A New Atomic Broadcast Protocol for Asynchronous Distributed Systems*

Nadjette Rebouh[1,*,†], Louiza Bouallouche-Medjkoune[2,†]

[1]Research Unit LaMOS, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria
[1]Research Unit LaMOS, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria

Abstract

Blockchains are decentralized and immutable ledgers that ensure secure transaction recording. In these systems, the atomic broadcast is pivotal, ensuring consistent transaction delivery to all participants. It ensures either all nodes receive a transaction or none, preserving ledger integrity. This paper addresses the atomic broadcast problem in asynchronous systems. The existing atomic broadcast protocols suffer from relying on strong temporal assumptions and generating a significant number of messages. In response, we propose a novel protocol based on the rotating coordinator principle for message ordering and the $\diamond S$ failure detector for fault tolerance. It has been demonstrated that $\diamond S$ is the minimal and sufficient class of failure detectors to solve the atomic broadcast in asynchronous systems. Simulation results, conducted using the Neko simulator, demonstrate significant enhancements in latency and message throughput compared to two existing protocols, implemented within the same simulator. The robustness and efficiency of the proposed solution are convincingly demonstrated through extensive simulations. This research contributes valuable insights into enhancing the reliability and performance of atomic broadcast protocols, crucial for the development of resilient and scalable distributed systems in blockchain technology and beyond.

Keywords

Blockchain Systems, Atomic Broadcast, Distributed System, Failure Detectors, Rotating Coordinator, Agreement Problem

1. Introduction

Blockchains represent a fascinating intersection of two prominent fields in computer science: distributed systems and cryptography. At its core, blockchains are distributed systems composed of a network of interconnected nodes, each maintaining a copy of the ledger. and participating in the consensus process. Blockchains, as decentralized and immutable ledgers, ensure secure transaction recording, resilience, fault tolerance, and consistency, making it a revolutionary technology for various applications, from financial transactions to supply chain management [1], [2]. However, maintaining consistency across distributed systems or distributed ledgers presents significant challenges. This is primarily due to the inevitable concurrency in such

TACC'2024, The 4th Tunisian-Algerian Conference on applied Computing, December 17-18, 2024, Constantine, Algeria

*You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

*Corresponding author.

†These authors contributed equally.

nadjette.rebouh@univ-bejaia.dz (N. Rebouh); louiza.medjkoune@univ-bejaia.dz (L. Bouallouche-Medjkoune)
0000-0002-8821-8543 (N. Rebouh); 0000-0003-2849-6258 (L. Bouallouche-Medjkoune)

systems, combined with the difficulty of providing global system control and the presence of failures (crashes or byzantine). This difficulty is significantly reduced by relying on group communication primitives that offer better guarantees than standard point-to-point communication. In these systems, the atomic broadcast (with reduction to consensus) is pivotal, ensuring consistent transaction delivery to all participants. It ensures either all nodes receive a transaction or none, preserving the systems integrity [3], [4], [5].

Several algorithms solving the atomic broadcast problem in a distributed systems have been proposed in the literature. These algorithms can be classified according to two main criteria: The message ordering approach and the fault tolerance mechanism implemented [6], [7]. Several approaches have been employed to enforce message order, either relying on a single entity to enforce delivery order uniqueness (the coordinator principle) or leveraging the system structure, which demands fewer control messages but at the expense of other characteristics [8]. However, the chosen fault tolerance approach plays a pivotal role in the correctness, efficiency, and complexity of the atomic broadcast protocol. Furthermore, the type of failures supported by the system significantly impacts the protocol performance and the system as a whole. The challenge then lies in designing a simple and efficient atomic broadcast protocol based on minimal temporal assumptions inherent in the necessary and sufficient failure detector for fault tolerance, the optimal process structure within the system, and the appropriate message ordering technique, while also considering the type of tolerated failures.

This paper presents a new atomic broadcast protocol for all types of asynchronous distributed systems. The protocol is based on the utilization of unreliable failure detectors $\diamond S$ for fault tolerance [3]. This class of failure detectors incorporates the minimal synchrony assumptions to address a given agreement problem [9]. Consequently, the protocol requires a majority of correct processes relative to the number of tolerated failures. Accordingly, the protocol generates fewer control messages necessary for the system execution and correction. Consequently, the protocol leverages the unique output of the failure detector to designate it as the coordinator for imposing message delivery order. Participating processes behave symmetrically to ensure system integrity through a fully centralized approach.

The paper is structured as follows. Section 2 surveys some related work. Section 3 introduces the system model and, formally, defines the concepts. Then, Section 4 presents the proposed atomic broadcast protocol and Section 5 proves its correctness. Section 6 evaluates the performances of the proposed protocol. Finally, Section 7 concludes the paper.

2. Related Work

In [6], authors discuss various protocols and approaches for achieving atomic broadcast, including both pessimistic and optimistic strategies. Pessimistic approaches typically rely on atomic broadcast protocols, where messages are delivered to all processes in a predefined order.

Two classes of atomic broadcast protocols have been identified according to the ordering of messages and the fault tolerance mechanism used. Token-based solutions rely on the ring structure of processes to transmit the order information handled by a token. They have been divided into two classes according to the fault tolerance mechanism used: [10], [11], [12] are based on group membership service while [13], [14] are failure detector based protocols.

In [3], they rely on a collaboration between the coordinator principle and the consensus to achieve the ordering task in a centralized manner. However, in [15], they use the same approach as [3] but in a decentralized manner. In this paper, we use the same approach but without the use of the consensus problem as a building block. The processes communicate directly with the coordinator of the round and get the appropriate information to decide on the set of messages to be atomically delivered.

In blockchain systems, the atomic broadcast protocols tolerate Byzantine failures when the processes are distributed across a decentralized network of nodes. These protocols ensure fault tolerance, allowing the system to maintain the consistency and integrity of data even in the presence of malicious processes or Byzantine faults [16], [17], [18], [19], [20].

3. Computation Model and Definitions

In this section, we present the system model and some definitions related to our study.

3.1. Computation Model

We consider an asynchronous distributed system composed of a set $\Pi = \{p_0, ..., p_{n-1}\}$ of n processes, connected by reliable communication channels forming a fully connected network. Processes can only tolerate failures by permanent crashes. We assume a majority of correct processes ($f < n/2$; where f represents the maximum number of faulty processes). The system is augmented with the unreliable failure detector $\diamond S$.

3.2. Unreliable Failure Detectors

Informally, a failure detector consists of a set of modules, each attached to a process: The module attached to a process maintains a set of suspected faulty processes. A failure detector is considered unreliable because it may suspect a correct process as well as fail to suspect a genuinely faulty process. Chandra and Toueg [3] defined eight classes of failure detectors based on two properties: Completeness (a liveness property ensuring that faulty processes will eventually be suspected) and Accuracy (a safety property restricting false suspicions about correct processes). In this paper, we consider the $\diamond S$ class defined as follows:

- *Strong completeness:* Every faulty process will eventually be permanently suspected by every correct process.
- *Weak ultimate accuracy:* There exists a moment from which at least one correct process is never suspected again.

Chandra et al. [9] showed that $\diamond S$ is the weakest class of failure detectors and is the minimal and necessary class to solve the atomic broadcast problem in an asynchronous distributed system (by reduction to the consensus problem).

3.3. Reliable Broadcast

This problem ensures an atomic delivery of messages to the processes of the system (a message sent by a process is received by all processes) [4]. It is defined using two communication primitives $R - broadcast()$ and $R - deliver()$. Formally, it is specified by the following properties:

1. *Agreement:* If a correct process broadcasts a message m, then inevitably all correct processes deliver this message;
2. *Validity:* If a process delivers a message m, then m has been broadcast by at least one process;
3. *Integrity:* A message delivery occurs at most once.

3.4. Atomic Broadcast

The atomic broadcast is an extension of the reliable broadcast. In addition to ensuring that all processes receive the same number of messages (reliable broadcast), it also ensures the same delivery order of these messages [4]. It is defined using two communication primitives: $A - broadcast()$ and $A - deliver()$. Atomic validation in the context of transactional databases and blockchain systems is an example of an application using the atomic broadcast primitives. This problem concerns the dissemination and delivery of messages by processes. More formally, atomic broadcast is completely defined by the properties of reliable broadcast plus the following order property:

- *Total order:* If a correct process delivers a message m before message m', then all processes deliver m before message m'.

3.5. Data structures and messages

Each correct process, participating in the atomic broadcast protocol, emits and receives messages of various types:

1. *vote*: To elect the new coordinator for the current round;
2. *cancel*: When the process wants to cancel the processing done during the current round;
3. *decision*: As soon as the coordinator receives a sufficient number of votes, it broadcasts this message to inform the other processes of its decision regarding the order of the message list to be atomically delivered.
4. *simple*: A simple message broadcast by a correct process to participate in the application tasks.

The first three messages are control messages, while the last message leads to a system evolution (its execution). To achieve this, a process manipulates the following data structures:

- $ordered\_list_i$: A list containing the messages identifiers (id_{msg}) and it does not allow duplication of elements.
- id_{msg}: A data structure composed of two integers, the first representing the message sender number and the second is the timestamp of this message. The use of this structure guarantees the uniqueness of broadcast messages.

- *unordered_list$_i$*: The list of broadcast messages that have not been yet ordered.
- *round*: An integer representing the current round, initially set to zero.
- *tab_vote$_i$*: An array of integers containing the votes of the processes according to their identifiers ($\forall j \in \Pi$: *tab_vote$_i$*$[j] = ?$).
- *trust$_i$*: An integer containing the number of positive votes (vote=1) present in *tab_vote$_i$*.
- *distrust$_i$*: An integer containing the number of negative votes (vote=2) present in *tab_vote$_i$*.
- *temp*: A list containing all messages belonging to rounds succeeding the current round.

4. The Atomic Broadcast Protocol

The section is devoted to the description and the algorithm of the proposed solution.

4.1. The proposed protocol description

The principle of the protocol is quite straightforward. It operates in a series of asynchronous rounds. During each round, a process is elected (rotating coordinator) to enforce the list of messages to be delivered by all correct processes. The algorithm relies on the use of the failure detector $\diamond S$ to provide the list of faulty processes in the system. To achieve this, the use of communication primitives, reliable broadcast, is paramount to ensure messages sending and receiving. The algorithm unfolds in two tasks:

1. ***Task1***: It consists of two phases:
   a) ***Phase1***: Each process broadcasts its messages by invoking the *R−Broadcast* primitive (which is the indirect implementation of $A - broadcast$).
   b) ***Phase2***: During this phase, each process p_i must vote for one of two values: (1) If it does not suspect the coordinator process p_c; (2) If it suspects it to be faulty based on the list of faulty processes provided by the failure detector $\diamond S$ attached to this process. The process p_i can only cast one vote during a round.
2. ***Task2***: Upon receiving a message m, the process p_i must perform one of the following treatments depending on the type of the received message:
   a) m is a simple message: p_i adds m to its *unordered_list$_i$* and places the message identifier in *ordered_list$_i$*.
   b) m is a vote message: p_i updates *tab_vote$_i$* and its local variables *distrust$_i$* and *trust$_i$*. Two cases can be considered:
      - *distrust$_i$* $\geq (n - f)$: p_i broadcasts a *cancel* message to start a new round.
      - *trust$_i$* $\geq (n - f)$ and p_i is the coordinator: In this case, p_i decides and broadcasts its *ordered_list$_c$*.
   c) $m \in$ *ordered_list$_i$*: If the round number of m (*round$_m$*) is equal to the current round, p_i delivers its recorded messages in the *ordered_list$_i$* according to the coordinator imposed order and initiates a new round. However, if *round$_m$* is greater than the current round, the process p_i adds the message m to the temporary list *temp*.
   d) m is a *cancel* message: If *round$_m$* is equal to the current round, p_i initiates a new round. However, if *round$_m$* is greater than the current round, the process p_i adds the message m to the temporary list *temp*.

4.2. The proposed protocol

This section presents the atomic broadcast solution (Algorithm 1).

Algorithm 1 *The atomic broadcast protocol*

```
TASK1
  PHASE1: Message sending
    $R - broadcast(simple)$
  PHASE2: Vote sending
    $p_c \leftarrow$ round $mod$ n ;
    if $(p_c \in suspected_i)$ then $tab\_vote_i[i] \leftarrow 1$ ;
                 else
                     $tab\_vote_i[i] \leftarrow 2$
    $R - broadcast(round_i, vote_i)$
TASK2
    upon $receiving\ m$ do
      $switch(m)$:
        $case\ simple$
          $ordered\_list_i \leftarrow ordered\_list_i \cup id_{msg}$;
          $unordered\_list_i \leftarrow unordered\_list_i \cup m$;
        $case\ vote$
          $tab\_vote_i[j] \leftarrow vote_j$;        $vote_j$ refers to the vote casted by the process $p_j$ ;
          update ( $trust_i$ and $distrust_i$)
          if $((p_c = p_i)$ and $(trust_i \geq (n - f))$ then
                   $R - broadcast(ordered\_list_i, decision)$ ;
          if $(distrust_i \geq (n - f))$ then $R - broadcast(round_i, cancel)$ ;
        $case\ decision$
          if $(m_{round} = round)$ then atomically deliver all messages in some
                   deterministic order imposed by the
                   coordinator process and start a new round ;
                 else
                     $temp_i \leftarrow temp_i \cup m$;
        $case\ cancel$
          if $(m_{round} = round)$ then start a new round;
                 else
                     $temp_i \leftarrow temp_i \cup m$;
      $endswitch$
```

5. Correctness Proof

The atomic broadcast protocol must ensure the two properties of safety and liveness encapsulated within the properties defining this problem.

Lemma 1. *If a process delivers a message m, then m has been broadcast by at least one process (validity property).*

Proof. The use of reliable broadcast primitives ensures this lemma. □

Lemma 2. *If a correct process delivers a message m, then inevitably all correct processes deliver m (agreement property).*

Proof. The use of the reliable broadcast primitives ensures that inevitably all processes deliver the same set of messages or none of them. By using the coordinator principle, we ensure that any message sent by a correct process will be delivered by all correct processes as soon as the coordinator broadcasts its final ordered list of messages. □

Lemma 3. *For any message m, each correct process delivers the message m at most once, and only if m has previously been broadcast by a certain correct process (integrity property).*

Proof. This property is guaranteed by the use of the reliable broadcast primitives. □

Lemma 4. *If two correct processes p and q deliver two messages m and m', then p delivers m before m' if, and only if, q delivers m before m' (total order property).*

Proof. This property is guaranteed by broadcasting the coordinator list of messages. Once a process is elected as the coordinator of the current round, it will impose its list of messages to be atomically delivered by all the correct processes. This list will be broadcast to the processes through the reliable broadcast primitive $R - broadcast$. Therefore, all correct processes will deliver their messages according to the order imposed by the coordinator, which will be the same for every process. □

Theorem.*The algorithm in Algorithm 1 implements the primitives of the atomic broadcast.*

Proof. The proof follows directly from the four lemmas. □

6. Performance Evaluation

In this section, we evaluate the performance of the proposed protocol. To do so, we utilize a well-known simulator in distributed systems, Neko [21].

6.1. Simulation model

In our work, we employ the same simulation model as [22] Fig. 1.

The parameter λ ($\lambda \geq 0$) indicates the relative speed of processing a message on a process compared to its transmission on the network. Different values for modeling the network environment were chosen ($\lambda = 0.1$, $\lambda = 1$, $\lambda = 10$). The values of λ are selected based on the approach followed in [23]:

- $\lambda = 1$: Indicates that the processor processing and network transmission have the same cost;

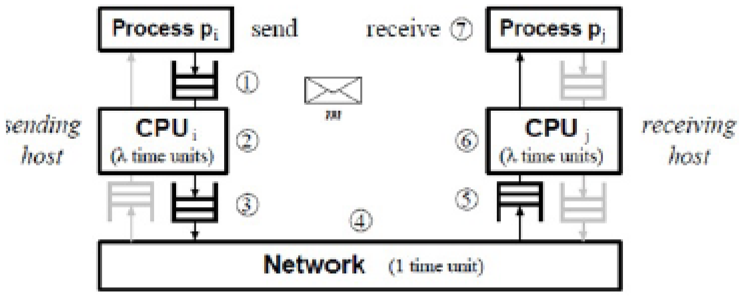

Figure 1: The simulation model.

- $\lambda = 10$: Indicates that the processor processing is higher compared to network transmission;
- $\lambda = 0.1$: Indicates that network transmission is higher compared to processor processing.

6.2. Simulation results

In this section, we present the simulation results obtained by comparing the proposed protocol with two other protocols existing in the literature [3] (referred to as $CT96$) and [13] (referred to as $ES11$). To undertake this task, we consider two execution scenarios:

1. Execution without process crashes (failure-free execution),
2. Execution with process crashes.

6.2.1. Performance Metrics

We consider a single metric for evaluating the performance of our protocol in two execution scenarios, with and without process crashes. This metric is the Latency, denoted as L, which is measured by varying the overall atomic emission rate, referred to as throughput (T: Representing the number of messages broadcast per second). For a single atomic emission, the latency L is calculated as follows:

$$L = \frac{1}{n}\left(\sum_{i=1}^{n-1} t_i\right) - t_a$$

Where:

- $i \in 0, 1, ..., n-1$: Represents the number of processes in the system;
- t_a: Represents the time of broadcasting $R - Broadcast(m)$;
- t_i: Represents the time of consumption $A - deliver(m)$ (message validation by the process p_i).

6.2.2. Execution without process crashes

In this section, we present the simulation results in terms of latency/ throughput. We consider a number of processes $n = 5$ for our proposal and that of $CT96$, and $n = 7$ for $ES11$, [1], while varying the values of λ (0.1, 1, 10), in a scenario without process failures.

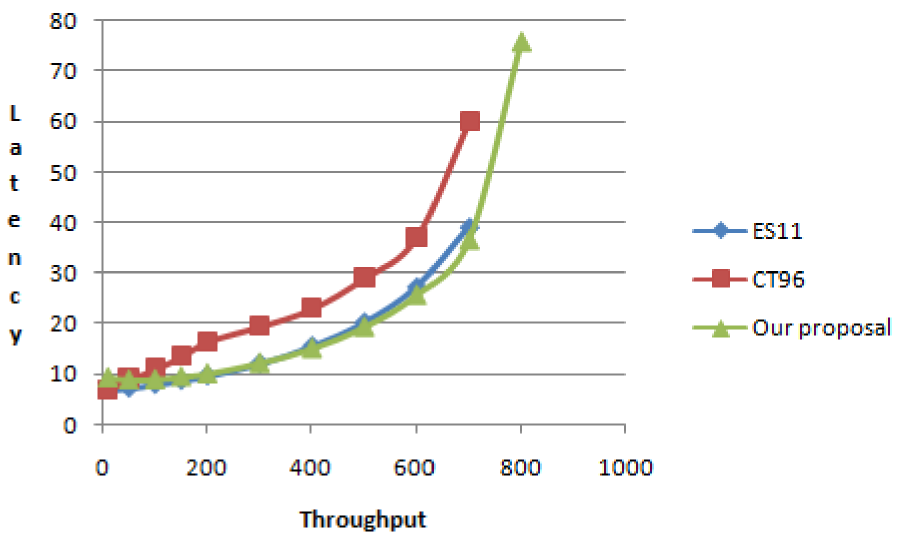

Figure 2: Latency vs Throughput: Execution without crashes (λ = 0.1).

Through the obtained results in Fig. 2, Fig. 3, and Fig. 4, we notice that our proposal outperforms in terms of latency compared to $CT96$ and $ES11$. In the case where $\lambda = 0.1$, our proposal practically overlaps with that of $ES11$. As soon as the throughput exceeds 800 msgs/s (Fig. 4), the protocol $ES11$ reaches its limits, unlike our protocol which experiences a considerable increase but remains operational.

6.2.3. Execution with process crashes

In this section, we present the simulation results in terms of latency and throughput. We consider $n = 5$ as the total number of processes for our proposal, $n = 7$ for $ES11$, and $f = 2$ as the maximum number of crashed processes for both protocols. We also vary the values of λ (0.1, 1, 10) in an environment with process failures.

Through the obtained results in Fig. 5, Fig. 6, and Fig. 7, we observe that $ES11$ outperforms our proposal in the case of low throughput. However, once this throughput exceeds a certain threshold, the latency of our protocol improves significantly, while the protocol of $ES11$ becomes non-operational.

[1]Since the number of processes required to support two failures is $n = 7$ in [13], and $n = 5$ in our proposal and [3]

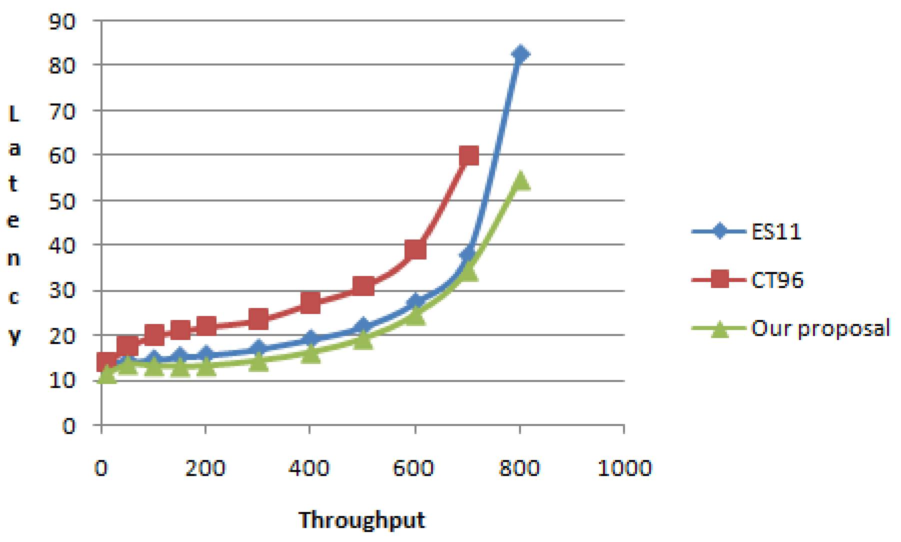

Figure 3: Latency vs Throughput: Execution without crashes ($\lambda = 1$).

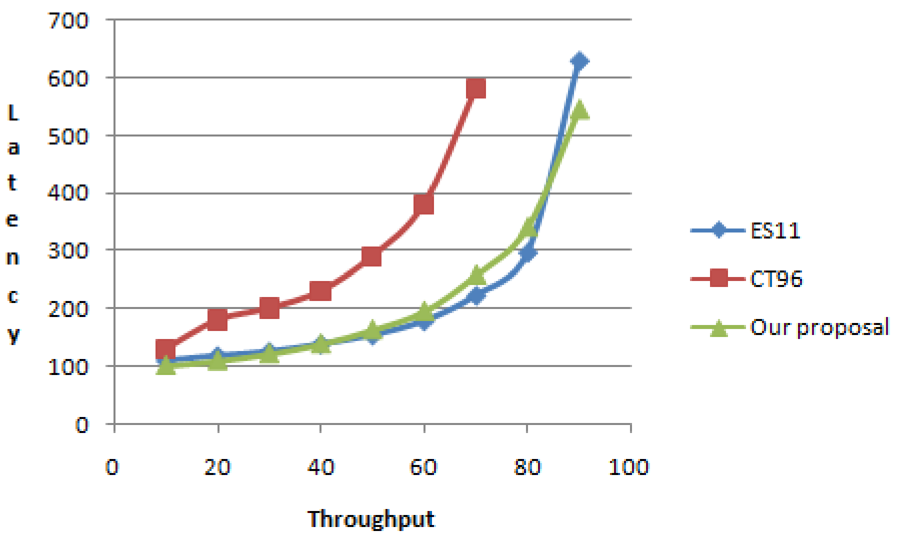

Figure 4: Latency vs Throughput: Execution without crashes ($\lambda = 10$).

7. Conclusion

This paper proposed a distributed protocol addressing the atomic broadcast problem in an asynchronous system. This protocol combines two approaches for message ordering and fault tolerance, including the rotating coordinator principle and the unreliable failure detector $\diamond S$. Through simulation results obtained using the distributed systems simulator Neko, we observed

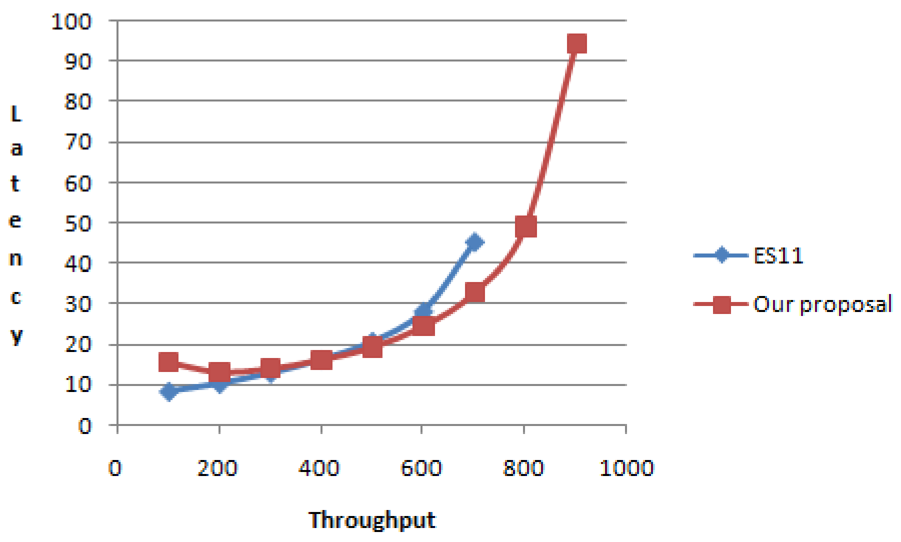

Figure 5: Latency vs Throughput: Execution with crashes ($\lambda = 0.1$).

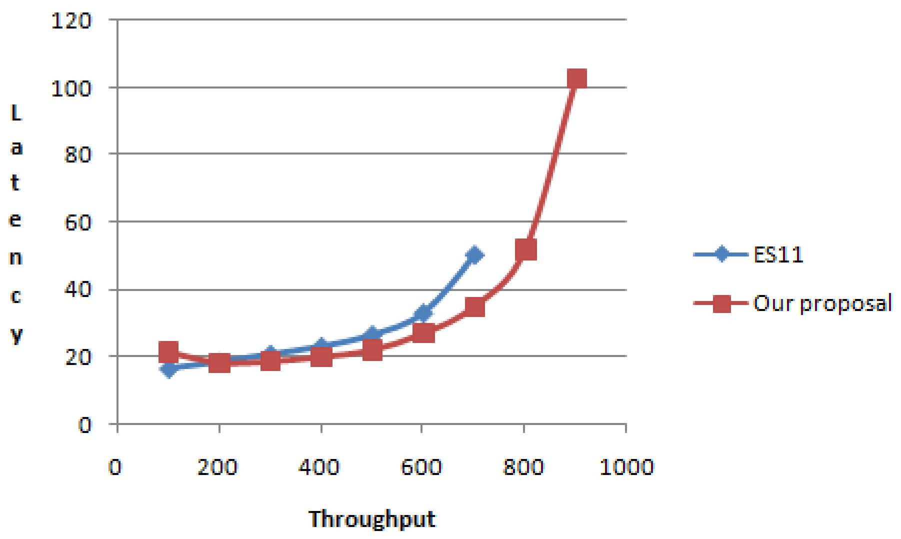

Figure 6: Latency vs Throughput: Execution with crashes ($\lambda = 1$).

that our proposal exhibits a remarkable improvement in terms of protocol latency and the number of messages generated compared to two other reference protocols implemented in the same simulator.

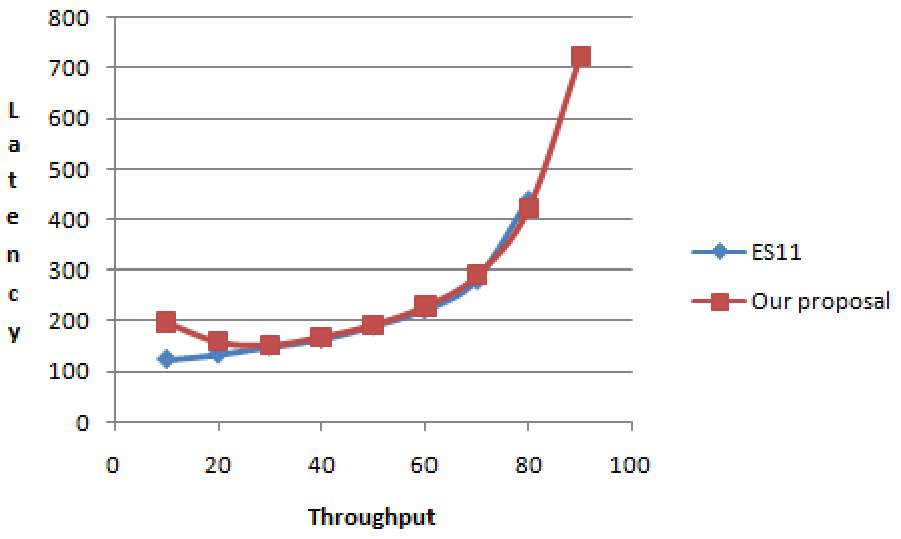

Figure 7: Latency vs Throughput: Execution with crashes ($\lambda = 10$).

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Bitcoin.org (2008). URL: https://bitcoin.org/bitcoin.pdf.

[2] R. Wattenhofer, Blockchain science: Distributed ledger technology, Inverted Forest Publishing (2019).

[3] T. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM (JACM) 43 (1996) 225–267.

[4] V. Hadzilacos, S. Toueg, A modular approach to fault-tolerant broadcasts and related problems, Technical Report, Technical Report, Cornell University, 1994.

[5] M. Ferdous, M. Chowdhury, M. Jabed, M. Hoque, A survey of consensus algorithms in public blockchain systems for crypto-currencies, Journal of Network and Computer Applications 182 (2021) 103035.

[6] X. Defago, A. Schiper, P. Urban, Total order broadcast and multicast algorithms: Taxonomy and survey, ACM Computing Surveys (CSUR) 36 (2004) 372–421.

[7] Y. Xiao, N. Zhang, W. Lou, Y. Hou, A survey of distributed consensus protocols for blockchain networks, IEEE Communications Surveys & Tutorials 22 (2020) 1432–1465.

[8] R. Ekwall, A. Schiper, A fault-tolerant token-based atomic broadcast algorithm, IEEE Transactions on Dependable and Secure Computing 8 (2010) 625–639.

[9] T. Chandra, V. Hadzilacos, S. Toueg, The weakest failure detector for solving consensus, Journal of the ACM (JACM) 43 (1996) 685–722.

[10] J. Chang, N. Maxemchuk, Reliable broadcast protocols, ACM Transactions on Computer Systems (TOCS) 2 (1984) 251–273.

[11] Y. Amir, L. Moser, P. Melliar-Smith, D. Agarwal, P. Ciarfella, The totem single-ring ordering and membership protocol, ACM Transactions on Computer Systems (TOCS) 13 (1995) 311–342.

[12] P. Jalili-Marandi, M. Primi, N. Schiper, F. Pedone, Ring paxos: High-throughput atomic broadcast, The Computer Journal 60 (2017) 866–882.

[13] R. Ekwall, A. Schiper, A fault-tolerant token-based atomic broadcast algorithm, IEEE Transactions on Dependable and Secure Computing 8 (2011) 625–639.

[14] N. Rebouh, A. Ifeticene, N. Aidoun, L. Bouallouche-Medjkoune, Failure detector-ring paxos-based atomic broadcast algorithm, International Journal of Critical Computer-Based Systems 7 (2017) 78–90.

[15] A. Mostefaoui, M. Raynal, Low cost consensus-based atomic broadcast, in: Proceedings. 2000 Pacific Rim International Symposium on Dependable Computing, IEEE, 2000, pp. 45–52.

[16] M. Yin, D. Malkhi, M. K. Reiter, G. Gueta, I. Abraham, Hotstuff: Bft consensus with linearity and responsiveness, in: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, 2019, pp. 347–356.

[17] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, E. Wong, Zyzzyva: speculative byzantine fault tolerance, in: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, 2007, pp. 45–58.

[18] S. Liu, P. Viotti, C. Cachin, V. Quema, M. Vukolic, {XFT}: Practical fault tolerance beyond crashes, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 485–500.

[19] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich, Algorand: Scaling byzantine agreements for cryptocurrencies, in: Proceedings of the 26th symposium on operating systems principles, 2017, pp. 51–68.

[20] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, in: OsDI, volume 99, 1999, pp. 173–186.

[21] P. Urban, X. Defago, A. Schiper, Neko: A single environment to simulate and prototype distributed algorithms, in: Proceedings 15th International Conference on Information Networking, IEEE, 2001, pp. 503–511.

[22] P. Urban, X. Defago, A. Schiper, Contention-aware metrics for distributed algorithms: Comparison of atomic broadcast algorithms, in: Proceedings Ninth International Conference on Computer Communications and Networks (Cat. No. 00EX440), IEEE, 2000, pp. 582–589.

[23] P. Urban, I. Shnayderman, A. Schiper, Comparison of failure detectors and group membership: Performance study of two atomic broadcast algorithms, in: 2003 International Conference on Dependable Systems and Networks, 2003. Proceedings., IEEE Computer Society, 2003, pp. 645–645.