ReLI: Real-Time Lightweight Byzantine Consensus in Low-Power IoT-Systems

Himanshu Goyal, Manish Kausik H, Sudipta Saha
School of Electrical Sciences, Indian Institute of Technology Bhubaneswar.
Email: {*hg11*, *mkh10*, *sudipta*}@iitbbs.ac.in

***Abstract*—IoT/WSN assisted smart-systems are making our living easier and more comfortable in various aspects. However, there is always a chance of malfunctioning in such massive decentralized systems in crucial moments because of one or more components of the system getting compromised. For instance, monitoring systems installed to watch the status of a bridge may unknowingly suppress the recent deterioration in the status because of some compromised sensing devices. Byzantine fault tolerance support is highly essential in combating the presence of such smart devices with malicious intentions. However, existing solutions for consensus or data aggregation in IoT/WSN systems either assume non-Byzantine node failures or use only simulation/theoretical models to address the existence of Byzantine nodes. Theoretically, a decentralized system can effectively tolerate Byzantine characteristics of up to a certain fraction of the nodes. However, to achieve even that, the nodes need to interact extensively and share data with each other which makes it challenging for such solutions to get practically realized and produce outcomes in real-time, especially in resource-constrained IoT systems. In this work, we adopt *Synchronous-Transmission* based mechanisms and propose a framework *ReLI* to efficiently achieve Byzantine consensus in low-power IoT systems. We show that ReLI can operate up to 80% faster and consume up to 78% lesser radio-on time compared to the traditional implementation of the strategy in a publicly available IoT/WSN testbed containing 45 nodes.**

***Index Terms*—Byzantine Fault Tolerance, Synchronous-Transmission, Concurrent-Transmissions, Wireless Sensor Networks, Internet of Things (IoT).**

I. Introduction

Human civilization tends to depend more on the use of smart systems. Technologies such as Internet-of-Things (IoT) and Wireless Sensor Networks (WSN) play crucial roles in building these smart systems. A smart system operate through decentralized collaboration among a large number of independent IoT devices. For instance, in an *automated surveillance system*[1], the independent devices equipped with a camera or microphone are installed at various places to cover the area under surveillance and collaboratively monitor the area. Similarly, many devices with appropriate sensing capabilities are installed in a structure monitoring system [2], covering the vital parts of the structure (e.g., a bridge or a building). The status of the target structure is obtained through a collaborative effort among these devices. Similar scenarios can be obtained from other smart systems [3] such as smart-grid, intelligent-transportation systems, industry-4.0, etc.

Unfortunately, any IoT device in any such smart system is susceptible to various unexpected issues, e.g., software errors, failures, or even various types of attacks. These issues can significantly disturb the overall integrity of a smart-system and may also result in false reporting while interacting with the end-users-leading to catastrophic effects. For instance, while monitoring a critical structure, a compromised IoT device may wrongly report the status of a component. Similarly, in a flock of UAVs with some mission, such issues may cause some of the UAVs to deviate from the target, which in turn may lead to an overall failure of the mission incurring a significant loss.

Fault-tolerant consensus protocols [4] play a vital role in establishing the trustworthiness of a system despite node failures. Such failures can be either *non-Byzantine* or *Byzantine*. Consensus protocols to handle non-Byzantine failure assume a weak failure model, e.g., simple *fail-stop* or *node crash*. They are primarily used in systems within a controlled environment, e.g., a data center, where only the authenticated users operate them. In contrast, consensus protocols capable of handling *Byzantine* failures [5], entail more comprehensive failure models where nodes can be compromised and have malicious intent behind participation. They are applicable for both open and decentralized systems, e.g., BlockChain [6].

Unlike handling non-Byzantine failures, the challenges in dealing with Byzantine failures are quite different and much harder to address. The works [7, 8] show how to manage non-Byzantine failures in low-power IoT-system efficiently just by using a network-wide *max finding operation* where the nodes do not need to share the actual data with each other. However, in contrast, to manage Byzantine faults, the devices do need to share the actual opinions/messages to come to a conclusion that naturally involves multiple rounds of many-to-many/all-to-all data-sharing among the nodes. This makes the protocol quite complex and communication-intensive for resource-constrained IoT/WSN systems.

RF transmissions under traditional *Asynchronous-Transmission* (AT) based communication strategies are primarily independent and uncoordinated. Due to the broadcast-driven nature of the wireless medium, AT-based communications, therefore, incur quite high chance of collision among the packets transmitted by different nodes, especially when data traffic goes higher. Complex applications like many-to-many/all-to-all data-sharing, hence, drastically degrade under AT with the increase in the number of source nodes. In contrast, in many recent works *Synchronous-Transmission* (ST) based strategies [9] have shown their superiority over AT. In particular, under ST, the transmissions of the packets from different nodes are explicitly timealigned with each other. This helps the protocols to exploit special physical layer phenomena called *Capture-Effect* (CE)/ *Constructive-Interference* (CI) to achieve highly reliable end-to-end data-sharing consuming very low-energy in the devices. In this work we adopt ST based communication mechanisms to design a framework, referred to as *ReLI*, to mitigate the Byzantine failures in low-power IoT-systems.

The primary contribution of the work is summarized below.

- We leverage ST-based communication mechanisms to design an efficient framework ReLI to accomplish Byzantine fault tolerance for resource-constrained low-power IoT systems.
- Special design considerations are adopted to make the strategy scalable and further optimize the overall completion time as well as energy consumption in the devices.
- ReLI has been implemented in Contiki operating system for TelosB devices. It has been extensively tested in both network simulator as well as publicly available IoT/WSN testbeds.

The rest of the paper is organized as follows. The section II provides a glance at the existing works to address non-Byzantine and Byzantine fault-tolerant protocols and their applicability in the context of low-power IoT systems. Section III provides the necessary background regarding Byzantine fault tolerance as well as the ST-based data-sharing strategies used in this work. Section IV provides details of the design and implementation of the proposed framework ReLI. Finally, Section VI demonstrates an in-depth evaluation of ReLI in network simulator and testbeds.

II. Related Work

Non-Byzantine node failures have been considered in several works [10]. Paxos [11] is one of the earliest and most well-known non-Byzantine consensus protocols, which is later expanded to the protocol Raft [10]. Both Raft and Paxos have been extensively used in many permission-based environments, e.g., Google's globally distributed File System [12], BlockChain enabled Hyperledger Fabric, [13] etc. Byzantine consensus is a significant component in the implementation of BlockChain technology [4]. For instance, BlockChain-assisted crypto-currency BitCoin [6] uses Proof-of-work (PoW); PPCoin[14] uses Proof-of-Stake (PoS) to mitigate the possibility of Byzantine node failures. However, such solutions are computation intensive and it make them unfeasible for IoT systems.

Efficient many-to-many interactions, data sharing, and consensus in resource-constrained IoT/WSN systems have been addressed in many works [9]. However, very few of these works deal with the issue of fault tolerance. A recent work, *Wireless-Paxos* [8], ports the protocol Paxos [11] to a low-power IoT/WSN system. It achieves an in-network fault-tolerant consensus solution by leveraging the aggregation property of another protocol, Chaos [15]. However, it targets only crash faults and cannot take care of Byzantine failures. Some recent works [16, 17, 18] try to bring the BlockChain service to IoT devices where the Byzantine fault-tolerant consensus is carried out with the help from the cloud. Such split architecture, although works, induces a significant delay and potential issues under higher demand.

The work *Practical Byzantine Fault Tolerant* [19] (PBFT) consensus demonstrates a substantially different approach. In particular it shows how Byzantine fault tolerance can be achieved through appropriate collaboration among the participants instead of solely depending on specific issues such as computation capabilities in the participating nodes (as done by PoW, PoS). Gradually PBFT became quite popular. The work [20] shows the application of PBFT in BlockChain-based audit systems to solve multiple security concerns in the consensus algorithms. The work [21] done at Renault Automobile Corp. demonstrates the application of PBFT for the smooth processing of insurance claims. PBFT has also gained attention in industrial IoT [22] applications.

However, communication overhead has been one of the significant challenges in realizing PBFT for real-world smart-systems. There have been attempts to improve the performance of the basic PBFT protocol. For instance, the work [19] reduces the complexity from exponential [23] to polynomial. A class of works [24, 25] has been done to make the protocol suitable for IoT systems. For instance, Pengs et al. [26] propose a credit-based mechanism using reinforcement learning to reduce the communication among the participants. The set of works [27, 28] attempts to make PBFT scalable by adopting a divide-and-conquer approach where the system is first divided into multiple layers/groups and subsequently, the problem is addressed separately in each of these groups. Finally, the group leaders collaborate with each other to come to a conclusion. However, these works only show a simulation or theoretical validation of the concepts. In contrast, in the current work, we take an endeavor to design a lightweight, low-latency, and scalable realization of PBFT for real-IoT systems.

III. Background

A class of existing solutions to mitigate the existence of Byzantine nodes exploit the computational capability of the participating nodes. However, a typical IoT/WSN device mostly lacks enough processing capability to realize these strategies. The approach shown in PBFT [19] demonstrates how to achieve Byzantine Consensus with only inter-device communication. Unfortunately, strict energy-constraint in the tiny devices used in IoT/WSN systems heavily restricts their communication ability. In addition, uncoordinated transmissions in traditional *Asynchronous-Transmission* (AT) based protocols in these systems waste a lot of throughput and energy in the devices due to collisions among the packets, making it even harder to meet the communication requirements to realize PBFT.

Recently there have been quite a lot of developments in ST-based strategies [9]. These works demonstrate high reliability and low-latency communication in systems comprising a large number of nodes. We leverage ST in designing our framework ReLI to achieve PBFT in IoT/WSN. In the following, we first briefly describe ST and the specific protocols used in this work.

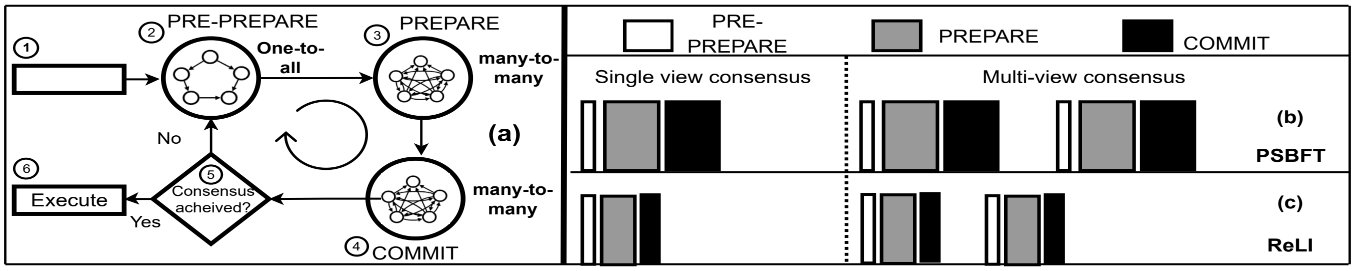

Fig. 1. (a) Flow diagram of a standard PBFT execution. Timing diagram of (b) *Practical Synchronous Byzantine Fault Tolerant* (PSBFT) and (c) ReLI.

A. ST-based protocols

Achieving tight time synchronization is the primary requirement for ST. The pioneering work Glossy [29] shows how to accomplish the same through lightweight software-based strategy and hence gain the benefit of ST in resource-constrained off-the-shelf IoT devices through a purely decentralized setting. There has been immense development in this field through many recent works [9]. In the current work, we mainly use three distinct protocols, Glossy [29], MiniCast [30], and Chaos [15], to serve our purpose. These are briefly described below.

Glossy: Glossy[29] demonstrates how ST can be exploited to achieve efficient one-to-all flooding over a large multi-hop network setting. The protocol starts with a designated *initiator* node transmitting the first packet, triggering the transmissions from its first-hop neighbors. Unlike AT, all the first-hop nodes simultaneously transmit their packets (having precisely the same content), which results in physical layer CE in the receiver radios of the second-hop nodes. The process continues this way until the flood reaches the entire network.

Chaos: Data aggregation is typical in WSN/IoT [31]. Chaos[15] exploits ST to achieve fast network-wide data aggregation. Unlike Glossy, it allows different nodes to transmit different data at the same time slot. It has been shown that specific data aggregations operations which do not depend on the number of times a node can contribute, i.e., not-sensitive to "*double counting*", such as max, min, etc., can be very efficiently computed by Chaos over an extensive decentralized system. Chaos can also carry out network-wide sharing of data from multiple source nodes. However, due to much larger packet size, it takes much longer to converge.

MiniCast: Many-to-many/All-to-all data-sharing has been an essential requirement in IoT/WSN systems. Glossy successfully conveys the data from one node to all other nodes in the network. The work LWB naively [32] achieves many-to-many sharing through sequential repetition of Glossy floods one-by-one from all the source nodes. However, wide separation among the floods makes LWB perform poorly with more nodes and large areas. Conversely, Chaos achieves many-to-many data sharing by allowing different nodes to share different data at the same time slot and taking advantage of CE to achieve all-to-all data sharing. However, the performance of data-sharing in Chaos is poor than aggregation, especially when the amount of data to be shared is large. The work MiniCast [30] achieves compact execution of multiple floods originating from different source nodes by minimizing the inter-flood gaps with the help of a packet-level TDMA schedule. MiniCast has been shown to outperform LWB and Chaos with a wide margin for all-to-all-data sharing.

As shown in [9], there are quite a few other existing all-to-all data sharing strategies. However, these works either extend LWB or Chaos or are based on the concept of MiniCast. These works mostly use multi-channel facilities or efficient network-coding strategies to enhance the performance of the base protocols. A prime part of the realization of PBFT in IoT/WSN needs to deal with massive data-sharing among the participating nodes. Therefore, in-principle any efficient data-sharing strategy can work. However, use of these protocols in raw mode will not bring the desired performance because of massive requirements and scalability issues. MiniCast has been used to serve the need in several recent works in different domains, e.g., Electric-Vehicles [33], SmartGrid [34, 35], Multi-Party-Computation [36]. There has been recent works to show how MiniCast can be made more efficient and scalable too [37, 38, 39]. To keep it simple and focus on these prime targets, in this work, we exploit MiniCast, and Glossy to fulfill the needs. We also use Chaos to support a special need in the design, as detailed in Section IV.

B. PBFT

To resolve non-Byzantine failures, first, the query under consideration is shared with all the nodes in the system, and then replies are collected. Since the nodes are not supposed to reply incorrectly, the reception of at least 50% of the system participants would ensure a consensus. In contrast, under Byzantine failures, a node forms agreement by discussing the issue with the other nodes in the system. Specifically, in PBFT, how many other nodes are replying with the same answer for a particular query is considered as a crucial issue. Formally, a *quorum* is defined as *the minimum number of nodes to pass a decision in an assembly*. In PBFT, a node to achieve consensus amidst various types of failures needs to reach a *quorum*. The minimum number of nodes necessary to be present in a system to enable every node to reach a quorum despite the presence of at most f non-Byzantine failures is $2f + 1$ [19]. Conversely, under non-Byzantine failures, for N number of nodes, the quorum size is $\frac{N}{2} + 1$. Under Byzantine failures, in contrast, quorum size is $\frac{2N}{3} + 1$ [40].

A node that requests for consensus is referred to as a *client node*. The request from the client is conveyed to a designated *primary node* which triggers the consensus process composed of five phases, as depicted in Fig. 1(a). A simplified description of these phases is given below.

1) **LAUNCH:** A client node (c) initiates the process by sending a request message *Req* to the primary node.
2) **PRE-PREPARE:** The primary node broadcasts *Req* to all the participating nodes.
3) **PREPARE:** All the nodes in the system validate the correctness of the request in *Req* and broadcast a *Prepare* message in the system conveying their opinion about it. In this work we assume that the opinion shared by the nodes follow a specific application dependent format [41].
4) **COMMIT:** In this phase, every node first individually decides whether it has reached quorum or not based on the number of similar messages received in the previous phase. If the number of alike messages is more than $2f+1$, a node decides to go for commit otherwise, not. In this phase also, the nodes share their decisions with each other. We assume the data to be shared in this phase to be binary (1 for YES and 0 for NO).
5) **EXECUTE REPLY:** After COMMIT, every node checks how many nodes have achieved consensus. The client considers its request *Req* to achieve consensus if it receives $f+1$ similar opinions. Thus, if $2f+1$ nodes have reached quorum, a global consensus is assumed to be achieved. However, when consensus is not achieved, it goes to the VIEW-CHANGE phase.
6) **VIEW-CHANGE:** In PBFT, the three consecutive phases PRE-PREPARE, PREPARE, and COMMIT together are considered to be a *view* formed under the leadership of the current primary node. If a certain view fails to result in consensus, a new primary node is decided based on a globally defined rule. Next, the new primary node starts the change of view and restarts the process with the PRE-PREPARE phase. With f number of traitors in the system, in the worst-case maximum $f+1$, view formations may be needed to achieve a consensus.

In the following, we describe the design of ReLI. We assume the sink/initiator node itself to play the role of a client as well as the initial primary node.

IV. Basic Idea

The three phases employed by PBFT are pictorially explained in Fig. 1(a). In the PRE-PREPARE phase, the primary node carries out a network-wide *one-to-all* sharing of the query message while both the PREPARE and the COMMIT phase execute a network-wide all-to-all interaction among the nodes to exchange the opinion regarding the message, and check how many nodes could reach the consensus, respectively. All these phases thus involve an *extensive number of messages passing among the nodes*. Moreover, possible change of view may happen several times, which would bring *many repetitions of the three phases*. ST-based data-sharing protocols fundamentally try to optimize the communication process by minimizing the packet collision and thus reducing the use of the RF transceivers in the energy-constrained IoT devices. However, to fulfill such heavy communication needs to achieve Byzantine fault tolerance in IoT, even ST-based protocols, if used in a very straightforward/naive way, may quickly drain a considerable amount of energy in low-power IoT/WSN devices.

Therefore, to optimize the performance, we make an in-depth study of the Byzantine consensus process, especially under ST, and come up with a set of observations as summarized below.

A. All-to-all vs. many-to-many

Time and energy consumed in any phase in PBFT heavily depend on the constraint that how many opinions a node must acquire to proceed. In PREPARE and COMMIT phase of PBFT every node needs to acquire enough alike messages to reach quorum. From the definition of the quorum size for Byzantine consensus, it can be observed that strict all-to-all data-sharing may not always be necessary. Rather, it depends on the actual number of traitors or faulty nodes. In a system with N nodes, although PBFT can support max f traitors, where $N = 3f + 1$, the actual number of traitors present in the system in practice may be much lesser than f, say k ($k < f$). In this situation, the necessary size of the quorum is $2k+1$. Therefore, it's enough to ensure that each node receives at least $2k + 1$ alike messages to reach quorum.

Note that for k traitors, the number of distinct messages/opinions that may differ from each other would be maximum k. Thus, by the *pigeonhole principle*, it can be said that a node would require to wait for messages from at max $3k + 1$ distinct nodes, which ensures at least $2k + 1$ messages from the non-traitor nodes, and hence alike. For a random distribution of k traitors, a node may need to wait for even less as it may receive $2k + 1$ alike messages much earlier. Thus, *knowledge regarding the number of traitors as well as their distribution in the system can be exploited for easier and faster convergence*. We use this concept to simplify the PREPARE phase further.

From a design point of view, a data-sharing mechanism must provide enough control on the data-sharing process so that system-wide execution can be done only until the desired degree of coverage is achieved. In an ST based protocol, there is a common parameter known as NTX which controls the reliability of the process. Fundamentally it determines how many times a node is supposed to repeat/forward the packets from other nodes. In MiniCast it explicitly controls the outreach of the floods originated from each source node. In this work we appropriately tune the value of NTX to accomplish the goal.

B. Restricted behavior of traitors under ST

Under traditional wired communication systems, unicast is the default communication mode. A traitor node in unicast gains enough scope to communicate differently to each of its neighboring nodes as a response to a given query. In a wireless medium, any communication happens by default in broadcast mode. However, under AT, broadcasts are often abstracted as unicast, which again can be exploited by the traitors to create confusion. In contrast, ST explicitly uses a

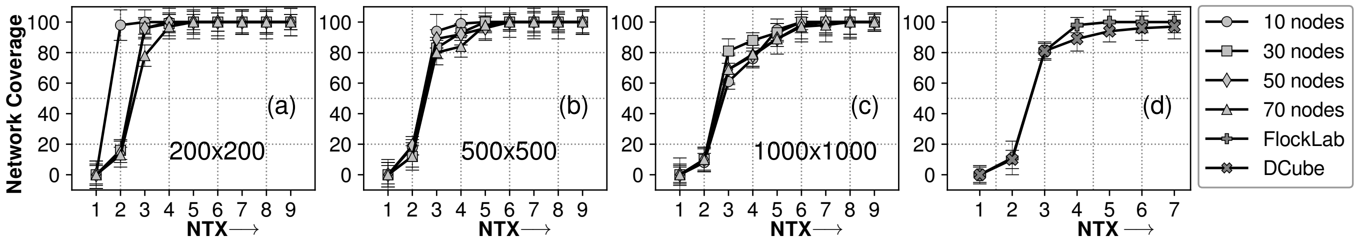

Fig. 2. Percentage of data received by the nodes with the variation of NTX.

broadcast-driven communication framework to achieve a faster spread of information. Under ST, the traitors, in order to avoid quick detection and identification, hence, are kind of forced to behave consistently to all of their neighbors. In addition, if a traitor tries to share different data in consecutive or different time-slots, it results in disturbances in any ST based process that are easily visible. Thus, in turn, ST refrains a traitor from changing the data while even forwarding. In conclusion, thus, ST, in general, severely restricts the Byzantine capability of a traitor. However, note that the traitors are still free to pass wrong/imperfect opinions in their own packet/slot.

Based on these observations, we optimize the PBFT consensus protocol and develop our proposed framework ReLI.

V. Protocol Design

The main design is discussed in the following section. The PRE-PREPARE phase is designed using a customized instance of Glossy. The query under consideration is first disseminated by the initiator node (primary) to all the other nodes in the system. Execution of Glossy, in general, happens very fast over a large network setting. Subsequently, the nodes start the PREPARE phase, where everyone shares its opinion regarding the query with each other. An instance of the protocol MiniCast [30] is used for this phase, as detailed below.

PREPARE: This phase is supposed to use a round of MiniCast based data-sharing among the nodes. A random distribution of the traitors is assumed. The primary node first makes an assessment of the approximate number of the traitors in the system (say k). Based on k, it also derives the desired value of NTX (see Section V-A for details) and shares the same with the query packet (in PRE-PREPARE phase). In the design of ReLI we tune the value of NTX appropriately so that every node obtains enough opinions from its neighbors to reach the quorum while the latency of the PREPARE phase is minimized. The exact steps are detailed in the following section. During PREPARE phase, every node keeps track of how many alike messages are received.

COMMIT: In the COMMIT phase, it is enough to share a single bit of information from each node indicating whether the node could reach the quorum in the PREPARE phase. In particular, a node sets a flag in case the variable $nAlike$ goes above $2k+1$. Since no explicit data sharing is needed in this phase, we use a customized version of the protocol Chaos to serve the purpose. Chaos originally uses a single flag bit for every node to track how many nodes have contributed to the aggregation process. The execution stops when all the nodes have contributed successfully. To serve our purpose, we add one more flag bit for each node. It is set to 'one' in case the corresponding node reaches the quorum. Thus, using just two bits of contributions from each node allows the whole COMMIT phase to run very fast and consume much less energy even when the number of nodes is large.

VIEW-CHANGE or NTX-reassessment: At the end of the COMMIT phase, every node in the network gets to know how many nodes in the system have reached the consensus. In case if the count goes higher or equal to $3k+1$, a global consensus is assumed to be achieved. However, there can be two distinct cases for consensus not getting reached. It may be either due to a *faulty primary node* or *due to insufficient spread of information caused by limited execution of MiniCast w.r.t the number of traitors in the system.* Since exploring the exact reason is not possible, at this stage, we first repeat the PREPARE phase with the value of NTX one more than the last used value (i.e., set NTX = NTX+1). We observe the change in the COMMIT phase. If considerably more number of nodes are found to be reaching quorum with the increased NTX, it indicates a wrong assessment of the traitors in the previous round. To rectify the same, we repeat the process until a global consensus is reached. We save this new NTX and use it next time onward. In case significant improvement is not visible after increasing NTX by 1, the system goes for VIEW-CHANGE, i.e., the primary node is changed as per a globally pre-decided rule, and the same is repeated at max k+1 number of times until consensus is reached.

Correct assessment of NTX is, thus, a very significant step in the whole process. In the following, we detail this specific issue.

A. Restricted data-sharing

During execution of MiniCast, transmissions happen in chain of packets. The marker packets HD (Header) and TL (Trailer) indicate the beginning and end of a chain. It starts with the initiator node transmitting the data packet received by the first-hop nodes. The chain transmitted by the first-hop nodes triggers the transmission by the second-hop nodes and so on. To optimize the data-sharing process in ReLI, NTX is set to the minimum value that is enough for every node to reach the quorum. Appropriate tuning of NTX brings a huge difference in the performance as depicted in Section VI.

Local-coverage in MiniCast: When MiniCast runs with a low NTX, every node is still supposed to successfully receive a certain amount of data from its local neighborhood. To verify this, we first study the behavior of MiniCast in Contiki network simulator called Cooja and IoT/WSN testbeds, FlockLab [42] and DCube [43]. Fig. 2 highlights the rise in the network cov-

TABLE I
ASSESSMENT OF NTX UNDER DIFFERENT EXPERIMENTAL SETTINGS.

Failure percentage (%)	**Simulation**			**Testbeds**	
	200X200	***500X500***	***1000X1000***	***FlockLab***	***DCube***
0	2	2	3	2	2
33	3	3	4	3	3
66	3	4	5	3	5
100	4	5	7	6	7

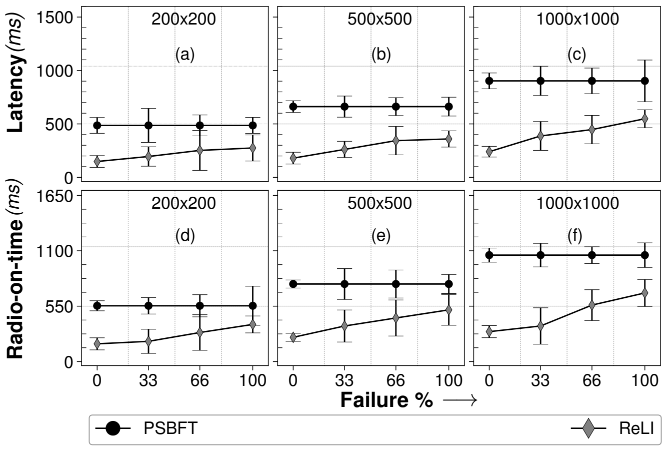

Fig. 3. Latency and Radio-on-time in a 70-node simulation of PSBFT and ReLI in over various network diameter.

erage with NTX for different network configurations. Based on the number of nodes and the simulation area in Cooja, while the value of NTX required to get 100% coverage is above 7, to achieve even upto 70% coverage the required value of NTX is found to be quite lesser. Fig. 2 (a)-(c) show these results. The same trend is visible in testbed networks of DCube and FlockLab, as shown in Fig. 2 (d). It is also visible that the exact relationship depends on the specific network setting. Therefore, deciding the correct value of NTX surely improves the protocol performance.

B. PBFT based on naive use of ST

To compare the performance of ReLI, we also design another platform where the ST-based primitives are applied in a straightforward way without any optimization. We refer to this strategy as *Practical Synchronous Byzantine Fault Tolerant* or PSBFT protocol. It uses one Glossy instance for PRE-PREPARE and two consecutive MiniCast instances for PREPARE and COMMIT phases. In all these MiniCast instances, we set NTX as the maximum value necessary in the target network. Moreover, Fig. 1(b) and Fig. 1(c) show the timing diagram of PSBFT and ReLI, respectively.

VI. EVALUATION

We implement both PSBFT and ReLI in Contiki OS for TelosB motes. We experiment with the implementations in both Cooja as well as IoT/WSN testbeds DCube and FlockLab composed of 45 and 24 TelosB motes, respectively. The success of ReLI largely depends on the right assessment and determination of NTX. Therefore, we first carry out a detailed evaluation of the NTX-assessment, but due to space limitation, we skip the detailing. Table I shows the final values of NTX obtained from this study. These are used for further experiments with ReLI.

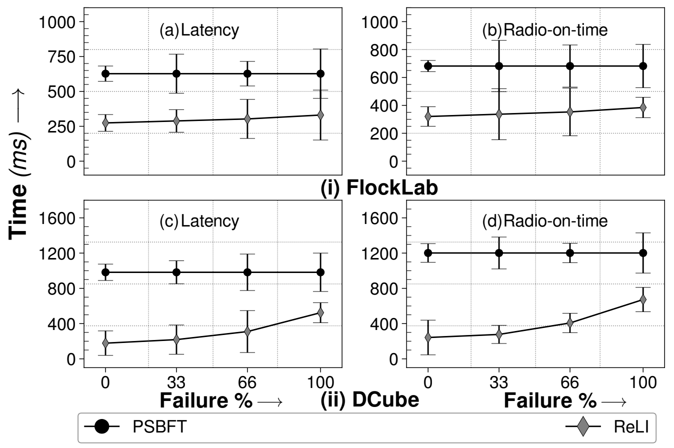

Fig. 4. Latency and Radio-on-time in execution of PSBFT and ReLI in FlockLab and DCube.

A. Metrics

To compare the performance of ReLI and PSBFT, we use following two metrics.

Latency: It is the time taken for a process to achieve the consensus.

Radio-on time: It is the total time necessary for a node to keep its radio ON to complete the execution of a protocol. It is used to understand the energy consumption in a device. The metrics are calculated in each device. They are presented as an average over all the devices and all the iterations.

B. Results

ReLI and PSBFT are compared under the presence of the Byzantine traitors in different network configurations through simulation as well as testbeds. The number of traitors is varied from zero to the maximum number that can be supported by the algorithm (i.e., one-third of the number of nodes). Overall, in all cases ReLI is found to be performing much better than PSBFT, even when there are many traitors in the system.

Fig. 3 shows comparison results through simulation of the protocols in a 70-node network over different deployment areas, namely, 200×200 m^2, 500×500 m^2, and 1000×1000 m^2. ReLI is found to be showing maximum improvement when the deployment area is the largest. In particular, it performs on average 55% and up to 73% faster and consumes on average 53% and up to 71% lesser radio-on time compared to PSBFT for a wide variation in the number of traitors assumed to be randomly distributed over the network. Intrinsically, higher area of deployment of the same number of devices increases the diameter of the network which in turn cause MiniCast to take substantially more time to carry out all-to-all communication in PSBFT. In contrast, ReLI leverages the concept of local coverage in MiniCast and manages to acquire the minimum necessary messages only from the surroundings accomplishing the goal faster.

In Flocklab, ReLI is found to perform on average 51% and up to 55% faster while consuming on average 48% and up to 53% lesser radio-on time compared to PSBFT. Similarly, in DCube, ReLI performs on average 68% and up to 80% faster while consuming on average 66% and up to 78% lesser radio-on time compared to PSBFT. Note that the number of nodes in DCube is almost double compared to FlockLab. This is one of the prime reasons behind PSBFT taking larger time in DCube than FlockLab. To further understand the scenario, we separately run neighbor discovery protocol in both the tetsbeds separately and found that compared to FlockLab, DCube is substantially denser with many stronger links among the nodes. This demonstrates that even under a large network setting, ReLI can successfully exploit the strong connectivity links in the network to execute substantially faster than PSBFT.

VII. Conclusion

Increasing dependence on smart systems not only brings ease to living but also introduces the possibility of unexpected problems because of the presence of smart traitors in the system. Under resource-constrained IoT-setting its quite challenging to resolve such issues in general. Existing consensus mechanisms for IoT/WSN can manage only non-Byzantine faults. Byzantine faults, in contrast, has been either shown through theory/simulations or are designed in a way that is not suitable for low-power IoT-systems. To the best of our knowledge, the current work is the first attempt to design an efficient framework for execution of *Byzantine fault resilient consensus* efficiently and practically in the low-power IoT/WSN systems. To achieve the goal, we leverage Synchronous-Transmission based communication mechanism and propose strategies to optimize the performance under practical settings. Clever use of frequency allocation [44] or Start-of-frame delimiter [45] based lightweight separation of the tasks can further improve the performance of the proposed strategy which is one of the future directions of the current work.

References

[1] V. A. Memos, K. E. Psannis, Y. Ishibashi, B.-G. Kim, and B. Gupta, “An efficient algorithm for media-based surveillance system (eamsus) in iot smart city framework,” *Future Generation Computer Systems*.

[2] A. P. Plageras, K. E. Psannis, C. Stergiou, H. Wang, and B. Gupta, “Efficient iot-based sensor big data collection–processing and analysis in smart buildings,” *Future Generation Computer Systems*.

[3] V. C. Gungor, B. Lu, and G. P. Hancke, “Opportunities and challenges of wireless sensor networks in smart grid,” *IEEE Trans. Ind. Electron.*

[4] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, “Survey of consensus protocols on blockchain applications,” in *IEEE ICACCS*, 2017.

[5] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Trans. Program. Lang. Syst.*, 1982.

[6] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.”

[7] B. Al Nahas, S. Duquennoy, and O. Landsiedel, “Network-wide consensus utilizing the capture effect in low-power wireless networks,” in *ACM SenSys*, 2017.

[8] V. Poirot, B. Al Nahas, and O. Landsiedel, “Paxos made wireless: Consensus in the air,” in *EWSN, 2019*.

[9] M. Zimmerling, L. Mottola, and S. Santini, “Synchronous transmissions in low-power wireless: A survey of communication protocols and network services,” *ACM Comput. Surv.*, 2020.

[10] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *USENIX*, 2014.

[11] L. Lamport, “Paxos made simple,” *ACM SIGACT News*, 2001.

[12] J. C. Corbett, J. Dean, and E. et al., “Spanner: Google’s globally distributed database,” *ACM Trans. Comput. Syst.*, 2013.

[13] E. Androulaki, A. Barger, and B. et al., “Hyperledger fabric: A distributed operating system for permissioned blockchains,” in *EuroSys, 2018*.

[14] S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake,” 2012.

[15] O. Landsiedel, F. Ferrari, and M. Zimmerling, “Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale,” in *ACM SenSys, 2013*.

[16] E. Schiller, E. Esati, S. R. Niya, and B. Stiller, “Blockchain on msp430 with ieee 802.15.4,” in *IEEE LCN, 2020*.

[17] C. Profentzas, M. Almgren, and O. Landsiedel, “Iotlogblock: Recording off-line transactions of low-power iot devices using a blockchain,” in *IEEE LCN, 2019*.

[18] C. Profentzas, M. Almgren, and O. Landsiedel, “Tinyevm: Off-chain smart contracts on low-power iot devices,” in *IEEE ICDCS, 2020*.

[19] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *OSDI, 1999* (M. I. Seltzer and P. J. Leach, eds.), USENIX Association, 1999.

[20] A. Ahmad, M. Saad, and A. Mohaisen, “Secure and transparent audit logs with blockaudit,” *J. Netw. Comput. Appl*, 2019.

[21] L. Gerrits, C. N. Samuel, R. Kromes, F. Verdier, S. Glock, and P. Guitton-Ouhamou, “Experimental scalability study of consortium blockchains with bft consensus for iot automotive use case,” in *ACM SenSys*, 2021.

[22] S. K. Dwivedi, P. Roy, C. Karda, S. Agrawal, and R. Amin, “Blockchain-based internet of things and industrial iot: a comprehensive survey,” *Security and Communication Networks*, 2021.

[23] M. Pease, R. Shostak, and L. Lamport, “Reaching agreement in the presence of faults,” *Journal of the ACM (JACM)*, 1980.

[24] J. Chang and F. Liu, “A byzantine sensing network based on majority-consensus data aggregation mechanism,” *Sensors*, 2021.

[25] Y. Meshcheryakov, A. Melman, O. Evsutin, V. Morozov, and Y. Koucheryavy, “On performance of pbft blockchain consensus algorithm for iot-applications with constrained devices,” *IEEE Access*, 2021.

[26] P. Chen, D. Han, T.-H. Weng, K.-C. Li, and A. Castiglione, “A novel byzantine fault tolerance consensus for green iot with intelligence based on reinforcement,” *JISA, Elsevier*, 2021.

[27] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, “A scalable multi-layer pbft consensus for blockchain,” *IEEE TPDS*, 2020.

[28] L. Zhang and Q. Li, “Research on consensus efficiency based on practical byzantine fault tolerance,” in *IEEE ICMIC*, 2018.

[29] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, “Efficient network flooding and time synchronization with glossy,” in *ACM IPSN, 2011*.

[30] S. Saha, O. Landsiedel, and M. C. Chan, “Efficient many-to-many data sharing using synchronous transmission and tdma,” in *IEEE DCOSS, 2017*.

[31] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “Tinydb: an acquisitional query processing system for sensor networks,” *ACM TODS*, 2005.

[32] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, “Low-power wireless bus,” in *ACM SenSys, 2012*.

[33] J. Debadarshini and S. Saha, “Efficient coordination among electrical vehicles: An iot-assisted approach,” in *IEEE INFOCOM*, 2022.

[34] J.Debadarshini and S. Saha, “Collaborative load management in smart home area network,” IEEE ICDCS 2022 [To appear].

[35] J. Debadarshini, S. Saha, and S. Samantaray, “Decentralized load management in han: An iot-assisted approach,” IEEE SmartGridComm 2022 [To appear].

[36] H. Goyal and S. Saha, “Multi-party computation in iot for privacy-preservation,” IEEE ICDCS 2022 [To appear].

[37] M. Tummala and S. Saha, “Concurrent transmission based data sharing with run-time variation of tdma schedule,” in *IEEE LCN*, 2020.

[38] M. Tummala, M. Kausik H, and S. Saha, “Flexicast: A structure-adaptive protocol for efficient data-sharing in iot,” IEEE CNSM 2022 [To appear].

[39] J. Debadarshini and S. Saha, “Divide, conquer and merge for internet-of-things,” IEEE DCOSS 2022 [To appear].

[40] M. Van Steen and A. Tanenbaum, “Distributed systems principles and paradigms,” *Network*, 2002.

[41] T. Distler, “Byzantine fault-tolerant state-machine replication from a systems perspective,” *ACM Comput. Surv.*, 2021.

[42] R. Trüb, R. Da Forno, L. Sigrist, L. Mühlebach, A. Biri, J. Beutel, and L. Thiele, “FlockLab 2: Multi-Modal Testing and Validation for Wireless IoT,” in *CPS-IoTBench 2020*.

[43] M. Schuß, C. A. Boano, M. Weber, and K. Römer, “A competition to push the dependability of low-power wireless protocols to the edge,” in *EWSN, 2017*.

[44] J. Debadarshini, C. Shekhar, and S. Saha, “Fine-grained frequencies for simultaneous intra-group one-to-all dissemination,” in *IEEE MASS*, 2020.

[45] J. Debadarshini, S. Saha, O. Landsiedel, and M. Choon Chan, “Start of frame delimiters (sfds) for simultaneous intra-group one-to-all dissemination,” in *IEEE LCN*, 2020.