Practical Asynchronous Interactive Consistency

Nikos Chondros, Stathis Maneas, Christos Patsonakis, Panos Diamantopoulos and Mema Roussopoulos*

Department of Informatics and Telecommunications
National and Kapodistrian University of Athens
Athens, Greece
{n.chondros,smaneas,c.patswnakis,panos_10d,mema}@di.uoa.gr

Abstract—Interactive consistency is the problem in which $n$ nodes, where up to $t$ may be byzantine, each with its own private value, run an algorithm that allows all non-faulty nodes to infer the values of each other node. This problem is relevant to critical applications that rely on the combination of the opinions of multiple peers to provide a service. Examples include monitoring a content source to prevent equivocation or to track variability in the content provided, and reliable time-stamping of events in state machine replication.

Previous algorithms assume a synchronous system, where one can make assumptions regarding message delivery delays and/or detection of absent messages. Practical, real world systems are mainly asynchronous in nature, thus requiring a different approach. In this paper, we present the first study on practical asynchronous interactive consistency. We leverage the vast prior work on broadcast and byzantine consensus algorithms to design, implement and evaluate a suite of algorithms, each with its own unique trade-off regarding processing overhead and message complexity, that can be used to achieve interactive consistency in real distributed systems.

We provide a complete, open-source implementation of each proposed interactive consistency algorithm by building a multi-layered stack of protocols that include several broadcast protocols, as well as a binary and a multi-valued consensus protocol. Most of these protocols have been proposed, but never been implemented and evaluated in a real system before. We analyze the performance of our suite of algorithms empirically by engaging in both single instance and multiple parallel instances of each alternative.

Keywords—Interactive consistency, Asynchronous, Consensus, Agreement, Byzantine

I. INTRODUCTION

Interactive consistency is defined in a system of $n$ distinct nodes, each having its own private value, and where up to $t$ may be byzantine (faulty). The goal is for all non-byzantine nodes to compute the same vector of values, where each slot contains the private value of the corresponding node.

This problem is useful to critical applications that rely on the reliable recording of the opinions of multiple witnesses. For example, some applications may employ multiple peers to monitor the content that is delivered by a single source as a means to verify its integrity. This will prevent the source from equivocating, i.e. distributing different content to different peers, with the additional benefit of being able to prove reliably if such equivocation took place. Another closely related application is the recording of the variability in the content offered as a means to track censorship ([1]), or other forms of personalization ([2]). Other applications include the ability of sensors to reliably compute complicated functions that depend on the combination of inputs from other sensors ([3]), the reliable time stamping of events, as well as the handling of other non-deterministic values, for which there is no consistent approach in state machine replication ([4]), system diagnosis such as failure detection and group membership, cloud computing ([5]), and others.

To date, related work regarding interactive consistency has focused exclusively in providing solutions that are tailored for synchronous systems ([6], [7], [8], [9], [5]). These algorithms deliver useful theoretical insight to the issue and may be suitable in cases where one can assume negligible message delivery delays ([10]) or has the ability to detect absent messages ([5], [6]), such as in shared memory multi-processor systems. However, this line of work is ill-suited when it comes to practical, real world distributed systems which, in their vast majority, are mainly asynchronous.

Closely related topics to that of interactive consistency are byzantine agreement and byzantine consensus, for which, researchers have proposed several asynchronous algorithms. One might assume that interactive consistency can be easily achieved by a simple synthesis of one or more steps of these algorithms. However, this is not the case since it is impossible in an asynchronous system to guarantee simultaneously that all honest parties inputs’ are included in the computation (or in our case, in the resulting vector of values) and that no honest party will “hang” indefinitely ([11]).

In this paper, we present the first, to our knowledge, approach of solving interactive consistency in practical asynchronous systems. We describe how we directly address, or circumvent, both theoretical and practical challenges that arise in solving this problem. The theoretical challenges are a result of the system’s asynchrony, meaning that they apply to all asynchronous algorithms. Such examples are the well-known FLP impossibility result ([12]) and the impossibility of simultaneously achieving input completeness and guaranteed termination ([11]). The practical challenges on the other hand, are a result of assumptions that several theoretical papers use to prove their algorithms, but that, unfortunately, do not hold in practice. Examples are unbounded memory

*Student author names appear in random order.
We present a suite of algorithms that can be used to achieve interactive consistency. We leverage prior work on broadcast and byzantine consensus protocols to design our algorithms. Since byzantine fault-tolerant algorithms are generally known to exhibit high message complexities (e.g., \cite{13} has a message complexity of \(O(n^3)\)), we design our algorithms to tradeoff lower computational overhead for lower message complexity depending on the intended environment of use. For example, some of our algorithms rely on digital signatures and require less communication. In cases where network communication is the bottleneck, such as wide area networks (WANs) and the Internet, it may be more efficient to verify a few signatures than to suffer increased message complexity. Other algorithms in our suite rely on the more efficient scheme of message authentication codes (MACs). They do, however, require more messages to be exchanged, which makes them more suitable for environments where processing power is limited.

To evaluate our algorithms, we first analytically compare their message complexity and signature complexity. We then implement all of the proposed algorithms and we present empirical measurements that consist of both single instance and multiple parallel instances. We compare the performance of the algorithms in terms of throughput and latency and draw conclusions as to the appropriateness of using each algorithm in varied network environments.

In summary, we make the following contributions:

- We present the first study of interactive consistency in practical asynchronous systems and illustrate the practical and theoretical challenges that need to be addressed.
- We propose a suite of algorithms that can be used to achieve interactive consistency. Each alternative provides its own unique trade-off regarding message complexity and processing overhead.
- We provide an open-source implementation of each of the proposed alternatives. This required the development of a protocol stack that includes an asynchronous reliable message exchange protocol, various broadcast primitives, as well as a binary and a multi-valued byzantine consensus protocol. Some of these protocols, such as \cite{13,14}, have never been implemented and evaluated (to our knowledge) in a real system.
- We evaluate our algorithms empirically by running both serial and parallel executions of each algorithm and compare their performance in terms of throughput and latency. We find that it is favorable to employ digital signatures in order to reduce message complexity in WAN settings and that simple protocol variations that restrict the behavior of malicious nodes manage to deliver significant performance improvements.

The remainder of this paper is organized as follows. In Section II we present background information needed for the non-expert reader and prior related works. In Section III we describe our system model. In Section IV we describe in detail the suite of algorithms we design to solve interactive consistency in asynchronous environments. In Section V we describe the implementation of our protocol stack and the prototypes of each of our algorithms. In Section VI we present results from an empirical performance comparison of our algorithms and in Section VII we conclude.

II. BACKGROUND AND RELATED WORK

Our study has unveiled that there is a large incoherence in the bibliography regarding, otherwise, well-studied problems. For instance, the terms “Byzantine Agreement” and “Byzantine Consensus” are often used to refer to the same problem (e.g. \cite{12} and \cite{15}). Others, e.g. \cite{16}, use the terms interchangeably, even though these are two distinct problems. To alleviate any confusion and for clarity, we start off this section by citing some basic definitions.

Synchronous System. There are a priori known bounded delays regarding message delivery and node clock drifts. Algorithms designed for these systems progress in a series of lock-step rounds. The number of rounds, as is the time required for one round to complete, are built-into the system. Typically, the number of rounds required by synchronous algorithms is directly dependent on the number of faults in the system. The higher the number of faults, the higher the number of rounds required to compute a solution and terminate.

Asynchronous System. There are no bounds regarding message delays and node speeds. Algorithms for these systems also operate in rounds, however, they tend to be implemented as a collection of message handlers. Thus, it is possible to receive and process an incoming message that refers to a round that is different from the one that the receiving node is currently in. In contrast to synchronous algorithms, these algorithms do not have a given completion time, meaning that they might require more, or even less, than their synchronous counterpart to terminate.

Byzantine Agreement. Assume a system of \(n\) nodes, where a single source \(n_i\) has a private value \(v_i\), and the following must be achieved:

- Agreement: All non-faulty nodes must agree on the same value.
- Validity: If the source node is non-faulty, then the agreed upon value by all the non-faulty nodes is the same as the initial value of \(n_i\).
- Termination: All non-faulty nodes must eventually decide on a value.

This problem, also known as the “Byzantine Generals Problem”, was introduced by Lamport et al. \cite{17}. Earlier work has proven that there is no solution for the asynchronous case \cite{13}, when the source is faulty. Agreement algorithms that tolerate byzantine failures of (non-source)
nodes in asynchronous systems are presented in [18] and [19].

**Byzantine Consensus.** Assume a system of $n$ nodes, where each node $n_i$ has a private value $v_i$, and the following must be achieved:

- **Agreement**: All non-faulty nodes must agree on the same value $v \in \{v_1, \ldots, v_n\}$.
- **Validity**: If all non-faulty nodes have the same initial value $v$, then the agreed upon value by all non-faulty nodes is $v$.
- **Termination**: All non-faulty nodes must eventually decide on a value.

The byzantine consensus problem is one of the most studied topics in distributed systems and the main topic of the well-known FLP impossibility result [12]. There are several types of consensus protocols. The first distinction revolves around determinism (or non-determinism). In a deterministic consensus protocol, given the set of input values on all nodes, the message schedule and the failures that occur (if any), the result will always be the same (hence the determinism). Deterministic consensus protocols require a synchronous system [20]. In a purely asynchronous system, consensus can only be achieved by randomization, which means the result may not always be the same across executions with the same inputs. Examples of such randomized protocols are the ones introduced by Bracha [13], Toueg [21] and Ben-Or [22] that guarantee eventual termination after a probabilistic number of rounds. In [15] a trusted, non-faulty dealer is also employed, on top of randomization, to bound the number of rounds that are required to achieve consensus. Its function is to, initially, compute a random bit sequence (secret) and then to reliably distribute shares of that secret to all of the $n$ nodes in the system. While we leverage the randomization approach by Bracha to ensure termination, we believe it is controversial to assume a trusted entity in an otherwise byzantine environment. Furthermore, distributing pieces of a shared secret among nodes is a fairly expensive procedure and should thus be avoided [13].

The second distinction regarding consensus protocols revolves around the agreed upon value. All of the aforementioned protocols are binary consensus protocols, i.e., the agreed upon value is $v \in \{0, 1\}$. In the multi-valued consensus protocol of Correia et al. [14], the set of values $V$ is of arbitrary size. Consensus here is achieved by using primitives such as reliable broadcast (described below) and binary consensus.

**Failure Detectors.** In [23], Chandra and Toueg proposed a solution for the consensus problem, in an asynchronous crash-fault environment, introducing a module called failure detector (FD). There is extensive bibliography that expands the family of FDs to a number of applications including the detection of arbitrary behaviour [24, 25, 26, 27]. The muteness failure detector presented in [24] is a failure detector that copes with the arbitrary behaviour of nodes and can detect nodes that willingly stay “mute” during the execution of the protocol. A mute node refuses to send messages when it should do so. More specifically, a node $n_i$ is mute to node $n_j$ if there is a time $t$ after which, either $n_i$ crashes, or stops sending messages to $n_j$, or sends only incorrectly signed messages or unsigned messages to $n_j$. In [25], FDs are used to solve the transaction commit problem in distributed databases (non-blocking atomic commit). Mostefaiou et al. [26] use perfect failure detectors to compute global data, used in atomic commit and distributed termination detection problem. As mentioned in the paper, the computation of global data is equivalent to the interactive consistency problem. Finally, Kihlstrom et al. [27] extend the work of Chandra and Toueg and propose a failure detector that can effectively detect Byzantine behaviour.

Despite their extensive use in theoretical models, the implementation and use of FDs remains infeasible in real distributed systems, due to the fact that their assumptions cannot be met. In [28], Chandra and Toueg define the weakest FD capable of solving consensus in asynchronous crash fault systems. However, this failure detector requires known bounds on node processing speed and message delivery, that hold after a Global Stabilization Time [29]. These assumptions are unlikely to hold in real distributed systems, rendering the weakest FD unimplementable [30, 31]. The same assumptions hold for the Byzantine Failure detector introduced in [27]. To the best of our knowledge, there is no failure detector that requires weaker synchrony assumptions and can detect byzantine faults.

**Broadcast Primitives.** All algorithms that achieve asynchronous consensus employ some form of reliable broadcast protocol, which satisfies the following properties [32]:

- **Validity**: If a non-faulty node broadcasts a message $m$, then it eventually delivers $m$.
- **Agreement**: If a non-faulty node delivers a message $m$, then all non-faulty nodes eventually deliver $m$. Often described as two individual properties [33]:
  - **Consistency**: If some non-faulty node delivers a message $m$ with an $ID$ and another non-faulty node delivers a message $m'$ with the same $ID$, then $m = m'$.
  - **Totality**: If some non-faulty node delivers a message $m$ with an $ID$, then all non-faulty nodes will eventually deliver some message with the same $ID$.
- **Integrity**: For any message $m$, every non-faulty node delivers $m$ at most once iff $m$ was previously broadcast by $sender(m)$.

In [13], Bracha introduced a $\frac{3}{4}$-resilient reliable broadcast primitive (RBB, for Reliable Broadcast of Bracha) to solve the consensus problem. Another type of broadcast primitive, with lower message complexity, is consistent broadcast, which is designed not to satisfy the totality property that is
often too expensive. Consistent broadcast replaces reliable broadcast in applications where totality can be ensured by other means. The standard implementation of consistent broadcast is Reiter’s echo broadcast [34].

**Interactive consistency.** Assume a system of $n$ nodes, where each node $n_i$ has a private value $v_i$, and the following must be achieved:

- **Agreement:** All non-faulty nodes must agree on the same vector of values $V = [v_1,...,v_n]$.
- **Validity:** If the private value of the non-faulty node $n_i$ is $v_i$, then all non-faulty nodes agree on $V[i] = v_i$. If $n_i$ is faulty, then all non-faulty nodes can agree on any value $v_i$.
- **Termination:** All non-faulty nodes must eventually decide on a vector $V$.

Interactive consistency (IC) was first introduced and studied by Lamport et al. [10]. Since, it has been the topic of several research papers ([6], [7], [8], [9], [5]). This line of work proposes algorithms for synchronous systems where one can assume negligible, or bounded, message delays ([10], [6], [8]), or even detect the absence of messages ([5], [7]). While these approaches might be feasible in environments such as shared memory multi-processors or digital flight control systems, they are ill-suited for practical, real-world distributed systems which are mainly asynchronous in nature. To our knowledge, and to date, interactive consistency has not been studied in an asynchronous setting (L. Lamport, private communications). Thus, we consider our work as the first attempt of solving this problem in a practical, real world setting.

### III. System Model

We assume an asynchronous distributed system comprised of $n$ nodes that are connected over a network. Each node has a public/private key pair and all nodes know the others’ public keys.

Nodes communicate by exchanging messages. We assume that the receiver of a message can always identify its sender. This is commonly achieved by employing authenticated channels, i.e. using asymmetric cryptography to establish a shared symmetric key that is used to secure pair-wise communication. The network can drop, delay, duplicate, or deliver messages out of order. However, we assume that messages are eventually delivered, provided that the corresponding senders keep on retransmitting them.

We assume a Byzantine failure model where nodes may deviate arbitrarily from the protocol. We allow for a strong adversary that can coordinate faulty nodes. However, we assume he cannot delay neither the delivery of messages, nor processing on correct nodes beyond the system’s synchrony assumptions. The adversary is also assumed to be computationally bound, meaning he cannot subvert common cryptographic techniques such as signatures and message authentication codes (MACs).

### IV. Asynchronous Interactive Consistency

#### A. Adapting approaches from synchronous systems

The original algorithm of Lamport et al. ([10]) requires a total of $t + 1$ rounds to achieve Interactive Consistency (IC) in a synchronous system, tolerating up to $t$ faults, with a total message complexity of $(t + 1)n^2$. Our first approach is to adapt the same algorithm in an asynchronous system, by simulating synchronous rounds with barriers implemented with timeouts. However, this requires the added assumption that all messages originating from correct nodes are delivered to all non-faulty nodes before each timeout expires. Messages delivered after the given time frame will be counted towards the $t$ system faults according to the model presented in ([55]).

There are a couple of issues related with the use of timeouts. The first one is efficiency. Assuming a timeout value of $T_r$ for each round, the system will always require a constant amount of time, i.e. $(t + 1)T_r$, to execute a request even in the presence of a single failure. The second is choosing a correct value for $T_r$. If we choose a conservative approach and set a large value for $T_r$, we could increase the execution time of the algorithm dramatically, thus, making it less practical. On the contrary, a small value might cause some slow nodes, who are otherwise correct, to be considered faulty. If this occurs multiple times, as is the case when one relies on multiple timeouts, it is possible that we will exceed the upper bound $t$ of total failures in the system.

To avoid the issues associated with multiple timeouts, one could attempt to leverage the vast prior work on asynchronous Byzantine Agreement (BA) and consensus algorithms. A simple solution to IC would be to run $n$ parallel instances of BA, where each node $n_i$ would spread its private value $v_i$ to the rest of the system. In a synchronous setting, this would result in all non-faulty nodes having the same vector of values. However, this approach is inadequate in a completely asynchronous environment, as a crashed node may never even start its instance of BA, and nodes cannot distinguish between crashed nodes and slow nodes. In this case, the non-faulty nodes need to decide, at a certain point, to exclude the crashed nodes from IC and store a default (e.g. null) value at the slot corresponding to each crashed node. Thus, they need a synchronization barrier, at which point to decide on the result vector. This synchrony assumption allows for the circumvention of the impossibility of simultaneously achieving input completeness and guaranteed termination in a purely asynchronous system ([11]).

The introduction of the barrier introduces a new challenge as, at that point, a BA instance may have delivered the result in some nodes but not yet in others; this, for example, may be triggered by an adversary starting his own BA instance near the barrier. Thus, correct nodes will need to achieve consensus, for each individual slot of the result vector, on the value to be placed in that slot. We observe that the barrier...
splits the procedure in two phases; we call the first phase the **value dissemination phase**, and the second the **result consensus phase**. We employed the costly BA approach for the first phase, yet we showed that a consensus phase is still required.

**B. Avoiding Byzantine Agreement**

With these observations, we seek less costly alternatives for the first phase, i.e. avoiding BA. Our first approach is to use a simple point-to-point message exchange, where each node announces its own private value to the others. As this exchange is unrestricted, it may result in each correct node receiving a different value from a malicious node. Thus, during the result consensus phase, nodes will again agree on the value to be placed in each slot of the result vector. We employ the multi-valued consensus (MC) algorithm from [14]; recall that this algorithm utilizes a binary consensus and a reliable broadcast primitive. We want to refrain from making any further synchrony assumptions, thus, making the result consensus phase completely asynchronous. In order to circumvent FLP, which states that achieving deterministic consensus is impossible in purely asynchronous systems, we employ a randomized consensus protocol. We use Bracha’s binary consensus (BC) and reliable broadcast (RBB) primitives from [13], and we run \( n \) parallel instances of MC, one for each value of the vector. The resulting IC algorithm (IC,MC-RBB) uses only one synchrony barrier, as opposed to the adaptation of Lamport’s algorithm which needs \( t+1 \). Its overall message complexity is \( 10n^4 + 5n^3 + n^2 \). For full derivation details of all message complexities we give in this paper, we refer the reader to [36]. Figure 1 demonstrates the message exchanges for this protocol.

Our next approach reduces this message complexity. We observe multi-valued consensus uses one binary consensus and two reliable broadcast instances. We avoid the use of MC by changing the subject on which consensus is required. In the previous algorithm, the consensus question is “what is the actual value to be placed in the corresponding slot of the result vector?”, because the first (value dissemination) phase is insecure. We make the first phase secure by using Consistent Broadcast [34]. Here, the source \( n_i \) first sends its value \( v_i \) to each node; then it collects signed endorsement responses. Once \( n − t \) such responses are accumulated, the sender forms a certificate \( c_i \) that includes them, and sends \(<n_i,c_i>\) to the rest of the nodes. Each node delivers \( v_i \) iff \( c_i \) has at least \( n − t \) valid signatures. Assuming signatures are unforgeable, malicious nodes cannot construct two valid certificates for two different values. Thus, this protocol bounds the sender to either send a single value to another node, or not send a value at all. As this value is now unique, we change the question of the consensus round to “is there a value to be placed in the corresponding slot of the result vector?”. This question can now be answered by a binary consensus protocol, and we utilize Bracha’s protocol [13] in our approach. Figure 2 depicts message exchanges for this protocol.

---

**Figure 1.** Diagram of message exchange for IC,MC-RBB, for a single value of the result vector (repeated \( n \) times to achieve IC).

**Figure 2.** Diagram of message exchanges for IC,BC-RBB, for a single value of the result vector (repeated \( n \) times to achieve IC).
Note that there are cases however, where a consensus instance may produce a result different than what a correct node originally knew at the barrier. This can happen when some of the correct nodes have delivered the value $v_i$, while others have not. Thus, a node may have had a value for this slot, and the result of consensus may by 0, in which case it drops the value. However, the contrary may also happen, where a node did not have a value but consensus resulted in 1. For this case, we add a final recovery step, where a node asks all other nodes for the $<v_i, c_i>$ tuple they have. Since at least one correct node has voted 1 for this specific consensus instance (otherwise there is no possibility for such an outcome), there is at least one node that will respond, and the correct value will be discovered. The overall complexity of this IC algorithm (IC,BC-RBB) is $6n^4 + 3n^3 + 3n^2$ messages and $n^4 + 2n^2$ signature operations.

**C. Reliable Broadcast using Signatures**

Bracha’s binary consensus algorithm utilizes a signature-free reliable broadcast primitive. However, the message complexity of Bracha’s reliable broadcast is $2n^2 + n$ messages for a single node. In an attempt to further minimize the overall message complexity of our algorithm we introduce a reliable broadcast primitive that uses digital signatures (RBS).

We enhance the Consistent Broadcast primitive of [4], to additionally satisfy the totality property, thus achieving agreement. To achieve this, we add an additional step of communication across all nodes after the final message is sent. The algorithm presented in Figure 3 is a reliable broadcast protocol, since it satisfies the agreement property. The proof of correctness can be found in [5]. The message complexity of the new reliable broadcast primitive is $n^2 + 3n$ with $n^2 + 2n$ signature operations. By caching already validated signatures, we can further reduce the cost of signature verifications, since nodes can refrain from revalidating cached signatures.

In Section VI we demonstrate signature verification cost can be masked in network topologies with high latencies, such as WANs.

**D. Improvements using signatures**

We revisit our two approaches to IC from section IV-B by utilizing the introduced Reliable Broadcast with signatures (RBS). First, we modify Bracha’s Binary Consensus protocol to use RBS instead of Bracha’s own Reliable Broadcast. This creates a new configuration for our binary consensus approach (IC,BC-RBS) which has a message complexity of $3n^3 + 9n^2 + 3n$ and $3n^4 + 7n^3 + 2n^2$ signature operations.

Next, we adapt our multi-valued consensus approach (IC-MC-RBB) to use RBS both directly, and indirectly via binary consensus. This new configuration (IC,MC-RBS) has a message complexity of $5n^4 + 15n^3 + n^2$ and $5n^4 + 10n^3$ signature operations.

---

**Step 1.** Upon $r$-broadcast$(m)$ do

send message(initial, m) to all nodes

**Step 2.** Upon receiving message (initial, m) from $P_s$ do

compute signature $\sigma$ on (vouch, m, s)

send message (vouch, m, $\sigma_i$) to $P_s$ only once

**Step 3.** Upon receiving $n - t$ messages (vouch, m, $\sigma_i$) with valid signatures $\sigma_i$ do

build a set $\Sigma$ of all valid signatures

send message (final, m, $\Sigma$) to all nodes

**Step 4.** Upon receiving message (final, m, $\Sigma$) with $n - t$ valid signatures in $\Sigma$ do

send message (final, m, $\Sigma$) to all nodes only once

**Step 5.** Upon receiving $n - t$ messages (final, m, $\Sigma$) with same set $\Sigma$

$r$-deliver($m$)

Figure 3. Reliable Broadcast using digital signatures.

To reduce the overhead caused by signature operations, we use authenticators as suggested by [4]. In this scheme, nodes exchange pair-wise messages to announce to the receiving party a symmetric secret key to use when sending messages to the sending party. This exchange is repeated often enough to make the symmetric key secure. When a node wishes to multicast a message to $n$ nodes, it composes an authenticator, which is a vector of $n$ HMACs, one for each receiving node, by using the corresponding key as an input to the HMAC function. The receiving party uses its corresponding entry of the authenticator to verify both the integrity and the authenticity of the message, making this scheme analogous to digital signatures (for the closed world for which the authenticator provides HMACs). The performance improvement is vast as, in a simple evaluation on a contemporary desktop CPU, we can calculate approximately 300 SHA-1 based HMACs at the same time required to produce a single digital signature using RSA with a 1024 bits key. We have adapted all of our protocols using digital signatures, to use authenticators instead, significantly improving on performance, and we present our results in section VII.

**E. Eventual Interactive Consistency**

So far, we present solutions to IC using one or more barriers, as the problem is unsolvable in a completely asynchronous setting. There is, however, a relaxed version of the problem, which can be solved without timing assumptions, which we introduce and briefly outline two solutions. We call this Eventual Interactive Consistency (EIC). We change the Agreement part of the problem’s definition as follows:

- **Agreement:** All non-faulty nodes must eventually agree on the same vector of values $V = [v_1, ..., v_n]$,
In this scheme, a non-faulty node will eventually build the result vector, containing all private values from all non-faulty nodes. Until it does however, it may have empty slots for values it does not yet know about. In practice, the result vector will be built slot by slot, and instead of a single up-call to deliver the complete vector, multiple up-calls will take place to inform the application the vector was augmented by one more value. This version of the problem is suitable, for example, for applications which gather opinions. The idea is, the application can serve the already gathered opinions to clients, with empty slots for the currently unknown ones, either immediately upon request, or when a system-defined threshold of entries has been filled in the vector. Eventually, when all non-faulty nodes provide their opinions, empty slots in the result vector will represent failed nodes. If the vector is used before it is completed, the only guarantee provided is that on a second access, the previous entries in the vector will still be included, potentially along with newer ones.

One simple approach to achieve this is by using a version of Reliable Broadcast (RB). All nodes start one instance and, by definition of RB, eventually all correct started broadcasts deliver the intended value. Once every one of these RB instances completes, the corresponding slot of the vector will be filled and a new notification will be sent to the upper level application. This approach leaves management of the result vector completely up to the application.

A more involved approach, which however preserves and manages the result vector as well, is the use of a byzantine fault tolerant Replicated State Machine (RSM), such as [4], enhanced to handle byzantine clients ([37], [38]). Each node of EIC becomes both a replica and a client of the same RSM. Each IC node, as a client to the RSM, posts its private value to the RSM as a write operation. The application running on top of the RSM receives this write operation and accepts it only when no prior value is known for the sending node and this instance of EIC. Whenever an external client requests the EIC result vector, it is dynamically compiled from entries already posted from nodes. A malicious node cannot harm the system as long as the RSM’s fault tolerance level (typically \(n \geq 3t + 1\)) is not breached, while as a client it is prohibited from posting more than one value by the aforementioned functionality running at the application layer behind the RSM.

V. IMPLEMENTATION

Figure 4 illustrates the open-source protocol stack we developed in Java in order to implement and evaluate our suite of algorithms. At the foundation lies an authenticated channels layer that uses SSL and manages message passing and timeout events; SSL provides for authentication and message integrity. We also provide alternatives for direct TCP/IP communication (without strong authentication), as well as, an in-process communication infrastructure that allows us to run our unit tests and verify the correctness of our implementation. The remaining layers are agnostic of the network layout or communication means, as they simply register event handlers to process incoming messages. Finally, we simulate loss-less channels by creating one output queue for each node, where each queue is handled by a different thread. A message is deleted from a queue only when the sender receives an acknowledgment for that specific message by the destination.

![Figure 4](image-url)

On top of this foundation, we implement a series of broadcast primitives, starting with Consistent Broadcast, which we then enhance to derive Reliable Broadcast with signatures (RBS). Lastly, we implement the signature-free Reliable Broadcast primitive of Bracha (RBB).

We then implement Bracha’s binary consensus (BC) protocol [13], which can be used in conjunction with both RBS and RBB. By combining BC with any of the reliable broadcast primitives, we implement the multi-valued consensus protocol (MC) of Correia et al. [14].

Bracha’s binary consensus algorithm operates in a probabilistic number of phases. It requires nodes to buffer all messages, even the ones referring to future phases to guarantee termination. This, in conjunction with the fact that the number of rounds is not bounded, may require nodes to buffer an arbitrary number of messages. In a practical system, it is unrealistic to assume nodes with unbounded memory. Thus, malicious nodes could bombard non-faulty nodes with spurious messages which, since they are required to buffer them, would result in a state-explosion attack. Our approach on this matter is twofold. First, we identify that each phase of Bracha’s consensus protocol is independent from any previous phases. This means that once a node enters phase \(i+1\), it can safely discard any buffered messages from phase \(i\) since they are no longer needed. Second, in order to defend against the state-explosion attack, nodes
buffer messages whose current phase number \( i \) is a total of \( H \) phases ahead. However, this requires a recovery protocol for slow nodes that have fallen behind and are unable to progress due to the fact that the others have reached a phase \( j > i + H \), which we leave as future work.

Finally, we leverage this protocol stack to provide the following interactive consistency algorithm suite:

1. Lamport’s algorithm (IC,LAMP).
2. Consistent Broadcast for the value dissemination phase and Bracha’s binary consensus, in conjunction with the reliable broadcast of Bracha for the result consensus phase (IC,BC-RBB).
3. Consistent Broadcast for the value dissemination phase and Bracha’s binary consensus, in conjunction with the reliable broadcast primitive that uses signatures for the result consensus phase (IC,BC-RBS).
4. Multicast for the value dissemination phase and multi-valued consensus, in conjunction with the reliable broadcast of Bracha for the result consensus phase (IC,MC-RBB).
5. Multicast for the value dissemination phase and multi-valued consensus, in conjunction with the reliable broadcast primitive that uses signatures for the result consensus phase (IC,MC-RBS).

VI. EVALUATION

In this section we empirically evaluate the performance of our algorithm suite under various system settings. We conduct our experiments using a dedicated cluster with eight identical nodes, directly connected via an isolated 1Gbps Ethernet switch. Each node is configured with 4GB of RAM and dual Intel(R) Xeon(TM) CPUs running at 2.80GHz, which additionally support hyper-threading.

In Figure 5a, we illustrate the total time required to complete one instance of each of the proposed algorithms, without faults, in a local area network (LAN) setting with almost negligible link latency. As expected, the asynchronous adaptation of Lamport’s algorithm exhibits the best performance due to its significantly lower message complexity, as well as the fact that all messages are delivered before any timeout (barrier) expires. From the remaining alternatives, the ones relying on multi-valued consensus for the result consensus phase perform worse. This illustrates that the IC,BC-RBB approach, even though it has a more costly first phase, outperforms both multi-valued approaches because of its efficiency at the second phase.

Next, we examine the tradeoff between signature operations and message complexity in both LAN and WAN settings. As is depicted in Figure 5a, all of the algorithms that employ the RBS primitive (which involves digital signatures) perform worse than those that utilize the RBB primitive, illustrating that, in a LAN setting, verifying signatures can hamper the algorithm’s performance. In order to establish whether the same holds in a WAN setting as well, where latencies are much higher, we repeated the experiment of Figure 5a. However, this time, we artificially injected a uniform latency of 50ms before each message delivery. The results are presented in Figure 5b, which illustrates that employing digital signatures in order to reduce message complexity is actually favorable in such a networking environment.

In Figure 5c, we illustrate the performance of each algorithm in the presence of a single crash fault in a LAN setting. We set a modest timeout value of three seconds. Results illustrate the inefficiency of employing multiple timeouts in such an environment. The asynchronous adaptation of Lamport’s algorithm, which used to be the best alternative in the fault-free case, now, in the presence of a single fault, actually exhibits the worst performance. We repeated this experiment in a WAN environment as well in order to establish if the same will apply in such a setting. Our results are depicted in Figure 5d and illustrate that multiple barriers are a better option than the increased message complexity of all of the consensus based approaches. However, we note that we used the same timeout value in both LAN and WAN settings, i.e. three seconds, in order to provide for an even comparison. In a real deployment, one would employ a much larger timeout value for the WAN case, resulting in a more significant impact in the algorithms’ performance.

In all of the aforementioned cases, we used digital signatures for the algorithms that employ CB for the value dissemination phase and RBS as the reliable broadcast primitive of the result consensus phase (IC,BC-RBB, IC,BC-RBS, IC,MC-RBB). In Figure 5a we illustrate the performance benefits of employing the more efficient scheme that is based on authenticators, as was described in Section IV-D. While there are no significant changes in the curves, the approaches with authenticators have closed the gap with their non-signature based siblings, and even the worst performing algorithm finishes in under one second.

Lastly, we evaluate the throughput of our algorithm suite by running 100 parallel instances of each of the proposed algorithms in a LAN setting. We present our results in Figure 5f. Since it is a fault-free case, it comes as no surprise that Lamport’s variant exhibits the best performance. For the remaining alternatives, results illustrate that the algorithms that employ CB for the value dissemination phase, thus requiring only one BC instance for the result consensus phase, provide consistently better throughput. Thus, one can draw significant performance improvements by strengthening the value dissemination phase.

VII. CONCLUSION

In this paper, we tackle the problem of Interactive Consistency in practical asynchronous systems. Although this problem is unsolvable in this setting, due to several impossibility results, such as FLP, we present solutions that range from porting Lamport’s synchronous algorithm and
making multiple timing assumptions, to composing more sophisticated algorithms based on existing broadcast and consensus primitives with a single timing assumption. We provide a signature-based reliable broadcast primitive, which can substitute existing signature-free incarnations of reliable broadcast. We also define a more relaxed version of the problem, which we call Eventual Interactive Consistency, that is suitable for some applications, and we describe possible approaches solving the problem without any timing assumptions.

As we implemented all of the described IC algorithms, we share our experience of dealing with pragmatic issues such as unbounded memory requirements, and channels that are authenticated, reliable, and loss-less. We build a completely event-driven open-source software stack in Java from scratch, and we use it to measure the performance of all of our proposed algorithms. We use parameters for both LAN and WAN settings, combined with the fault-free case, and with one crash fault that causes all nodes to wait for their timeouts, demonstrating the cost of timing assumptions. We also implement the more efficient scheme of authenticators, which allows us to replace digital signatures in all of the algorithms that use the latter. We empirically compare the performance of all algorithms and highlight trade-offs that arise in different system settings. With this work, we hope to provide useful insight to interested system designers on the appropriate IC approach to use.

REFERENCES

[1] E. Athanasopoulos, S. Ioannidis, and A. Sfakianakis, “Censormon: A web censorship monitor,” in USENIX Workshop on Free and Open Communications on the Internet, FOCI 2011.

[2] G. Goodell, M. Roussopoulos, and S. Bradner, “A directory service for perspective access networks,” IEEE/ACM Trans. Netw., vol. 17, no. 2, pp. 501–514, Apr. 2009.

[3] S. Wang, K. Yan, C. Ho, and S. Wang, “The optimal generalized byzantine agreement in cluster-based wireless sensor networks,” Computer Standards & Interfaces, vol. 36, no. 5, pp. 821–830, 2014.

[4] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in Proceedings of the Third Symposium on Operating Systems Design and Implementation, ser. OSDI 1999, pp. 173–186.

[5] S.-S. Wang, K.-Q. Yan, and S.-C. Wang, “Achieving efficient agreement within a dual-failure cloud-computing environment,” Expert Syst. Appl., vol. 38, no. 1, pp. 906–915, 2011.

[6] P. M. Thambidurai and Y.-K. Park, in Symposium on Reliable Distributed Systems, pp. 93–100.

[7] P. Lincoln and J. Rushby, “Formal verification of an interactive consistency algorithm for the draper FTP architecture under a hybrid fault model,” in Compass 1994, pp. 107–120.

[8] A. Gascón and A. Tiwari, “A synthesized algorithm for interactive consistency,” in NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings, 2014, pp. 270–284.
[9] J. H. Lala, “A byzantine resilient fault tolerant computer for nuclear power plant applications,” in *International Symposium on Fault Tolerant Computing Systems*, July 1986.

[10] M. Pease, R. Shostak, and L. Lamport, “Reaching agreement in the presence of faults,” *J. ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1980.

[11] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, “Universally composable synchronous computation,” in *TCC*, 2013, pp. 477–498.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[13] G. Bracha, “Asynchronous byzantine agreement protocols,” *Inf. Comput.*, vol. 75, no. 2, pp. 130–143, Nov. 1987.

[14] M. Correia, N. F. Neves, and P. Verissimo, “From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures,” *Comput. J.*, vol. 49, no. 1, pp. 82–96, Jan. 2006.

[15] S. Toueg, “Randomized byzantine agreements,” in *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC 1984, pp. 163–178.

[16] P. A. Patra and C. P. Rangan, “Communication optimal multi-valued asynchronous byzantine agreement with optimal resiliency,” in *Information Theoretic Security - 5th International Conference, ICTIS 2011*, 2011, pp. 206–226.

[17] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[18] G. Bracha and S. Toueg, “Asynchronous consensus and broadcast protocols,” *J. ACM*, vol. 32, no. 4, pp. 824–840, Oct. 1985.

[19] R. Canetti and T. Rabin, “Fast asynchronous byzantine agreement with optimal resilience,” in *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, ser. STOC 1993, pp. 42–51.

[20] D. Dolev, C. Dwork, and L. Stockmeyer, “On the minimal synchronism needed for distributed consensus,” *J. ACM*, vol. 34, no. 1, pp. 77–97, Jan. 1987.

[21] G. Bracha and S. Toueg, “Resilient consensus protocols,” in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC 1983, pp. 12–26.

[22] 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*, ser. PODC 1983.

[23] T. D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[24] A. Doudou and A. Schiper, “Muteness detectors for consensus with byzantine processes,” in *Proceedings of the 17th ACM Symposium on Principle of Distributed Computing*, Puerto Citeseer, 1997.

[25] R. Guerraoui and P. Kouznetsov, “On the weakest failure detector for non-blocking atomic commit,” in *Foundations of Information Technology in the Era of Network and Mobile Computing*. Springer, 2002, pp. 461–473.

[26] J.-M. Hélary, M. Hurfin, A. Mostefaoui, M. Raynal, and F. Tronel, “Computing global functions in asynchronous distributed systems with perfect failure detectors,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 9, pp. 897–909, Sep. 2000.

[27] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, “Byzantine fault detectors for solving consensus,” *The Computer Journal*, vol. 46, no. 1, pp. 16–55, 2003.

[28] T. D. Chandra, V. Hadzilacos, and S. Toueg, “The weakest failure detector for solving consensus,” *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 685–722, 1996.

[29] M. Larrea, A. Fernández, and S. Arévalo, “Optimal implementation of the weakest failure detector for solving consensus,” in *Reliable Distributed Systems, 2000. SRDS-2000. Proceedings The 19th IEEE Symposium on*. IEEE, 2000, pp. 52–59.

[30] V. K. Garg and J. R. Mitchell, “Implementable failure detectors in asynchronous systems,” in *Foundations of Software Technology and Theoretical Computer Science*. Springer, 1998, pp. 158–169.

[31] M. Correia, G. S. Veronese, N. F. Neves, and P. Verissimo, “Byzantine consensus in asynchronous message-passing systems: a survey,” *IJCCBS*, vol. 2, no. 2, pp. 141–161, 2011.

[32] V. Hadzilacos and S. Toueg, “A modular approach to fault-tolerant broadcasts and related problems,” 1994.

[33] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, “Secure and efficient asynchronous broadcast protocols,” in *Advances in Cryptology-Crypto 2001*. Springer, 2001, pp. 524–541.

[34] M. K. Reiter, “Secure agreement protocols: Reliable and atomic group multicast in rampart,” in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. ACM, 1994, pp. 68–80.

[35] D. Dolev and R. Strong, *Distributed Commit with Bounded Waiting*, 1982.

[36] N. Chondros, S. Maneas, C. Patsonakis, P. Diamantopoulos, and M. Roussopoulos, “Practical asynchronous interactive consistency (extended paper),” Department of Informatics and Telecommunications, University of Athens, Tech. Rep., October 2014. [Online]. Available: [http://cgi.di.uoa.gr/~mema/publications/ic-extended.pdf](http://cgi.di.uoa.gr/~mema/publications/ic-extended.pdf)

[37] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, “Making byzantine fault tolerant systems tolerate byzantine faults,” in *NSDI*, vol. 9, 2009, pp. 153–168.

[38] B. Liskov and R. Rodrigues, “Byzantine clients rendered harmless,” in *Distributed Computing*. Springer, 2005, pp. 487–489.