On the Latency Efficiency of Message-Parsimonious Asynchronous Atomic Broadcast

Dan Dobre* HariGovind V. Ramasamy† Neeraj Suri‡

Abstract

We address the problem of message-parsimonious asynchronous atomic broadcast when a subset t out of n parties may exhibit Byzantine behavior. Message parsimony *involves using only the optimal $\mathcal{O}(n)$ message exchanges per atomically delivered payload in the normal case. Message parsimony is desirable for Internet-like deployment environments in which message loss rates are non-negligible. Protocol PABC, the only previously-known message-parsimonious solution, suffered from two limitations vis-à-vis the solutions with $\mathcal{O}(n^2)$ message complexity: more communication steps and the use of digital signatures. We present a protocol termed AMP that for the first time provides signature-free message parsimony while at the same time reducing the number of communication steps to the minimum necessary. In contrast to many previous atomic broadcast solutions, our protocol satisfies both safety and liveness in the asynchronous model.*

1. Introduction

Byzantine fault-tolerant (BFT) replication has garnered strong interest as a means of enhancing the trustworthiness of distributed systems. Replicating a service at geographically dispersed sites offers protection against catastrophic failures of a single site, such as outages, intrusions, and denial-of-service (DoS) attacks. Atomic broadcast is a fundamental communication primitive for the coordination of the replicas of such a replicated service. It allows a group of n parties to agree on a set of payload messages to deliver as well as on their delivery order, despite the failure of up to t parties. Our goal is to develop atomic broadcast protocols that are suitable for building highly available and intrusion-tolerant services in the Internet.

Because of the non-negligible message loss rates on the Internet, it has been shown by Bakr and Keidar [1] and more recently by Shraer and Keidar [14] that a broadcast protocol with a *one-to-all* communication pattern among the parties is likely to have significantly better performance than protocols that rely on the *all-to-all* communication pattern. In view of this observation, Ramasamy and Cachin recently proposed a *leader-based* atomic broadcast protocol (Protocol PABC) [12], based on the one-to-all communication pattern. The protocol guarantees both *safety* and *liveness* in asynchronous networks, circumventing the FLP impossibility result [5] by relying on randomized agreement under unstable or faulty conditions. The protocol is optimally resilient (i.e., $t < n/3$) with an amortized message complexity of $\mathcal{O}(n)$ in the fully asynchronous system model. Message complexity, an important metric for analyzing the efficiency of atomic broadcast protocols, denotes the number of protocol messages generated per atomically delivered payload. Previously, the most efficient solutions in the asynchronous model were the Byzantine Fault Tolerant or BFT protocol by Castro and Liskov [4], the KS protocol by Kursawe and Shoup [9], and the Fast Byzantine or FaB consensus protocol by Martin and Alvisi [10], all of which use the *all-to-all* communication pattern incurring a message complexity of $\mathcal{O}(n^2)$.

The reduced message complexity of the PABC protocol comes with three limitations. Firstly, even the optimized version takes a minimum of seven communication steps between the atomic broadcast and the atomic delivery of a payload, which is more than twice the number of steps needed by the BFT protocol. Secondly, the protocol uses digital signatures even under normal conditions. Finally, the PABC protocol overlooks the possibility of only a finite number of payloads ever being atomically broadcast; as a consequence, there may be runs in the PABC protocol in which a subset of correct parties is lagging behind other correct parties forever.

We have developed an atomic broadcast protocol that addresses the above drawbacks of the PABC protocol while at the same time retaining its advantages, namely, message parsimony and correctness despite asynchrony. The proto-

*D. Dobre is with the Dept of CS, TU Darmstadt, Germany. Email: dan@informatik.tu-darmstadt.de

†H. V. Ramasamy is with the IBM Zurich Research Laboratory, Switzerland. Email: hvr@zurich.ibm.com

‡N. Suri is with the Dept of CS, TU Darmstadt, Germany. Email: suri@cs.tu-darmstadt.de

col is called AMP, which stands for the Authenticator-based Message-Parsimonious protocol. Message parsimony [11] leverages the observation that conditions are *normal* during most of a system's operation and uses only the optimal $\mathcal{O}(n)$ message exchanges per atomically delivered payload under such conditions. Normal conditions refer to a stable network and no intrusions. Unlike the BFT protocol, and just like the PABC protocol, our new protocol satisfies both safety and liveness in the asynchronous model. In particular, under unstable network conditions, the deterministic BFT protocol can generate a potentially unbounded number of protocol messages by repeatedly switching protocol modes without making progress, thereby violating liveness.

Like the PABC protocol, the AMP protocol proceeds in *epochs*, where an epoch consists of a *parsimonious mode* and a *recovery mode*. The key idea of our solution is to replace the primitive used in the parsimonious mode of the PABC protocol, namely, *consistent broadcast* (CB), with a signature-free version while at the same time reducing the number of communication steps in the parsimonious mode. This replacement necessitates additional steps in the recovery mode to ensure that safety and liveness are always satisfied. As failures and instability are the exception rather than the norm in many real systems, we decided to optimize performance as much as possible for the normal case, even if it means incurring a slightly higher performance penalty during periods of instability.

The AMP protocol substitutes digital signatures used in the implementation of consistent broadcast with message authenticators [4, 15] based on symmetric cryptography. The protocol uses a minimum of five communication steps (versus the PABC protocol's seven) between the atomic broadcast of a payload and the payload's atomic delivery. The protocol achieves the reduction in steps by using only two (instead of the PABC protocol's three) consecutive consistent broadcast instances for each payload atomically delivered in the parsimonious mode. Unlike the PABC protocol, the AMP protocol guarantees that even in runs in which only a finite number of payloads are broadcast, all correct parties eventually atomically deliver the same set of payloads.

Table 1 compares normal-case operation of the AMP protocol with that of the BFT, FaB, and PABC protocols in terms of reliance on Public-Key (PK) operations, latency, message complexity and resilience.

Table 1. Normal-Case Operation of Efficient Byzantine-Fault-Tolerant Atomic Broadcast Protocols

Protocol	PK Ops?	Latency		#Msgs	Resilience
		#CB Instances	#Comm. Steps		
BFT [4]	no	N/A	3	$2n^2 + n$	$3t + 1$
FaB [10]	no	N/A	2	$n^2 + n$	$5t + 1$
PABC [12]	yes	3	7	$7n$	$3t + 1$
AMP	no	2	5	$5n$	$3t + 1$

2. Preliminaries

2.1 System Model

We consider an asynchronous distributed system model equivalent to the one of Cachin et al. [3] in which there are no bounds on relative processing speeds and message delays. The system consists of n *parties* $P_1, \ldots, P_n$ and an *adversary*. Up to t parties are controlled by the adversary and are called *corrupted*; the other parties are called *correct*. We use a *static* corruption model, and there is an algorithm run by a trusted *dealer* for generating the state information (e.g. keys) used to initialize each party. All computations by the parties, the adversary, and the dealer are probabilistic, polynomial-time algorithms. As our model is based on the formal approach in cryptography, we allow for a negligible probability of failure in the specification of our protocols. The system model includes a digital signature scheme that is secure against existential forgery using chosen-message attacks [6].

Each pair of parties is linked by an *authenticated asynchronous channel* that provides message integrity. Message authentication codes or MACs can be used to implement such a channel between two parties, e.g. using HMAC [2]. For this purpose, each pair of parties share a common key.

Some protocol messages that are sent to all parties contain *message authenticators*. A message authenticator is a vector of MACs, one per party. The sender computes the MAC for each party using the key it shares with that party. To verify the authenticity of a message containing an authenticator, the receiver just checks its corresponding MAC in the authenticator.

Messages on the channels are scheduled by the adversary. However, we assume that every message on a channel between two correct parties is *eventually* delivered. Every protocol instance is identified by a unique string ID, called the *tag*. Formally, the local interface to our protocols consists of *input actions*, which are messages of the form $(ID, \texttt{in}, type, \ldots)$, and *output actions*, which are messages of the form $(ID, \texttt{out}, type, \ldots)$. The parties receive and generate *protocol messages* of the form $(ID, type, \ldots)$, delivered to other parties over the channels. Before a party starts to process messages for an instance ID, the instance with that ID must be *initialized*.

2.2 Protocol Primitives

2.2.1 Extensions of Consistent Broadcast

Standard Consistent Broadcast. Consistent broadcast (CB) provides a way for a designated sender P_s to broadcast a payload to all parties and guarantees that any two correct parties that deliver the payload, deliver the same content. Formally, every broadcast protocol instance is identified by a tag ID. At the sender P_s, the CB protocol instance is invoked by an input action of the form $(ID, \texttt{in}, \texttt{c-broadcast}, m)$, with $m \in \{0,1\}^*$. When that occurs, we say P_s *c-broadcasts* m *with tag* ID. Only P_s executes this action; all other parties initialize the protocol instance with tag ID in the role of receivers. When an output action of the form $(ID, \texttt{out}, \texttt{c-deliver}, m)$ occurs at a party P_i, we say that P_i *c-delivers* m *with tag* ID.

Definition 1 (*Consistent Broadcast*)**.** A protocol for CB satisfies the following conditions except with negligible probability.
Termination: If a correct party *c-broadcasts* m with tag ID, then all correct parties eventually *c-deliver* m with tag ID.
Agreement: If two correct parties P_i and P_j *c-deliver* m and m' with tag ID, respectively, then $m = m'$.
Integrity: Every correct party *c-delivers* at most one payload m with tag ID. Moreover, if the sender P_s is correct, then m was previously *c-broadcast* by P_s with tag ID.

The standard protocol for implementing ordinary CB is Reiter's *echo broadcast* [13]; it involves $\mathcal{O}(n)$ messages, has a latency of three communication steps, and relies on a digital signature scheme. The sender starts the protocol by sending the payload m to all parties; then it waits for a quorum of $\lceil \frac{n+t+1}{2} \rceil$ parties to issue a signature on the payload and to "echo" the payload and signature to the sender. When the sender has collected and verified sufficient signatures, it composes a `final` protocol message containing the signatures and sends it to all parties. Upon receiving a valid `final` message with $\lceil \frac{n+t+1}{2} \rceil$ signatures on the same payload, a party can *c-deliver* the payload.

Signature-free Consistent Broadcast We obtain a simple signature-free version of CB from the echo-broadcast implementation simply by replacing signed echoes by echo messages carrying authenticators. Thus, the `final` protocol message contains authenticators from $\lceil \frac{n+t+1}{2} \rceil$ parties. When a party P_i receives the `final` protocol message, the party checks the i^{th} entry in each authenticator. If the i^{th} entries in all $\lceil \frac{n+t+1}{2} \rceil$ authenticators have been computed on the same payload, then P_i can *c-deliver* the payload. One can easily see that if two correct parties P_i and P_j *c-deliver* m and m' for the same *c-broadcast* instance, then $m = m'$.

Although the above implementation satisfies agreement and integrity, any corrupt party P_i may cause the termination property of CB (stated above) to be violated even if the designated sender P_s behaves correctly. In the echo-broadcast implementation, when a party P_j receives a `final` message from a correct P_s, P_j can immediately *c-deliver* because the $\lceil \frac{n+t+1}{2} \rceil$ signed echoes will be valid. However, in the signature-free implementation, when the designated sender receives an echo message from P_i, it can verify only the validity of its corresponding MAC in the authenticator before including the echo as part of its `final` message. The MACs for other parties contained in the authenticator may be invalid or computed on a different payload. We call such an echo message *partially corrupt*. Partially corrupt messages are examples of *malicious failures* (as opposed to muteness failures [8]); malicious failures are exhibited only by parties that are actually corrupted by the adversary. If partially corrupt echo messages are part of the `final` message, then one or more correct parties may not be able to *c-deliver*.

In fact, signature-free CB satisfies the following weaker termination property:

Weak Termination: If no party is corrupted and some correct party *c-broadcasts* m with tag ID, then all correct parties eventually *c-deliver* m with tag ID.

Dual-Mode Consistent Broadcast The CB protocol used in the parsimonious mode of the AMP protocol is a dual-mode protocol that operates in the signature-free mode until some party complains about receiving a partially corrupt `final` message. Upon receiving a partially corrupt `final` message, the CB implementation at the recipient party sends a `pc-complaint` message to the designated sender (i.e., the AMP protocol's leader). Even if the complaining party is lying about receiving a partially corrupt `final` message, it is clear at that point that there is at least one malicious failure because only corrupt parties lie. Upon receiving the `pc-complaint` message, the CB implementation at the leader switches to the signature-based echo-broadcast mode and re-sends the payload to all parties, indicating that signed echoes are required. All subsequent CB protocol instances will directly operate in the echo-broadcast mode.

2.2.2 Multi-Valued Byzantine Agreement

We use a protocol for multi-valued Byzantine agreement (MVBA) as defined by Cachin et al. [3], which allows agreement values from an arbitrary domain instead of being restricted to binary values. Unlike previous MVBA protocols, their protocol does not allow the decision to fall back on a *default* value if not all correct parties propose the same value, but uses a protocol-external mechanism instead. This so-called *external validity condition* is specified by a global, polynomial-time computable predicate Q_{ID} that is known to all parties and is typically determined by an external application or higher-level protocol. Each party proposes a value that contains certain validation information. The protocol ensures that the decision value has been proposed and that it satisfies Q_{ID}.

When a party P_i starts an MVBA protocol instance with tag ID and an input value $v \in \{0,1\}^*$ that satisfies predicate Q_{ID}, we say that *P_i proposes v for MVBA with tag ID and predicate Q_{ID}*. Correct parties only propose values that satisfy Q_{ID}. When P_i terminates the MVBA protocol instance with tag ID and outputs a value v, we say that it *decides v for ID*.

Definition 2 (*Multi-valued Byzantine agreement (MVBA)*)**.** A protocol for *multi-valued Byzantine agreement* with predicate Q_{ID} satisfies the following conditions except with negligible probability.
External Validity: Any correct party that decides for ID decides v such that $Q_{ID}(v)$ holds.
Agreement: If some correct party decides v for ID, then any correct party that decides for ID decides v.
Integrity: If all parties are correct and if some party decides v for ID, then some party proposed v for ID.
Termination: All correct parties eventually decide for ID.

2.3 Definition of Atomic Broadcast

Atomic broadcast provides a "broadcast channel" abstraction [7], such that all correct parties deliver the same sequence of messages broadcast on the channel. A party P_i *atomically broadcasts* (or *a-broadcasts*) a payload $m \in \{0,1\}^*$ with tag ID when an input action of the form $(ID, \texttt{in}, \texttt{a-broadcast}, m)$ is delivered to P_i. Broadcasts are parameterized by the tag ID to identify their corresponding broadcast channel. A party *atomically delivers* (or *a-delivers*) a payload m with tag ID by generating an output action of the form $(ID, \texttt{out}, \texttt{a-deliver}, m)$.

Definition 3 (*Atomic broadcast*)**.** A protocol for atomic broadcast satisfies the following properties except with negligible probability.
Validity: If $t+1$ correct parties *a-broadcast* some payload m with tag ID, then some correct party eventually *a-delivers* m with tag ID.
Agreement: If some correct party has *a-delivered* m with tag ID, then all correct parties eventually *a-deliver* m with tag ID.
Total Order: If two correct parties both *a-delivered* distinct payloads m_1 and m_2 with tag ID, then they have *a-delivered* them in the same order.
Integrity: For any payload m, a correct party P_j *a-delivers* m with tag ID at most once. Moreover, if all parties are correct, then m was previously *a-broadcast* by some party with tag ID.

The above properties are similar to the definitions of PABC [12], Cachin et al. [3], and Kursawe and Shoup [9]. We do not formalize their *fairness* condition, although our protocols satisfy an equivalent notion. We measure the *latency* of atomic broadcast as the minimum number of communication steps between the atomic broadcast and the atomic delivery of a payload at all correct parties.

3 The Authenticator-Based Message-Parsimonious Protocol (AMP)

We now describe the Authenticator-Based Message-Parsimonious (AMP) atomic broadcast protocol. The protocol has a structure similar to that of the PABC protocol [12]. Like the PABC protocol, the AMP protocol is optimally resilient (i.e., $n \geq 3t+1$) and has an amortized message complexity of $\mathcal{O}(n)$. Unlike the PABC protocol, the AMP protocol uses dual-mode CB in the parsimonious mode. Furthermore, the protocol has a latency of five communication steps, which is two steps fewer than that of the PABC protocol.

3.1 Protocol Overview

During normal operation, a leader determines the delivery order of payloads and conveys the order to the other parties using dual-mode CB. The parties atomically deliver the payloads in the order chosen by the leader. If a party observes the leader to be slow or exhibiting faulty behavior, the party switches to recovery mode. When a sufficient number of correct parties have switched to recovery mode, the protocol ensures that all correct parties start the recovery mode. The goal of the recovery mode is to start the next epoch in a consistent state and with a new leader. The difficulty lies in determining which payloads have been atomically delivered in the parsimonious mode of the past epoch.

Consistent broadcast ensures agreement among the correct parties that deliver the payload. However, a corrupted leader may cause the fate of some payloads to be *undefined* in the sense that there might be only one correct party that has *c-delivered* a payload, but no way for other correct parties to learn about this fact. The AMP protocol solves this problem by (1) initializing a new CB instance only after a payload has been *c-delivered* by the current CB instance and (2) delaying the atomic delivery of a *c-delivered* payload until *one* more payload has been *c-delivered*. In contrast, the PABC protocol delays atomic delivery until two additional *c-deliveries* have occurred.

Upon entering recovery mode, a party requests and collects *signed* proofs from "enough" correct parties to be able to show (to any correct party) that its last and second-to-last *c-deliveries* were indeed valid. Subsequently, a first round of multi-valued Byzantine agreement (MVBA) ensures that all correct parties agree on a synchronization point. Then, the protocol ensures that the correct parties deliver all payloads up to that synchronization point. To implement this step, every party must store all payloads that were delivered in the parsimonious mode. A second MVBA instance is used to atomically deliver at least one payload, ensuring progress in every epoch.

3.2 Parsimonious and Recovery Modes

We now describe the parsimonious and the recovery modes in detail. The line numbers refer to the detailed protocol description in Algorithms 1 and 2.

3.2.1 Parsimonious Mode: Algorithm 1

When a party P_i *a-broadcasts* a payload m, it appends m to a local queue $\mathcal{I}$ and forwards m using an `initiate` message to the leader P_l of the current epoch e, where $l = e \bmod n$ (lines 1.19–1.21). When this happens, we say that P_i *initiates* the payload. Upon receiving the `initiate` message, P_l appends m to a local FIFO buffer $\mathcal{B}$ (line 1.22).

The leader binds sequence numbers to the payloads that it receives in `initiate` messages, and conveys the bindings to the other parties through dual-mode CB. For this purpose, all parties start with an instance of dual-mode CB (line 1.12). The leader acts as the designated sender of the CB instance, and the tag contains the epoch e and a sequence number s. Here, s starts from 0 in every epoch. The leader *c-broadcasts* the next available initiated payload (lines 1.15–1.17 and lines 1.42–1.44), and every party waits to *c-deliver* some payload m. When m is *c-delivered*, P_i stores it in *log*, but does not yet *a-deliver* it (line 1.29). At this point in time, we say that P_i has *committed* sequence number s to payload m in epoch e. Then, P_i *a-delivers* the payload to which it has committed the sequence number $s - 1$ (if available, lines 1.30–1.32). It increments s (line 1.33) and starts the next CB instance (line 1.35).

The delay of an extra *c-delivery* between the *c-delivery* and the *a-delivery* of a payload m is the key to avoiding undefined payloads that may be caused by a faulty leader. However, as it is possible that no further payloads with sequence numbers higher than s may be *c-delivered*, the leader *c-broadcasts* a `dummy` payload to trigger the *a-delivery* of m. The leader *c-broadcasts* such a `dummy` message whenever a corresponding timer T expires (line 1.25); T is activated whenever the current sequence number is committed to a payload (line 1.28), and T is disabled when the leader *c-broadcasts* a payload (line 1.41).

To guarantee that all correct parties eventually catch up with each other in terms of *a-delivered* payloads, the parsimonious mode ensures that once an *a-delivery* has happened at a correct party, all correct parties eventually transition to the recovery mode. That is guaranteed by the following scheme:

> When a correct party P_i commits a sequence number x, it (re)activates a timer T (line 1.28); if T expires before the next *c-delivery*, then P_i sends a `request` message carrying a uniquely identifiable `dummy` payload m to all parties (line 1.27), requesting them to "adopt" m and atomically broadcast m themselves. Thereby, all correct parties send an `initiate` message to the leader P_l, start the failure-detection mechanism (lines 1.19–1.21), and expect the leader to make progress in terms of further *c-deliveries*. If $t + 1$ correct parties *a-deliver* m before the failure detection has been triggered, then $t + 1$ correct parties have necessarily committed a sequence number $x + 1$.

After at most X repetitions of the above scheme, $t + 1$ correct parties switch to the recovery mode. Otherwise, the failure detection invokes the $\mathit{transition}()$ function at $t + 1$ correct parties. In either case, all correct parties switch to recovery mode, as explained below.

3.2.2 Transition to Recovery Mode: Algorithm 1

The protocol transitions from the parsimonious to the recovery mode when: (1) X payloads have been *c-delivered* (line 1.38) or (2) the leader is not functioning properly. The first condition is needed to to ensure that eventually all correct parties agree on the set of payloads *a-delivered* in the parsimonious mode. The second condition is needed to prevent a corrupted leader from violating liveness.

Any party that *a-broadcasts* a payload m calls $\mathit{update}_{\mathcal{FD}}(\texttt{initiate}, m)$ (line 1.21); this starts a timer unless it has already been activated. When a payload is *a-delivered* during the parsimonious mode, the call to $\mathit{update}_{\mathcal{FD}}(\texttt{deliver}, m)$ (line 1.31) checks whether $\mathcal{I}$ contains further undelivered payloads, and if so, restarts the timer. Else, the timer is disabled. When the timer expires, $\mathcal{FD}$ invokes $\mathit{transition}()$.

When the $\mathit{transition}()$ function is invoked at a party P_j, the party sends a `transition` message to all parties (lines 1.53) and does not initialize any further CB instances (line 1.34). When a correct party receives $2t + 1$ `transition` messages, it enters recovery mode (lines 1.50–1.52). There is a transition "amplification" mechanism [9, 12] by which a correct party that has received $t + 1$ `transition` messages and has not yet sent out a `transition` message itself joins the other parties by sending its own `transition` message.

Transition amplification ensures that when some correct party enters recovery mode, all other correct parties eventually enter it as well. It is easy to see why. A correct party

Algorithm 1: AMP at P_i (Parsimonious Mode)

```
intialization:
1.1   $e \leftarrow 0$                                          {current epoch}
1.2   $\mathcal{I} \leftarrow []$        {queue of a-broadcast but not a-delivered payloads}
1.3   $\mathcal{D} \leftarrow \emptyset$                         {set of a-delivered payloads}
1.4   $init\_view()$
function $init\_view()$:
1.5   $l \leftarrow (e \bmod n) + 1$                          {$P_l$ is leader of epoch $e$}
1.6   $log \leftarrow []$        {size $X$ array of payloads c-delivered in epoch $e$}
1.7   $s \leftarrow 0$                {sequence number of next payload in epoch $e$}
1.8   $want\_transition \leftarrow \texttt{false}$
      {indicates whether a transition to recovery phase is desired}
1.9   $c \leftarrow 0$ {number of $\texttt{transition}$ messages received for $P_l$ in epoch $e$}
1.10  $\mathcal{B} \leftarrow []$                          {initiated payloads buffered at $P_l$}
1.11  $\mathcal{S} \leftarrow \mathcal{D}$                              {set of c-broadcast payloads}
1.12  initialize c-broadcast instance with tag $ID|\texttt{bind}.e.s$
1.13  if $i = l$ then                                     {c-broadcast the first payload}
1.14      wait until $\mathcal{B} \neq \emptyset$
1.15      $m \leftarrow head(\mathcal{B})$
1.16      $\mathcal{S} \leftarrow \mathcal{S} \cup \{m\}$
1.17      c-broadcast $m$ with tag $ID|\texttt{bind}.e.s$
upon receiving $(ID, \texttt{request}, m)$
1.18  trigger $(ID, \texttt{in}, \texttt{a-broadcast}, m)$

upon $(ID, \texttt{in}, \texttt{a-broadcast}, m)$
1.19  send $(ID, \texttt{initiate}, e, m)$ to $P_l$
1.20  $append(m, \mathcal{I})$
1.21  $update_{\mathcal{FD}}(\texttt{initiate}, m)$
upon receiving $(ID, \texttt{initiate}, e, m) : m \notin S$
1.22  $append(m, \mathcal{B})$
upon $timeout(T)$
1.23  $m \leftarrow \texttt{dummy}$
1.24  if $i = l$ then
1.25      $append(m, \mathcal{B})$
1.26  else
1.27      send $(ID, \texttt{request}, m)$ to all parties
upon c-delivery of $m$ with tag $ID|\texttt{bind}.e.s$ : $m \notin log$
1.28  $restart(T)$
1.29  $log[s] \leftarrow m$
1.30  if $s \geq 1$ then
1.31      $update_{\mathcal{FD}}(\texttt{deliver}, log[s-1])$
1.32      $deliver(log[s-1])$
1.33  $s \leftarrow s + 1$
1.34  if $\neg want\_transition$ then               {leader $P_l$ is not suspected}
1.35      initialize c-broadcast instance with tag $ID|\texttt{bind}.e.s$
1.36  if $s = X$ then
1.37      $transition()$
1.38      $recovery()$
1.39  if $i = l$ then                                              {am the leader}
1.40      wait until $\mathcal{B} \neq \emptyset$
1.41      $stop(T)$
1.42      $m \leftarrow head(\mathcal{B})$
1.43      $\mathcal{S} \leftarrow \mathcal{S} \cup \{m\}$
1.44      c-broadcast the message $m$ with tag $ID|\texttt{bind}.e.s$
function $deliver(m)$
1.45  if $m \notin \{\texttt{dummy}\} \cup D$ then
1.46      $remove(m, \mathcal{I})$
1.47      $\mathcal{D} \leftarrow \mathcal{D} \cup \{m\}$
1.48      output $(ID, \texttt{out}, \texttt{a-deliver}, m)$
upon receiving $(ID, \texttt{transition}, e)$ from $P_j$ for the first time
1.49  $c \leftarrow c + 1$
1.50  if $c = t + 1$ then
          $transition()$
1.51  if $c = 2t + 1$ then
1.52      $recovery()$
function $transition()$                            {called by the $\mathcal{FD}$ module}
1.53  send $(ID, \texttt{transition}, e)$ to all parties
1.54  $complained \leftarrow \texttt{true}$
```

Algorithm 2: AMP at P_i (Recovery Mode)

```
function $recovery()$:
 {/***** Part 1: build recovery sets for $s-2$ and $s-1$ *****/}
2.1   send $(ID, \texttt{proof\_request}, e, s-1)$ to all parties
2.2   $R_{s-2} \leftarrow \emptyset$; $R_{s-1} \leftarrow \emptyset$
2.3   while $\neg[Q_{ID|\texttt{equality}.e}(R_{s-2}, s-2)$ and
              $Q_{ID|\texttt{consistency}.e}(R_{s-1}, s-1)]$ do
2.4       wait for a message $(ID, \texttt{proof}, e, M_j, \Sigma_j)$ from some $P_j$
2.5       $R_{s-2} \leftarrow R_{s-2} \cup \{(M_j[0], \Sigma_j[0])\}$
2.6       $R_{s-1} \leftarrow R_{s-1} \cup \{(M_j[1], \Sigma_j[1])\}$

2.7   $\mathcal{R} \leftarrow \langle R^{'}_{s-2}, R^{'}_{s-1} \rangle$, where $R^{'}_{s-2}$ and $R^{'}_{s-1}$ are the compacted
      versions of $R_{s-2}$ and $R_{s-1}$, respectively
 {/***** Part 2: agree on watermark *****/}
2.8   send $(ID, \texttt{candidate}, e, s-1, \mathcal{R}, \sigma)$ to all parties, where
          $\sigma$ is a valid signature on $(ID, \texttt{candidate}, e, s-1)$
2.9   wait for valid messages $(ID, \texttt{candidate}, e, s_j, \mathcal{R}_j, \sigma_j)$
          from $\lceil \frac{n+t+1}{2} \rceil$ distinct parties $P_j$
2.10  $W \leftarrow \langle (s_1, \mathcal{R}_1, \sigma_1), \ldots, (s_n, \mathcal{R}_n, \sigma_n) \rangle$
2.11  propose $W$ for MVBA with tag $ID|\texttt{watermark}.e$ and predicate
      $Q_{ID|\texttt{watermark}.e}$
2.12  wait for MVBA with tag $ID|\texttt{watermark}.e$ to decide
      $\bar{W} = \langle (\bar{s}_1, \bar{\mathcal{R}}_1, \bar{\sigma}_1) \ldots (\bar{s}_n, \bar{\mathcal{R}}_n, \bar{\sigma}_n) \rangle$
2.13  $w \leftarrow \max\{\bar{s}_1, \ldots, \bar{s}_n\}$
 {/***** Part 3: synchronize up to watermark $w$ *****/}
2.14  pick $(w, \bar{\mathcal{R}}, *) \in \bar{W}$ deterministically
2.15  if $s \geq 1$ then
2.16      if $s \leq w$ then
2.17          $deliver(log[s-1])$
2.18      if $s = w + 1$ then
2.19          $deliver(\bar{m})$ from $\bar{\mathcal{R}}[1]$
2.20  if $s > w - 2 \geq 0$ then
2.21      $\mathcal{M} \leftarrow \{(log[k], k)\}$ for $k = 0, \ldots, w-2$
2.22      send $(ID, \texttt{complete}, e, \mathcal{M})$ to all
2.23  while $s \leq w - 2$ do
2.24      wait for messages $(ID, \texttt{complete}, e, \bar{\mathcal{M}}_j)$ from $t+1$
              distinct $P_j$ such that $\forall \bar{\mathcal{M}}_j : (\bar{m}, s) \in \bar{\mathcal{M}}_j$ for some $\bar{m}$
2.25      $deliver(\bar{m})$
2.26      $s \leftarrow s + 1$
2.27  while $s \leq w$ do
2.28      $deliver(\bar{m})$ from $\bar{\mathcal{R}}[s - w + 1]$
2.29      $s \leftarrow s + 1$
 {/***** Part 4: deliver some messages and start next epoch *****/}
2.30  send $(ID, \texttt{queue}, e, \mathcal{I}, \sigma)$ to all parties, where
          $\sigma$ is a valid signature on $(ID, \texttt{queue}, e, \mathcal{I})$
2.31  wait for messages $(ID, \texttt{queue}, e, \mathcal{I}_j, \sigma_j)$ from
          $n - t$ distinct $P_j$ such that $\mathcal{I}_j \cap \mathcal{D} = \emptyset$
2.32  $Q \leftarrow \langle (\mathcal{I}_1, \sigma_1), \ldots, (\mathcal{I}_n, \sigma_n) \rangle$
2.33  propose $Q$ for MVBA with predicate $Q_{ID|\texttt{deliver}.e}$
2.34  wait for MVBA to decide $\bar{Q} = \langle (\bar{\mathcal{I}}_1, \bar{\sigma}_1), \ldots, (\bar{\mathcal{I}}_n, \bar{\sigma}_n) \rangle$
2.35  for $m \in \bigcup_{j=1}^{n} \bar{\mathcal{I}}_j \setminus \mathcal{D}$, in some deterministic order do
2.36      $deliver(m)$
2.37  $init\_view()$
2.38  for $m \in \mathcal{I}$ do send $(ID, \texttt{initiate}, e, m)$ to $P_l$
 {/***** end of recovery mode *****/}

upon receiving $(ID, \texttt{proof\_request}, e, s_j - 1)$ from $P_j$
      for the first time for any value of $s_j$
2.39  $M \leftarrow \langle \texttt{committed}(s_j - 2), \texttt{committed}(s_j - 1) \rangle$, where
      $\texttt{committed}(x) = \begin{cases} log[x] & \text{if } 0 \leq x \leq s-1 \\ \perp & \text{otherwise} \end{cases}$
2.40  $\Sigma[0] \leftarrow \{(ID, \texttt{proof}, e, s_j - 2, M[0])\}_i$
2.41  $\Sigma[1] \leftarrow \{(ID, \texttt{proof}, e, s_j - 1, M[1])\}_i$
2.42  send $(ID, \texttt{proof}, e, M, \Sigma)$ to $P_j$
```

enters recovery mode only after receiving $\texttt{transition}$ messages from at least $t+1$ correct parties. These messages will be eventually received at every correct party, which would result in a transition amplification by every correct party that had not previously sent a $\texttt{transition}$ message.

3.2.3 Recovery Mode: Algorithm 2

The recovery mode can be divided into four distinct parts:

1. Every correct party P_i constructs *recovery sets* $R_{s_i - 2}$

and R_{s_i-1} for the second-highest and the highest sequence number *c-delivered*, respectively.
2. All correct parties agree on a common synchronization point, a sequence number w termed *watermark*.
3. All correct parties synchronize up to w, i.e. they atomically deliver all payloads to which sequence numbers $\leq w$ have been committed.
4. All correct parties *a-deliver* every payload that was *a-broadcast* by $t+1$ correct parties and switch to the parsimonious mode of the next epoch.

To determine the synchronization point, the parties first have to exchange information about how much progress they have made in the parsimonious mode. As the *c-deliveries* in the parsimonious mode could have been from signature-free CB instances, P_i may not be able to directly prove the progress it has made. For this purpose, P_i gathers *recovery sets* R_{s_i-2} and R_{s_i-1} for the second-highest and highest committed sequence numbers, respectively; these sets can be considered as a signed "show of support" from a sufficient quorum. To construct the recovery sets, P_i sends a `proof_request` message with the sequence number of the last *c-delivery* (i.e., $s_i - 1$) to all parties (line 2.1). Upon receiving a `proof_request` message for sequence number $s_i - 1$ from P_i, every correct party P_j responds with a signed `proof` message containing the payloads to which sequence numbers $s_i - 2$ and $s_i - 1$ were committed (lines 2.39–2.42). Subsequently, P_i constructs recovery sets R_{s_i-2} and R_{s_i-1} from the `proof` messages received (lines 2.3–2.7). The elements of a recovery set R_x (where $x = s_i - 2$ or $s_i - 1$) are tuples of the form (m_j, σ_j), where m_j denotes the payload to which party P_j has committed the sequence number x, and σ_j is a valid signature by P_j on $(ID, \texttt{proof}, e, x, m_j)$. If there is no payload to which sequence number x has been committed by P_j, then m_j has the value $\perp$.

Construction of the recovery sets is complete at a party P_i when the *equality predicate* holds for R_{s_i-2} and the *consistency predicate* holds for R_{s_i-1}.

The equality predicate, $Q_{ID|\texttt{equality}.e}$, holds iff the set R_{s_i-2} contains entries (m, σ_j) from $t+1$ distinct parties P_j that each substantiate P_i's claim that sequence number $s_i - 2$ was committed to a payload m with a valid signature $\sigma_j = \{(ID, \texttt{proof}, e, s_i - 2, m)\}_j$. Formally,

$Q_{ID|\texttt{equality}.e}(R, \tilde{s}) \equiv \exists R' \subseteq R$, such that $R' = \{(m, \sigma_j)\}$ **and** $|R'| = t + 1$, where σ_j is a valid signature by P_j on $(ID, \texttt{proof}, e, \tilde{s}, m)$ **and** $(\tilde{s} \geq 0 \Rightarrow m \neq \perp)$.

We use R'_{s_i-2} to denote the particular subset of R_{s_i-2} that satisfies the equality predicate. R'_{s_i-2} is called the *compacted version* of R_{s_i-2}.

Every correct party P_i is able to construct R_{s_i-2} because there exists a set C of $t+1$ correct parties that have committed sequence number $s_i - 2$. By the Agreement property of CB, every party in C has committed sequence number $s_i - 2$ to m_i. Thus, the following property holds:

P1: At any correct party, if $Q_{ID|\texttt{equality}.e}(R_{s-2}, s-2)$ holds for $s \geq 2$, and if $R'_{s-2} = \{(m, \sigma_j)\}$ is the compacted version of R_{s-2}, then a correct party has committed sequence number $s - 2$ to m.

It is easy to see that the property P1 holds: By construction, all payloads in the compacted recovery set R'_{s_i-2} are the same (m), and the set contains an entry from $t+1$ distinct parties (at least one of which is correct) claiming to have committed $s_i - 2$ to that payload m.

The consistency predicate, $Q_{ID|\texttt{consistency}.e}$, holds iff the set R_{s_i-1} contains a subset R'_{s_i-1} of $\lceil \frac{n+t+1}{2} \rceil$ entries from distinct parties P_j. Each entry is of the form (m_j, σ_j), where each σ_j is a valid signature by P_j on $\{(ID, \texttt{proof}, e, s_i - 1, m_j)\}_j$. Here, m_j can be $\perp$, but all non-$\perp$ payloads in the subset R'_{s_i-1} must be identical. Formally,

$Q_{ID|\texttt{consistency}.e}(R, \tilde{s}) \equiv \exists R' \subseteq R$, such that $R' = \{(m_j, \sigma_j)\}$ **and** $|R'| = \lceil \frac{n+t+1}{2} \rceil$, where σ_j is a valid signature by P_j on $(ID, \texttt{proof}, e, \tilde{s}, m_j)$ **and** $(\tilde{s} \geq 0 \Rightarrow \exists m_j \neq \perp)$ **and** $\forall (m_j \neq \perp, \sigma_j), (m_k \neq \perp, \sigma_k) \in R'$: $m_j = m_k$.

We use R'_{s_i-1} to denote the particular subset of R_{s_i-1} that satisfies the consistency predicate. R'_{s_i-1} is called the *compacted version* of R_{s_i-1}.

Every correct party P_i can construct R_{s_i-1} because there exists a set C of at least $n - t \geq \lceil \frac{n+t+1}{2} \rceil$ correct parties such that every $P_j \in C$ either has committed $s_i - 1$ to $m_j = m_i$ or has not committed $s_i - 1$ yet, in which case $m_j = \perp$. The recovery set R_{s_i-1} has the following property:

P2: Suppose that $Q_{ID|\texttt{consistency}.e}(R_{s_i-1}, s_i - 1)$ holds at P_i and a correct party has *a-delivered* a payload m with sequence number $s_i - 1$. If m_j is any non-$\perp$ payload appearing in the compact recovery set R'_{s_i-1}, then $m_j = m$.

To see why P2 holds, consider a correct party P_k that has *a-delivered* m with sequence number $s_i - 1$. P_k must have committed sequence number s_i. Hence, a set C of $\lceil \frac{n+t+1}{2} \rceil - t = \lceil \frac{n-t+1}{2} \rceil$ *correct* parties have participated in the CB instance s_i, which means that every party $P_j \in C$ previously committed sequence number $s_i - 1$. By the Agreement property of CB, every party $P_j \in C$ must have committed $s_i - 1$ to m. By construction, $|R'_{s_i-1}| = \lceil \frac{n+t+1}{2} \rceil$, which means that $|C| + |R'_{s_i-1}| > n$; hence, there exists some $P_j \in C$ such that $(m_j, \sigma_j) \in R'_{s_i-1}$ and thus R'_{s_i-1} contains at least one entry for m. The fact that $Q_{ID|\texttt{consistency}.e}(R_{s_i-1}, s_i - 1)$ holds at P_i implies that all non-$\perp$ payloads that are part of R'_{s_i-1} are necessarily for the same payload, and thus $m = m_j$.

In the second part of the recovery mode (lines 2.8–2.13), a watermark sequence number w is determined such that (a) no correct party has committed sequence number $w + 1$ or higher and (b) some correct party has committed sequence number $w - 1$. While (a) ensures that w is high enough such that no correct party has *a-delivered* a payload with a higher sequence number, (b) guarantees that w is small enough such that every correct party is able to deliver every payload up to w. To determine w, every correct party P_i sends a watermark `candidate` message for sequence number $s_i - 1$, the *highest* sequence number P_i has committed in the parsimonious mode. The `candidate` message contains the compact recovery sets R'_{s_i-2} and R'_{s_i-1} for sequence numbers $s_i - 2$ and $s_i - 1$, respectively. Subsequently P_i collects $\lceil \frac{n+t+1}{2} \rceil$ `candidate` messages and forms a *watermark vector* W with the following property:

P3: No correct party has *a-delivered* a payload with a sequence number larger than w, where $w = \max(W)$ denotes the highest sequence number for which W has an entry.

To see why P3 holds, consider a correct party P_i that has *a-delivered* a payload to which sequence number $w + 1$ was committed. Then, $\lceil \frac{n-t+1}{2} \rceil$ correct parties have committed $w + 1$. Hence, every watermark vector W built from $\lceil \frac{n+t+1}{2} \rceil$ `candidate` messages must necessarily contain one entry for sequence number $w + 1$ or higher. That would contradict the assumption that w is the highest sequence number for which W has an entry.

After constructing the watermark vector W_i, every correct party P_i proposes $W_i = \langle(s_1, \mathcal{R}_1, \sigma_1) \ldots (s_n, \mathcal{R}_n, \sigma_n)\rangle$ for MVBA with tag $ID|\texttt{watermark}.e$ and predicate $Q_{ID|\texttt{watermark}.e}$ (line 2.11), where

$Q_{ID|\texttt{watermark}.e}(W) \equiv W$ has entries $(s_j, \mathcal{R}_j, \sigma_j)$ from $\lceil \frac{n+t+1}{2} \rceil$ distinct P_j **and** $\forall (s_j, \mathcal{R}_j, \sigma_j) \in W : \sigma_j$ is a valid signature by P_j on $(ID, \texttt{candidate}, e, s_j)$ **and** $|\mathcal{R}_j[0]| = t + 1$ **and** $Q_{ID|\texttt{equality}.e}(\mathcal{R}_j[0], s_j - 1)$ **and** $|\mathcal{R}_j[1]| = \lceil \frac{n+t+1}{2} \rceil$ **and** $Q_{ID|\texttt{consistency}.e}(\mathcal{R}_j[1], s_j)$.

By the validity property of MVBA, the decision $\bar{W}$ will be one of the proposed watermark vectors and satisfy the predicate $Q_{ID|\texttt{watermark}.e}$. Consequently, by property P3, no correct party has *a-delivered* a payload with a sequence number larger than $w = \max(\bar{W})$. Based on this observation, we argue that all correct parties *a-deliver* the same sequence of payloads in epoch e.

We now focus our correctness arguments on the consistency property only and refer the reader to Section 3.3 for complete proofs that cover both safety and liveness. *Consistency* refers to the property that if two correct parties *a-deliver* payloads m and m' with sequence number $\tilde{s}$ in the parsimonious mode and in the recovery mode of epoch e respectively, then $m = m'$. We distinguish three cases: **(a)** $\tilde{s} \leq w - 2$, **(b)** $\tilde{s} = w - 1$, and **(c)** $\tilde{s} = w$, and argue that for all three cases, the third phase of the recovery mode (lines 2.20–2.29) ensures consistency. Note that by P3, it is safe to ignore the case where $\tilde{s} > w$.

Let $(w, \mathcal{R}, *) \in \bar{W}$. In the first case **(a)**, property P1 implies that some correct party has committed sequence number $w - 1$. Consequently, for every sequence number $\tilde{s} \leq w - 2$, $t + 1$ correct parties have committed $\tilde{s}$. Consider a correct party P_i that has *a-delivered* m with sequence number $\tilde{s}$ in the parsimonious mode and a correct party P_j that *a-delivers* payload m' from `complete` messages for m' received from $t + 1$ distinct parties (lines 2.23–2.26). Among the $t + 1$ `complete` message received, at least one message was from a correct party P_k that has committed $\tilde{s}$ to m' in the parsimonious mode. On the other hand, P_i has committed $\tilde{s}$ to m in the parsimonious mode. By the Agreement property of CB, it follows that $m = m'$. Consistency for case **(b)**, where $\tilde{s} = w - 1$, results from P1 as follows: for every $(m_j, \sigma_j) \in \mathcal{R}[0]$, some correct party has committed sequence number $\tilde{s}$ to m_j. Let m' be the payload a correct party *a-delivers* from R_{s_i-2} (line 2.28). By the Agreement property of CB, $m' = m_j = m$. Consistency for case **(c)**, where $\tilde{s} = w$, follows directly from property P2, analogous to property P1 for case **(b)**.

Finally, in the fourth part of the recovery mode (lines 2.30–2.38), the correct parties deliver every payload atomically broadcast by $t + 1$ correct parties to ensure the validity property of atomic broadcast. To agree on the set of payloads to atomically deliver, all correct parties execute an instance of MVBA on a vector $\langle(\mathcal{I}_1, \sigma_1), \ldots, (\mathcal{I}_n, \sigma_n)\rangle$ with predicate $Q_{ID|\texttt{deliver}.e}$, where

$Q_{ID|\texttt{deliver}.e}(\langle(\mathcal{I}_1, \sigma_1), \ldots, (\mathcal{I}_n, \sigma_n)\rangle) \equiv$ for at least $n - t$ distinct j, $\big(\mathcal{I}_j \cap \mathcal{D} = \emptyset$ **and** σ_j is a valid signature by P_j on $(ID, \texttt{queue}, e, \mathcal{I}_j)\big)$.

Subsequently, the correct parties switch to the parsimonious mode of the next epoch.

3.3 Correctness of the AMP Protocol

Organization We first give a brief overview of the lemmas established and how they contribute to showing the correctness of AMP. Lemmas 1 and 2 correspond to the properties P1 and P2 of the compacted recovery sets, R_{s-2} and R_{s-1}. Lemma 3 shows that the number of payloads possibly *a-delivered* during the parsimonious mode of an epoch is limited to $w + 1$ (i.e., sequence numbers 0 to w). Lemmas 1–3 are used as building blocks in Lemma 6, which proves that before starting the next epoch, all correct parties first catch up with every *a-delivery* done in the preceding epoch. Lemma 4 states that once an *a-delivery* has happened at a correct party, eventually all correct parties enter the recovery mode. This feature guarantees that (unlike in the PABC protocol) all correct parties catch up with each other even if no further "real" payloads are *a-broadcast*. Lemma 5 shows that no correct party blocks (stays forever) in recovery mode. Lemmas 4–6 are used to show that all correct parties agree on the *set* of *a-deliveries* in each epoch (Lemma 7). Lemma 8 shows that all correct parties agree on the *order* in which they *a-deliver* payloads. Finally, Total Order and Agreement (Lemma 9) follow from the fact that all correct parties agree on the *sequence* of payloads *a-delivered* in every epoch (Lemmas 7 and 8).

Lemma 1 (Property P1)**.** *Let $\bar{W}$ be the watermark vector of epoch e, $(\bar{s}, \bar{\mathcal{R}}, \bar{\sigma}) \in \bar{W}$, and $(\bar{m}_j, \bar{\sigma}_j) \in \bar{\mathcal{R}}[0]$. If $\bar{s} \geq 1$, then some correct party has committed sequence number $\bar{s} - 1$ to $\bar{m}_j$.*

Proof. By the Validity property of MVBA, the predicate $Q_{ID|\texttt{watermark}.e}(\bar{W})$ holds, and hence, the equality predicate $Q_{ID|\texttt{equality}.e}(\bar{\mathcal{R}}[0], \bar{s}-1)$ must also be *true*. By the definition of $Q_{ID|\texttt{equality}.e}$, $\bar{\mathcal{R}}[0]$ contains valid signatures $\bar{\sigma}_j$ on $(ID, \texttt{proof}, e, \bar{s}-1, m)$ from $t+1$ distinct parties P_j that claim to have committed sequence number $\bar{s}-1$ to payload m. This implies that some correct party has committed $\bar{s}-1$ to payload m. As $|\bar{\mathcal{R}}[0]| = t+1$, for all $(\bar{m}_j, \bar{\sigma}_j) \in \bar{\mathcal{R}}[0]$, $\bar{m}_j$ must be equal to m. □

Lemma 2 (Property P2)**.** *Let $(\bar{s}, \bar{\mathcal{R}}, \bar{\sigma}) \in \bar{W}$ and let $(\bar{m}_i \neq \perp, \bar{\sigma}_i) \in \bar{\mathcal{R}}[1]$. If $\lceil \frac{n-t+1}{2} \rceil$ correct parties have committed sequence number $\bar{s}$ to payload m, then $\bar{m}_i = m$.*

Proof. By the Validity property of MVBA, $Q_{ID|\texttt{watermark}.e}(\bar{W})$ holds. Hence, $Q_{ID|\texttt{consistency}.e}(\bar{\mathcal{R}}[1], \bar{s})$ is *true*. By the definition of $Q_{ID|\texttt{consistency}.e}$, $\bar{\mathcal{R}}[1]$ contains entries $(\bar{m}_i, \bar{\sigma}_i)$ from $\lceil \frac{n+t+1}{2} \rceil$ distinct parties P_i where $\bar{\sigma}_i$ is a valid signature on $(ID, \texttt{proof}, e, \bar{s}, \bar{m}_i)$ and $\forall (\bar{m}_i \neq \perp, \bar{\sigma}_i), (\bar{m}_j \neq \perp, \bar{\sigma}_j) \in \bar{\mathcal{R}}[1] : \bar{m}_i = \bar{m}_j$. By the hypothesis, a set C of $\lceil \frac{n-t+1}{2} \rceil$ correct parties have committed sequence number $\bar{s}$ to payload m. As any two sets of $\lceil \frac{n-t+1}{2} \rceil$ and $\lceil \frac{n+t+1}{2} \rceil$ elements intersect, there exists an entry $(\bar{m}_i, \bar{\sigma}_i) \in \bar{\mathcal{R}}[1]$ such that $P_i \in C$, which implies that $m = \bar{m}_i$. As $|\bar{\mathcal{R}}[1]| = \lceil \frac{n+t+1}{2} \rceil$, for all $(\bar{m}_i \neq \perp, \bar{\sigma}_j) \in \bar{\mathcal{R}}[1]$, $\bar{m}_i$ must be equal to m. □

Lemma 3 (Property P3)**.** *Let w be the watermark of epoch e. If $\lceil \frac{n-t+1}{2} \rceil$ correct parties have committed sequence number $\tilde{s}$ in epoch e, then $\tilde{s} \leq w$.*

Proof. Let C be any set of $\lceil \frac{n-t+1}{2} \rceil$ correct parties that have committed $\tilde{s}$ in epoch e. The proof is by contradiction. Suppose that $\tilde{s} \geq w+1$. Then, every party $P_j \in C$ sends a `candidate` message for sequence number $s_j \geq w+1$ (line 2.8). Let $\bar{W}$ be the watermark vector of epoch e. By $Q_{ID|\texttt{watermark}.e}(\bar{W})$, $\bar{W}$ contains an entry from $\lceil \frac{n+t+1}{2} \rceil$ distinct parties P_j. As $|C| + |W| > n$, there exists an entry $(\bar{s}_j, \bar{\mathcal{R}}_j, \bar{\sigma}_j) \in \bar{W}$ such that $P_j \in C$. As $\bar{s}_j \geq w+1$, the watermark of epoch e equals $\max(\bar{W}) \geq w+1 > w$, a contradiction. □

Lemma 4. *If any correct party P_i* a-delivers *a payload m in epoch e, then every correct party eventually enters the recovery mode of epoch e.*

Proof. We first show the Lemma using the two claims established below, and then prove the two claims.

Claim 1: If $t+1$ correct parties have committed sequence number $\tilde{s}$ in epoch e, then every correct party eventually enters the recovery mode of epoch e.

Claim 2: If any correct party has entered the recovery mode of epoch e, then all correct parties eventually enter the recovery mode of epoch e.

If P_i *a-delivers* m in the parsimonious mode, then $t+1$ correct parties have *c-delivered* m in epoch e. By *Claim 1*, every correct party eventually enters the recovery mode of epoch e. If P_i's *a-delivery* of m did not happen in the parsimonious mode, then it must have happened in the recovery mode of epoch e. Then, by *Claim 2*, all correct parties eventually enter the recovery mode of e.

Claim 1: Let C be any set of $t+1$ correct parties that commit sequence number $\tilde{s}$ in epoch e. Upon the *c-delivery* of some payload with sequence number $\tilde{s}$, every party $P_j \in C$ (re)starts a timer T (line 1.28). There are two cases to consider: **(a)** every $P_j \in C$ commits $\tilde{s}+1$ before T expires, and **(b)** T expires at some party $P_j \in C$.

Case (a): If every $P_j \in C$ commits sequence number $\tilde{s}+1$ before T expires, then *Claim 1* can be applied recursively up to sequence number $X-1$. If $t+1$ correct parties commit sequence number $X-1$, then they enter the recovery mode at line 1.38. By transition amplification, all correct parties eventually enter the recovery mode of epoch e.

Case (b): If any $P_j \in C$ times out on T, then P_j triggers an *a-broadcast* event for a `dummy` payload m at all correct parties (lines 1.27 and 1.18). Thus, eventually $2t+1$ correct parties *a-broadcast* m. If t or fewer correct parties *a-deliver* m before the failure detector $\mathcal{FD}$ triggers, then $\mathcal{FD}$ calls the *transition*() function at $t+1$ or more correct parties. Hence, at least $t+1$ correct parties sent out a `transition` message (lines 1.53–1.54). By the transition amplification mechanism (line 1.50), all correct parties eventually enter recovery mode (line 1.52). On the other hand, if $t+1$ correct parties *a-deliver* m before $\mathcal{FD}$ triggers, then they have all committed sequence number $\tilde{s}+1$. Then, as in **Case (a)**, *Claim 1* follows from the recursive application of the hypothesis.

Claim 2: If any correct party P_i enters recovery mode at line 1.52, then it has received a `transition` message from $t+1$ correct parties. Therefore, every correct party eventually receives $t+1$ `transition` messages and sends out its own `transition` message. By the transition amplification mechanism (line 1.50), all correct parties eventually enter the recovery mode of epoch e. If P_i has entered recovery mode at line 1.38, then it has committed sequence number $X-1$. As $X \geq 2$, $t+1$ correct parties have committed sequence number $X-2$. By *Claim 1*, all correct parties eventually enter the recovery mode of epoch e. □

Lemma 5. *If all correct parties have entered the recovery mode of epoch* e*, then all correct parties eventually enter epoch* $e+1$.

Proof. We need to show that no correct party P_i blocks in the recovery mode of epoch e. If P_i blocks in the recovery mode, then it blocks at one of the following **wait** statements found in part 1 (line 2.4), in part 2 (lines 2.9 and 2.12), in part 3 (line 2.24), or in part 4 (lines 2.31 and 2.34) of the recovery mode.

Part 1 Assume that P_i blocks at line 2.4. Thus, the composite predicate $Q_{ID|\texttt{equality}.e}(R_{s-1}, s_i-2)$ **and** $Q_{ID|\texttt{consistency}.e}(R_{s-1}, s_i-1)$ equals *false*. P_i goes to line 2.4 only after sending a `proof_request` message for sequence number s_i-1 (line 2.1). We distinguish the following cases **(a)** $s_i = 0$, **(b)** $s_i = 1$, and **(c)** $s_i \geq 2$, and show that in all three cases both predicates eventually become *true*, contradicting the assumption.

Case (a): If $s_i = 0$, then according to the code in lines 2.39–2.42, every correct party responds with a `proof` message carrying $\perp$ and valid signatures on $(ID, \texttt{proof}, e, \tilde{s}, \perp)$, where $\tilde{s} \in \{-2, -1\}$. Thus, P_i eventually receives `proof` messages from $n-t$ correct parties for sequence numbers -2 and -1 carrying $\perp$, such that predicate $Q_{ID|\texttt{equality}.e}$ **and** $Q_{ID|\texttt{consistency}.e}$ eventually becomes *true*.

Case (b): If $s_i = 1$, then P_i has committed sequence number 0 to some payload m. In this case, every correct party P_j (i.e., at least $n-t$ parties) responds with a `proof` message carrying $\perp$ together with a valid signature on $(ID, \texttt{proof}, e, -1, \perp)$ and $m_j \in \{m, \perp\}$ together with a valid signature on $(ID, \texttt{proof}, e, 0, m_j)$. Moreover, P_i eventually receives a `proof` message for sequence number 0 carrying m. It is easy to verify that P_i collects sufficient tuples (m_j, σ_j) such that $Q_{ID|\texttt{equality}.e}$ **and** $Q_{ID|\texttt{consistency}.e}$ holds.

Case (c): As $s_i \geq 2$, P_i has committed sequence number s_i-1. Thus, a set C of $\lceil \frac{n-t+1}{2} \rceil \geq t+1$ correct parties have initialized CB instance with sequence number s_i-1 (line 1.35). As $s_i \geq 2$, every party $P_j \in C$ necessarily has committed sequence number s_i-2 (lines 1.29 and 1.33). By the Agreement property of CB, every $P_j \in C$ has committed sequence number s_i-2 to the same payload $m \neq \perp$. Thus, upon receiving a `proof_request` message from P_i, every $P_j \in C$ generates a `proof` message for m and a valid signature $\sigma_j = \{(ID, \texttt{proof}, e, s_i-2, m)\}_j$. Thus, P_i eventually receives consistent `proof` messages for sequence number s_i-2 and payload m from every party in C and adds the tuple (m, σ_j) to R_{s-2}. Therefore, eventually the equality equality predicate $Q_{ID|\texttt{equality}.e}(R_{s-2}, s_i-2)$ holds. The same reasoning as in **Case (b)** above can be applied to show that $Q_{ID|\texttt{consistency}.e}$ eventually becomes true.

Part 2 Assume that a correct party P_i blocks at the **wait** statement found at line 2.9. As no correct party blocks in part one of the recovery mode, every correct party P_j sends a `candidate` message at line 2.8. Thus P_i eventually receives a valid `candidate` message from $n-t \geq \lceil \frac{n+t+1}{2} \rceil$ correct parties and hence no correct party blocks at line 2.9. Consequently, every correct party P_i proposes watermark vector W_i for MVBA at line 2.11. By Termination of MVBA, eventually P_i decides some value v and thus no correct party blocks at line 2.12.

Part 3 If some correct party P_i blocks at the **wait** statement found at line 2.24, then $w \geq s_i + 2 \geq 2$. As $w \geq 2$, Lemma 1 implies that a correct party has committed sequence number $w-1$. Thus, a set C of $t+1$ correct parties have initialized CB instance with sequence number $w-1$ (line 1.35). As $w-2 \geq 0$, every party $P_j \in C$ necessarily has committed sequence number $w-2$ (line 1.29). Thus, the condition at line 2.20 ($s_j > w-2$) evaluates to *true*. Therefore, every party $P_j \in C$ sends a `complete` message carrying every sequence number $s' \leq w-2$ committed in epoch e and $log[s']$. Thus, P_i eventually receives consistent `complete` messages from every $P_j \in C$ for all sequence numbers $\leq w-2$. This ensures that no correct party blocks at line 2.24.

Part 4 Assume that a correct party blocks at the **wait** statement found at line 2.31. Every correct party P_j sends a `queue` message carrying the initiation queue $\mathcal{I}_j$, where $\mathcal{I}_j \cap \mathcal{D}_j = \emptyset$. For any two correct parties P_i and P_j that are in Part 4 of the recovery phase of a given epoch, $D_i = D_j$ (by Lemma 7). Thus, eventually P_i receives a `queue` message from $n-t$ distinct parties P_j such that $\mathcal{I}_j$ and $\mathcal{D}_i$ are disjoint, a contradiction. By the Termination condition of MVBA, no correct party blocks at line 2.34. □

Lemma 6. *If any correct party* P_i *enters epoch* $e+1$*, then* P_i *has* a-delivered *every payload to which* $\lceil \frac{n-t+1}{2} \rceil$ *correct parties have committed a sequence number in epoch* e.

Proof. Let $\tilde{s}$ be the sequence number committed to some payload m by $\lceil \frac{n-t+1}{2} \rceil$ correct parties in epoch e. We have to show that P_i *a-delivers* m in epoch e. Let w be the watermark of epoch e. By Lemma 3, $\tilde{s} \leq w$. Let $s-1$ be the highest sequence number that P_i committed before entering the recovery mode of epoch e. If $\tilde{s} < s-1$, then P_i must have *a-delivered* a payload m' to which it committed sequence number $\tilde{s}$ before entering the recovery mode. By

the Agreement property of CB, $m = m'$. If, on the other hand, $\tilde{s} \geq s-1$, we distinguish the following two cases **(1)** $\tilde{s} = s-1$, and **(2)** $\tilde{s} > s-1$.
Case (1) ($\tilde{s} = s-1$): If $\tilde{s} < w$, then $s \leq w$. Thus, P_i *a-delivers* $log[s-1]$ at line 2.16. By the Agreement property of CB, $log[s-1] = m$. If $\tilde{s} = w$, then $s = w+1$, and hence, P_i *a-delivers* some payload $\bar{m}$ from recovery set $\bar{\mathcal{R}}[1]$ for sequence number w (line 2.19). By Lemma 2, $\bar{m} = m$.
Case (2) ($\tilde{s} > s-1$): We distinguish the following three subcases: (2a) $\tilde{s} \leq w-2$, (2b) $\tilde{s} = w-1$, and (2c) $\tilde{s} = w$. Case (2a) implies that $s \leq w-2$. Thus, P_i *a-delivers* all payloads to which some correct party has committed a sequence number smaller than or equal to $w-2$ at line 2.25. Consequently, P_i *a-delivers* a payload $\bar{m}$ to which some correct party has committed sequence number $\tilde{s}$. By the Agreement property of CB, $\bar{m} = m$. Case (2b) implies that $s \leq w-1$, and thus, P_i *a-delivers* a payload $\bar{m}$ from recovery set $\bar{\mathcal{R}}[0]$ for sequence number $\tilde{s} = w-1$ at line 2.28. By Lemma 1, some correct party has committed $\tilde{s}$ to $\bar{m}$. Again, by the Agreement property of CB, $\bar{m} = m$. Case (2c) implies that $s \leq w$. Thus, P_i *a-delivers* a payload $\bar{m}$ from recovery set $\bar{\mathcal{R}}[1]$ for sequence number $\tilde{s} = w$ at line 2.28. By Lemma 2, $\bar{m} = m$. □

Lemma 7. *All correct parties* a-deliver *the same set of payloads in epoch e before entering Part 4 of the recovery mode.*
Proof. We have to show that if a correct party P_i *a-delivers* a payload m in epoch e, then eventually all correct parties *a-deliver* m in epoch e. By the transitive application of Lemmas 4 and 5, all correct parties eventually enter epoch $e+1$. We distinguish between the *a-delivery* of m in **(1)** the parsimonious mode, and **(2)** part 3 of the recovery mode. **Case (1)** implies that there exists a sequence number $\tilde{s}$ such that P_i has committed $\tilde{s}$ to payload m and that P_i has committed sequence number $\tilde{s}+1$. Therefore a set C of $\lceil \frac{n-t+1}{2} \rceil$ correct parties have initialized CB instance with sequence number $\tilde{s}+1$. Thus, every $P_j \in C$ has committed sequence number $\tilde{s}$ to m in epoch e. By Lemma 6, every correct party *a-delivers* payload m in epoch e. **Case (2):** Let $\tilde{s}$ be the sequence number committed to m. It is easy to verify that $w-1 \leq \tilde{s} \leq w$ holds. As no correct party has *a-delivered* m before Part 3 of the recovery mode, $s_j \leq \tilde{s}+1$ holds for all correct parties P_j. If $\tilde{s} = w-1$, then all correct parties *a-deliver* m from $log[s-1]$ (line 2.16) or $\bar{\mathcal{R}}[0]$ (line 2.28). If $\tilde{s} = w$, then all correct parties *a-deliver* m from $\bar{\mathcal{R}}[1]$ (line 2.19 or line 2.28). □

Lemma 8 (Consistency). *If any two correct parties P_i and P_j* a-deliver *payload m and m', respectively, as the $\tilde{s}^{\text{th}}$ payload (where, $\tilde{s} \geq 0$) in epoch e before entering part 4 of the recovery mode, then $m = m'$.*

Proof. The hypothesis implies that $\tilde{s} \leq w$. When both P_i and P_j *a-deliver* m and m' in the parsimonious mode, then they *a-deliver* m and m' from $log[\tilde{s}]$. By the Agreement property of CB, $m = m'$.

Suppose that only P_i *a-delivers* m in the parsimonious mode. We distinguish the following three cases: **(1)** $\tilde{s} \leq w-2$, **(2)** $\tilde{s} = w-1$, and **(3)** $\tilde{s} = w$, and show that if P_j *a-delivers* m' as payload number $\tilde{s}$ in the recovery mode, then $m = m'$. It is clear that if P_j *a-delivers* m' from $log[\tilde{s}]$ (line 2.16) then by the Agreement property of CB, $m = m'$. **Case (1):** P_j *a-delivers* m' from the `complete` messages received from $t+1$ distinct parties that claim to have committed sequence number $\tilde{s}$ to payload m'. As at least one of those is correct, by the Agreement property of CB, $m = m'$. **Case (2):** P_j *a-delivers* m' from $\bar{\mathcal{R}}[0]$ at line 2.28. By Lemma 1 (Property P1), at least one correct party has committed $\tilde{s}$ to m'. By the Agreement property of CB, $m = m'$. **Case (3):** P_j *a-delivers* m' from $\bar{\mathcal{R}}[1]$. By Lemma 2 (P2), $m = m'$.

Now, let us consider the case when both P_i and P_j *a-deliver* m and m' in the recovery mode. If $\tilde{s} \leq w-2$, then some correct party has *a-delivered* m in the parsimonious mode, thus $m = m'$. Further, we distinguish two cases: **(1)** $\tilde{s} = w-1$, and **(2)** $\tilde{s} = w$. **Case (1):** Both P_i and P_j *a-deliver* m and m' either from $log[\tilde{s}]$ or from $\bar{\mathcal{R}}[0]$. By Lemma 1 (Property P1) and the Agreement property of CB, $m = m'$. **Case (2):** P_i and P_j *a-deliver* m from $\bar{\mathcal{R}}[1]$. By the Agreement property of MVBA and the deterministic choice of $\bar{\mathcal{R}}[1]$, $m = m'$. □

Lemma 9. AMP *satisfies Agreement and Total Order.*
Proof. By Lemma 7 and Lemma 8, all correct parties deliver the same sequence of payloads before entering part 4 of the recovery mode of epoch e. Due to the Agreement property of MVBA and the fact that all payloads *a-delivered* in part 4 are *a-delivered* in some deterministic order, all correct parties deliver the same sequence of payloads in epoch e. □

Lemma 10. AMP *satisfies Validity.*
Proof. Let e be the largest epoch number at any correct party at the point in time when a set C of $t+1$ correct parties *a-broadcast* payload m. We will show that some correct party P_i *a-delivers* m in epoch e. The proof is by contradiction. Assume that no correct party *a-delivers* m in epoch e. This implies that no correct party *a-delivers* m in the parsimonious mode of epoch e. Therefore, the failure detector $\mathcal{FD}$ will cause a timer expiry at every $P_j \in C$, and eventually some correct party P_i will receive a `transition` message from every $P_j \in C$. The transition amplification mechanism ensures that every correct party eventually enters recovery mode of epoch e. Lemma 5 implies that every $P_j \in C$ eventually enters Part 4 (lines 2.30–2.38) of the recovery mode of epoch e. As m is contained in $t+1$ `queue` messages and every vector Q consists of $n-t$ such messages, there is at least one input for m in $\bar{Q}$. Therefore, P_i *a-delivers* m in epoch e, contradicting the assumption. □

4 Discussion and Conclusion

Atomic Broadcast Using Consistent Broadcast The weak specification of the Consistent Broadcast abstraction is key to the message parsimony of the AMP and PABC protocols. The fact that CB lacks a strong Agreement property such as "if one correct party delivers a payload m, then all correct parties eventually deliver m" is a double-edged sword. The advantage lies in the fact that CB can be implemented employing a centralized communication pattern, as described. The drawback lies in the fact the CB alone cannot be relied on to satisfy the Agreement property of atomic broadcast. Thus, a faulty leader may choose not to involve up to t correct parties in the atomic broadcast protocol. In the PABC protocol, such a leader may cause some subset of correct parties to lag behind other correct parties without being able to catch up. Consequently, Agreement may be violated. In fact, the PABC protocol satisfies a weaker Agreement property conditioned by the fact that an infinite number of payloads are being broadcast by some correct party. To address this drawback, the AMP protocol uses failure detection to guarantee that once a payload has been *a-delivered* by a correct party, all correct parties eventually switch to recovery mode. The recovery mode then ensures that *all* the correct parties synchronize.

WAN Deployment Owing to non-negligible message-loss rates in WANs, message complexity is an important metric to consider when designing protocols for the Internet. In this paper, we have presented a novel Byzantine-fault-tolerant atomic broadcast protocol called AMP that under normal conditions (i.e., when the network is stable and there are no intrusions) has an amortized message complexity of $O(n)$ per atomically delivered payload, does not require expensive public-key cryptography, and has a latency of only five communication steps. In contrast, even under the most benign conditions, the only other asynchronous atomic broadcast protocol with $O(n)$ message complexity, Protocol PABC, uses digital signatures and has a latency of seven communication steps. All other optimally-resilient atomic broadcast protocols that we are aware of incur a cost of $O(n^2)$ messages per atomically delivered payload. Recently, it has been shown [14] that a centralized communication pattern can often outperform decentralized ones despite incurring additional communication steps. Hence, we expect our protocol to offer significant advantages in Internet-like settings over previous work.

Acknowledgments: We gratefully acknowledge the funding support from Microsoft Research via the European PhD Fellowship and also from the EC IP DECOS and NoE ReSIST projects. We thank Charlotte Bolliger for her editorial comments.

References

[1] O. Bakr and I. Keidar. Evaluating the Running Time of a Communication Round over the Internet. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (PODC-2002)*, pages 243–252, 2002.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keyed Hash Functions and Message Authentication. In *Proceedings of CRYPTO*, pages 1–15, 1996.

[3] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and Efficient Asynchronous Broadcast Protocols. In *Advances in Cryptology: CRYPTO 2001 (J. Kilian, ed.), LNCS 2139*, pages 524–541. Springer, June 2001.

[4] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, Nov 2002.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):372–382, Apr 1985.

[6] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[7] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In S. Mullender, editor, *Distributed systems (2nd Ed.)*, pages 97–145. ACM Press - Addison Wesley, 1993.

[8] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. Byzantine Fault Detectors for Solving Consensus. *The Computer Journal*, 46(1):16–35, 2003.

[9] K. Kursawe and V. Shoup. Optimistic Asynchronous Atomic Broadcast. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), LNCS 3580 (L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, eds.)*, pages 204–215, Oct 2005.

[10] J.-P. Martin and L. Alvisi. Fast Byzantine Consensus. In *Proceedings of International Conference on Dependable Systems and Networks (DSN-2005)*, pages 402–411, June 2005.

[11] H. V. Ramasamy. *Parsimonious Service Replication for Tolerating Malicious Attacks in Asynchronous Environments*. PhD thesis, University of Illinois at Urbana-Champaign, 2005.

[12] H. V. Ramasamy and C. Cachin. Parsimonious Asynchronous Byzantine-Fault-Tolerant Atomic Broadcast. In *Proceedings of the 9th International Conference On Principles Of Distributed Systems (OPODIS-2005), LNCS 3974*, pages 88–102, December 2005.

[13] M. K. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communication Security*, pages 68–80, Nov 1994.

[14] A. Shraer and I. Keidar. How to Choose a Timing Model? In *Proceedings of Dependable Systems and Networks (DSN)*, pages 389–398, 2007.

[15] G. Tsudik. Message Authentication with One-Way Hash Functions. *ACM Computer Communications Review*, 22(5):29–38, 1992.