2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Byzantine Fault-Tolerant Atomic Multicast

Paulo Coelho*†, Tarcisio Ceolin Junior‡§, Alysson Bessani¶, Fernando Dotti§ and Fernando Pedone*
*Università della Svizzera italiana - Switzerland
†Universidade Federal de Uberlândia - Brazil
‡Universidade Federal de Santa Maria - Brazil
§Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul - Brazil
¶LaSIGE, Faculdade de Ciências, Universidade de Lisboa - Portugal

***Abstract*—Atomic multicast is an important building block in the architecture of scalable and highly available services. Atomic multicast reliably propagates and orders messages addressed to one or more groups of processes. Despite the large body of literature on atomic multicast, existing protocols target benign failures. This paper presents ByzCast, the first Byzantine Fault-Tolerant atomic multicast. Byzantine Fault Tolerance has become increasingly appealing as services can be deployed in inexpensive hardware (e.g., cloud environments) and new applications (e.g., blockchain) become more sensitive to malicious behavior. ByzCast has two important characteristics: it was designed to use existing BFT abstractions and it scales with the number of groups, for messages addressed to a single group. We discuss the design of ByzCast and how it can be optimized for particular workloads. Besides proposing a novel atomic multicast protocol, we extensively assess its performance experimentally.**

I. Introduction

Modern online services are expected to be always available and scalable. High demand for services that can tolerate failures and sustain ever-increasing load has led to different designs and tradeoffs. According to the guarantees provided to service users, two classes of solutions exist. Systems that provide *weak consistency* (e.g., [1], [2], [3], [4]) can typically deliver high performance at the cost of exposing non-intuitive application behavior to the users. Systems that focus on *strong consistency* (e.g., linearizability [5]) provide more intuitive service behavior but require requests to be ordered across the system before they can be executed by the servers [6], [7].

Atomic multicast is a fundamental communication abstraction in the design space of strongly consistent distributed systems. Atomic multicast abstracts the complexity involved in reliably propagating and ordering requests, and in doing so it provides stronger communication guarantees than "best-effort" network-level communication (e.g., IP multicast). With atomic multicast, processes can multicast messages to different groups of destination processes (e.g., different shards) with the guarantee that destinations will reliably deliver these messages in acyclic order. Acyclic order implies that destinations deliver common messages consistently. As a result, application programmers can focus on the inherent complexity of a service and rely on atomic multicast to handle communication that scales (with the number of destinations) and tolerates failures.

Although research on efficient atomic multicast protocols is relatively mature [8], [9], [10], [11], to date all existing protocols target benign failures (e.g., crash failures) [12], [13], [14], [15]. In this paper, we introduce ByzCast, the first Byzantine Fault-Tolerant (BFT) atomic multicast protocol. Byzantine fault tolerance has become increasingly appealing as service providers can deploy their systems in increasingly inexpensive hardware (e.g., cloud environments) and new applications become more and more sensitive to malicious behavior (e.g., blockchain [16]).

ByzCast's design was motivated by two driving forces: (i) The desire to reuse existing BFT tools and libraries, instead of coming up with protocols that would require an implementation from scratch. (ii) The perception that the usefulness of atomic multicast lies in its ability to deliver scalable performance. On the one hand, much effort has been put into designing, implementing, debugging and performance-tuning BFT atomic broadcast protocols (i.e., a special case of atomic multicast in which messages always address the same set of destinations) [17], [18], [19], [20], [21]. We would like to build on these solutions and thereby shorten the development cycle of our BFT atomic multicast protocol. On the other hand, it would not be difficult to achieve the first goal above with a naive atomic multicast protocol that trivially relies on atomic broadcast. For example, one could use a fixed group of processes to order all multicast messages (using atomic broadcast) and then relay the ordered messages to their actual destinations. Instead, one should aim at atomic multicast protocols that are *genuine*, that is, only the message sender and the message destinations should communicate to order multicast messages [22]. A genuine atomic multicast is the foundation of scalable systems, since it does not depend on a fixed group of processes and does not involve processes unnecessarily.

ByzCast conciliates these goals with a compromise between reusability and scalability: the resulting protocol is more complex than the naive variant described above and *partially genuine*. ByzCast is partially genuine in that messages atomically multicast to a single group of processes only require coordination between the message sender and the destination group; messages addressed to multiple groups of processes, however, may involve processes that are not part of the destination (i.e., these processes help order the messages though). We motivate partially genuine atomic multicast protocols with the observation that when sharding a service state for performance, service providers strive to maximize the number of requests that can be served by a single shard alone.

2158-3927/18/$31.00 Â©2018 IEEE
DOI 10.1109/DSN.2018.00017

ByzCast is a hierarchical protocol. It uses an overlay tree where a node in the tree is a group of processes. Each group of processes runs an instance of atomic broadcast that encompasses the processes in the group. Hence, ordering messages multicast to a single group is easy enough: it suffices to use the atomic broadcast instance implemented by the destination group. Ordering messages that address multiple groups is trickier. First, it requires ordering such a message in the lowest common ancestor group of the message's destinations (in the worst case the root). Then, the message is successively ordered by the lower groups in the tree until it reaches the message's destination groups. The main invariant of ByzCast is that the lower groups in the tree preserve the order induced by the higher groups.

In addition to proposing a partially genuine atomic multicast protocol that builds on multiple instances of atomic broadcast (one instance per group of processes), we also consider the problem of building an efficient overlay tree. The structure of the overlay tree is mostly important for messages that address multiple groups. In its simplest form, one could have a two-level tree: any messages that address more than one destination would be first ordered by the root group and then by the destination groups, the leaves of the tree. In this simple tree, however, the root could become a performance bottleneck. More efficient solutions, based on more complex trees, are possible if one accounts for the workload when computing ByzCast's overlay tree. We frame this discussion as an optimization problem.

This paper makes the following contributions:

- We present a partially genuine atomic multicast protocol that builds on multiple instances of atomic broadcast, a problem that has been extensively studied and efficient libraries exist (e.g. [23], [24], [25], [18]).
- We define the problem of building an overlay tree as an optimization problem. Our optimization model takes into account the frequency of messages per destination and the performance of a group alone.
- We describe a prototype of ByzCast developed using BFT-SMaRt [18], a well-established library that implements BFT atomic broadcast.
- We provide a detailed experimental evaluation of ByzCast and compare it to a naive atomic multicast solution.

The rest of the paper is organized as follows. Section II introduces the system model and definitions. Section III presents ByzCast, its performance optimizer, and correctness proof. Section IV details our prototype. Section V describes our experimental evaluation. Section VI surveys related work and Section VII concludes the paper.

II. System Model and Definitions

In this section, we detail our system model (§II-A) and recall the definitions of atomic multicast (§II-B), atomic broadcast (§II-C), and state machine replication (§II-D).

A. Processes, groups, and communication

We consider a distributed system with an unbounded set of client processes $\mathcal{C} = \{c_1, c_2, ...\}$ and a bounded set of server processes $\mathcal{S} = \{p_1, ..., p_n\}$, where clients and servers are disjoint. Processes communicate by exchanging messages and do not have access to a shared memory or a global clock. The system is asynchronous: messages may experience arbitrarily large (but finite) delays and there is no bound on relative process speeds.

Client and server processes can be *correct* or *faulty*. A correct process follows its specification whilst a faulty process can present arbitrary (i.e., Byzantine) behavior. We define $\Gamma = \{g_1, ..., g_m\}$ as the set of server process groups in the system. Groups are disjoint, non-empty, and satisfy $\bigcup_{g \in \Gamma} g = \mathcal{S}$. We assume each group contains $3f + 1$ processes, where f is the maximum number of faulty server processes per group [26], [27].

We use cryptographic techniques for authentication, and digest calculation. We assume that adversaries (and Byzantine processes under their control) are computationally bound so that they are unable, with very high probability, to subvert the cryptographic techniques used. Adversaries can coordinate Byzantine processes and delay correct processes in order to cause the most damage to the system. Adversaries cannot, however, delay correct processes indefinitely.

B. Atomic Multicast

For every message m, $m.dst$ denotes the groups to which m is multicast. If $|m.dst| = 1$ we say that m is a *local* message; if $|m.dst| > 1$ we say that m is a *global* message.

A process atomically multicasts a message m by invoking primitive a-multicast(m) and delivers m with a-deliver(m). We define the relation $<$ on the set of messages correct processes a-deliver as follows: $m < m'$ iff there exists a correct process that a-delivers m before m'.

Atomic multicast satisfies the following properties [28]:

- *Validity*: If a correct process p a-multicasts a message m, then eventually all correct processes $q \in g$, where $g \in m.dst$, a-deliver m.
- *Agreement*: If a correct process p a-delivers a message m, then eventually all correct processes $q \in g$, where $g \in m.dst$, a-deliver m.
- *Integrity*: For any correct process p and any message m, p a-delivers m at most once, and only if $p \in g$, $g \in m.dst$, and m was previously a-multicast.
- *Prefix order*: For any two messages m and m' and any two correct processes p and q such that $p \in g$, $q \in h$ and $\{g, h\} \subseteq m.dst \cap m'.dst$, if p a-delivers m and q a-delivers m', then either p a-delivers m' before m or q a-delivers m before m'.
- *Acyclic order*: The relation $<$ is acyclic.

An atomic multicast algorithm $\mathcal{A}$ is *genuine* if and only if for any admissible run R of $\mathcal{A}$ and for any correct process p in R, if p sends or receives a message, then some message m is a-multicast, and either (a) p is the process that a-multicasts m or (b) $p \in g$ and $g \in m.dst$ [22].

C. Atomic Broadcast

Atomic broadcast is a special case of atomic multicast in which there is a single group of server processes. In this paper, we assume that each group implements FIFO atomic broadcast, which in addition to the properties presented above, also ensures the following property.

- *FIFO order*: If a correct process broadcasts a message m before it broadcasts a message m', no correct process delivers m' unless it has previously delivered m.

D. State machine replication

State machine replication is a well-established approach to fault tolerance [6], [7]. The idea is that by executing service requests deterministically in the same order, correct replicas will transition through the same sequence of state changes and produce the same output for every request. In a system with f Byzantine replicas, a client knows that a request was successfully executed when it receives $f+1$ identical responses from the servers. Atomic broadcast can be used to guarantee that replicas deliver requests in the same order.

With state machine replication, every server has a full copy of the service state. Several approaches have proposed to shard the service state and handle each shard as a replicated state machine (e.g., [29], [30], [31], [32]). Atomic multicast is a natural abstraction to order requests in a sharded replicated system. Requests that can be entirely executed within a shard are multicast to the required shard; requests that involve data in multiple shards must be consistently multicast to all target shards.

State machine replication provides linearizability, a consistency criteria. A system is linearizable if it satisfies the following requirements [5]: (i) It respects the real-time ordering of requests across all clients. There exists a real-time order among any two requests if one request finishes at a client before the other request starts at a client. (ii) It respects the semantics of the requests as defined in their sequential specification.

III. Byzantine Fault Tolerant Atomic Multicast

In this section, we explain the rationale behind the design of ByzCast (§III-A), present the protocol in detail (§III-B), show how to optimize ByzCast for different workloads (§III-C), and then argue about its correctness (§III-D).

A. Rationale

The design of ByzCast was guided by two high-level goals:

1) Building on existing solutions: Research on Byzantine Fault Tolerant agreement protocols is mature (see §VI). One of our main goals was to devise an atomic multicast protocol that could reuse existing BFT software, instead of designing a protocol that would require an implementation completely from scratch.

2) Striving for scalable protocols: Genuineness is the property that best captures scalability in atomic multicast. By requiring only the groups in the destination of a message to coordinate to order the message, a genuine atomic multicast protocol can scale with the number of groups while saving resources.

B. Protocol

For clarity, we describe a version of ByzCast that uses additional groups of servers to help order messages. Hereafter, we refer to the groups in $\Gamma = \{g_1, ..., g_m\}$ as *target groups* and the additional server groups in $\Lambda = \{h_1, ..., h_n\}$ as *auxiliary groups*. As with target groups, each auxiliary group has $3f+1$ processes, with at most f faulty processes.

Each group x in ByzCast (both target and auxiliary) implements a FIFO atomic broadcast. The atomic broadcast in group x is implemented by x's members and independent from the atomic broadcast of other groups. We distinguish between the primitives of atomic multicast, denoted as a-multicast and a-deliver, and the primitives of the atomic broadcast of group x, denoted as x-broadcast and x-deliver.

ByzCast arranges groups in a tree overlay where the leaves of the tree are target groups and the inner nodes of the tree are auxiliary groups. We define the *reach* of a group x, $reach(x)$, as the set of target groups that can be reached from x by walking down the tree. In Fig. 1 (a), $reach(h_1) = \{g_1, g_2, g_3, g_4\}$, $reach(h_2) = \{g_1, g_2\}$, and $reach(h_3) = \{g_3, g_4\}$. We denote the children of a group x in the tree as $children(x)$.

To a-multicast a message m to a set of target groups in $m.dst$ (see Algorithm 1), a process first x_0-broadcasts m in the *lowest common ancestor* group x_0 of (the groups in) $m.dst$, denoted $lca(m.dst)$.

Alg. 1 ByzCast

```
1:  Initialization
2:      T is an overlay tree with groups Γ ∪ Λ
3:      A-delivered ← ∅
4:  To a-multicast message m:
5:      x_0 ← lca(m.dst)          {lowest common ancestor of m.dst}
6:      x_0-broadcast(m)
7:  Each server process p in group x_k executes as follows:
8:      when x_k-deliver(m)
9:          if k = 0 or x_k-delivered m (f + 1) times then
10:             for each x_{k+1} ∈ children(x_k) such that
                        m.dst ∩ reach(x_{k+1}) ≠ ∅ do
11:             x_{k+1}-broadcast(m)
12:         if x_k ∈ m.dst and m ∉ A-delivered then
13:             a-deliver(m)
14:             A-delivered ← A-delivered ∪ {m}
```

When m is x_k-delivered by processes in x_k, each process x_{k+1}-broadcasts m in x_k's child group x_{k+1} if x_{k+1}'s reach intersects $m.dst$. This procedure continues until target groups in $m.dst$ x_k-deliver m, which triggers the a-deliver of m.

To account for Byzantine processes in group x_k, processes in x_{k+1} only handle m once they x_{k+1}-deliver m $f+1$ times. This ensures that m was x_{k+1}-broadcast by at least one correct process in x_k and, by inductive reasoning, m was a-multicast by a client (and not fabricated by a malicious server).

Intuitively, ByzCast atomic order is a consequence of two invariants:

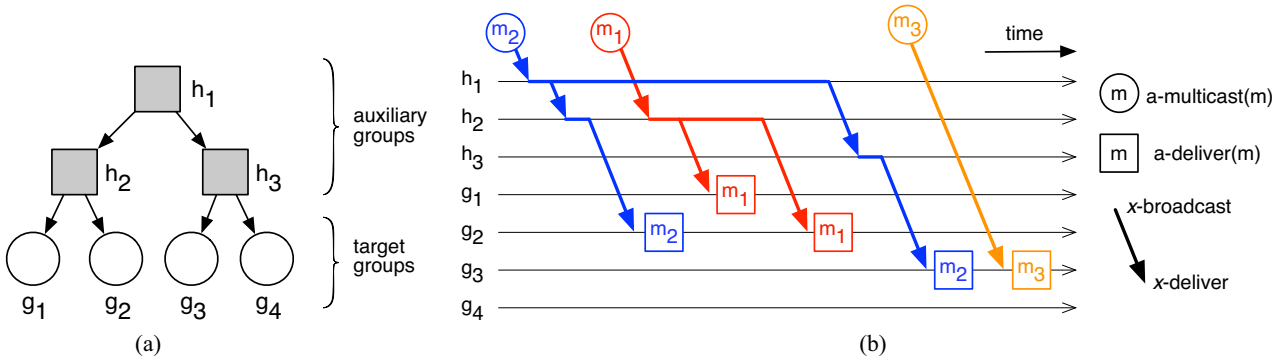


Fig. 1: (a) An overlay tree used in ByzCast with four target groups and three auxiliary groups. (b) An execution of ByzCast with three messages: m_1 is a-multicast to $\{g_1, g_2\}$, m_2 to $\{g_2, g_3\}$, and m_3 to g_3. For clarity, each group has one (correct) process.

1) Any two messages m and m' atomically multicast to common destinations are ordered by at least one inner group x_k in the tree.
2) If m is ordered before m' in x_k, then m is ordered before m' in any other group that orders both messages (thanks to the FIFO atomic broadcast used in each group).

We illustrate an execution of ByzCast in Fig. 1 (b) with messages m_1, m_2 and m_3 a-multicast to groups $\{g_1, g_2\}$, $\{g_2, g_3\}$, and $\{g_3\}$, respectively. Assuming the overlay tree shown in Fig. 1 (a), m_1 is first h_2-broadcast in group h_2. Upon h_2-delivering m_1, processes in h_2 atomically broadcast m_1 in g_1 and in g_2. Message m_2 is first h_1-broadcast, and then it continues down the tree until it is delivered by g_2 and g_3, its destination target groups. Message m_3 is g_3-broadcast in g_3 directly since it is addressed to a single group. The order between m_1 and m_2 is determined by their delivery order at h_2 since h_2 is the highest group to deliver both messages.

ByzCast is a partially genuine atomic multicast protocol. While messages addressed to a single group are ordered by processes in the destination group only, messages addressed to multiple groups may involve auxiliary groups. For example, in Fig. 1, the atomic multicast of m_1 (resp., m_2) involves h_2 (resp., h_1, h_2 and h_3), which is not a destination of m_1 (resp., m_2). Since m_3 involves a single destination group, only m_3's sender and g_3, m_3's destination, must coordinate to order the message. The performance of messages multicast to multiple groups largely depends on the overlay tree, as we discuss in the next section.

Finally, even though we described ByzCast with auxiliary groups as inner nodes of the tree, Algorithm 1 does not need this restriction: target groups can be inner nodes in the overlay tree, or we can have a tree that contains target groups only.

C. Optimizations

Laying out ByzCast overlay tree is an optimization problem with conflicting goals: on the one hand, we aim at short trees to reduce the latency of global messages; on the other hand, when laying out the tree, we must avoid overloading groups. For example, in Fig. 1, the height of the lowest common ancestor of m_1 and m_2 are two and three, respectively. A two-level tree where the four target groups descend directly from one auxiliary group would improve the latency of global messages. However, in a two-level tree all global messages must start at the root group, which could become a performance bottleneck.

We now formulate the problem of laying out an optimized ByzCast tree. The following parameters are input:

- Γ and Λ as already defined, and $\mathcal{N} = \Gamma \cup \Lambda$;
- $D \subseteq \mathcal{P}(\Gamma)$: all possible destinations of a message, where $\mathcal{P}(\Gamma)$ is the power set of Γ;
- $F(d)$: maximum load in messages per second multicast to destinations d in the workload, where $d \in D$; and
- $K(x)$: maximum performance in messages per second that group x can sustain, $\forall x \in \mathcal{N}$.

Given this input, the problem consists in finding the directed edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ of the optimized overlay tree $\mathcal{T} = (\mathcal{N}, \mathcal{E})$. To more precisely state the optimization function with constraints, we introduce additional definitions.

- $P(\mathcal{T}, d)$: the set of groups involved in a multicast to d (i.e., groups in the paths from $lca(d)$ to all groups in d);
- $H(\mathcal{T}, d)$: the height of the lowest common ancestor of groups in d;
- $T(\mathcal{T}, x) = \{d \mid d \in D \text{ and } x \in P(\mathcal{T}, d)\}$: set of destinations that involve group x; and
- $L(\mathcal{T}, x) = \sum_{d \in T(\mathcal{T}, x)} F(d)$: load imposed on group x.

Among the candidate overlay trees, respecting the above restrictions, we are interested in those that minimize the height of the various destinations.

$$\text{minimize} \sum_{d \in D} H(\mathcal{T}, d)$$

In addition to topological constraints, we have that the load imposed to each group respects its capacity.

$$\text{subject to } \forall x : L(\mathcal{T}, x) \leq K(x)$$

D. Correctness

In this section, we prove that ByzCast satisfies all the properties of atomic multicast (§II-B).

Lemma 1: For any message m atomically multicast to multiple groups, let group x_0 be the lowest common ancestor

of $m.dst$. For all $x_d \in m.dst$, if correct process p in x_0 x_0-delivers m, then all correct processes in the path $x_1, ..., x_d$, x_k-deliver m $(f+1)$ times, where $1 \leq k \leq d$.

PROOF: By induction. (Base step.) Since p x_0-delivers m, x_1 is a child of x_0, and $reach(x_1) \cap m.dst \neq \emptyset$, p x_1-broadcasts m. The claim follows from the validity of atomic broadcast and the fact that there are $2f+1$ correct processes in x_0. (Inductive step.) Assume each correct process r in x_k x_k-deliver m at least $(f+1)$ times. From Alg.1, and the fact that x_{k+1} is a child of x_k and $reach(x_{k+1}) \cap m.dst \neq \emptyset$, r x_{k+1}-broadcasts m. From the validity of atomic broadcast and the fact that there are $2f+1$ correct processes in x_k, every correct process in x_{k+1} x_{k+1}-delivers m. □

Lemma 2: For any atomically multicast message m, let group x_0 be the lowest common ancestor of $m.dst$. For all $x_d \in m.dst$, if correct process p in x_d x_d-delivers m, then all correct processes in the path $x_0, ..., x_d$, x_k-deliver m, where $0 \leq k \leq d$.

PROOF: By backwards induction. (*Base step.*) The case for $k = d$ follows directly from agreement of atomic broadcast in group x_d. (*Inductive step.*) Assume that every correct process $r \in x_k$ x_k-delivers m. We show that correct processes in x_{k-1} x_{k-1}-deliver m. From Alg.1, r x_k-delivered m $(f+1)$ times. From integrity of atomic broadcast in x_k, at least one correct process s in x_{k-1} x_k-broadcasts m. Therefore, s x_{k-1}-delivers m, and from agreement of atomic broadcast in x_{k-1} all correct processes x_{k-1}-deliver m. □

Proposition 1: (*Validity*) If a correct process p a-multicasts a message m, then eventually all correct processes $q \in g$, where $g \in m.dst$, a-deliver m.

PROOF: Let group x_0 be the lowest common ancestor of $m.dst$ and x_d a group in $m.dst$. From Alg.1, p x_0-broadcasts m and from validity of atomic broadcast, all correct processes in x_0 x_0-deliver m. From Lemma 1, all correct processes in x_d, x_d-deliver m $(f+1)$ times. Hence, every correct process in x_d a-delivers m. □

Proposition 2: (*Agreement*) If a correct process p in group x_d a-delivers a message m, then eventually all correct processes $q \in g$, where $g \in m.dst$, a-deliver m.

PROOF: From Lemma 2, all correct processes in x_0, x_0-deliver m. Thus, from Lemma 1, all $x_d \in m.dst$ x_d-deliver m $(f+1)$ times. It follows from Alg.1 that all $q \in x_d$ a-deliver m. □

Proposition 3: (*Integrity*) For any correct process p and any message m, p a-delivers m at most once, and only if $p \in g$, $g \in m.dst$, and m was previously a-multicast.

PROOF: From Alg.1, it follows immediately that a correct process $p \in g$ a-delivers m at most once, only if $g \in m.dst$ and m is a-multicast. □

Lemma 3: If m and m' are two messages atomically multicast to one or more destination groups in common, then $lca(m) \in subtree(m')$ or $lca(m') \in subtree(m)$.

PROOF: Assume group x is a common destination in m and m' (i.e., $x \in m.dst \cap m'.dst$). Let $path(x)$ be the sequence of groups in the overlay tree $\mathcal{T}$ from the root until x. From Alg.1, in order to reach x, $lca(m)$ (resp., $lca(m')$) must be a group in $path(x)$. Without loss of generality, assume that $lca(m)$ is higher than $lca(m')$ or at the same height as $lca(m')$. Then, $lca(m') \in subtree(m)$, which concludes the lemma. □

Lemma 4: If a correct process in group x_0 x_0-delivers m before m', then for every ancestor group x_d of x_0, where $x_d \in m.dst \cap m'.dst$, every correct process in x_d x_d-delivers m before m'.

PROOF: By induction on the path $x_0, ..., x_k, ..., x_d$. (*Base step.*) Trivially from the properties of atomic broadcast in group x_0. (*Inductive step.*) Let $p \in x_k$ x_k-deliver m before m'. Thus, p x_{k+1}-broadcasts m before m' and from the FIFO guarantee of atomic broadcast in x_{k+1}, every correct process $q \in x_{k+1}$ x_{k+1}-delivers m before m'. □

Proposition 4: (*Prefix order*) For any two messages m and m' and any two correct processes p and q such that $p \in g$, $q \in h$ and $\{g, h\} \subseteq m.dst \cap m'.dst$, if p a-delivers m and q a-delivers m', then either p a-delivers m' before m or q a-delivers m before m'.

PROOF: The proposition holds trivially if p and q are in the same group, so assume that $g \neq h$. From Lemma 3, and without loss of generality, assume that $lca(m') \in subtree(m)$. Thus, $lca(m')$ will order m and m'. From Lemma 4, both p and q a-deliver m and m' in the same order as $lca(m')$. □

Proposition 5: (*Acyclic order*) The relation $<$ is acyclic.

PROOF (SKETCH): For a contradiction, assume there is an execution of ByzCast that results in a cycle $m_0 < ... < m_d < m_0$. Since all correct processes in the same group a-deliver messages in the same order, the cycle must involve messages a-multicast to multiple groups. Let x be the highest lowest common ancestor of all messages in the cycle. We define $subtree(x, 1)$, $subtree(x, 2)$, ... as the subtrees of group x in $\mathcal{T}$. Since the cycle involves groups in the subtree of x, there must exist messages m and m' such that (a) m is a-delivered before m' in groups in $subtree(x, i)$ and (b) m' is a-delivered before m in groups in $subtree(x, j), i \neq j$. From Lemma 4, item (a) implies that processes in x x-deliver m and then m', and item (b) implies that processes in x x-deliver m' and then m, a contradiction. □

IV. IMPLEMENTATION

We implemented ByzCast on top of BFT-SMaRt, a well-known library for BFT replication [18]. This library has been used in many academic projects and a few recent blockchain systems (e.g., [16], [33]).

BFT-SMaRt message ordering is implemented through the Mod-SMaRt algorithm [34], which uses the Byzantine-variant of Paxos described in [35] to establish consensus on the i-th (batch of) operation(s) to be processed by the replicated state machine. The leader starts a consensus instance every

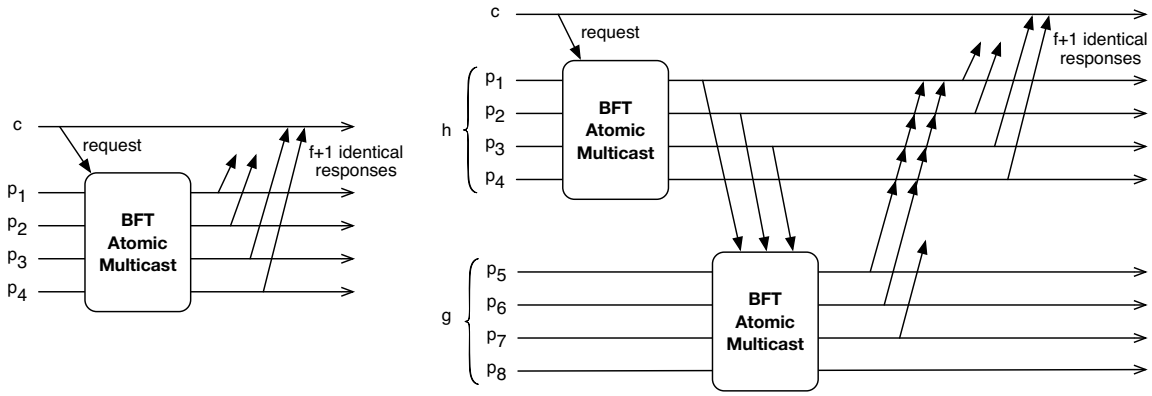


Fig. 2: Executions of ByzCast with (a) a local message and (b) a global message. Each group has four processes, one of which may be Byzantine. For clarity, the execution of ByzCast shows a single target group only.

time there are pending client requests to be processed and there are no consensus being executed. Consensus follows a message pattern similar to PBFT [17]: the leader *proposes* a batch of messages to be processed, the replicas validate this proposal by *writing* the proposal in the other replicas; the replicas *accept* the proposal if a Byzantine quorum of $n - f$ replicas perform the write. When a replica learns that $n - f$ replicas accepted the proposal, it executes the operation and sends replies to the clients. In case of leader failure or network asynchrony, a new leader is elected. BFT-SMaRt also implements protocols for replica recovering (i.e., state transfer), and group reconfiguration [18].

In ByzCast, each group (either target or auxiliary) corresponds to a BFT replicated state machine. Each replica in auxiliary groups connects to all the replicas in the next level. We implemented two overlay trees. A three-level tree, as the one presented in Fig. 1 and a two-level tree that uses a single auxiliary group to order global messages.

Replicas only process messages from a higher-level group when they (FIFO) a-deliver them $f + 1$ times. Target groups execute a-delivered messages and reply either to clients or to auxiliary groups whether the message is local or global. Both clients and auxiliary groups wait for $f + 1$ correct replies. Fig. 2 depicts the described logic in executions of a request from a client in a local and a global message. Except for client requests, which are single messages, all messages exchanges between groups need $f + 1$ equal responses before they can be processed. Even though multiple processes in group invoke the broadcast of a message in another group, thanks to BFT-SMaRt's batching optimization, it is likely that all such invocations are ordered in a single instance of consensus.

Clients run in a closed loop (i.e., only send a new message after the previous message reply) and forward messages to every replica in the lowest common ancestor group of the message. ByzCast was implemented in Java and the source code is publicly available.[1]

[1] https://github.com/tarcisiocjr/byzcast.

V. Performance Evaluation

In this section, we describe the main motivations that guided the design of our experiments (§V-A), detail the environments in which we conducted the experiments (§V-B), and then present and discuss the results (§V-C–V-H).

A. Evaluation rationale

In the following, we explain our choices for environments, benchmarks, and protocols.

1) Environments: We consider a local-area network (LAN) and a wide-area network (WAN). The LAN provides a controlled environment, where experiments run in isolation; the WAN represents a more challenging setting.

2) Benchmarks: We use a microbenchmark with 64-byte messages to evaluate particular scenarios in isolation. We vary the number of groups (up to 8 groups, the largest configuration we can accommodate in our local infrastructure) and the number of message destinations. We assess two layouts for the ByzCast tree: a 2-level and a 3-level tree. We consider executions with a single client to understand the performance of ByzCast without queueing effects, and with multiple clients to evaluate our solution under stress. Finally, we consider workloads with and without locality (i.e., skewed access).

3) Protocols: We compare ByzCast to BFT-SMaRt and to a non-genuine 2-level atomic multicast protocol, which we call Baseline. BFT-SMaRt uses a single group and provides a reference to the performance of ByzCast with local messages. The Baseline protocol has one auxiliary group that orders all messages regardless of the message destination. After the message is ordered, it is forwarded to its destinations. Each process in the target group waits until it receives the message from $f + 1$ processes in the auxiliary group. Although the non-genuine protocol does not scale, it provides a performance reference for global messages.

B. Environments and configuration

We now detail the environments where we performed the evaluation of the three protocols.

1) Local-area network (LAN): This environment consisted of a set replica nodes with an eight-core Intel Xeon L5420 processor working at 2.5GHz, 8GB of memory, SATA SSD disks, and 1Gbps ethernet card; and clients nodes with a four-core AMD Opteron 2212 processor at 2.0GHz, 4GB of memory, and 1Gbps ethernet card. Each node runs CentOS 7.1 64 bits. The RTT (round-trip time) between nodes in the cluster is around 0.1ms.

2) Wide-area network (WAN): We used Amazon EC2, a public wide-area network. All nodes are c4.xlarge instances, with 4 vCPUs and 7.5GB of memory. We allocated nodes in four regions: California (R1), North Virginia (R2), Frankfurt (R3) and Tokyo (R4). Table I summarizes the latency between pairs of regions in milliseconds.

	EU	CA	VA	JP
CA	165	–	70	112
VA	88	70	–	175
JP	239	112	175	–

TABLE I: Latencies within Amazon EC2 infrastructure.

3) Configuration: In all experiments, groups contain four processes, each process running in a different node. The number of groups depends on the tree layout. In the 2-level tree we have from 2 to 8 target groups and 1 auxiliary for global messages. In the 3-level tree we fix the number of target groups to 4 and the number of auxiliary groups to 3, as depicted in Fig. 1. In the WAN setup, we distribute clients along all the regions and deploy each process of a group in a different region. Consequently, the system can tolerate the failure of a whole region.

C. Overlay tree versus workload

We start by assessing how the workload and the performance of groups affect the overlay tree. We consider a system with four target groups and up to three auxiliary groups subject to two workloads. In both workloads we assume global messages only since local messages are multicast directly to target groups and do not affect the tree layout. In the *uniform workload*, clients multicast messages to two groups and all combinations of destinations have an equal probability of being chosen. In the *skewed workload*, clients multicast messages to either groups $\{g_1, g_2\}$ or to $\{g_3, g_4\}$. Moreover, we inject higher load in the skewed workload. Table II details the two workloads. Based on the experiments reported in §V-D, an auxiliary group can sustain approximately 9500 messages/sec (i.e., $K(h_i) = 9500$ m/s).

Table III shows outcomes for the two workloads with two-level ($\mathcal{T}_2$) and three-level ($\mathcal{T}_3$) trees (for the three-level tree depicted in Fig. 1). For the uniform workload, a two-level tree is the best option since the root can sustain the load (i.e., $L_u(\mathcal{T}_2, h_1) < K(h_1)$) and the sum of heights is lower than in the three-level tree (12 instead of 16). For the skewed workload, a two-level tree would impose too high a load on the root (i.e., $L_s(\mathcal{T}_2, h_1) > K(h_1)$) and therefore it is not a viable solution. In this case, in a three-level tree the traffic is divided among the two branches of the tree (h_2 and h_3).

Uniform workload	
$D_u = \{\{g_i, g_j\} \mid 1 \leq i, j \leq 4 \wedge i \neq j\}$	$\forall d \in D_u : F_u(d) = 1200$ m/s
Skewed workload	
$D_s = \{\{g_1, g_2\}, \{g_3, g_4\}\}$	$\forall d \in D_s : F_s(d) = 9000$ m/s

TABLE II: Uniform and skewed workloads.

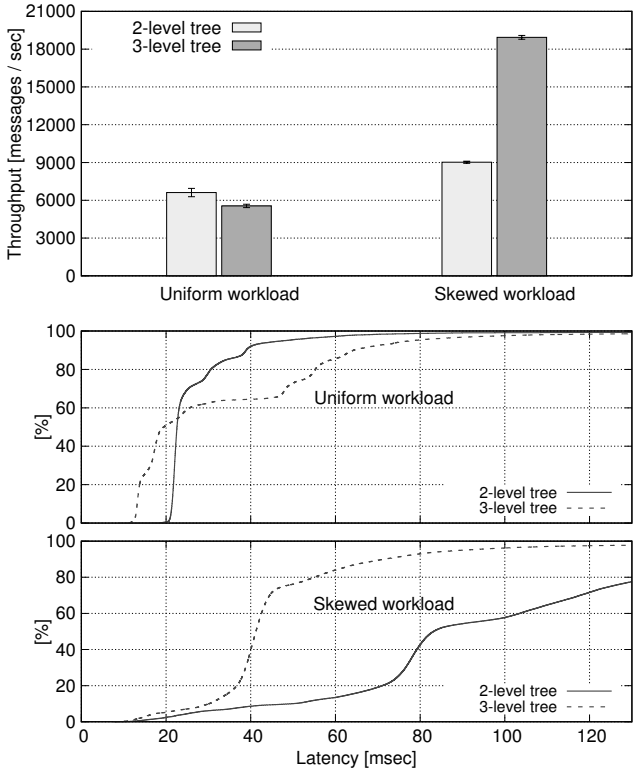


Fig. 3: ByzCast global messages throughput and latency CDF with 2-level and 3-level trees. Whiskers show 95% confidence interval.

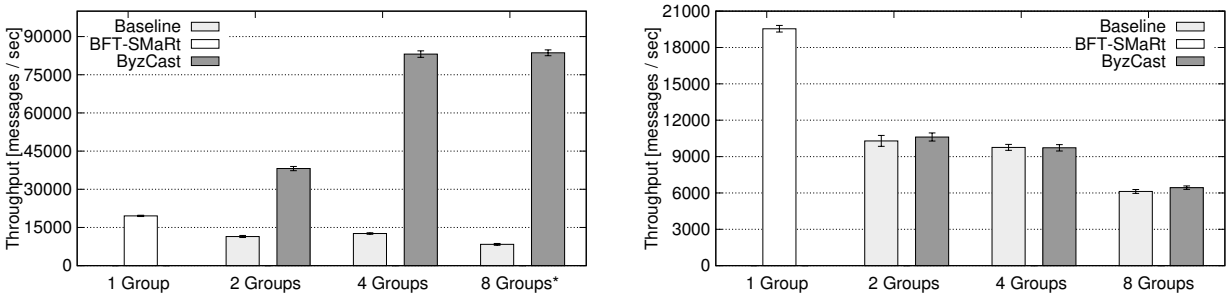
Fig. 3 exhibits the experimental results in terms of throughput and latency Cumulative Distribution Function (CDF) for each scenario. For the uniform workload, the average latency with a two-level tree is lower than with a three-level tree, although about 55% of messages have lower latency. This happens because the three-level tree distributes the load more uniformly among inner groups. In the skewed workload, the high load on the root of the two-level tree leads to much higher latencies than the three-level tree. The experiments presented next (both LAN and WAN) use the 2-level tree.

D. Scalability of ByzCast in LAN

This experiment assesses the performance of ByzCast and compares it to BFT-SMaRt (using a single group) and to Baseline, a non-genuine atomic multicast approach. Fig. 4(a) shows the throughput in messages per second versus the number of groups, when 200 clients per group multicast local messages only (except for the 8-group setup where there are 100 clients per group since we do not have enough client nodes to deploy

Uniform workload			
$T_u(\mathcal{T}_2, h_1) = D_u$	$L_u(\mathcal{T}_2, h_1) = 7200$ m/s	$\sum_{d \in D_u} H(\mathcal{T}_2, d) = 12$	Best choice (lowest heights)
$T_u(\mathcal{T}_3, h_1) = D_u \setminus \{\{g_1, g_2\}, \{g_3, g_4\}\}$ $T_u(\mathcal{T}_3, h_2) = D_u \setminus \{\{g_3, g_4\}\}$ $T_u(\mathcal{T}_3, h_3) = D_u \setminus \{\{g_1, g_2\}\}$	$L_u(\mathcal{T}_3, h_1) = 4800$ m/s $L_u(\mathcal{T}_3, h_2) = 6000$ m/s $L_u(\mathcal{T}_3, h_3) = 6000$ m/s	$\sum_{d \in D_u} H(\mathcal{T}_3, d) = 16$	Poor choice
Skewed workload			
$T_s(\mathcal{T}_2, h_1) = D_s$	$L_s(\mathcal{T}_2, h_1) = 18000$ m/s	$\sum_{d \in D_s} H(\mathcal{T}_2, d) = 4$	Not viable (load exceeds capacity)
$T_s(\mathcal{T}_3, h_1) = \emptyset$ $T_s(\mathcal{T}_3, h_2) = \{\{g_1, g_2\}\}$ $T_s(\mathcal{T}_3, h_3) = \{\{g_3, g_4\}\}$	$L_s(\mathcal{T}_3, h_1) = 0$ m/s $L_s(\mathcal{T}_3, h_2) = 9000$ m/s $L_s(\mathcal{T}_3, h_3) = 9000$ m/s	$\sum_{d \in D_s} H(\mathcal{T}_3, d) = 4$	Best choice

TABLE III: Optimization model outcomes for uniform and skewed workloads.



(a) Local messages.

(b) Global messages.

Fig. 4: Throughput in a LAN. Whiskers show 95% confidence interval.

200 clients per group without saturating the nodes). The results show that the genuineness of ByzCast with respect to local messages pays off. The throughput scales linearly with the number of groups with respect to BFT-SMaRt (single group), delivering more than 83000 messages/sec with 4 groups and 200 clients per group and the same with 8 groups and 100 clients per group. This happens because, for single-group messages, ByzCast only involves the sender of a message and the destination target group. Since a single group must order all the messages with the Baseline protocol, it becomes nearly saturated with 400 clients. Thus, the performance with four groups is only slightly higher than with two groups, from 11000 to 12000 messages/sec), and even smaller with eight groups. Fig. 4(b) shows that ByzCast's throughput when all the clients multicast global messages only is at most half the throughput of BFT-SMaRt: 9700 messages/sec against 19500 messages/sec in the best case. Differently from BFT-SMaRt, a global message in ByzCast has to be ordered by both the auxiliary group and the target groups, impacting the message latency and the overall throughput. The same observation holds for the Baseline protocol, which behaves similarly to ByzCast.

E. Throughput versus latency in LAN

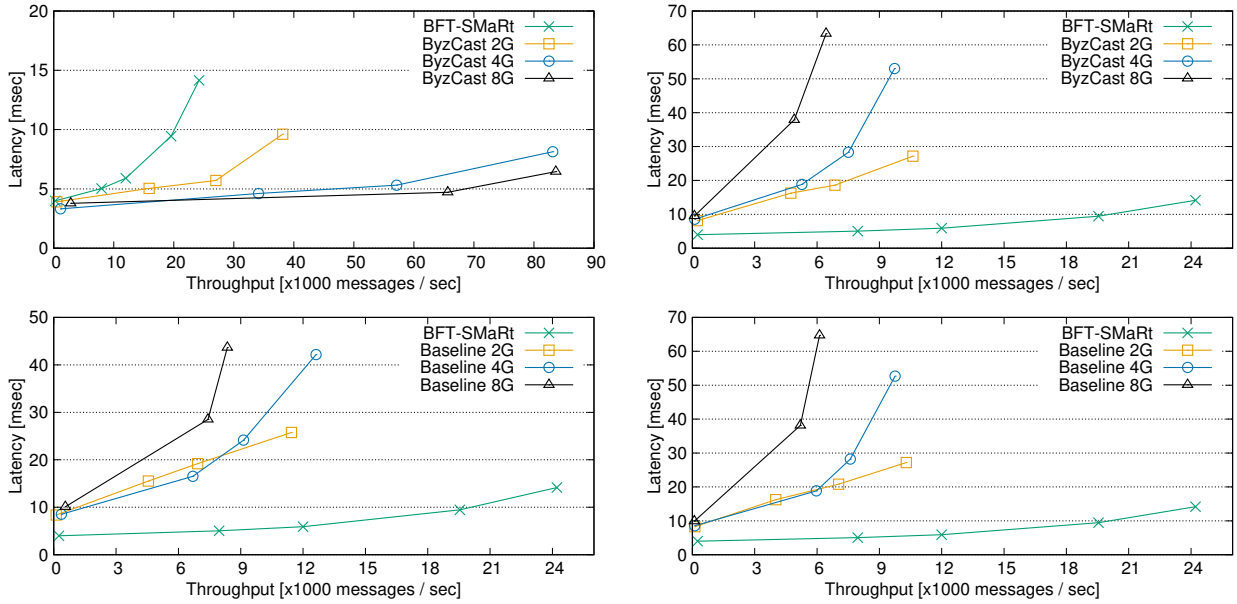
In Fig. 5(a) we can observe how the mean latency behaves as the number of clients increase. ByzCast (top) is at least twice as fast and has half the Baseline's latency even with only 2 groups. In executions where all request are global messages, even for small number of clients, BFT-SMaRt has always the best performance, as depicted in Fig. 5(b). This results reinforces the observation that an atomic broadcast (BFT-SMaRt) is preferable over an atomic multicast when most messages are global [36]. ByzCast and Baseline for 2, 4 and 8 groups perform very alike and the latency saturates with less than half BFT-SMaRt's throughput.

F. Latency without contention in LAN

The next experiments assess latency with a single client. This setup aims to check how the protocols perform in the absence of contention or queuing effects. We consider configurations with an increasing number of groups with both local and global messages. We can see in Fig. 6 that regarding local messages ByzCast performs as well as BFT-SMaRt no matter the number of groups, with latency around 4 msec. The fact that groups do not interact with each other when ordering local messages guarantees this expected latency. Global messages have twice the latency of local messages in ByzCast because they go through the auxiliary group before reaching the target groups. Besides, global messages latency increases slightly as we add more target groups as replicas in the auxiliary group need to perform multiple broadcasts to all the groups in message destination.

G. Performance with mixed workload in LAN

The last experiment in LAN assesses the performance of ByzCast with both local and global messages. In a 2-level overlay tree with 4 target groups, 160 equally distributed clients multicast local and global messages in a proportion of 10:1. Fig. 7 shows the latency CDF for both Baseline and ByzCast. Since in the Baseline protocol (Fig. 7(a)) all messages are ordered in the same auxiliary group before reaching the target group(s), the latency for both local and global messages



(a) Local messages: ByzCast (top) and Baseline (bottom).

(b) Global messages: ByzCast (top) and Baseline (bottom).

Fig. 5: Throughput vs. latency in a LAN.

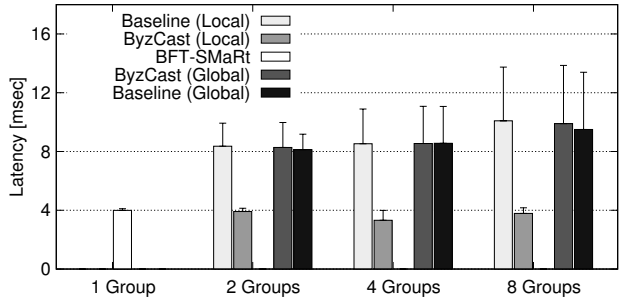


Fig. 6: Single-client latency in a LAN. Bars show median latency and whiskers show 95-th percentile.

are similar. ByzCast, on the contrary, is genuine for local messages, which have a considerably smaller latency up to the 99.5-th percentile, as exhibited in Fig. 7(b). For global messages, ByzCast and Baseline have similar performance. It is worth noticing that in ByzCast local messages do not suffer from the "convoy effect", a phenomenon in which the slower ordering of global messages can impact the latency of local ones [37]. In fact, the local-message latency CDF for ByzCast with 10% of global messages is very similar to the latency for 100% local messages.

H. Latency without contention in WAN

The first experiment in WAN measures the latency of ByzCast without any queuing effect or resources overload. A single client from each region multicasts local and global messages in a closed loop. The conclusions, shown in Fig. 8, are similar to those we drew in a LAN. ByzCast has latency as good as a single group (BFT-SMaRt) for local messages and twice the value for global ones. In ByzCast, clients multicast global messages via an auxiliary group that totally orders all messages before broadcasting them to target groups, what explains the doubled latency. The Baseline protocol pays this double ordering for every message.

I. Performance with mixed workload in WAN

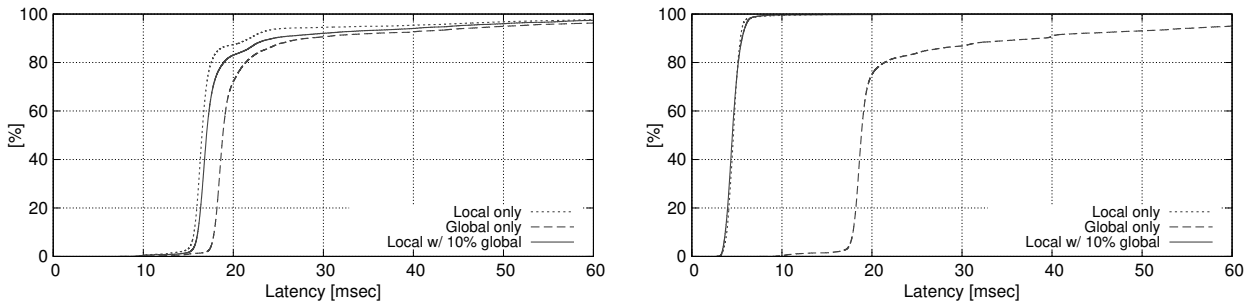
The last experiment evaluates ByzCast with a mix of local and global messages in a proportion of 10:1, which we believe would represent a more realistic workload. The setup comprehends 4 target groups, 1 auxiliary group to order global messages, and 40 clients per target group equally distributed among the 4 geographical regions. The results presented in Fig. 9 shows that ByzCast is 2x to 3x faster than the Baseline protocol in terms of throughput. Fig. 10 shows the latency CDF for global and local messages. As expected, ByzCast has local latency 2x to 4x smaller than the values for the Baseline protocol. Regarding global messages, both protocols behave similarly as exposed by previous experiments in LAN and WAN. The latency CDF also confirms that ByzCast does not suffer from the convoy effect, as the local latency is stable even in the presence of global messages.

VI. Related Work

ByzCast is at the intersection of two topics: atomic multicast (§VI-A) and BFT protocols (§VI-B).

A. Atomic Multicast

Several multicast and broadcast algorithms have been proposed [38]. Moreover, many systems use "ad hoc" ordering



(a) Baseline (b) ByzCast

Fig. 7: Latency CDF with 10% of global messages.

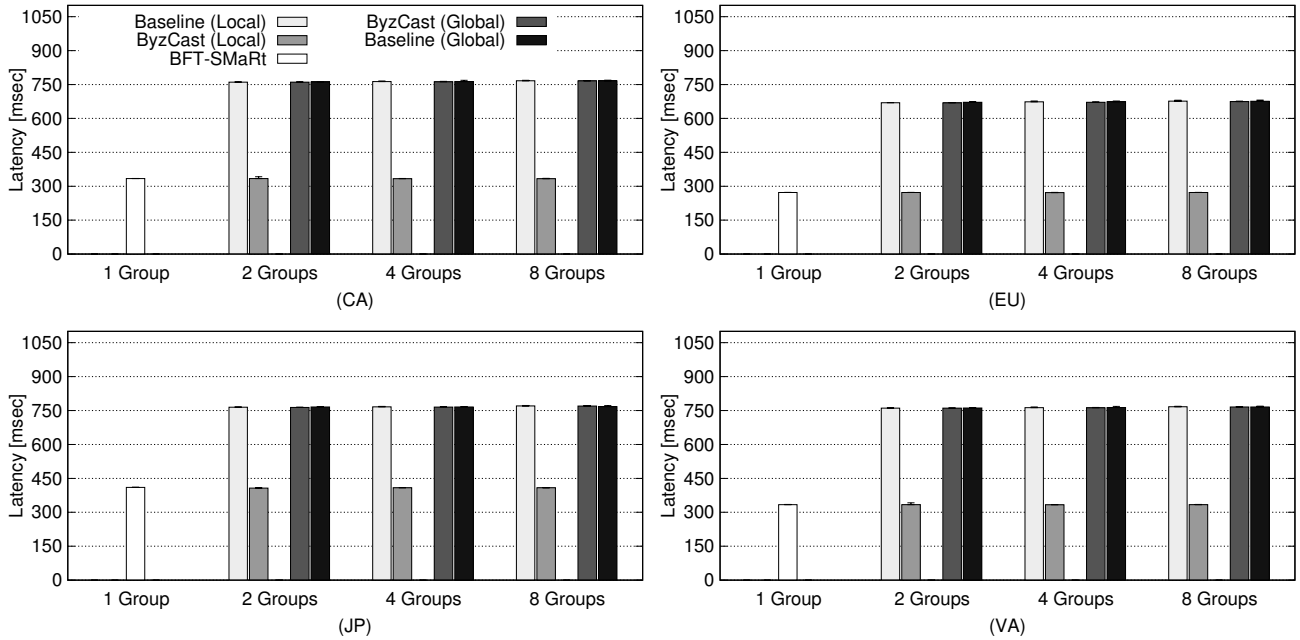


Fig. 8: Latency with single client in WAN. Bars show median latency and wihiskers represent the 95-th percentile.

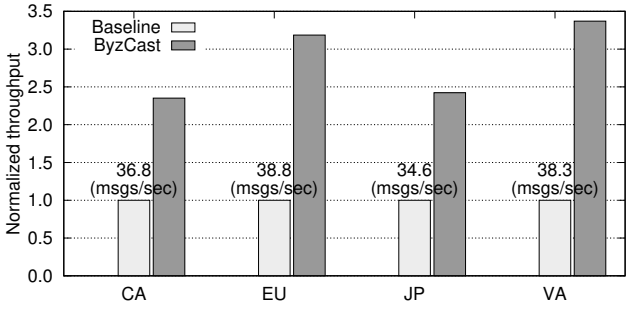
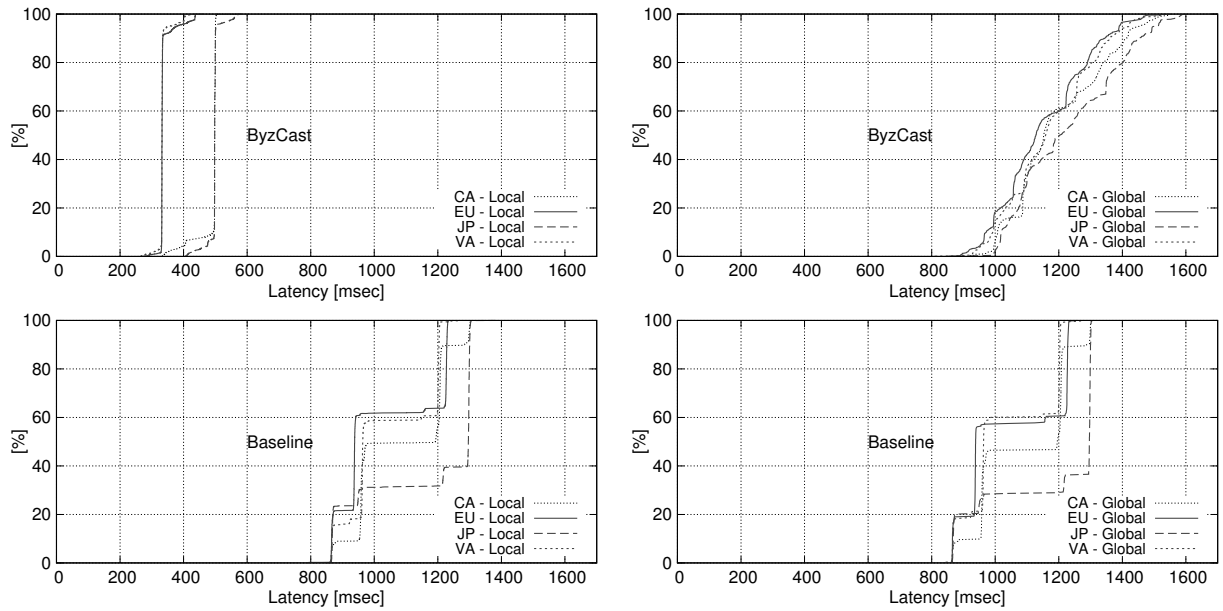


Fig. 9: Normalized throughput with mixed workload in a WAN.

protocols that do not implement all the properties of atomic multicast (e.g., [29], [39], [40]). We focus next on atomic multicast algorithms that tolerate benign failures, since no atomic multicast algorithm exists for Byzantine failures.

Existing atomic multicast algorithms fall into one of three categories: *timestamp-based*, *round-based*, and *ring-based*. Algorithms based on timestamps (i.e., [8], [9], [15], [36]) are genuine and variations of an early atomic multicast algorithm [41], designed for failure-free systems. In these algorithms, processes assign timestamps to messages, ensure that destinations agree on the final timestamp assigned to each message, and deliver messages following this timestamp order. The algorithm in [9] ensures another property besides genuineness called *message-minimality*. This property states that the messages of the algorithm have a size proportional to the number of destination groups of the multicast message, and not to the total number of processes. Although ByzCast is not genuine with respect to global messages, it satisfies this



(a) Local messages.

(b) Global messages.

Fig. 10: Latency CDF with 40 clients per group and 10% of global messages.

property for local messages which can be delivered as fast as the underlying atomic broadcast algorithm.

In round-based algorithms, processes execute an unbounded sequence of rounds and agree on messages delivered at the end of each round. A round-based atomic multicast algorithm that can deliver messages in 4δ is presented in [36]. Differently from ByzCast, this algorithm may penalize local messages, as they may be slowed down by global messages.

Ring-based algorithms propagate messages along a predefined ring overlay and ensure atomic multicast properties by relying on this topology. An atomic multicast algorithm in this category is proposed in [10], where consensus is run among the members of each group. The time complexity of this algorithm is proportional to the number of destination groups. Multi-Ring Paxos [12], Spread [11], [14], and Ridge [13] are ring-based non-genuine atomic multicast protocols. On the one hand, to deliver a message m, they require communication with processes outside of the destination groups of m and local messages may also suffer from convoy effect. On the other hand, these protocols do not require disjoint groups.

B. Scalable BFT

Despite the large amount of work on BFT replication in the last two decades (e.g., [17], [18], [19], [20], [21], [42], [43], [44], [45], [46]), the scalability of BFT protocols is still a relatively unexplored topic, which we discuss in this section.

A common observation of BFT protocols is that their performance degrades significantly as the number of faults tolerated increase [43]. This lack of *fault-scalability* comes mostly from the all-to-all communication used in these protocols, which implies in a quadratic amount of messages. This limitation can be mitigated either by using protocols with linear message pattern [42], [43], [44], by using protocols with a smaller ratio between n and f [45], [46], or by exploring erasure codes and large message batches [21]. Independently on the trade-offs explored by these protocols, all of them lose performance as the number of replicas increase, contrary to ByzCast.

There are few BFT protocols that target wide-area networks [19], [20]. These protocols tend to use more replicas to decrease the relative quorum size or the distance between replicas in the quorums. Similarly to the scalable protocols described before, the performance of these protocols tends to decrease with the number of replicas.

The natural way of scaling replicated systems is sharding the state in multiple replica groups and running ordering protocols only in these groups. To the best of our knowledge, there are only three works that consider partitionable replication for BFT systems. Augustus [47] and Callinicos [48] introduces protocols for executing transactions in multiple shards of a key-value store implemented on top of multiple BFT groups. A recent work by Nogueira et al. [49] introduces protocols for splitting and merging replica groups in BFT-SMaRt, without discussing ways to disseminate messages to more than one of these groups with Byzantine failures. ByzCast complements these works by providing a protocol for disseminating requests on multiple partitions, enabling thus the efficient support for services that require multi-partition operations.

VII. Conclusion

Atomic multicast is a fundamental communication abstraction in the design of scalable and highly available strongly consistent distributed systems. This paper proposes ByzCast,

the first Byzantine Fault-Tolerant atomic multicast, designed to build on top of existing BFT abstractions. ByzCast is partially genuine, i.e., it scales linearly with the number of groups, for messages adressed to a single group. In addition to introducing a novel atomic multicast algorithm, we also assessed its performance in two different environments. The results show that ByzCast outperforms BFT-SMaRt in most cases, as well as a non-genuine BFT atomic multicast protocol.

ACKNOWLEDGEMENTS

We thank the reviewers for the constructive suggestions. This work is supported in part by the Swiss Government Excellence Scholarships, Hasler Foundation, CNPq (GDE Project 204558/2014-0), CAPES (PVE Project 88887.124751/2014-00), and FCT through projects LaSIGE (UID/CEC/00408/2013) and IRCoC (PTDC/EEI-SCR/6970/2014).

REFERENCES

[1] J. Baker, C. Bond, J. Corbett, J. J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore: Providing scalable, highly available storage for interactive services," in *CIDR*, 2011.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. on Computer Systems*, vol. 26, no. 2, 2008.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *SOSP*, 2007.

[4] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in *SOSP*, 2011.

[5] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *Trans. on Programming Languages and Systems*, vol. 12, pp. 463–492, July 1990.

[6] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *CACM*, vol. 21, pp. 558–565, July 1978.

[7] F. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, vol. 22, pp. 299–319, Dec. 1990.

[8] U. Fritzke, P. Ingels, A. Mostéfaoui, and M. Raynal, "Fault-tolerant total order multicast to asynchronous groups," in *SRDS*, 1998.

[9] L. Rodrigues, R. Guerraoui, and A. Schiper, "Scalable atomic multicast," in *IC3N*, 1998.

[10] C. Delporte-Gallet and H. Fauconnier, "Fault-tolerant genuine atomic multicast to multiple groups," in *OPODIS*, 2000.

[11] D. A. Agarwal, L. E. Moser, P. M. Melliar-Smith, and R. K. Budhia, "The totem multiple-ring ordering and topology maintenance protocol," *ACM Trans. on Computer Systems*, vol. 16, pp. 93–132, May 1998.

[12] P. J. Marandi, M. Primi, and F. Pedone, "Multi-ring paxos," in *DSN*, 2012.

[13] E. Bezerra, D. Cason, and F. Pedone, "Ridge: high-throughput, low-latency atomic multicast," in *SRDS*, 2015.

[14] A. Babay and Y. Amir, "Fast total ordering for modern data centers," in *ICDCS*, 2016.

[15] P. Coelho, N. Schiper, and F. Pedone, "Fast atomic multicast," in *DSN*, 2017.

[16] C. Cachin and M. Vukolic, "Blockchain consensus protocol in the wild (invited paper)," in *DISC*, 2017.

[17] M. Castro and B. Liskov, "Practical Byzantine fault-tolerance and proactive recovery," *ACM Trans. on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.

[18] A. Bessani, J. Sousa, and E. Alchieri, "State machine replication for the masses with BFT-SMaRt," in *DSN*, 2014.

[19] J. Sousa and A. Bessani, "Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines," in *SRDS*, 2015.

[20] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, and D. Z. Josh Olsen, "STEWARD: Scaling Byzantine fault-tolerant replication to wide area networks," *IEEE Trans. on Dependable and Secure Computing*, vol. 7, no. 1, 2010.

[21] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *CCS*, 2016.

[22] R. Guerraoui and A. Schiper, "Genuine atomic multicast in asynchronous distributed systems," *Theoretical Computer Science*, vol. 254, no. 1-2, pp. 297–316, 2001.

[23] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *SOSP*, 2013.

[24] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm.," in *USENIX ATC*, 2014.

[25] A. Turcu, S. Peluso, R. Palmieri, and B. Ravindran, "Be general and don't give up consistency in geo-replicated transactional systems," in *OPODIS*, 2014.

[26] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[27] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[28] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," tech. rep., Cornell University, 1994.

[29] J. C. C. *et al.*, "Spanner: Google's globally distributed database," in *OSDI*, 2012.

[30] C. E. Bezerra, F. Pedone, and R. van Renesse, "Scalable state-machine replication," in *DSN*, 2014.

[31] B. Li, W. Xu, M. Z. Abid, T. Distler, and R. Kapitza, "SAREK: optimistic parallel ordering in Byzantine fault tolerance," in *EDCC*, 2016.

[32] L. L. Hoang, C. E. B. Bezerra, and F. Pedone, "Dynamic scalable state machine replication," in *DSN*, 2016.

[33] J. Sousa, A. Bessani, and M. Vukolic, "A Byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform," in *DSN*, 2018.

[34] J. Sousa and A. Bessani, "From Byzantine consensus to BFT state machine replication: A latency-optimal transformation," in *EDCC*, 2012.

[35] C. Cachin, "Yet another visit to Paxos," Tech. Rep. RZ 3754, IBM Research Zurich, 2009.

[36] N. Schiper and F. Pedone, "On the inherent cost of atomic broadcast and multicast in wide area networks," in *ICDCN*, 2008.

[37] N. Schiper, P. Sutra, and F. Pedone, "P-Store: Genuine partial replication in wide area networks," in *SRDS*, 2010.

[38] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Computing Surveys*, vol. 36, no. 4, pp. 372–421, 2004.

[39] J. Cowling and B. Liskov, "Granola: Low-overhead distributed transaction coordination," in *USENIX ATC*, 2012.

[40] D. Sciascia, F. Pedone, and F. Junqueira, "Scalable deferred update replication," in *DSN*, 2012.

[41] K. Birman and T. Joseph, "Reliable communication in the presence of failures," *Trans. on Computer Systems*, vol. 5, pp. 47–76, Feb. 1987.

[42] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine fault tolerance," *ACM Trans. on Computer Systems*, vol. 27, no. 4, 2009.

[43] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie, "Fault-scalable Byzantine fault-tolerant services," in *SOSP*, 2005.

[44] P.-L. Aublin, R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 BFT protocols," *ACM Trans. on Computer Systems*, vol. 32, no. 4, pp. 12:1–12:45, 2015.

[45] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic, "XFT: practical fault tolerance beyond crashes," in *OSDI*, 2016.

[46] G. S. Veronese, M. Correia, A. Bessani, L. C. Lung, and P. Verissimo, "Efficient Byzantine fault-tolerance," *IEEE Trans. on Computers*, vol. 62, no. 1, 2013.

[47] R. Padilha and F. Pedone, "Augustus: Scalable and robust storage for cloud applications," in *EuroSys*, 2013.

[48] R. Padilha, E. Fynn, R. Soulé, and F. Pedone, "Callinicos: Robust transactional storage for distributed data structures," in *USENIX ATC*, 2016.

[49] A. Nogueira, A. Casimiro, and A. Bessani, "Elastic state machine replication," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 9, 2017.