $RnotA\_msgs_p$ all the messages whose identifiers are in seq_p^{k-1} and *A.delivered* them.

Lemma 3 For all processes p (whether correct or faulty) and for any message m. Message m is *A.delivered*, and thus removed from $RnotA\_msgs_p$, only if it exists $k \geq 1$ such that *id(m)* = First($seq_p^k(i)$), with $0 \leq i < |seq_p^k|$ and First($seq_p^k(i)$), the $(i+1)^{th}$ message identifier of seq_p^k, with $0 \leq i < |seq_p^k|$.

Proof: Straightforward from Task 3 (Task 3 is the only task in which a message m can be removed from $RnotA\_msgs_p$).

Lemma 4 For all $k \geq 1$, for all processes p (whether correct or faulty) and for any two messages m and *m'*. If p *A.delivers* m such that *id(m)* = First($seq_p^k(i)$) with $0 \leq i < |seq_p^k|$, and successively [5] *A.delivers* *m'* such that *id(m')* = First($seq_p^{k+d}(i')$), then $i' \equiv (\ i\ +\ 1) \bmod (|seq_p^k|)$ and $d = 1$ if $i' = 0$, $d = 0$ otherwise.

Proof : Straightforward from Task 3 and Lemma 3.

In the following lemma, we say that process p is the "first" process to *A.deliver* some message m if for some $k \geq 1$, p *A.delivers* m such that $id(m) \in seq_p^k$ and for all correct processes, μ_q^k = Prefix(μ_p^k) with *id(m)* $\in \mu_p^k$.

Lemma 5 For all $k \geq 1$, and for any process p (whether correct or faulty):

- If p executes *propose*(∗,∗,k), then all correct processes q eventually execute *propose*(∗,∗,k).
- If p *decides* μ_p^k, then all correct processes q eventually *decide* μ_q^k, with μ_p^k and μ_q^k related by the prefix property.
- If p is the first process to *A.deliver* message m such that *id(m)* = First($seq_p^k(k_l)$), with $0 \leq k_l < |seq_p^k|$, then all correct processes q eventually *A.deliver* m and *id(m)* = First($seq_q^{k+h}(k_l')$), with:

$h = \mathcal{H}(\sum_{j=1}^{k-1}|seq_p^j| - \sum_{j=1}^{k}|seq_q^j| + k_l)$ and
$k_l' = \sum_{j=1}^{k-1}|seq_p^j| - \sum_{j=1}^{k-1+h}|seq_q^j| + k_l$,

with $\mathcal{H}$ the Heaviside(t) unit step function. By definition, $\mathcal{H}(t) = 0$ for $t < 0$, and $\mathcal{H}(t) = 1$ for $t \geq 0$.

Proof: For lack of space, the proof is omitted from this paper but is given in [2].

Lemma 6 (Uniform Agreement) If a process *delivers* a message m, then all correct processes eventually *deliver* m.

Proof: Immediate from part three of Lemma 5.

Lemma 7 (Uniform Integrity) For any message m, each process *A.delivers* m at most once, and only if m was previously *A.broadcast* by *sender(m)*.

Proof: Suppose a process p *A.delivers* m. By Lemma 2, p must have *R.delivered* m and $id(m)$ must have been in seq_p^k for some $k \geq 1$. Before p *A.delivers* m, p removes m from $RnotA\_msgs_p$. By the Uniform Integrity of Reliable Broadcast p cannot *R.deliver* m again. Thus p will never *A.deliver* m again.

From the algorithm, p executed *decide*(μ_p^k,∗,k) for some $k \geq 1$ with $id(m) \in \mu_p^k$. By Uniform Integrity of Prefix Agreement, some process q must have executed *propose*($RnotA\_ids_q$,∗,k) with $RnotA\_ids_q^k \subseteq \mu_p^k$ and $id(m) \in RnotA\_ids_q$. So q previously *R.delivered* m. By Uniform Integrity of Reliable Broadcast, some process r *R.broadcast* m. So r *A.broadcast* m (Task 1). This concludes the proof of the lemma.

Lemma 8 (Uniform Total Order) If any two processes p and q both *A.deliver* two messages m and *m'* then p *A.delivers(m)* before it *A.delivers(m')* if and only if q *A.delivers(m)* before it *A.delivers(m')*.

Proof: Since both p and q *A.deliver* m, by Lemma 3, it must exist:

- k_1 with $k_1 \geq 1$, such that $id(m)$ = First($seq_p^{k_1}(i_{k_1})$) with $0 \leq i_{k_1} < |seq_p^{k_1}|$, and
- k_2 with $k_2 \geq 1$, such that $id(m)$ = First($seq_q^{k_2}(i_{k_2})$) with $0 \leq i_{k_2} < |seq_q^{k_2}|$.

By Lemma 5, either $k_2 = k_1 + h$ with:

$$\begin{cases} h &= \mathcal{H}\ (\ \sum_{j=1}^{k_1-1}|seq_p^j| - \sum_{j=1}^{k_1}|seq_q^j|\ +\ i_{k_1}) \\ i_{k_2} &= \sum_{j=1}^{k_1-1}|seq_p^j| - \sum_{j=1}^{k_1-1+h}|seq_q^j|\ +\ i_{k_1} \end{cases}$$

[4] For simplicity, we replace by the minus sign "∗" the primitive parameters that are not determining for the comprehension of the text.

[5] We say that a process p *A.delivers* message m and successively *A.delivers* message *m'* if there is no message *m"* such that p *A.delivers m"* after m and before *m'*, with the *after* and *before* relations as defined by Lamport [20].

$$\text{i.e.}: \sum_{j=1}^{k_1-1+h} |seq_q^j| + i_{k_2} = \sum_{j=1}^{k_1-1} |seq_p^j| + i_{k_1} = N_1 \quad (1)$$

or $k_1 = k_2 + h$ with:

$$\begin{cases} h = \mathcal{H}\left(\sum_{j=1}^{k_2-1} |seq_q^j| - \sum_{j=1}^{k_2} |seq_p^j| + i_{k_2} \right) \\ i_{k_1} = \sum_{j=1}^{k_2-1} |seq_q^j| - \sum_{j=1}^{k_2-1+h} |seq_p^j| + i_{k_2} \end{cases}$$

$$\text{i.e.}: \sum_{j=1}^{k_2-1+h} |seq_p^j| + i_{k_1} = \sum_{j=1}^{k_2-1} |seq_q^j| + i_{k_2} = N_2 \quad (2)$$

Equations 1 and 2 mean that both p and q *A.delivered* the same number of messages, N_1 — case 1 or N_2 — case 2, before *A.delivering* message m.

A similar argument for message m' shows that both p and q *A.deliver* the same number of messages before *A.delivering* message m'. Thus p and q both *A.deliver* m and m' in the same order. This concludes the proof of the lemma.

Lemma 9 (Validity)
If a correct process *A.broadcasts* a message m, then it eventually *A.delivers* m.

Proof: The proof is by contradiction. Suppose some correct process p *A.broadcasts* m, and some correct process never *A.delivers* m. By Uniform Agreement of Atomic Broadcast, no correct process *A.delivers* m. By Task 1, p *R.broadcasts* m. By Validity of Reliable Broadcast, every correct process q eventually *R.delivers* m, and inserts m into $RnotA\_msgs_q$ (Task 2). Since the correct processes q never *A.deliver* m, there is a time after which m is permanently in $RnotA\_msgs_q$. From Figure 1 and Lemma 5, there is a k_1, such that for all $l \geq k_1$, all correct processes execute *propose(∗,∗,l)*, and they do so with sets of identifiers that always include $id(m)$. Since all faulty processes eventually crash, there is a k_2, such that no faulty process executes *propose(∗,∗,l)* with $l \geq k_2$. Let $k = max(k_1, k_2)$. Since all correct processes execute *propose(RnotA_msgs,∗,k)* with $id(m) \in RnotA\_msgs$, by Termination and Uniform Agreement of Prefix Agreement, all correct processes execute *decide(μ^k,∗,k)* with $id(m) \in \mu^k$. Thus all correct processes *A.deliver* m - a contradiction that concludes the proof.

Note that when Atomic Broadcast has no more messages to be *A.delivered*, one *dummy* execution of Prefix Agreement is run to guarantee the liveness property (see termination test of Task 3).

Theorem 1 *Consider any system (synchronous or asynchronous) subject to benign failures and where Uniform Reliable Broadcast can be implemented. Figure 1 uses the Uniform Prefix Agreement to solve Uniform Atomic Broadcast.*

Proof: Immediate from Lemmata 8, 6, 9, and 7.

5 Prefix Agreement algorithm

In this section we present an algorithm to solve Prefix Agreement. As for the Atomic Broadcast algorithm, this algorithm is developed modularly. It uses the services provided by a Reliable Broadcast module and those provided by any Eventual Strong failure detector $\diamond\mathcal{S}$ module. With regard to $\diamond\mathcal{S}$, a complete characterization of this failure detector is given in [5].

5.1 Assumptions

The algorithm in Figure 2 solves Prefix Agreement using any Eventual Strong failure detector $\diamond\mathcal{S}$. Each process periodically consults its failure detector module and can use the set of suspects returned in solving Prefix Agreement. With $\diamond\mathcal{S}$, all processes may be erroneously added to the list of suspects at any time. However, there is a correct process and a time after which that process is not suspected to have crashed [5]. Prefix Agreement is an extension of the Chandra and Toueg consensus algorithm with $\diamond\mathcal{S}$. The main differences between these algorithms come from the handling of flows of *propose* primitives during the algorithm to guarantee the construction of consistent decision values (see Lemma 11) and their consistency along different executions of the algorithm (see Lemma 12).

5.2 Algorithm

The algorithm, presented in Figure 3 uses the *rotating coordinator* paradigm [6, 8, 4]. Computation proceeds in asynchronous "rounds". All the processes execute the same sequence of rounds. Each round $r \geq 1$ is divided into 4 asynchronous phases. We assume that all processes have a priori knowledge that during round r, the coordinator is process $c \equiv (k+r)[n] + 1$, with k denoting the k^{th} execution of Prefix Agreement[6]. At any given time, two different processes may be in two different rounds and thus may have two different coordinators. Every time a process becomes a coordinator, this process tries to *decide* on the sequence of messages to be provided to the Atomic Broadcast module: the current coordinator locks a sequence of message identifiers, and if it is correct and is not suspected by any surviving process, then it succeeds: it *R.broadcasts* a request to *decide* the locked sequence of message identifiers. Upon *R.delivery* of a *decide* request, a process *decides* accordingly. Note that before *deciding*, a process p updates ν_p^k with the largest estimate of the decision value it has acknowledged or with the *R.delivered* decision value, if its

[6]The notation $x \equiv y[z]$ stands for $x \equiv y$ modulo z.

The code for process p running Prefix Agreement is the following one:

CurrentBuffList: list of buffers where the proposed messages identifiers received during the repeated executions of Prefix Agreement are stored, initially $\emptyset$.
CurrentBuff(k): buffer shared with the the thread running the Propose procedure, initially $\emptyset$.

```
do forever
    Upon receipt of propose(μ, ν, k)
    if (k = current_k + 1) then
        allocate CurrentBuff(k)
        append μ to CurrentBuff(k)
        create a thread running Propose(ν, k)
        current_k:= current_k + 1
    else
        if (k ≤ current_k) then
            append μ to CurrentBuff(k)
        else //empty
```

Figure 2: **Solving Uniform prefix Agreement with the Eventual Strong Failure Detector**

largest estimate is "older" than the decision value. It then updates μ_p^k with the decision value.

Recall that processes may invoke the *propose* primitive several times during the same execution of Prefix Agreement. From an operational point of view, receipt of these multiple *propose* primitives are handled via a thread mechanism running our algorithm described here above. Our Prefix Agreement algorithm, upon receipt of a *propose* primitive, appends the proposed message identifiers to a buffer, denoted *CurrentBuff[k]*. If this *propose* primitive is the first one of a new execution k then a k^{th} thread *Propose(*,k)* is created. The Prefix Agreement algorithm is presented in Figure 2.

We have to show that the Termination, Uniform Integrity, Uniform Prefix Agreement and Uniform Agreement of Prefix Agreement hold. Some of the proofs are omitted by lack of space, but they are detailed in [2].

Lemma 10 (Termination) For any execution $k \geq 1$ of Prefix Agreement, if a process (whether correct or faulty) executes *decide*(*,*,k) then all correct processes eventually execute *decide*(*,*,k) exactly once.

Sketch of the proof: There are two cases.

• Some process *decides*. This process must have *R.delivered* a *(*,*,decide,k)* message. By the Uniform Agreement property of Reliable Broadcast and Task 2 all correct processes eventually *R.deliver* this message and decide.

• No process decides. We show that no correct process remains blocked forever at one of the "wait until" statements. The proof is by contradiction. By assumption, a majority of processes are correct. Thus, in Phase 2, either the current coordinator c receives a majority of replies and thus does not block or, c crashes. If c crashes then, by the Eventual Strong Completeness of $\diamond\mathcal{S}$, all correct processes eventually suspect c and thus do not block at the "wait until" statement in Phase 3. Since there are at least $\lceil(n+1)/2\rceil$ correct processes, c cannot block at "wait until" statement of Phase 4. This shows that all correct processes complete round r. This is a contradiction that completes the proof of the claim.

We then show, by the Eventual Weak Accuracy property of $\diamond\mathcal{S}$, that there is a correct process q and a time t_1 such that no correct process suspects q after time t_1. Thus, all processes that suspect q after time t_1 eventually fail, and there is a time $t' \geq t_1$, such that no process sends a *nack* message to q (from Figure 3, q eventually becomes coordinator). Thus, q eventually *R.broadcasts* a decide message, and by the Validity property of Reliable Broadcast, eventually all correct processes *R.deliver* this *decide* message and *decide*. This is a contradiction with the assumption of the case. Thus this case is impossible, and this concludes the proof.

Lemma 11 (Uniform Prefix Agreement) For any execution $k \geq 1$, and for any process p (whether correct or faulty). If p *decides* μ_p^k then each correct process *p'* eventually *decides* $\mu_{p'}^k$ with μ_p^k and $\mu_{p'}^k$ satisfying the prefix property.

Sketch of the proof: If any process decides, it must be the case that a coordinator *R.broadcast* a message of the type *(*,*,decide,k)* in Phase 4. This coordinator must have received $\lceil(n+1)/2\rceil$ acknowledgments in Phase 4. Let r be the smallest round number in which $\lceil(n+1)/2\rceil$ acknowledgments are sent to a coordinator in Phase 3. Let c denote the coordinator of round r, i.e., $c \equiv (r+k)[n]+1$. Let est_c^k denote c's estimate at the end of Phase 2 of round r. We claim that for all rounds r', with $1 \leq r \leq r'$, if coordinator c' sends $est_{c'}^k$ in Phase 2 of round r', then $est_c^k =$ Prefix($est_{c'}^k$). The proof of the claim is by induction on the round number. Briefly it relies on a majority argument — there exists some process p such that p sends an acknowledgment in Phase 3 of round r and such that the coordinator of the next round selects p's estimate with p's estimate the most recent estimate of the execution — and on the fact that the coordinator concatenates estimates.

We now show that if any two processes p and p' execute respectively *decide*(μ_p^k,*,k) and *decide*($\mu_{p'}^k$,*,k), then μ_p^k and $\mu_{p'}^k$ meet the prefix property. Suppose that some process p *R.delivers* (r_q,est_q^k,*decide,k*) message, and then executes *decide*(μ_p^k,*,k) with $\mu_p^k = est_q^k$.

procedure **Propose**(ν^{k-1}, k)

ν_p^{k-1}	:	two fields vector:
		$\nu_p^{k-1}[estimate]$:= suffix of the latest estimate of acknowledged by p in Prefix Agreement$(k-1)$.
		$\nu_p^{k-1}[ts]$:= timestamp of this latest estimate of the decision value.
μ_p^k	:	k^{th} decision value.
est_p^k	:	k^{th} estimate of process p, initially $= \nu_p^{k-1}[estimate]$.
new_ids_p	:	buffer of message identifiers accumulated during the current execution of Prefix Agreement.
ts_p	:	timestamp of process p, initially 0.
r_p	:	round of the current execution of Prefix Agreement, initially 0.
$state_p$	:	state of process p, initially undecided.

Rotate through coordinators until a decision is reached
{Task 1} while $state_p = undecided$
  $r_p := r_p + 1$
  $c_p := ((k + r_p) \bmod n) + 1$ *//c_p is the current coordinator*
  Phase 1: *// all the processes p send est_p as well as* CurrentBuff *to c_p*
    move *$CurrentBuff_p$* into new_ids_p *//$CurrentBuff_p$ is emptied in new_ids_p*
    if $(k \neq 1 \wedge r_p \neq 1)$ then
      send$(p, r_p, est_p^k, new\_ids_p, \nu_p^{k-1}[ts], ts_p, k)$ to c_p
  Phase 2: *// if $(k \neq 1 \wedge r_p \neq 1)$, c_p gathers $\lceil (n+1)/2 \rceil$ estimates and proposes a new estimate*
    if $(p = c_p)$ then
      if $(k = 1 \wedge r_p = 1)$ then
        $est^k := new\_ids_p$ from which has been removed duplicates *//c directly proposes its estimate*
      else
        wait until $\{(q, r_p, est_q^k, new\_ids_q, \nu_q^{k-1}[ts], ts_q, k)$ is received from $\lceil (n+1)/2 \rceil$ processes $q\}$
        $MsgSet_p^k[r_p]$:= the $\lceil (n+1)/2 \rceil$ $(q, r_p, est_q^k, new\_ids_q, \nu_q^{k-1}, ts_q, k)$ p received$\}$
        t_{max}:= max $\{ts_q \mid (q, r_p, est_q^k, new\_ids_q, \nu_q^{k-1}[ts], ts_q, k) \in MsgSet_p^k[r_p]$
        if $(t_{max} \neq 0)$ then
          $est^k := est_q^k$ where $(q, r_p, est_q^k, new\_ids_q, \nu_q^{k-1}[ts], t_{max}, k) \in MsgSet_p^k[r_p]$
        else
          $\nu_{max}[ts]$:= max $\{\nu_q^{k-1}[ts] \mid (q, r_p, est_q, new\_ids_q, \nu_q^{k-1}[ts], t_{max}, k) \in MsgSet_p^k[r_p]\}$
          $est^k := est_q^k$ where $(q, r_p, est_q^q, new\_ids_q, \nu_{max}[ts], t_q, k) \in MsgSet_p^k[r_p]$
        new_ids := union of all the new_ids_q in $MsgSet_p^k[r_p]$ from which duplicates are removed
        $new\_ids := new\_ids \setminus est^k$
        $est^k := est^k \odot new\_ids$ (see footnote †)
      send(p, r_p, est^k, k) to all
  Phase 3: *//all processes wait for c_p estimate*
    wait until {received(c_p, r_p, est^k, k) or $c_p \in \Diamond\mathcal{S}_p\}$
    if (received(c_p, r_p, est^k, k)) then
      $ts_p := r_p$
      $est_p^k := est^k$
      send(p, r_p, ack, k) to c_p
    else
      send$(p, r_p, nack, k)$ to c_p *//p suspects that c_p crashed*
  Phase 4: *//c_p waits for $\lceil (n+1)/2 \rceil$ replies. If there are $\lceil (n+1)/2 \rceil$ acks then c R.broadcast a decide message*
    if $(p = c_p)$ then
      wait until {for $\lceil (n+1)/2 \rceil$ q: received *(q, r_p, ack, k)* or *$(q, r_p, nack, k)$*}
      if (for $\lceil (n+1)/2 \rceil$ q: received *(q, r_p, ack, k)*) then
        R.broadcast*$(r_p, est_p^k, decide, k)$*
      else *//empty*
//when p R.delivers a decide message, it decides accordingly
{Task 2}
  when {**R.deliver***$(r_q, est_q^k, decide, k)$*}
  if $(r_q > ts_p)$ then *// p has not received the current* estimate *in Phase 3*
    $\nu_p^k[estimate] := est_q^k$
    $\nu_p^k[ts] := r_q$
  else
    $\nu_p^k[estimate] := est_p^k$
    $\nu_p^k[ts] := ts_p$
  $\mu_p^k := est_q^k$
  decide*(μ_q^k, ν_p^k, k)*
  $state_p := decided$

†The concatenation operator is denoted by "$\odot$", and is defined as follows: let $\mathcal{A}$ and $\mathcal{B}$ be any two sequences. $\mathcal{A} \odot \mathcal{B}$ is equal to the sequence containing each element of $\mathcal{A}$ followed by each element of $\mathcal{B}$, after having removed from $\mathcal{B}$ the maximal subsequence such that Suffix$(\mathcal{A})$ = Prefix$(\mathcal{B})$. More explicitly, if $\mathcal{A} = \prec e_1, e_2, e_3, e_4 \succ$ and $\mathcal{B} = \prec e_3, e_4, e_5, e_6 \succ$, then $\mathcal{A} \odot \mathcal{B} = \prec e_1, e_2, e_3, e_4, e_5, e_6 \succ$.

Figure 3: **Solving the k^{th} Uniform prefix Agreement with the Eventual Strong Failure Detector**

Process q must have previously *R.broadcast* the message $(r_q, est_q^k, decide, k)$ in Phase 4 of round r_q. From Figure 3, q must have received a majority of acknowledgments in Phase 4 of round r_q. By the definition of r, $r \leq r_q$. From the above claim, est_c^k = Prefix(est_q^k). Similarly, suppose that some process p' *R.delivers* $(r_{q'}, est_q^k, decide, k)$ message, and executes $decide(\mu_{p'}^k, *, k)$ with $\mu_{p'}^k = est_{q'}^k$. With a similar argument, we have est_c^k = Prefix($est_{q'}^k$). From the proof of the above claim, if $r \leq r_q \leq r_{q'}$ then $est_q^k = Prefix(est_{q'}^k)$, otherwise $est_{q'}^k$ = Prefix(est_q^k). Thus, if $r \leq r_q \leq r_{q'}$ then μ_p^k = Prefix($\mu_{p'}^k$), otherwise $\mu_{p'}^k$ = Prefix(μ_p). This completes the proof of the lemma.

Lemma 12 (Uniform Agreement) For any execution $k > 1$ of Prefix Agreement, and for any process p (whether correct or faulty). If p executes $decide(\mu_p^k, *, \text{k})$, then all correct processes q eventually execute $decide(\mu_q^{k-1}, *, \text{k-1})$ and μ_q^{k-1} = Prefix($\odot_{j=k-1}^{k} \mu_p^j$).

Proof: the proof is omitted.

Lemma 13 (Non blocking) No correct process remains forever in an execution of Prefix Agreement.

Proof: the proof is omitted.

Lemma 14 (Uniform Integrity) If a process (whether correct or faulty) *decides* μ assuming ν, then each message in μ was *proposed* by some processes, and ν = Prefix(μ).

Proof: From the algorithm, it is clear that all the *estimates* that a coordinator receives in Phase 2 are proposed values. Therefore, the decision value that a coordinator selects from theses *estimates* must be the values proposed by some processes. Thus Uniform Integrity of Prefix Agreement is satisfied.

Theorem 2 *The algorithm described in Figures 2 and 3 solves Uniform Prefix Agreement.*

Proof: Immediate from Lemmata 10, 11, 12, and 14.

6 Related Work

The problem of atomic broadcast in asynchronous systems has received considerable attention in the distributed processing community. Different approaches have been adopted to circumvent the impossibility result of [12]. The first one is due to [8] for which the authors consider two models of partial synchrony. The authors give a solution to Consensus in both models, and although they do not give a solution to Atomic Broadcast, we think that it will be straightforward to extend their solution of Consensus to Prefix Agreement and thus to Atomic Broadcast. F. Cristian et al. in [11] propose an Atomic Broadcast Algorithm, called *On-demand atomic broadcast*. Their algorithm tolerates crash, omission and performance failures and has good performance with bursts of messages. Their system, that they called *Timed-Asynchronous* system, is different from ours since they assume that local clocks measure time with a known accuracy, that processes react to message arrival or timeout within some delay, and that most messages are transmitted within a timeout delay. I. Keidar et al. present an algorithm for totally ordering messages for asynchronous systems [19]. Their algorithm uses an underlying transport layer that serves as a failure detector. Contrary to our solution, their algorithm handles network partitions and process failures. However they assume that their transport layer supplies reliable locally ordered group multicast service. U. Wilhelm et al. in [26] identify a hierarchy of total order communication paradigms. Their model of computation follows the one of [5]. They present an algorithm incorporating the two extremes of total order requirement, namely the weakest and the strongest total order. Contrary to our work in which we focus on the efficiency of the algorithm solving the strong total order reliable broadcast, their solution can be used to optimize the communication cost of an application by selecting the cheapest communication paradigm for the required task.

7 Conclusion

In this paper, we have presented a lightweight algorithm to solve Uniform Atomic Broadcast for asynchronous systems augmented with an Unreliable Failure Detector [5]. Our Uniform Atomic Broadcast algorithm tolerates crash failures and requires a majority of correct processes. It uses the services provided by the Uniform Prefix Agreement, an extension of the Consensus problem. This extension enables all correct processes to propose a flow of messages during an execution, instead of one as in Consensus, avoiding accordingly the idle period of time inherent to Consensus. Uniform Prefix Agreement uses all these proposed messages to compose the decision value, increasing accordingly the number of messages decided per execution, at a cost equal to the one exhibited in [5] for one message.

Using the eventual strong failure detector $\diamond\mathcal{S}$ of [5] and requiring a majority of correct processes ensures that even if the failure detector does not meet its specification the safety properties of our algorithms will never be violated.

Furthermore, as in [5] and contrary to [19, 11], we do not handle group membership changes: even even if a process is suspected to be faulty by all the other processes, by asking their failure detector $\diamond\mathcal{S}$, this process will never be "considered" as faulty by them. That is, this process may be silent for a while by not acknowledging messages, however it will never be prevented from atomically broadcasting messages in the group, and as long as it will not have been really crashed, it will atomically deliver the same sequence of messages as all the correct processes do.

Acknowledgment

I am grateful to Sam Toueg for extremely useful discussions on the extension of Consensus, and to Laurent Amsaleg for his invaluable help. I would like to thank the anonymous referees for their useful comments and suggestions on the paper.

References

[1] E. Anceaume. *A comparison of fault-tolerant atomic broadcast protocols.* In *4th Workshop on Future Trends of Distributed Computing Systems*, sept., 1993.

[2] E. Anceaume. A lightweight solution to uniform atomic broadcast for asynchronous systems: proofs. Tech. Report PI 1066, IRISA, http://www.irisa.fr/EXTERNE/projet/solidor/doc/pub96.html, nov. 1996.

[3] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Trans. on Computer Systems*, 5(1):47–76, feb. 1987.

[4] T. Chandra and S. Toueg. Time and message efficient reliable broadcasts. In *4th Int. Workshop on Distributed Algorithms*, pages 289–300. Springer-Verlag, sept. 1990.

[5] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. In *10th ACM Symp. on Principles of Distributed Computing*, pages 325–340, aug. 1993.

[6] J. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM Trans. on Computer Systems*, 2(3):251–273, aug. 1984.

[7] F. Cristian. Issues in the design of highly available computing service. In *Annual Symp. of the Canadian Information Processing Society*, pages 9–16, july 1987.

[8] C. Dwork et al. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, april 1988.

[9] F. Cristian et al. Atomic broadcast: from simple message diffusion to byzantine agreement. In *5th Int. Symposium on Fault-Tolerant Computing*, pages 200–206, june 1985.

[10] F. Cristian et al. Fault-tolerance in the advanced automation system. Technical Report 7424, IBM Reasearch Laboratory, april 1990.

[11] G. Alvarez et al. On-demand asynchronous atomic broadcast. In *5th IFIP Working Conf. on Dependable Computing and Critical Applications*, sept. 1995.

[12] M. Fisher et al. Impossibility of distributed consensus with one faulty process. *Journal of the ACM Symp. on Principles of Distributed Computing*, 32(2):374–382, april 1985.

[13] P. Ezhilchelvan et al. Newtop: a fault-tolerant group communication protocol. In *15th Int. Conference on Distributed Computing Systems*, june 1995.

[14] R. Van Renesse et al. A risc approach to process groups. In *5th ACM SIGOPS Worshop*, 1992.

[15] Y. Amir et al. Transis: A communication sub-system for high availability. In *22th FTCS conference*, july 1992.

[16] Y. Amir et al. Fast message ordering and membership using a logical token-passing ring. In *13th Int. Conference on Distributed Computing Systems*, pages 551–560, may 1993.

[17] A. Gopal and S. Toueg. Reliable broadcast in synchronous and asynchronous environments. In J.C. Bremond and Raynal editors, editors, *3rd Int. Worshop on Distributed Algorithms*, 1989.

[18] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In Sape J. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.

[19] I. Keidar and D. Dolev. Efficient message ordering in dynamic networks. In *15th ACM Symp. on Principles of Distributed Computing*, 1996.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. of the ACM*, 21(7):558–565, july 1978.

[21] L. Lamport and M. Fisher. Byzantine generals and transaction commit protocols. Technical Report 62, SRI International, april 1982.

[22] S. Mullender, editor. *The Amoeba Distributed operating system.* Center for Mathematics and Computer Science, 1987.

[23] G. Neiger. *Techniques for simplifying the design of distributed systems.* PhD thesis, Cornell University, aug. 1988.

[24] L. Peterson, N. Bucholz, and R. Schlichting. Preserving and using context information in interprocess communication. *ACM Trans. on Computer Systems*, 7(3):217–246, aug. 1989.

[25] D. Powell, editor. *Delta-4: A Generic Architecture for Dependable Distributed Computing.* Springler-Verlag, 1991.

[26] U. Wilhem and A. Schiper. A hierarchy of totally ordered multicasts. In *14th Symp. on Reliable Distributed Systems*, 1995.