Using Randomized Byzantine Consensus To Improve Blockchain Resilience Under Attack

Afonso Garcia Louro do Nascimento e Oliveira
afonso.n.oliveira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2021

Abstract

The rise in popularity of blockchains has lead to an increasing interest in the development Byzantine Fault-Tolerant (BFT) state machine replication systems for a variety of use cases and operational scenarios. A common approach when implementing these systems is to build on top of leader-based partially synchronous consensus protocols, which rely on critical assumptions about the underlying network in order to guarantee liveness. Alternatively, randomized protocols are able to avoid these pitfalls by operating over a fully asynchronous model. However, existing constructions still fall significantly behind, in terms of optimal performance, when compared to their deterministic counterparts. In this thesis we present Alea-BFT, an asynchronous BFT protocol, which leverages randomization in order to guarantee liveness without relying on timing assumptions about the underlying network and provides significant asymptotic and practical improvements over the state of the protocols in this model. We implemented Alea-BFT and compare its performance with HoneyBadgerBFT and Dumbo1/2, under a deployment scenario using up to 128 replicas, uniformly distributed across the globe. The experimental results demonstrate that Alea-BFT is able achieve multi-fold improvements over the remaining protocols, specially as the system size increases.
Keywords: Consensus, Blockchain, Asynchronous, Randomization, Byzantine Fault Tolerance

1. Introduction

The rise in popularity of cryptocurrencies has led to an increasing interest in deploying blockchain based systems for a variety of use cases and operational scenarios. The need for higher performance guarantees and legal compliance forced organizations to steer away from the permissionless model of Bitcoin [19] in favor of permissioned solutions. In a permissioned scenario, where the identity of network participants is known and the adversary's power is strictly bounded by the fraction of nodes under its control, a classic Byzantine fault tolerant state-machine replication approach is often used as a building block to assemble blockchain systems. The topic of Byzantine Fault Tolerance is not new [15], in fact it has been extensively studied over the last decades and a wide variety of solutions can be found in the literature. The traditional approach for BFT consensus algorithms follows two major design patterns: partial synchrony and leader-based. Assumptions regarding the underlying network allow partially synchronous protocols to escape the limitations of the FLP result [5] whereas relying on a leader to coordinate consensus instances allows for a decrease in the overall algorithm complexity. Protocols such as PBFT [9] and its derivatives mostly fit into the previous model and have been considered the standard in BFT consensus for years. The problem with protocols in this family is that, by operating over a partially synchronous model, progress is only guaranteed during periods where these timing assumptions hold, causing performance to deteriorate or even completely stall during uncivil intervals [22]. Additionally a leader-based approach introduces a single point of failure into the system, for example a malicious leader can purposely slow down the system throughput up until the minimum threshold required to avoid being replaced [2, 10]. Finally, recent development efforts focused mostly on optimizing performance under the assumption that failures do not occur [10], falling back to a more expensive strategies when forced to recover from failures. We argue that in spite of achieving impressive results under relatively controlled operational conditions, protocols in this class, underperform when subject to the more hostile deployment characteristics of blockchain systems, such as wide-area networks of mutually untrusting peers were nodes are not only allowed but actually expected to fail, in an attempt to slow down or subvert critical system

properties, and advocate for a different approach based on non-determinism. Consensus protocols based on randomization [3] bypass FLP by relaxing one of the defining properties of the consensus problem to be probabilistic, therefore being able to operate on a completely asynchronous model, eliminating the possibility for a malicious network scheduler to thwart performance. BFT solutions in this model have been around since 1983 [4, 21], but despite presenting characteristics that make them very interesting from a theoretical standpoint have usually been considered impractical due to high communication costs and expected termination time [18]. We believe that randomized BFT protocols are not only practical but in fact, due to their leaderless design and asynchronous network model, present a more resilient solution than traditional deterministic, leader-base based approaches for deployment in adverse blockchain environments.

1.1. Contributions

The main focus of this thesis was to present a randomized BFT protocol capable of achieving better performance and scalability than existing solutions for the asynchronous model. Further bridging the gap between asynchronous and partially synchronous BFT protocols, while simultaneously providing higher resilience guarantees in the presence of adversarial network scheduling and malicious replicas. With these goals in mind, we designed and implemented a novel randomized atomic broadcast protocol called Alea-BFT, that presents the following characteristics:

- It provides optimal resilience for the Byzantine model, tolerating up to $f = \lfloor \frac{N-1}{3} \rfloor$ faulty processes out of N total processes, where faulty processes are allowed to arbitrarily deviate from the protocol spec and even collude with each other in an attempt to subvert its properties.
- It is completely asynchronous, meaning that no assumptions are made regarding the delivery schedule of messages by the network. This property is crucial to ensure robustness under adversarial network conditions and prevent performance attacks based on timing assumptions.
- It sidesteps from a design approach based on an asynchronous common subset framework, where most of recent efforts in developing asynchronous BFT protocols have been concentrated, in favour of a novel architecture based on a two phased pipeline design.
- It provides significant asymptotic improvements over the state of the art protocols in this model. Particularly, both expected message and communication complexities can be reduced by a factor of up to $\mathcal{O}(N)$, while still terminating in constant time.

Our experimental evaluation of Alea-BFT concluded that it consistently outperforms existing asynchronous atomic broadcast protocols for any system scale, further bridging the performance gap between deterministic and randomized protocols, while still providing all the resilience guarantees characteristic of the asynchronous model.

2. Background

2.1. Consensus Problem

The consensus problem is a fundamental part of distributed systems research, and consists of getting a set of distributed processes to agree on a common value from a collection of initial proposals. A strong form of consensus must satisfy the following properties [14]:

- **Agreement:** All processes that decide do so on the same value.
- **Termination:** Every non-faulty process eventually decides.
- **Integrity:** The decided value must have been proposed by some process.

The first and last properties are safety properties, ensuring that nothing bad happens, whereas termination is a liveness property stating that good things eventually happen [1]. Solving consensus in the absence of faults is a trivial task. However, we are interested in exploring this problem in the presence of faulty processors, particularly in the Byzantine failure model where replicas may behave arbitrarily or even collude in an attempt to subvert the properties of the protocol. Solving consensus also provides a solution to a series of higher-level problems. An important one, in the context of SMR, is the problem of atomic broadcast which informally states that all correct processes must deliver the same sequence of messages. This has been proven to be solvable by running a series of consensus instances deciding on which message to deliver next making both problems equivalent. ABC can be formally defined in terms of the the following properties [14]:

- **Validity:** If a correct process broadcasts a message m, then some correct process eventually delivers m.
- **Agreement:** If any correct process delivers a message m, then every correct process delivers m.
- **Integrity:** A message m appears at most once in the delivery sequence of any correct process.

- **Total Order:** If two correct processes deliver two messages m and m', then both processes deliver m and m' in the same order.

In the paradigm of blockchain an atomic broadcast primitive can be used to establish a total ordering on appends to the ledger.

2.2. FLP Impossibility

The FLP impossibility result [5] states that there is no deterministic solution for the consensus problem, capable of simultaneously ensuring both safety and liveness properties, in an asynchronous model where nodes can fail silently. In order to circumvent this result researchers have devised extensions to the original system model where consensus is possible. The most popular extension approaches can be grouped into the following categories: (i) Timing assumptions, (ii) Failure detectors, (iii) Randomization. The previous techniques are not mutually exclusive and can be used as either a complementary subsystem or combined into a hybrid approach.

2.3. Byzantine Consensus

Byzantine Fault Tolerance (BFT) refers to the ability of a system to tolerate arbitrary behaviour from a subset of its participants without compromising its critical operational properties. Achieving consensus in the event of Byzantine failures was formally proposed by Lamport et al. as the Byzantine Generals Problem [15] and has been the target of extensive academic research ever since. The FLP result is logically still valid in Byzantine model, forcing protocols to operate over the system models with stronger assumptions. Earlier work on this topic considered a *synchronous* model with the first solution was presented by Pease et al. [20], and later improved by Dolev and Strong [11]. A resilience bound of $\lfloor \frac{N-1}{3} \rfloor$ was proven optimal for any Byzantine solution without digital signatures by Ben-Or [4].

2.4. Randomization

Extending the system model using randomization can be used to escape the limitations of FLP. The most common approach is to extend the termination requirement to allow non-terminating executions with a collective probability of zero. The termination requirement can therefore be modified to reflect this probabilistic nature:

- **Probabilistic Termination:** Every non-faulty process eventually decides with probability one.

There are two ways to introduce non-determinism into the system. One is to assume the model itself is randomized and applicable operations on each state only occur probabilistically [6]. The other randomized algorithm approach considers a source of randomness located in the processes themselves. In this model processes have access to *coin-flip* operations that return random binary values according to a certain probability distribution. All of the randomized protocols referenced in our work operate based on this second approach. Another important characteristic of these algorithms is that by operating over an asynchronous model they are completely decoupled from the concept of "real time". For this reason their *running time* is usually characterized based on the expected number of rounds required for termination [8].

2.5. Asynchronous BFT

Randomization allows consensus protocols to operate over a fully asynchronous model, therefore eliminating the need for timing assumptions and the liveness issues associated with them. In order to prove the increases in robustness provided by randomized protocols, Miller et. al [17] devised an experiment where an adversarial scheduler with full control over the delivery of messages attempted to compromise the liveness properties of both PBFT and a novel randomized protocol HoneyBadgerBFT [17]. The experimental results showed that the scheduler indeed prevented PBFT from making any progress at all while HoneyBadgerBFT (and by extension any asynchronous protocol) was still able to continue executing operations. Most recent attempts of implementing practical atomic broadcast protocols for the asynchronous model are instantiated based on an ACS framework. In ACS every party proposes an input value, and outputs a common vector containing the inputs of at least $N - f$ distinct parties. HoneyBadgerBFT [17] is usually regarded as the first practical BFT protocol for the asynchronous model. The authors made the critical observation that atomic broadcast could be built based on an ACS framework by combining it with a threshold encryption scheme, following the structure of Algorithm .1. The protocol proceeds in epochs, every epoch each replica proposes a set of $\lfloor B/N \rfloor$ transactions, where B corresponds a configurable batch size parameter, and delivers $\Omega(B)$ transactions. Proposals are encrypted, using the shared public key distributed during the trusted setup, before being passed as input to to an ACS instance. The output vector of ACS, consisting of at least $N-f$ encrypted proposals, is then subject to a threshold decryption round, where replicas share decryption shares for the proposals included in the vector, before being canonically sorted and committed. The use of threshold encryption prevents an adversary from selectively censoring transactions, by selecting which proposals to include in the ACS output vector, since replicas commit into delivering a certain set of proposals before the adversary learns about the particular contents of each one. A particularly elegant as-

Algorithm .1 Reduction from ACS to ABC

constants:
  N
  B
  PK
  SK_i

state variables:
  $r \leftarrow 0$
  $buf \leftarrow \emptyset$

procedure START
  while true **do**
    // Step 1: Random selection and encryption
    $p \leftarrow buf[0 : \lfloor B/N \rfloor]$
    $v_i \leftarrow TPKE.Enc(PK, p)$

    // Step 2: Agreement on ciphertexts
    input v_i to ACS (r)
    wait until ACS (r) delivers $\{v_j\}_{j \in S}$, where $S \subset [1..N]$ **then**

    // Step 3: Decryption
    for each $j \in S$ **do**
      $e_{j,i} \leftarrow TPKE.DecShare(SK_i, v_j)$
      multicast $\langle \texttt{DEC}, r, j, i, e_j \rangle$
      wait until receive $f+1$ messages in the form $\langle \texttt{DEC}, r, j, k, e_{j,k} \rangle$ **then**
        $y_j \leftarrow TPKE.Dec(PK, \{(k, e_{j,k})\})$

    // Step 4: Delivery
    $block_r \leftarrow sorted(\cup_{j \in S}\{y_j\})$
    $buf \leftarrow buf - block_r$
    $r \leftarrow r + 1$
    output $block_r$

pect of HoneyBadgerBFT is its ACS construction, illustrated in Figure 1, resulting from the composition of two sub-protocols/phases RBC and ABA. During the broadcast phase, every replica starts an RBC instance in order to disseminate its proposal across all other replicas.

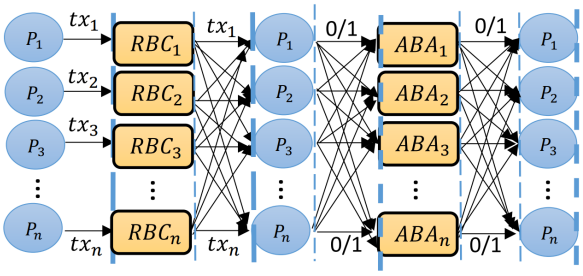

Figure 1: The structure of ACS in HoneyBadgerBFT [17].

Other asynchronous protocols, such as BEAT [12], EPIC [16] and Dumbo1/2 [13] all fit into the same ACS framework.

3. Implementation

Recent efforts in implementing practical asynchronous atomic broadcast protocols, have mostly concentrated efforts around an ACS framework. Protocols in this framework despite having been proven as practical [17], disproving the conception that asynchronous BFT protocols are naturally inefficient, and further refined in subsequent work [12, 16, 13], still present some significant drawbacks. In Alea-BFT we completely sidestep from an ACS based framework in favor of a novel pipelined architecture composed by broadcast and agreement components, executed in parallel, that communicate with each other by performing write and read operations over a shared data structure.

3.1. System Model

We consider a network composed of N processes, uniquely identified from the static set $S = \{P_0, ..., P_{N-1}\}$ out of which up to f may fail, as well as an unbounded number of clients. Our protocol provides an atomic broadcast channel characterized by local SEND and DELIVER events parameterized by a payload value. A replica may execute SEND an arbitrary number of times, triggered in response to client requests, and must be prepared to DELIVER as many messages as the atomic broadcast channel outputs. We assume a Byzantine failure model where up to $f = \lfloor \frac{N-1}{3} \rfloor$ processes can fail during the execution of the protocol, we will refer to these processes as corrupt. The adversary is given full control over the behaviour of these corrupted processes meaning that they can stop, deviate arbitrarily from the protocol's specification and even collude among each other in order to subvert the properties of the protocol. The remaining processes that do not fail during protocol execution are labeled as correct. The system is asynchronous, with the delivery schedule of messages being delegated under adversarial control without bounds on communication delays or processing times. We consider the processes to be fully-connected by reliable channels providing guarantees that messages are eventually delivered and no modifications to the messages occur in the channel. Lastly the adversary is said to be computationally bound and therefore unable to subvert the cryptographic primitives employed.

3.2. Building Blocks

Similarly to other protocols in this setting we provide an higher level protocol for atomic broadcast which invokes sub-protocols to carry out certain tasks. In this modular stack architecture, upper level protocols can provide input and receive output from sub-protocols down the stack. Now we present some baseline definitions and describe the underlying primitives that are used as building blocks when implementing the Alea-BFT protocol.

A threshold signature scheme (TSS) is a cryptographic primitive that allows for multiparty key generation and signing. A (t, n)-threshold signature

scheme allows a subset of t out of n participants to generate a valid signature while disallowing its creation when the number of protocol participants is smaller than t. In involves distributing shares of a signing key sk_i to each of the N parties as well as a common public key mpk and a public key vector PK. A (t, n)-TSS must provide two basic security requirements:

- **Unforgeability**: It is infeasible for a polynomial-time adversary to output a valid signature on a message that was submitted as a signing request to less than $n - t$ honest parties.

- **Robustness**: It is computationally infeasible for an adversary to produce t valid signature shares such that the output of the share combining algorithm is not a valid signature.

Particularly, Alea-BFT relies threshold schemes for digital signatures and coin-tossing. They are all non-interactive and therefore do not require any particular communication pattern and can be easily integrated into an asynchronous protocol.

A verifiable consistent broadcast protocol (VCBC) provides an extension of consistent broadcast that allows any party P_i, that has delivered the payload message m, to inform another party P_j about the outcome of the broadcast execution, allowing it to deliver m immediately and terminate the corresponding VCBC instance. More formally, a VCBC protocol guarantees the following additional properties over CBC [7]:

- **Verifiability:** If a correct party delivers a message m, then it can produce a single protocol message M that it may send to other parties such that any correct party that receives M can safely deliver m.

- **Succinctness:** The size of the proof σ carried by M is independent of the length of m.

An asynchronous binary agreement (ABA) primitive allows correct processes to agree on the value of a single bit. Each process P_i proposes a binary value $b_i \in \{0, 1\}$ and decides for a common value b from the set of proposals by correct processes. Formally, a binary agreement primitive can be defined by the following properties:

- **Agreement:** If any correct process decides b and another correct process delivers b', then $b = b'$.

- **Termination:** Every correct process eventually decides.

- **Validity:** If all correct processes propose b, then any correct process that decides must decide b.

Following the FLP impossibility result [5], there is no deterministic algorithm capable of satisfying all the previous proprieties in the asynchronous model of Alea-BFT. A solution to this is problem is resort to a randomized model that guarantees termination in a probabilistic way. As a result, the termination property is replaced with the following:

- **Termination:** Every correct process eventually decides with probability 1.

A priority queue is a custom data structure for storing elements, sorted according to their priority values. We refer to each position in a priority queue as a slot, uniquely identified by a priority value associated with it. Only a single element can ever be inserted in a given slot, even after being removed, as the slot is permanently labeled as used and cannot store another element. There is a special slot called the head slot, that always points to the lowest priority slot whose value hasn't been removed yet. The pointer to the head slot progresses incrementally, conditioned by the insertion and removal of elements from the queue. A priority queue exposes the following attributes:

- **id** : The unique identifier of the queue (static).

- **head** : The priority associated with the head slot of the queue (dynamic).

Additionally, a priority queue provides an interface for interacting with its contents as described bellow:

- **Enqueue** (v, s) : Add an element v with a given priority value s to the queue, ignore if the corresponding slot is not empty.

- **Dequeue** (v) : Remove the specified element v from the queue, if it is present.

- **Get** $(s) \rightarrow \{v, \perp\}$: Retrieve the element v contained in the slot specified by the priority s, or $\perp$ if the slot is empty.

- **Peek** $() \rightarrow \{v, \perp\}$: Retrieve the element v in the head slot of the queue, or $\perp$ if the slot is empty.

A queue mapping function is a function $F(r)$ that identifies the priority queue, over which the protocol should operate for a given round r. Informally it can be thought of as a leader election function, responsible for selecting which replicas pre-ordered proposals do operate over for any given r. This function can be any deterministic mapping from $\mathbb{N}$ to $i \in [0, N[$ as long as, for any given value r, there is are values $r' > r$ such that $F(r')$ spans over all elements in $[0, N[$, guaranteeing that a queue is always eventually revisited in a subsequent round. For our initial implementation of Alea-BFT we chose a queue mapping function $F(r) = r \ \% \ N$, which iterates over the priority queues following a round robin distribution.

3.3. Protocol

In Alea-BFT we completely sidestep from an ACS based framework in favor of a novel pipelined architecture composed by broadcast and agreement components, executed in parallel, that communicate with each other by performing write and read operations over a set of N priority queues. Processes maintain two state variables shared between components. The variable S_i consisting of the set of all messages delivered by the protocol, is initialized as empty upon a call to the START procedure and updated during the execution of the agreement component. The variable $\texttt{queues}_i$ contains N priority queues, each one mapping to a distinct replica $P_i, \forall_i \in [0, N[$. Algorithm .2 is responsible for initializing the shared state variables and starting the pipeline components upon a to call to the START procedure. The broadcast component is responsible for establishing a local pre-order over the client updates received and propagating that order to other replicas. The overall component flow, illustrated in Algorithm .3, starts with replicas receiving client requests and storing them in a buffer. When the number of requests in the buffer exceeds a certain threshold B, corresponding to the batch size, the requests are removed from the buffer to form a proposal. An incremental sequence number is attributed to the proposal and the pair is disseminated via a VCBC primitive. Note that a sequence number is local to each replica, meaning that proposals originating from different replicas may share the same sequence number. When a replica VCBC-delivers a proposal, it stores it in a backlog pertaining to the proposer, locally sorted according to the respective priority value, such that it can later be picked up by the next stage of the pipeline. Particularly, every replica maintains N separate proposal backlogs, consisting of undelivered pre-ordered proposals by each of the N replicas. Additionally, each replica maintains two local state variables, an buffer of pending requests buf_i, and an integer value, $priority_i$, indicating the next sequence number it should attribute to a proposal. The agreement component, presented in Algorithm .4, is responsible for establishing a total order over the backlogs of replica proposals created by the execution of the broadcast component. It proceeds in rounds, such that for each round the backlog of proposals pertaining to a certain replica (which for ease of description we refer to as the round leade) is selected for processing according to a mapping function F. A correct replica examines the leader's proposal backlog and inputs a value into an ABA execution depending on its contents. Particularly, if it had previously delivered a proposal from the current leader with priority s, it expects the backlog to contain a proposal with priority $s+1$ to be ordered next. Or $s+2$, if the $(s+1)$-th leader proposal happens to be a duplicate from an already ordered proposal originating from a distinct replica. If it contains such proposal, it inputs 1 into an ABA instance, or 0 otherwise. The outcome of the ABA will dictate whether the pre-ordered proposal should be totally ordered for that round or not. Due to the fact that the broadcast primitive used by Alea-BFT does not guarantee totality, some replicas may need execute a fallback sub-protocol to actively fetch this value from others. Processes maintain a single state variable r_i, serving as an unique identifier for the current agreement round. The execution of the agreement component starts with a call to the AC-START procedure, which initializes the local variable r_i to 0 and begins execut-

Algorithm .2 Alea-BFT - Initialization

```
1:  constants:
2:      N
3:      f

4:  state variables:
5:      S_i ← ∅
6:      queues_i ← ∅

7:  procedure START
8:      queues_i[x] ← new pQueue() , ∀x ∈ [0, N[
9:      async BC-START()
10:     async AC-START()
```

Algorithm .3 Alea-BFT - Broadcast Component

```
1:  constants:
2:      B

3:  state variables:
4:      buf_i
5:      priority_i

6:  procedure BC-START
7:      buf_i ← ∅
8:      priority_i ← 0

9:  upon receiving a message m, from a client do
10:     if m ∉ S_i then
11:         buf_i ← buf_i ∪ {m}
12:         if |buf_i| = B then
13:             input buf_i to VCBC (i, priority_i)
14:             buf_i ← ∅
15:             priority_i ← priority_i + 1

16: upon outputting m for VCBC (j, priority_j) do
17:     Q_j ← queues_i[j]
18:     Q_j.Enqueue(priority_j, m)
19:     if m ∈ S then
20:         Q_j.Dequeue(m)
```

ing the agreement loop.

Algorithm .4 Alea-BFT - Agreement Component

```
1: state variables:
2:     r_i

3: procedure AC-START
4:     r_i ← 0
5:     while true do
6:         Q ← queues_i[F(r_i)]
7:         value ← Q.Peek()
8:         proposal ← v ≠ ⊥ ? 1 : 0
9:         input proposal to ABA (r_i)
10:        wait until ABA (r_i) delivers b then
11:            if b = 1 then
12:                if Q.Peek() = ⊥ then
13:                    broadcast ⟨FILL-GAP, Q.id, Q.head⟩
14:                wait until (v ← Q.Peek()) ≠ ⊥ then
15:                    AC-DELIVER(v)
16:        r_i ← r_i + 1

17: upon receiving a valid ⟨FILL-GAP, q, s⟩ message from P_j
    do
18:     Q ← queues_i[q]
19:     if Q.head ≥ s then
20:         entries ←VCBC(queue, s').REQ ∀_{s'} ∈ [s, Q.head]
21:         send ⟨FILLER, entries⟩ to P_j

22: upon delivering a valid ⟨FILLER, entries⟩ message do
23:     for each ⟨ANS, *, *⟩ message M ∈ entries do
24:         deliver M to the corresponding VCBC

25: procedure AC-DELIVER(value)
26:     for each n ∈ N do
27:         queues[n].Dequeue(value)
28:     S ← S ∪ {value}
29:     output value
```

3.4. Efficiency Analysis

By analysing Alea-BFT we observe that, for every particular proposal payload to be delivered, message exchanges occur in three different places. First, during the execution of the broadcast component, a replica initiates a VCBC instance to disseminate the pre-ordered proposal across all replicas, which are queued in a priority queue slot according to the priority value assigned to it. Second, all replicas participate in successive ABA instances in order to decide, whether or not, to deliver a particular slot's contents. We denote by σ the average number of ABA instances executed over a single slot before it decides for 1 and its contents are scheduled for delivery. Finally, a fallback sub-protocol is triggered by replicas that did not VCBC-deliver the proposal before the corresponding ABA execution decided for 1, in order to actively fetch it from the other replicas.

Time (round) complexity is defined as the expected number of communication rounds before a protocol terminates, or before a given value is output in case of a continuous protocol with online inputs and outputs, such as Alea-BFT (atomic broadcast). The first and third steps terminate in constant time $\mathcal{O}(1)$, whereas the total number of rounds required for the agreement component to decide depend on the value of σ, therefore bounding the overall time complexity of Alea-BFT as $\mathcal{O}(\sigma)$.

Message complexity is defined as the expected number of messages generated by correct replicas during the execution of the protocol. The VCBC instance executed during the broadcast phase generates $\mathcal{O}(N)$ messages, every ABA instance exchanges $\mathcal{O}(N^2)$ messages and finally the third recovery phase incurs an overhead of $\mathcal{O}(N)$ messages per each replica that triggers this fallback protocol. Hence, the message complexity of Alea-BFT is $\mathcal{O}(\sigma N^2)$, due to the σ ABA instances that are executed per priority queue slot prior to delivery.

Communication complexity consists of the expected total bit-length of messages generated by correct replicas during the protocol execution. Let $|m|$ correspond to the average proposal size and λ the size of a threshold signature share. The execution of VCBC incurs a communication complexity of $\mathcal{O}(N(|m| + \lambda))$, each ABA instance requires correct nodes to exchange $\mathcal{O}(\lambda N^2)$ bits and finally each replica that triggers the recovery phase adds an additional communication cost of $\mathcal{O}(N(|m|+\lambda))$ bits. This results in an expected total communication complexity of $\mathcal{O}(\sigma\lambda N^2 + N^2(|m| + \lambda))$, due to the σ ABA executions and up to N recovery round triggers.

Table 1: Complexity of Alea-BFT decomposed by stages.

Stage	Message	Communication	Time
Broadcast	$\mathcal{O}(N)$	$\mathcal{O}(N(\|m\| + \lambda))$	$\mathcal{O}(1)$
Agreement	$\mathcal{O}(\sigma N^2)$	$\mathcal{O}(\sigma\lambda N^2)$	$\mathcal{O}(\sigma)$
Recovery	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2(\|m\| + \lambda))$	$\mathcal{O}(1)$
Total	$\mathcal{O}(\sigma N^2)$	$\mathcal{O}(\sigma\lambda N^2 + N^2(\|m\| + \lambda))$	$\mathcal{O}(\sigma)$

As previously mentioned, Alea-BFT does not provide a constant time reduction from VCBC and ABA to atomic broadcast. In particular, this is because multiple redundant zero deciding ABA instances may be executed over the same priority queue slot until its contents are considered to be totally ordered. However, we argue that despite being theoretically unbounded, the value of σ is in practice a very small constant, which ultimately converges to the optimal value of 1. This statement comes from the observation that given a round-robin queue mapping function F, the same queue is revisited every N epochs, meaning N sequential ABA instances must have been executed by the time a particular queue is revisited. Considering the validity property of ABA, which states that the decided value must have been proposed by a correct process, then the termination of a VCBC instance by $N - f$ correct replicas guarantees that the next

ABA execution pertaining to it will decide for 1. Therefore, in order for the average value of σ to increase by a single unit replicas must complete N sequential ABA executions for every single VCBC instance, a very unlikely scenario given the characteristics of these protocols.

4. Results

For our WAN experiments we deployed Alea-BFT, Dumbo1/2 and HoneyBadgerBFT on 4, 8, 16, 32, 64 and 128 Amazon EC2 t2.medium instances, uniformly distributed across 10 different regions (Paris, London, Frankfurt, Singapore, Tokyo, Mumbai, California, Virginia, Central Canada and St. Paulo) therefore spanning 4 continents. Each instance was equipped with 2 virtual CPUs, 4GB of memory and running Amazon Linux 2. We split our experiments in test groups based on the system scale N and a varying batch size B ranging from 4 up to 10^6 transactions. We use a fixed transaction size of 250 bytes across all our experiments.

4.1. Measuring σ and Message Complexity

In Section 3 we presented a theoretical analysis on the complexity of Alea-BFT. The analysis showed that all complexity metrics are dependent on the value of a variable σ, corresponding to average the number of ABA executions per delivered proposal, which is theoretically unbounded in the presence of a particularly adversarial network scheduler. In an attempt to quantify the actual value of σ under realistic network conditions we measured the number of messages generated by correct processes during protocol execution for different system scales using a constant batch size of 1000 transactions. Figure 2 compares the average number of messages generated by each correct process during the execution of Alea-BFT, amortized by the number of delivered proposals, against a theoretical simulation for different σ values. As we can see the experimental measurements follow very closely the theoretical simulations for which $\sigma = 1$, independently from the system scale, further supporting our hypothesis that despite being theoretically unbounded, under a realistic deployment scenario, the value of σ does in fact converge to optimal value of 1. A follow-up question is how does this translate, in practice, in terms of relative message complexity when compared to the state of the art. To this end, we measured experimentally the relative message complexity per replica, of Alea-BFT in comparison with HoneyBadgerBFT and Dumbo1/2. As we can see in Figure 3, this metric scales exponentially for the protocols based on ACS while staying linear for Alea-BFT. This is expected as in an ACS framework every replica must RBC broadcast its proposals for that batch which incurs $\mathcal{O}(N^2)$ messages per replica, while in Alea-BFT the broadcast primitive used has a message complexity of $\mathcal{O}(N)$. Additionally, to cover an unfavorable scenario where σ increases to a high value, we also include experimental measurements for Alea-BFT using a malicious scheduler that purposely delayed the delivery of VCBC instances by $N - f$ replicas in order to artificially increasing the value of σ to N. As illustrated in Figure 3, even under unrealistically adversarial network conditions Alea-BFT still requires fewer message exchanges than HoneyBadgerBFT, despite exceeding both Dumbo protocols. This makes sense, as for a scenario where $\sigma = N$, both Alea-BFT and HoneyBadgerBFT require N ABA executions per consensus instance while Dumbo1 and 2 reduce this number to a small value k (independent of N) and a constant value,

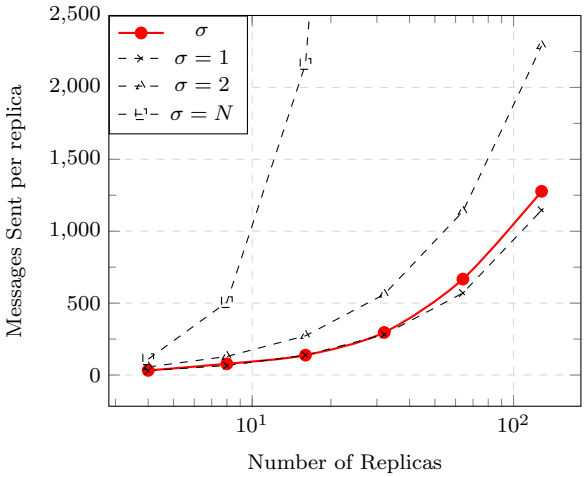

Figure 2: Messages generated per replica per batch delivered (Alea-BFT).

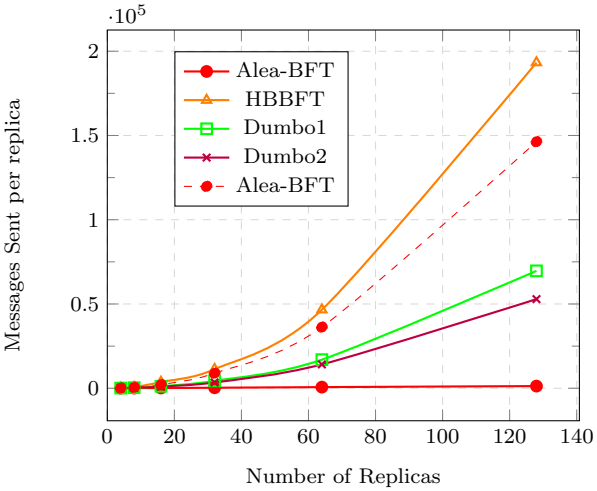

Figure 3: Messages generated per replica per batch delivered.

respectively. The previous experiments help corroborate our initial hypothesis regarding the practical value of σ converging into a constant. Table 2 presents the expected complexities of Alea-BFT, HoneyBadgerBFT and Dumbo1/2 when setting σ to the measured value of 1, further improving the state of the art, for asynchronous atomic broadcast protocols, by a factor of N in terms of both message and communication complexity.

Table 2: Comparison of atomic broadcast protocols, assuming σ is constant.

Protocol	Message	Communication	Time
HBBFT	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2\|m\| + \lambda n^3 \log N)$	$\mathcal{O}(\log N)$
Dumbo1	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2\|m\| + \lambda n^3 \log N)$	$\mathcal{O}(\log k)$
Dumbo2	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2\|m\| + \lambda n^3 \log N)$	$\mathcal{O}(1)$
Alea-BFT	$\mathcal{O}(N^2)$	$\mathcal{O}(\lambda N^2 + N^2(\|m\| + \lambda))$	$\mathcal{O}(1)$

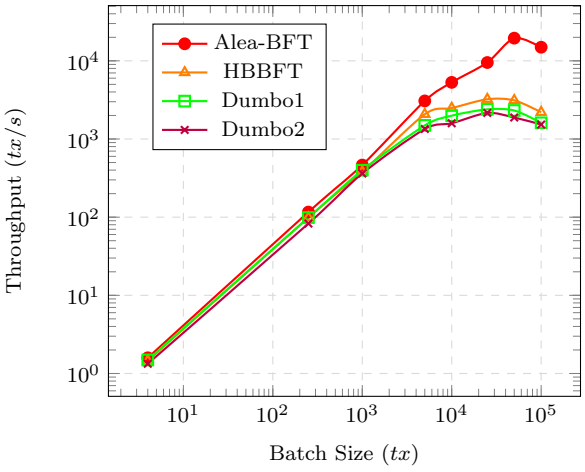

Figure 4: Throughput with varying batch sizes.

4.2. Throughput

Throughput is defined as the rate at which requests are serviced by the system, in case of a transaction processing system throughput is measured as the number of transactions committed by unit of time. In our experiments we measured the throughput by launching multiple replicas executing the protocol being evaluated while periodically registering the number of transactions committed during that interval. Figure 4.2, compares the throughput of Alea-BFT, HoneyBadgerBFT and Dumbo1/2, for different system scales, as the batch size increases. The positive slopes for the measurements pertaining to Alea-BFT indicate that our experiments did not fully saturate the available bandwidth, and it would be possible to attain higher throughput by increasing the batch size. In contrast, for HoneyBadgerBFT and Dumbo1/2, the overall throughput of the system initially increases linearly to the increase of system load, but eventually the throughput stops increasing in some cases even start decreasing. This result follows naturally from the differences in communication complexity between the protocols, as presented in Table 2. For the remaining of this section we will compare the performance of these protocols under the same batch size as we assume the available bandwidth is ample and not a bottleneck, However, readers should be aware that Alea-BFT will provide much higher throughput if using a larger batch size. In Figure 5, we present the throughput of the protocols using a fixed batch size of 5000 transactions. We start by noticing that Alea-BFT not only outperforms all other protocols for all systems scales, but this discrepancy in performance actually increases with the number of replicas in the system revealing better scalability. This is expected since Alea-BFT presents lower message and communication complexities that its counterparts while also reducing the number of expensive threshold cryptography operations required to commit a transaction batch. Another interesting remark is that for smaller values of N, HoneyBadgerBFT actually outperforms both Dumbo protocols despite being theoretically more expensive.

4.3. Latency

Latency is defined as the time interval between the instant the first correct replicas start the protocol and $(N - f)$ replicas deliver the result. In our experiments we measured the latency from the instant replicas select a transaction to propose, from the pending buffers, until the instant $(N - f)$ deliver it to the application layer. In HoneyBadgerBFT and Dumbo1/2 this corresponds to an instance of ACS plus the threshold decryption round, while in Alea-BFT it encompasses the full pipeline and including the period during which a transaction is waiting in the priority queues for the agreement component to select it for delivery. In Figure 6, we examine the average latency of the protocols under no contention for different system sizes, having each node propose a single transaction at the time while no other requests are being made. As we can see for small values of N, the basic latency of all four protocols is very similar. However, as the system size increases we start to observe some considerable difference, with the average latency of HoneyBadgerBFT increasing much faster than the remaining protocols. This discrepancy can be explained by the latency overhead associated with running multiple ABA instances, a factor that is greatly reduced in both Alea-BFT and the Dumbo protocols. Note that the basic latency of Alea-BFT is greatly influenced by the replica which proposes the transaction, as the priority queues containing ordered proposals are traversed in a round robin manner by replica id. Particularly, a proposal from a low id replica will re-

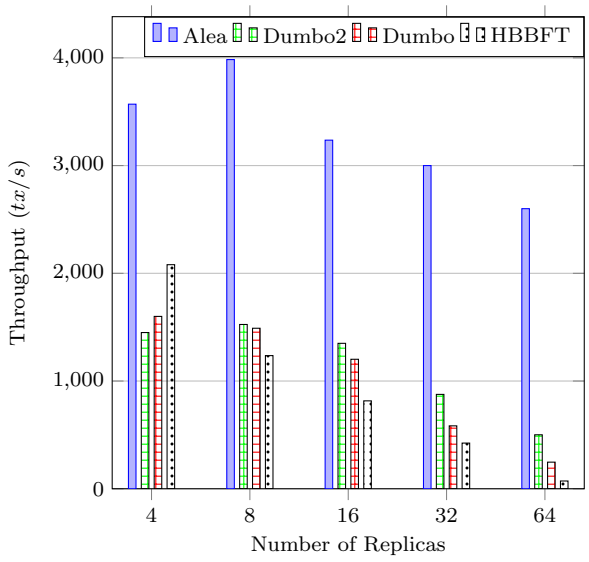

Figure 5: Throughput of Alea-BFT, HoneyBadgerBFT and Dumbo1/2 with a batch of $5 * 10^3$ txs.

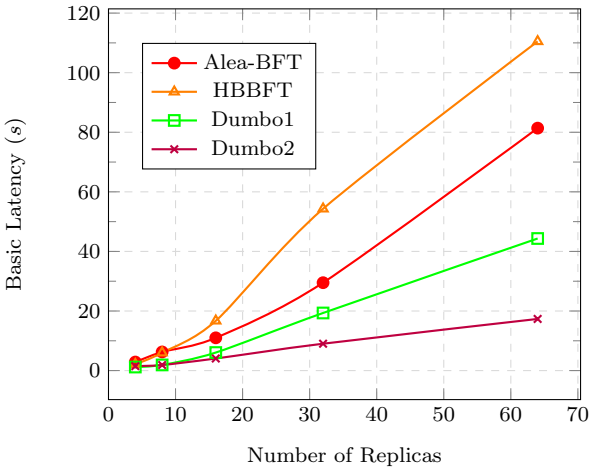

Figure 6: Basic latency.

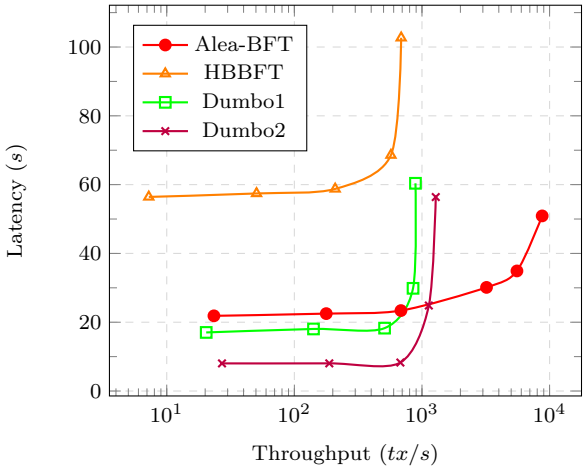

Figure 7: Throughput vs. Latency (n=32).

sult in a much smaller basic latency measurement than if the proposal originated from a replica with an higher identifier. For this particular experiment the proposer was randomly selected and the results averaged across multiple runs. Figure 7 shows how latency evolves as the system load increases for a medium system scale of 32 replicas. For all protocols, initially the latency stays relatively stable only presenting small increases as the system load grows. However, as we reach the nominal capacity of each protocol we start to notice a very steep increases in latency as the system resources stop being able to keep up with the increase in system load. We notice that Alea-BFT is able to sustain a stable latency for much higher system loads than all the other protocols due to its higher throughput and optimized bandwidth usage.

5. Conclusions

In this work, we were motivated by the fact that existing BFT protocols based on timing assumptions, present a series of characteristics that make them vulnerable to performance degradation attacks, and explored the use of randomization as possible a robustness mechanism. The main contribution consists on a different method for constructing asynchronous ABC protocols, which completely sidesteps from the ACS based model popularized by HoneyBadgerBFT where most recent work in the area of asynchronous BFT has been focused. Our protocol provides significant asymptotic and practical improvements over the state of the art protocols in this model. Particularly, an expected improvement by a factor of up to $\mathcal{O}(N)$ in terms of both message and communication complexities. The experimental evaluation of Alea-BFT concluded that it is able achieve multi-fold improvements over the current state of the art protocols, specially as the system size grows, therefore also providing better scalability guarantees. Overall, the main conclusion to retain from this thesis is that randomized BFT protocols can in fact be used to deploy high performance systems, while still providing all the resilience guarantees associated with a fully asynchronous operation model.

References

[1] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[2] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack.

IEEE Transactions on Dependable and Secure Computing, pages 564–577, 2011.

[3] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, (2–3):165–175, Sept. 2003.

[4] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, page 27–30, New York, NY, USA, 1983. Association for Computing Machinery.

[5] E. Borowsky and E. Gafni. Generalized flp impossibility result for t-resilient asynchronous computations. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 91–100, New York, NY, USA, 1993. Association for Computing Machinery.

[6] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, page 824–840, Oct. 1985.

[7] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. volume 2139, 04 2001.

[8] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 42–51, New York, NY, USA, 1993. Association for Computing Machinery.

[9] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

[10] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, page 153–168, USA, 2009. USENIX Association.

[11] D. Dolev and H. R. Strong. Polynomial algorithms for multiple processor agreement. STOC '82, page 401–407, New York, NY, USA, 1982. Association for Computing Machinery.

[12] S. Duan, M. K. Reiter, and H. Zhang. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2028–2041, New York, NY, USA, 2018. Association for Computing Machinery.

[13] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 803–818, New York, NY, USA, 2020. Association for Computing Machinery.

[14] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. 1994.

[15] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, page 382–401, July 1982.

[16] C. Liu, S. Duan, and H. Zhang. Epic: Efficient asynchronous bft with adaptive security. pages 437–451, 06 2020.

[17] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. CCS '16, page 31–42, New York, NY, USA, 2016. Association for Computing Machinery.

[18] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Ritas: Services for randomized intrusion tolerance. *IEEE Transactions on Dependable and Secure Computing*, pages 122–136, 2011.

[19] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008. Accessed: 2021-01-08.

[20] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, page 228–234, Apr. 1980.

[21] M. O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409, 1983.

[22] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe. Bft protocols under fire. NSDI'08, page 189–204, USA, 2008. USENIX Association.