XPC: Fast and Reliable Synchronous Transmission Protocols for 2-Phase Commit and 3-Phase Commit

Alberto Spina, Michael Breza, Naranker Dulay, Julie McCann
Imperial College, London
{alberto.spina15, mjb04, n.dulay, j.mccann}@imperial.ac.uk

Abstract

The improvement of software abstractions and frameworks for programmers is one of the major challenges for the engineering of reliable and efficient wireless sensing systems. We address this challenge with X Process Commit (XPC), an atomic commit protocol framework, and *Hybrid*, a Synchronous Transmission (ST) communication approach. *Hybrid* exploits the reliability of Glossy and the speed of Chaos, two Synchronous Transmission primitives, to get lower latency and higher reliability than either on their own. *Hybrid* is a general approach that can provide reliable communication for any round based protocol. We use XPC and *Hybrid* to build the classical 2-phase and 3-phase commit protocols. Through extensive experimentation, we compare the performance of the 2-phase and 3-phase commit protocols when they use *Hybrid*, Glossy, and Chaos for communication. Our results show that *Hybrid* is more robust than Chaos to radio interference, with almost 100% reliability in a network of nodes suffering from moderate radio interference, 13% to 50% faster than Glossy, and has comparable overheads to other state of the art ST atomic commit approaches A^2/Synchrotron.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Wireless Communication; C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Algorithms,Design,Experimentation,Performance

Keywords

Wireless Sensor Networks; Agreement Protocols; Concurrent Transmission

1 Introduction

Wireless Sensor Networks(WSN) are a key technology used in environmental and infrastructure monitoring. They are also being integrated into the Sense-Action control loops of IoT and Cyber-Physical Systems(CPS) applications for many sectors of industry, including manufacturing, electricity, gas and water supply, construction and agriculture [6,36].

One of the big challenges with WSN is to improve current software abstractions and frameworks to create full-featured WSNs that provide system-level services while ensuring efficiency and reliability [26, 30]. It is difficult to create software abstractions for WSN due to the limited capabilities of sensor nodes and the (often extreme) physical deployment environment. These two factors contribute to high rates of communication and node failure [10].

A commonly required system-level service for WSN used in IoT and CPS systems is information propagation to all of the nodes. Examples include parameter changes (e.g. change channel or sensing rates), transactions (all devices reboot after code updates), or physical actuation (activating a valve, turn right).

The distributed systems community have developed many high-level abstractions for building dependable distributed systems that provide: reliable broadcast, consensus, group membership, view-synchronous communication and information propagation. The WSN community have mostly assumed an asynchronous model and used simpler abstractions with eventual consistency guarantees such as Trickle [27] due to previous difficulties with reliable synchronisation and communication. Yet, programmers can benefit from higher-level abstractions with stronger consistency guarantees, such as the ability to reach agreement, if these abstractions are efficient and reliable [15, 21].

The need for stronger consistency guarantees in dissemination has been argued for sensor systems used as part of a CPS in [15], to enable consistent self-adaptation in WSN [2], and for the dissemination of distributed data tables for in-network query processing [23]. Other examples which require the strong consistency provided by 2PC and 3PC is communication for UAV Swarms [31]. If the UAVs in a swarm decide to turn left, transactional primitives must guarantee that their neighbour UAVs will turn left at the same time to avoid collisions. Eventual consistency is not strong enough. Cyber-Physical Systems and WSAN used in industrial control applications [33] require updates to be consistent

International Conference on Embedded Wireless Systems and Networks (EWSN) 2020
17–19 February, Lyon, France © 2020 Copyright is held by the authors.
Permission is granted for indexing in the ACM Digital Library
ISBN: 978-0-9949886-4-5

and time bounded before actuation to ensure system stability. For example, in the WaterBox test-bed [22], tank water levels are sensed, and the inflow and outflow valves are controlled. A new control command must be received and implemented by all of the controllers at the same time in a strongly consistent manner, otherwise there is a high probability of a tank overflow or underflow that may cause damage [37]. Unreliable wireless communication [38] is one of the biggest challenges in the provision of strong consistency guarantees. To address this problem, we investigate the use of Synchronous Transmission communication to improve reliability.

Synchronous Transmission (ST) is a communication approach where wireless nodes can synchronise and exchange data at the same time [20]. It allows us to implement abstractions with stronger guarantees based on the synchronous model for distributed systems, where there is a known upper bound on message transmission and processing time. Glossy [16] and Chaos [25] are two well-known ST examples.

This paper makes several contributions. We present XPC (X Process Commit), a new programming framework for the implementation of atomic commit protocols. We also introduce *Hybrid*, a novel ST approach that uses the Glossy one-to-all ST primitive and the Chaos all-to-all ST primitive together to achieve better reliability and speed than either on their own. We acknowledge that no wireless protocol can provide 100% reliability or latency guarantees for individual packet delivery. The aim of Hybrid is to achieve improved reliability of strong consistency at a level sufficient for applications such as [33]. The definition of sufficient for these applications is dependant on the dynamics of the phenomenon under control [37]. *Hybrid* combines ST primitives to improve the reliability and latency of strong consistency by leveraging the massive communication redundancy of ST, an approach that has been shown to be effective for the control of multiple pendulums [7].

Hybrid is a general ST communication approach that can be used for communication for protocols other than those created with XPC. To achieve good reliability and low latency, *Hybrid* is required to make decisions on the use of the appropriate ST primitive with the best parameters at the time. We provide a detailed evaluation and comparison of Glossy, Chaos, and *Hybrid* when used for the two-phase commit [18] and three-phase commit [35] protocols. Our results show that in a network of 20 nodes with two sources of high radio interference *Hybrid* can provide close to 100% reliability when Chaos can not, and latencies that are between 13% - 50% faster than Glossy. We also discuss the performance of our scheme in light of published results of the state-of-the-art agreement framework, A2/Synchrotron [3].

2 Background and Related Work

Atomic Commit Protocols [29] are important to all distributed systems that need to maintain a consistent global state across the entire system. Protocols for 2-phase commit (2PC) [18] and 3-phase commit (3PC) [35] are the most well established and used to ensure that the nodes in a distributed system agree to commit a transaction. These protocols provide guarantees and have limits that are well understood. 2PC is a fast algorithm that can guarantees liveness in a network with no failures. The guarantee of liveness for 2PC can be violated in the presence of a coordinator failure and at least one node failure. 3PC adds an extra communication phase to guarantee liveness in the case of multiple node failures. It too suffers from the loss of the coordinator and cannot account for partitioned networks [29]. Both 2PC and 3PC assume a fail-restart failure model. In this work, we assume that fail-restarts are caused only by communication failures due to time-varying communication links. These short term communication failures are due to the environment and common with the use of low power radios [9]. Node failures could be handled by logging protocol state to persistent storage like the on-board flash RAM.

2.1 Synchronous Transmission

Synchronous Transmission (ST) communication primitives aim to provide energy and time efficient network-wide broadcasts by synchronously transmitting packets from multiple wireless nodes. They depend upon the radio effects of constructive interference [11], the capture effect [17], or both. Constructive interference occurs when two identical radio messages are received within 0.5μseconds of each other and can be successfully decoded. The Glossy ST communication primitive [16] was one of the first to use constructive interference, followed by many others [11]. The requirement that all messages are identical makes constructive interference based schemes inherently one-to-many.

The Chaos communication primitive [25] was one of the first examples of the use of the capture effect. The relaxation of the message similarity requirement of constructive interference makes communication primitives using the capture effect all-to-all.

The existence of communication redundancy makes ST communication very reliable in practice. A notable exploration of this property has been the EWSN (Embedded Wireless Systems and Networks conference) Dependability Competition that has been held to assess the reliability of communication primitives, and propose an assessment methodology [8] for this purpose. The use of ST protocols is being explored for many high reliability applications [5].

2.2 A^2/Synchrotron

Synchrotron [3] is a transmission kernel inspired by Chaos and LWB [14]. It operates in time slots which include the time taken for the reception, processing and transmission of packets. In Chaos, reception rates degrade quickly in the presence of network interference and link unreliability. Synchrotron addresses this by using time-slotted channel hopping,

Synchrotron suffers from the same scalability and reliability problems as Chaos. Chaos uses a control message that uses a single bit per node in the network to keep track of which nodes have received the latest data. The control message size limits the number of nodes that can participate in a Chaos flood. Chaos is best effort because of its all-to-all communication and may not terminate, or take an unbounded amount of time to do so. This poses a problem for applications like control that require a time bound. We address this problem in our work by bounding the time of a Chaos flood and then using Glossy to reach the remaining network nodes in a reliable way.

2.3 Baloo

Although several ST communication primitives exist such as one-to-all (Glossy) and all-to-all (Chaos), they are very difficult to program because they rely on the low-level control of timers and radio events. Baloo is a middleware layer [20] that addresses this problem. Baloo exposes a well-defined interface to enable the run-time control of ST-primitives by the network layer and makes it possible to create higher level abstractions using ST. XPC uses the ST communication abstractions provided by Baloo in its design and implementation. Baloo offers a standardised ST layer so that various ST approaches can be developed in a comparable way.

In Baloo, the protocol implementation is separated from the lower level manipulation of data packets, data transfers and timing model. The underlying ST primitives may be changed without affecting the protocols themselves. Higher level protocol logic can then be implemented using callback functions.

Time Division Multiple Access (TDMA) [34] is used by Baloo to create execution rounds and requires a fixed execution time upper time bound for each round. Control packets are sent by a controller node at the beginning of each round and contain schedule information and configuration information. Nodes that successfully receive and decode control packets can transmit during subsequent data slots that they have been allocated. Baloo does not offer any services such as those required for voting.

3 XPC and *Hybrid*

In this section, we present XPC and *Hybrid*. XPC is a software library that provides abstractions for the implementation of atomic commit protocols. Hybrid is a way of using both Glossy and Chaos for fast and reliable flooding. XPC and Hybrid were implemented on the Contiki-NG operating system [24] using the Baloo middleware [20] for the TelosB motes [1] on the FlockLab test-bed [28].

3.1 XPC

XPC is designed to create atomic commit protocols for WSN. An overview of the layers of XPC can be seen in Figure 1.

1. **Application**: Some applications require atomic commit protocols with strong consistency guarantees and time bounds.
2. **Protocol implementation**: XPC is used to implement an atomic commit protocol that meets the strong guarantees and time bounds of the application.
3. **Common code**: Handles Packet buffers, message parsing, and all retransmission policies for all ST primitives.
4. **ST primitives**: Abstracts away the requirements of the ST primitive. Each ST primitive has completely different message packet structures and timing requirements. With XPC, a protocol can specify which ST primitive to use to exchange messages for each round.

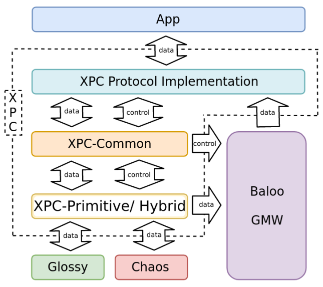


Figure 1: Layered overview of all XPC components. XPC lives alongside Baloo's implementation, processing all communication from the application.

XPC assumes that there is a single global host which manages the protocol phases. All of the other nodes act as participants. The XPC global host is in charge of the atomic commit protocol's overall progress from a network wide point of view. The other nodes either commit or abort a value specified by the global host.

XPC provides an API of 28 function calls for the development of atomic commit protocols. There are separate functions for the global host and the nodes. Please note that in Figure 1 *Hybrid* is depicted at the same level as XPC-Primitive, but can be used separately. The principle API functions are shown in Table 1.

XPC API		
Name	**Use**	
primitive_reset_schedule	Global Host	Prepare schedule for ST primitive
primitive_update_schedule	Global Host	Update schedule for next round
xpc_prepare_control_message	Global Host	Application control message
xpc_control_config_next_round	Global Host	Application round configuration
xpc_read_control_message	Nodes	Read application control message
xpc_commit	Nodes	Application commit
xpc_abort	Nodes	Application abort

Table 1: XPC-Common and XPC-Primitive APIs.

When a new phase begins the XPC global host generates a transmit schedule for all of the nodes in the network using *primitive_reset_schedule*. It sends the schedule in a control packet. Each phase may consist of many rounds, depending on how many nodes respond in the first. In a network with no interference and good communication links, all of the nodes may reply in one round. If some nodes do not reply, a retransmission round must be scheduled using *primitive_update_schedule* to request communication from the missing nodes.

Each schedule must be generated one round in advance. An additional final round is scheduled to handle the potential retransmissions. Retransmissions to collect lost responses from the nodes can only occur a maximum number of times, or transmission limit, so that the protocol does not exceed the application's time bound. Exceeding the time bound causes a time out, and the protocol terminates with an abort if the global host fails to hear from all of the nodes within the protocols required time bound.

Nodes that receive a control packet respond to the XPC global host with the requested information based on the phase of the protocol (a `vote` for the voting phase, or a `haveCommitted` for the commit phase). A node may miss a control packet from the global host due to interference in the network. A node will timeout and abort the transaction if it does not receive a control packet after a set time to prevent deadlock caused by infinitely waiting.

3.2 Baloo Control and Modifications

XPC uses and configures Baloo in the following ways:

1. **Single Initiator**. Baloo relies on the presence of a global host. This node is in charge of bootstrapping the network and sending control packets at the beginning of each flood. With XPC, the global host is in charge of the protocol and the protocol state machine.
2. **Retransmissions for Reliability**. WSN links are very unreliable, and lose packets due to interference or environmental conditions. To mitigate this issue XPC uses retransmissions to execute a phase more than once should there be missing replies.
3. **Additional Final Round**. At the beginning of a Baloo round, we cannot be sure whether the XPC global host will receive replies from all of the nodes. If it does not, XPC schedules a "retransmission" round to request information from the nodes that did not reply. XPC schedules a final, empty round to handle the potential for missing replies. In Figure 2, the XPC global host sends control packet C during rounds 1 to N, and expects all of the nodes to reply during their scheduled slots. If all of the nodes successfully reply by round N, a final empty round (denoted as E) is scheduled to communicate protocol termination. If not, the global host will send another communication schedule. The empty round was chosen to give the protocol the flexibility to schedule or cancel retransmission rounds dynamically based on the number of nodes that respond.

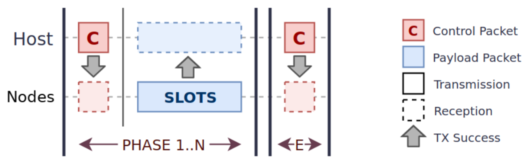


Figure 2: Example X-Phase protocol ported to Baloo's round structure.

Several common Baloo configurations are used by all protocols, regardless of their underlying ST primitive:

- `schedule.period`. All protocols share the same length of time allocated for the execution of the application after a successful iteration of the protocol. Application execution time can be adapted to the needs of the top-level application.
- `user_bytes`. All protocol information necessary for a given round is disseminated in a control packet. The host assigns two sections of the optional `user_bytes` configuration parameter: the first holds the message sent by the host to all nodes in the network, the second holds the value currently proposed by the host.

3.3 *Hybrid* Synchronous Transmission Scheme

2PC and 3PC can be implemented with communication primitives that support addressing, acknowledgements and retransmissions. To provide strong consistency guarantees, they require reliability and latency guarantees from the communication primitive that are as strong as possible. Current ST communication primitives have been shown to provide good reliability and latency guarantees [8].

Glossy can provide reliable floods but requires a separate flood for each node. While very fast, Glossy floods occur sequentially, one for each node. The sum of the time for all of the floods may be greater than the application's time bound on decision making. Here the commit should timeout, leaving the complete system in a consistent (old) state as updates are uncommitted. Chaos implements all-to-all communication and is faster than Glossy. Chaos can fail to receive information from specific nodes due to the random nature in which nodes send and recieve. Loss of information from a node can cause a commit timeout preventing the commit of the new state.

A simple sequential combination of Chaos and Glossy does not yield better performance. Baloo's Chaos implementation had to be modified to support time-boundedness and scheduling to schedule Glossy and Chaos together. Our adaptation to Chaos controls the interruption of Chaos so that we can maintain the node information, and pass it to another scheduled Chaos round, or Glossy flood. We added two new API functions to Baloo to set and get the list of responding nodes from the previous Chaos round.

In *Hybrid*, the first transmission of each phase executes using a Chaos flood with a slot duration selected to reach as many nodes as possible (see Figure 3). If the slot duration is too short, few nodes may reply. If the initial slot duration is too long, the latency may increase. We show in Section 4.2 our experimental process for the selection of the Chaos slot duration. If all of the nodes have not responded to the initial Chaos flood, Glossy is used for reliable retransmissions to communicate with the remaining nodes. Glossy should be used for as few nodes as possible because each node requires a separate flood. The maximum number of retransmissions using Glossy floods is bounded to preserve the application's time bounds. If a reply does not arrive from every node, the protocol times-out. We leave the question of dynamic slot sizes that adapt to network conditions for future work.

4 Evaluation

In this section, we report experimental results comparing the latency and reliability of Glossy, Chaos and our Hybrid approach. *Hybrid* uses Glossy and Chaos in their primitive form, without added features, so we compare against Glossy and Chaos on their own. Glossy and Chaos are used as ST primitives for many other higher level ST protocols such as Splash [12], LWB [14], Pando [13], and Mixer [19] that enhance ST with many other features such as network coding or fountain coding. We thought that a comparison against

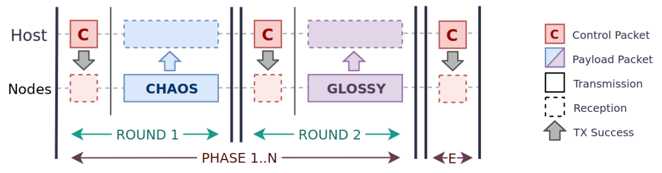


Figure 3: Overview of *Hybrid* execution across multiple phases. Each phase starts with a Chaos dissemination round, followed by a variable number of Glossy floods.

Glossy and Chaos on their own would be a better comparison for *Hybrid*.

We also compare XPC/*Hybrid* with *published* Synchrotron values running in the same testbed qualitatively comparing the different protocols operating on networks of similar density. This experiment will not account for variations in the network interference or the states of the nodes at the different times of the experiments, therefore this comparison is not robust.

We used XPC to create two reference atomic commit protocols for evaluation. We implemented 2-phase commit(2PC) [18] and 3-phase commit(3PC) [35] using XPC with Glossy only, with Chaos only, and with *Hybrid*. We evaluate the agreement outcome and the latency of each of the atomic commit protocol with each ST primitive. Then we experimentally evaluate the robustness of the atomic commit protocols with different degrees of radio interference. Finally, we qualitatively compare our latency results to the reported results of A^2/Synchrotron which uses Chaos [3].

Our analysis was performed on the FlockLab testbed [28] using the Tmote WSN platform [1] initially on nodes {1-4, 6-8, 10, 11, 13, 14-20, 22-28, 31-33}. All results shown are the average of 100 transaction runs. Given the inherent scale limitations of both Glossy and Chaos (scale is an open problem in ST research), we believe that the FlockLab testbed provides an adequate network size and network density for our evaluation.

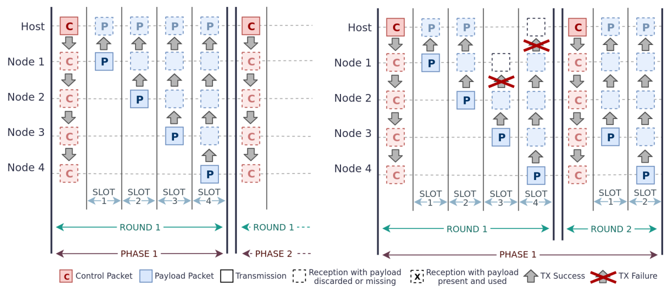
4.1 XPC using Glossy

XPC with Glossy uses a time-sliced data dissemination approach. Given a network of k nodes (where one is the XPC global host), each round a `schedule.n_slots` field is set to $k-1$. All nodes, except the global host, communicate in a given `schedule.slot` (see Figure 4a). The nodes receive round information from the global host's control packet. Nodes reply to the host by broadcasting during their scheduled slot. The `payload` exchanged during each round contains the messages sent by each node as a reply to the host.

XPC using Glossy keeps track of nodes that do not reply in their scheduled slots and schedules them to retransmit in the next round. The retransmission round has a slot for each node that did not reply. In Figure 4b, node 3 and node 4 did not successfully send their reply to the host node during the first round. The retransmission round (i.e. round 2) contains only two slots.

4.1.1 *Two-Phase Commit with Glossy*

Our first set of results show that 2PC is unable to reach all nodes in one round reliably. The introduction of retransmission rounds greatly boosts the overall reliability (Figure 5a). Note that there are very few timeout aborts with the use of retransmissions.



(a) Glossy round execution (b) Glossy retransmission round

Figure 4: Execution of Glossy rounds with XPC. When nodes do not reply in a given slot they are scheduled to retransmit in the subsequent round during the same phase.

The results in Figure 5b show that retransmissions do not significantly increase the latency of the protocol. With no retransmissions, the protocol reaches a timeout abort in approximately 30% of the runs.

4.1.2 *Three-Phase Commit with Glossy*

The results in Figure 5c show that "timeout commits" do not change the reliability of 3PC-Glossy when compared to 2PC-Glossy (Figure 5a). 3PC (Figure 5d) has one more phase, and a higher latency when compared to 2PC.

4.2 XPC using Chaos

Compared to Glossy, Chaos prioritises latency over reliability. As can be seen in Figure 6 Chaos floods occur in a "best-effort" fashion with no certainty that a flood is long enough for communication to reach all of the nodes in the network.

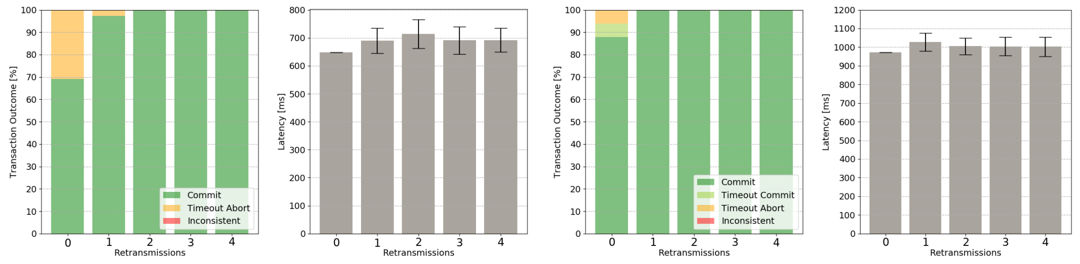
With XPC the challenge is to bound the maximum communication time of a Chaos flood, the slot duration. Chaos floods last until all of the nodes cease to receive new information from their neighbours. XPC bounds the time of a Chaos flood and schedules a round of the same slot duration as the previous round using the same `payload` (Figure 6). The Chaos flood resumes exactly from where XPC stops the previous flood and receives extra time to terminate. Communication ceases when no node sees new information in the packets being broadcast (as seen in Figure 6).

In our experiments with 2PC-Chaos and 3PC-Chaos, we explore two parameters: the number of retransmissions and the Chaos slot duration. The results in Figure 7 validated our assumptions: a longer slot duration (i.e. 100ms or 200ms) has near 100% reliability, very similarly to XPC using only Glossy.

4.2.1 *Two-Phase Commit with Chaos*

The results show very poor reliability for 25ms slots (Figure 7a) with improved results for 50ms slots (Figure 7b). For latency, the results show that 25ms slots (Figure 8a) have higher latency than 50ms slots (Figure 8b).

We analysed the cause for this unreliability using FlockLab traces of LEDs controlled by the General Purpose Input/Output pins. The red LED is the radio turned on, purple



(a) 2PC-Glossy transaction outcome (b) 2PC-Glossy retransmission latency (c) 3PC-Glossy transaction outcome (d) 3PC-Glossy retransmission latency

Figure 5: Evaluation of transaction outcome and latency for XPC 2PC-Glossy and XPC 3PC-Glossy in FlockLab.

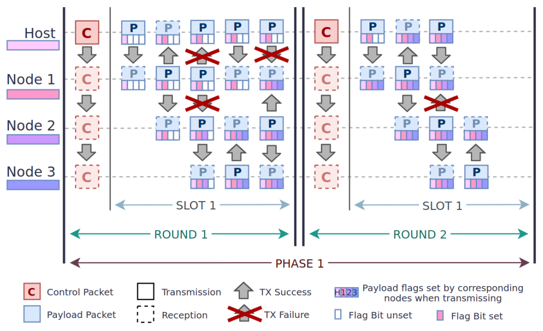


Figure 6: Execution of an XPC Chaos round with 1 retransmission. As nodes aggregate their vote into the payload they set their bits into the packet's flags bit-field.

LED is the reception of a message, and the yellow LED is message broadcast. Figures 9b and 9a come from the Flocklab web visualiser for GPIO outputs. The logs have matching timestamps for each node. We analysed a GPIO pin trace of 2PC-Chaos with 25ms slots and 1 retransmission because it was unable to commit without retransmissions (Figure 7a). Most of the executions present in the GPIO traces aborted with a time-out due to missing replies. Figure 9a is a representation of a successful commit that required 5 retransmissions: 2 retransmissions for Phase 1, 2 retransmissions for Phase 2, and a final retransmission to communicate the end of the 2PC round. The XPC Chaos uses 2 rounds for each communication phase, an indication of a slot duration that is too short. Multiple retransmissions are required because not all of the nodes are reached. The FlockLab GPIO trace for a 50ms slot duration was very different. Most of the executions in the traces were successful. Figure 9b shows a representation of a successful commit execution with 3 rounds.

The issue is that Chaos needs gap times for its callback functions. If the slot duration is short, the next round occurs before the end of the gap times. We see that a 100ms slot duration has a higher reliability and slightly lower latency than any number of retransmissions with 25ms slots.

The results for 100ms and 200ms slots in Figure 8 show that 2PC-Chaos can be reliable and have lower latency than 2PC-Glossy. Chaos floods with 100ms slots achieve above 95% reliability and 325ms latency. With 200ms slots, we see reliability close to 100% and latency of 525ms, around 40% less than 2PC-Glossy evaluated on FlockLab. With a longer Chaos flood duration the number of retransmissions does not significantly increase the overall latency.

4.2.2 *Three-Phase Commit with Chaos*

The results show that 3PC-Chaos suffers from the same reliability and latency concerns as 3PC-Glossy. The reliability of 3PC-Chaos (Figure 10) is worse than 2PC-Chaos, due to the addition of an extra Chaos round and the "timeout commit" that is a part of 3PC. Above 90% reliability can be achieved with 200ms transmission slots, which is similar to 2PC-Chaos.

3PC-Chaos latency increases with the introduction of an additional communication round, but remain lower than 3PC-Glossy (Figure 11). Chaos is a fast ST primitive with a low overall protocol execution time. Unfortunately, Chaos can be unreliable on low-power multi-hop networks.

4.3 XPC using *Hybrid*

In this section we evaluate XPC with *Hybrid*.

4.3.1 *Hybrid Two-Phase Commit*

The Chaos slot duration for 2PC-*Hybrid* is the same used for the evaluation of Chaos on its own. The slot duration only determines the length of the first round of each protocol phase. The subsequent retransmissions use Glossy.

We can see in Figure 12 that the Glossy retransmissions increase the protocol reliability to 100% for our experimental set-up. The latency (Figure 13) is also very close to that of 2PC-Chaos.

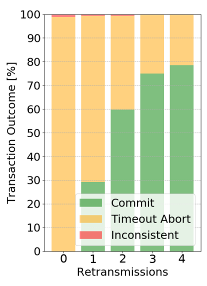
4.3.2 *Hybrid Three-Phase Commit*

The results for 3PC-*Hybrid* (Figure 14) show that agreement can be reached reliably with 4 or 5 retransmissions for any slot duration. A 100ms Chaos slot duration can reach 100% reliability with 3 retransmissions. A 200ms Chaos slot duration has high reliability with no retransmissions.

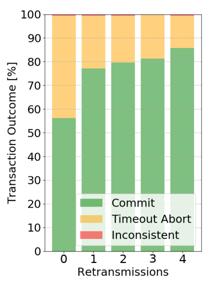
The latency of 3PC-*Hybrid* (see Figure 15) is very close to that of 3PC-Chaos. Our results show that XPC using *Hybrid* to schedule ST primitives can realise higher-level abstractions for use in synchronous WSNs with good performance and reliability.

4.4 Interference Analysis

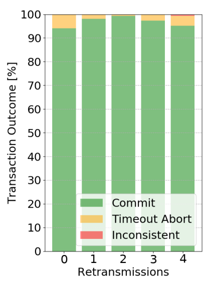
We focus our next set of experiments on the reliability of our *Hybrid* communication approach under varying amounts of radio interference. Interference causes nodes to miss broadcast packets, and potentially desynchronise from the



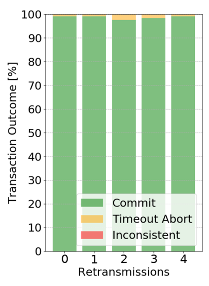
(a) 25ms slots



(b) 50ms slots

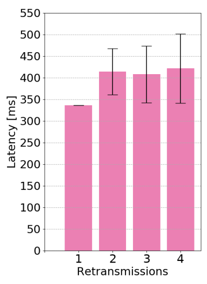


(c) 100ms slots

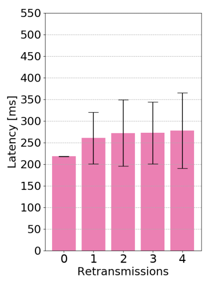


(d) 200ms slots

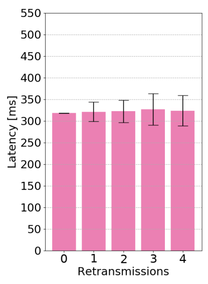
Figure 7: Agreement outcome of XPC 2PC-Chaos with varying slot duration in FlockLab.



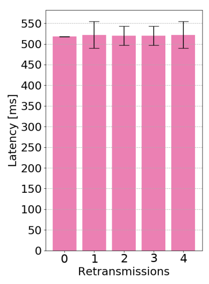
(a) 25ms slots



(b) 50ms slots

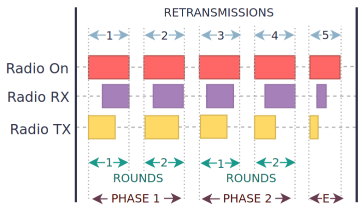


(c) 100ms slots

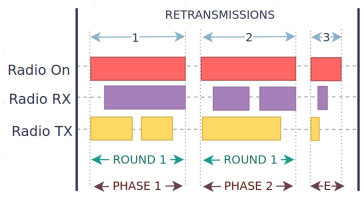


(d) 200ms slots

Figure 8: Latency of XPC 2PC-Chaos with varying slot duration in FlockLab.

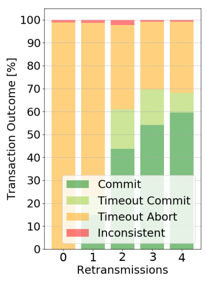


(a) GPIO trace for 2PC-Chaos with 25ms slots.

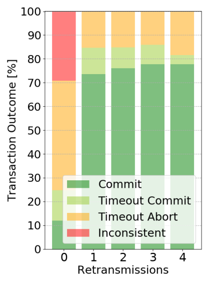


(b) GPIO trace for 2PC-Chaos with 50ms slots.

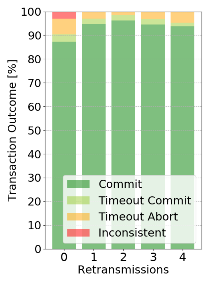
Figure 9: Representations of FlockLab Radio LED GPIO tracing for 2PC-Chaos with 25ms and 50ms slots.



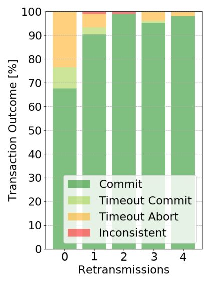
(a) 25ms slots



(b) 50ms slots



(c) 100ms slots



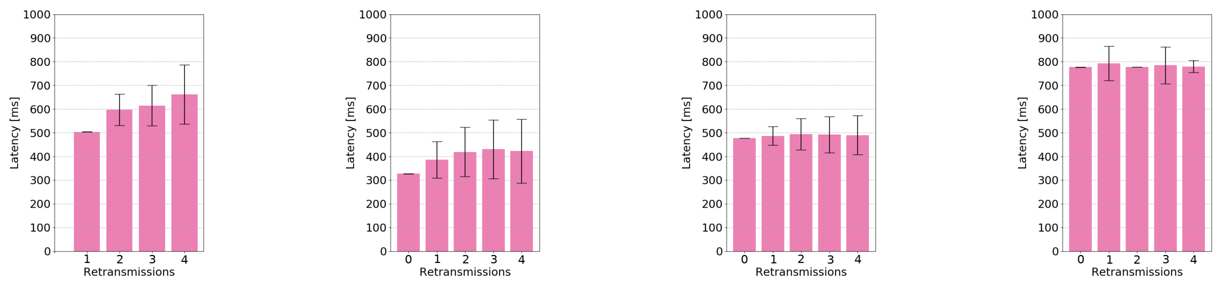
(d) 200ms slots

Figure 10: Agreement outcome of XPC 3PC-Chaos with varying slot duration in FlockLab.

network and miss transmission slots. We analyse protocol reliability in the presence of network interference.

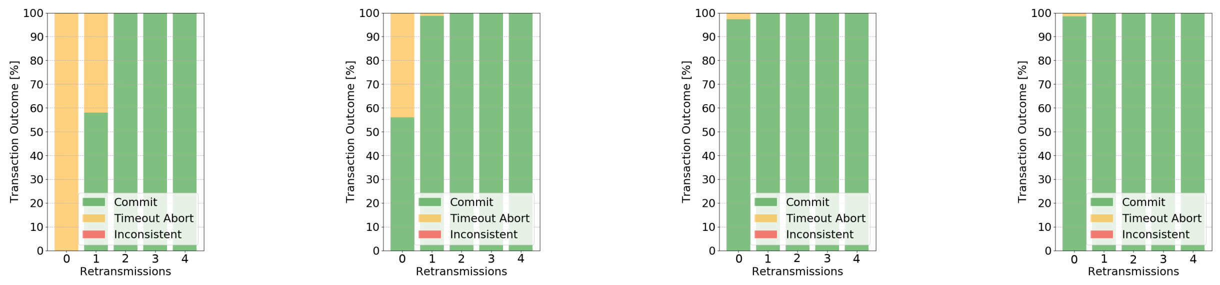
We performed our previous experiments during times of low radio interference (defined below). We established that *Hybrid* out-performs both Glossy and Chaos in such conditions. In the next set of experiments we evaluate Chaos, Glossy and *Hybrid* with more severe radio interference.

We performed experiments on the Flocklab [28] Tmotes {2-4, 6, 8, 10, 13, 15, 16, 11, 18-20, 22-28, 31-33}. Between 1 and 8 nodes were used from that group to inject interfer-



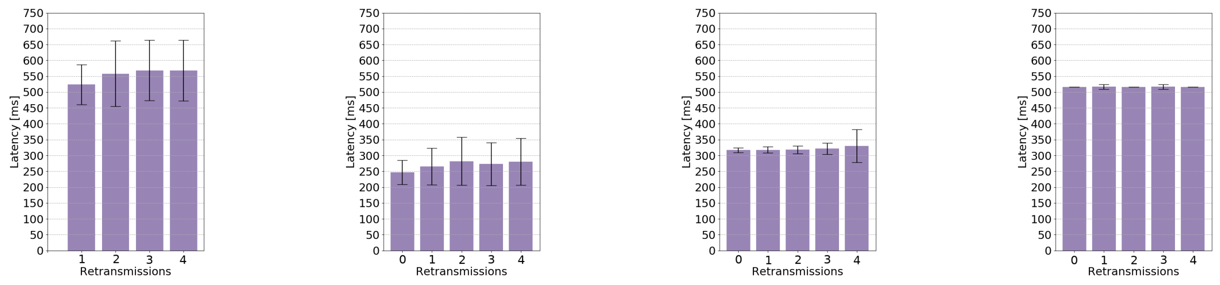
(a) 25ms slots (b) 50ms slots (c) 100ms slots (d) 200ms slots

Figure 11: Latency of XPC 3PC-Chaos with varying slot duration in FlockLab.



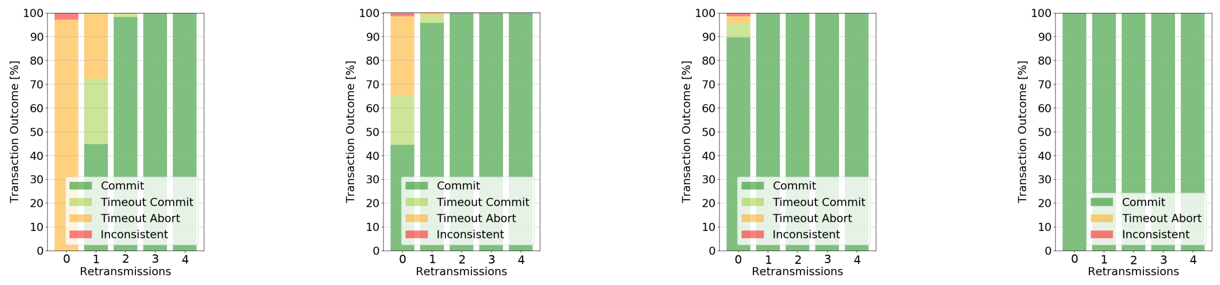
(a) 25ms slots (b) 50ms slots (c) 100ms slots (d) 200ms slots

Figure 12: Agreement outcome of XPC 2PC-*Hybrid* with varying slot duration in FlockLab.



(a) 25ms slots (b) 50ms slots (c) 100ms slots (d) 200ms slots

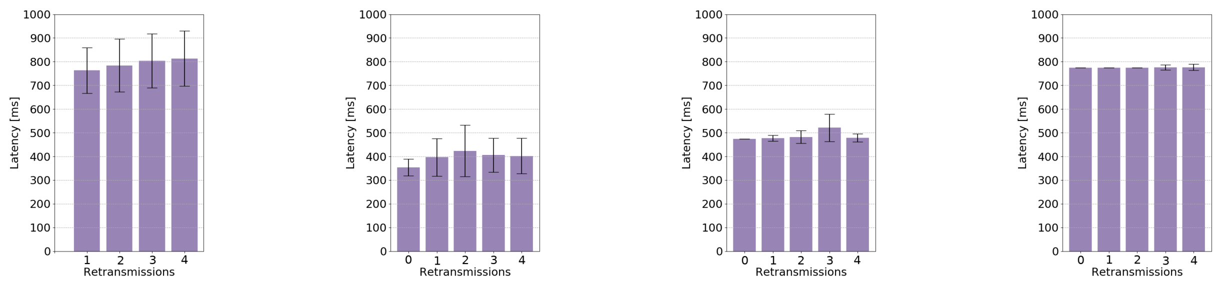
Figure 13: Latency of XPC 2PC-*Hybrid* with varying slot duration in FlockLab.



(a) 25ms slots (b) 50ms slots (c) 100ms slots (d) 200ms slots

Figure 14: Agreement outcome of XPC 3PC-*Hybrid* with varying slot duration in FlockLab.

ence into the network. We increase the interference by using different interference models, and different numbers of interfering nodes. The interfering nodes were {31, 20, 27, 28, 8, 6, 4, 3}, selected because they are physically close enough to



(a) 25ms slots (b) 50ms slots (c) 100ms slots (d) 200ms slots

Figure 15: Latency of XPC 3PC-*Hybrid* with varying slot duration in FlockLab.

another node to jam its radio reception. The jamming nodes use JamLab [9], a customisable off-the-shelf radio interference generation library for WSN motes. JamLab provides a set of interference profiles (explained below), that is becoming an accepted way to comparably evaluate WSN protocols. All results shown are the average of 150 transaction runs. The following interference patterns [9] were used to perform comparisons:

1. **Low Interference** Background noise on the FlockLab testbed during night-time hours (9pm-6am). This represents an ideal network deployment.
2. **High Interference** Background noise on the FlockLab testbed during day-time hours (7am-8pm). It provides an estimate of average real-world conditions.
3. **WiFi Interference** Noise generated by JamLab to emulate the interference of non-saturated WiFi file transfers and radio streaming.
4. **Microwave Interference** Noise generated by JamLab to emulate the periodic interference caused by microwave ovens over 802.15.4 transmission channels.

All nodes in the network vote in favour of all proposed values (100% agreement rate) and protocol phases are allowed up to 9 retransmissions before timing out and aborting. The Chaos slot duration for Chaos and *Hybrid* is 50ms. It is important to note that we consider WiFi and Microwave interference to both represent a high degree of radio interference. Both types of JamLab injected interference (WiFI and Microwave) were executed during night-time hours to minimise other external interference. We present the results using the following metrics:

- **Interference Model**. Low, High, WiFi and Microwave interference models were tested and evaluated individually for each protocol.
- **Average Reliability**. Protocol reliability measures the rate at which all nodes in the network commit the proposed transaction consistently. *If even just one node times out or aborts, the reliability is scored as zero for the given round.*
- **Latency**. The overall time of an XPC run. It starts when XPC pre-empts the application and ends when the application is resumed. Protocol latency is expected to increase with the interference.
- **First Round Coverage**. The percentage of network nodes reached, on average, during the first dissemination of each phase, denoted as P1, P2 and P3 depending on the number of phases.
- **Average Number of retransmissions**. This metric expresses the average number of retransmissions required for a protocol to switch to a subsequent stage.

4.5 Single Jammer

Low Interference	**Reliability (%)**	**Latency (ms)**	**Chaos Coverage (%)**	**Avg. Retr.**
2PC Glossy	100.00	564.43	**P1**: 99.72 [G] **P2**: 99.81 [G]	**P1**: 1.08 **P2**: 1.03
2PC Chaos	95.77	287.00	**P1**: 96.37 [C] **P2**: 96.36 [C]	**P1**: 1.27 **P2**: 1.41
2PC *Hybrid*	100.00	244.17	**P1**: 97.98 [C] **P2**: 97.14 [C]	**P1**: 1.13 **P2**: 1.21
3PC *Hybrid*	100.00	409.79	**P1**: 96.29 [C] **P2**: 97.57 [C] **P3**: 97.28 [C]	**P1**: 1.41 **P2**: 1.35 **P3**: 1.37
High Interference	**Reliability (%)**	**Latency (ms)**	**Chaos Coverage (%)**	**Avg. Retr.**
2PC Glossy	100.00	580.95	**P1**: 98.95 [G] **P2**: 98.54 [G]	**P1**: 1.15 **P2**: 1.25
2PC Chaos	89.09	481.66	**P1**: 94.77 [C] **P2**: 96.13 [C]	**P1**: 2.71 **P2**: 2.46
2PC *Hybrid*	100.00	284.79	**P1**: 96.64 [C] **P2**: 97.26 [C]	**P1**: 1.63 **P2**: 1.46
3PC *Hybrid*	100.00	465.52	**P1**: 94.19 [C] **P2**: 95.57 [C] **P3**: 95.88 [C]	**P1**: 1.63 **P2**: 1.70 **P3**: 1.68
Wifi Interference	**Reliability (%)**	**Latency (ms)**	**Chaos Coverage (%)**	**Avg. Retr.**
2PC Glossy	100.00	628.63	**P1**: 98.04 [G] **P2**: 98.04 [G]	**P1**: 1.60 **P2**: 1.58
2PC Chaos	56.76	875.74	**P1**: 89.53 [C] **P2**: 87.95 [C]	**P1**: 3.94 **P2**: 5.51
2PC *Hybrid*	100.00	357.29	**P1**: 91.36 [C] **P2**: 91.18 [C]	**P1**: 1.78 **P2**: 1.72
3PC *Hybrid*	100.00	499.00	**P1**: 93.72 [C] **P2**: 91.44 [C] **P3**: 93.89 [C]	**P1**: 1.71 **P2**: 1.62 **P3**: 1.67
Microwave	**Reliability (%)**	**Latency (ms)**	**Chaos Coverage (%)**	**Avg. Retr.**
2PC Glossy	100.00	593.33	**P1**: 98.89 [G] **P2**: 98.96 [G]	**P1**: 1.29 **P2**: 1.29
2PC Chaos	70.97	889.92	**P1**: 86.36 [C] **P2**: 89.18 [C]	**P1**: 5.51 **P2**: 4.51
2PC *Hybrid*	100.00	355.11	**P1**: 92.99 [C] **P2**: 92.44 [C]	**P1**: 1.73 **P2**: 1.86
3PC *Hybrid*	100.00	507.28	**P1**: 93.13 [C] **P2**: 91.83 [C] **P3**: 93.07 [C]	**P1**: 1.69 **P2**: 1.76 **P3**: 1.64

Table 2: Comparisons of XPC protocols for different interference patterns (1 jamming node).

The results of 2PC-Glossy, 2PC-Chaos, 2PC-*Hybrid* and 3PC-*Hybrid*, evaluated under the four interference model show in Table 2 the interference caused by one node with a variety of different interference patterns.

2-PC-Chaos has reduced latency and reliability across all experiments under challenging conditions. This result has been observed by others using Chaos based protocols [4]. There is a difference in latency results for Chaos in these experiments when compared to those in Figure 8. Here we measure the overall latency of both committed and aborted (due to timeout) transactions. In Figure 8, we only measure the latency of the commit transactions. Measuring the latency of both commits and aborts shows us the potential benefits of combining Chaos and Glossy in an operational system in the presence of interference.

We make the following further observations about the data presented in Table 2. As the interference models increase their disturbance over the channel, the protocol latencies increase linearly. The stronger WiFi and Microwave radio interference causes longer latency for 3PC-*Hybrid*. An increase of retransmissions causes an increase in latency and reflects the intensity of the channel's interference. The protocols aside from Chaos across all interference models achieve a 100% correct outcome, but when combined with Glossy into *Hybrid* it reduces latency by reaching over 90% of nodes during the first 50ms of each phase. Switching to reliable Glossy broadcasts handles the remaining nodes quickly and with high reliability.

4.6 Multiple Jammers

We extend our analysis to consider the impact of increased interference in the network. The purpose of this evaluation is to disrupt the network in degrees until we can see complete failure. We do this using multiple nodes generating jamming interference in the network. We select the microwave oven as an extreme form of interference.

By increasing the number of nodes generating microwave oven interference, we create a more challenging communication environment for the evaluation of the robustness of the protocol. It is important to note that all of the degrees of interference represented in this experiment are high. We consider that beyond three jamming nodes represents extreme interference beyond that of a normal operational environment. Tables 3 and 4 report the results of executions with between 2 and 8 interfering nodes.

The results in Tables 3 and 4 show that 2PC-*Hybrid* is faster than 2PC-Glossy for small amounts of interference. At one to two interfering nodes, 2PC-Glossy has higher latency than 2PC-*Hybrid* because it does not have the initial Chaos flood used by *Hybrid* to efficiently flood the network with data. Glossy does have a higher first transmission coverage and lower average retransmissions.

At three interfering nodes, the reliability of 2PC-Glossy and 2PC-*Hybrid* begin to degrade. We still see very similar reliability for both. The initial Chaos round reaches fewer nodes than before, and both protocols rely on Glossy floods. With four interfering nodes, 2PC-Glossy is more reliable than 2PC-*Hybrid*. Both are now reliant upon Glossy, and 2PC-Glossy has 10 glossy retransmissions while 2PC-*Hybrid* has 9 glossy retransmissions. This trend continues as the number of interfering nodes increases. With six interfering nodes, the protocols have mostly failed. None have reliability above 10%.

The unreliability of Chaos is seen with the performance of 2PC-Chaos. At four interfering nodes, 2PC-Chaos has completely failed. We also see that Chaos has a very high count of average retransmissions.

3PC-*Hybrid* behaves similarly to 2PC-*Hybrid*, with a 50% extra latency due to the extra phase. From the experiment traces, we can see that the 1st dissemination phase usually has the highest retransmissions. This is probably caused by nodes finding it harder to capture the control packet for the next round while under interference. At 8 jamming nodes we note that 3PC-*Hybrid* completely fails to receive any packets for the first Chaos transmit of the second and third phases. We also see the total number of retransmissions at their maximum value.

We see that *Hybrid* managed to achieve 100% reliability at or under 508ms for only 1 jamming node. This result suggests that *Hybrid* could be used for control applications [33] for low to moderate levels of interference, but high levels of interference may still cause failure. We leave investigation into the use of channel diversity for further resilience as an extension to this work.

4.7 Comparison with A^2 implementations

We qualitatively compare the latency of XPC to the execution times reported by A^2 in 2017 and 2019 [3,32] In both cases, tests are executed on Flocklab using all of the nodes.

In our analysis of XPC, we present the latency of com-

1 Jamming Node	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	100.00	593.33	**P1**: 98.89 [G] **P2**: 98.96 [G]	**P1**: 1.29 **P2**: 1.29
2PC Chaos	70.97	889.92	**P1**: 86.36 [C] **P2**: 89.18 [C]	**P1**: 5.51 **P2**: 4.51
2PC *Hybrid*	100.00	355.11	**P1**: 92.99 [C] **P2**: 92.44 [C]	**P1**: 1.73 **P2**: 1.86
3PC *Hybrid*	100.00	507.28	**P1**: 93.13 [C] **P2**: 91.83 [C] **P3**: 93.07 [C]	**P1**: 1.69 **P2**: 1.76 **P3**: 1.64
2 Jamming Nodes	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	100.00	787.45	**P1**: 91.28 [G] **P2**: 91.28 [G]	**P1**: 2.56 **P2**: 2.85
2PC Chaos	49.59	1030.10	**P1**: 60.64 [C] **P2**: 80.71 [C]	**P1**: 6.30 **P2**: 5.89
2PC *Hybrid*	100.00	648.85	**P1**: 82.67 [C] **P2**: 82.87 [C]	**P1**: 3.05 **P2**: 3.05
3PC *Hybrid*	100.00	976.25	**P1**: 77.43 [C] **P2**: 81.66 [C] **P3**: 81.29 [C]	**P1**: 3.37 **P2**: 3.09 **P3**: 2.96
3 Jamming Nodes	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	94.14	969.45	**P1**: 85.62 [G] **P2**: 86.50 [G]	**P1**: 3.77 **P2**: 3.60
2PC Chaos	34.31	987.03	**P1**: 73.75 [C] **P2**: 78.16 [C]	**P1**: 6.45 **P2**: 5.39
2PC *Hybrid*	95.00	841.74	**P1**: 78.28 [C] **P2**: 79.62 [C]	**P1**: 4.07 **P2**: 3.87
3PC *Hybrid*	97.79	1270.12	**P1**: 78.95 [C] **P2**: 77.60 [C] **P3**: 77.79 [C]	**P1**: 3.95 **P2**: 4.05 **P3**: 3.85
4 Jamming Nodes	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	52.10	1123.97	**P1**: 84.66 [G] **P2**: 86.39 [G]	**P1**: 6.37 **P2**: 5.68
2PC Chaos	0.20	1066.59	**P1**: 56.12 [C] **P2**: 67.55 [C]	**P1**: 8.96 **P2**: 7.94
2PC *Hybrid*	46.94	1134.01	**P1**: 76.71 [C] **P2**: 78.82 [C]	**P1**: 7.27 **P2**: 6.34
3PC *Hybrid*	40.63	1480.84	**P1**: 74.22 [C] **P2**: 78.95 [C] **P3**: 75.60 [C]	**P1**: 7.23 **P2**: 5.58 **P3**: 5.80

Table 3: Comparisons of XPC protocols with multiple jamming nodes (Microwave Interference).

mit only and abort only transactions. Aborted transactions complete in a shorter time. The inclusion of aborted transactions into the reported results lowers the upper-bound of protocol execution times. We see that *Hybrid* outperforms Glossy and Chaos for 2PC implementations. 2PC-*Hybrid* is able to match the 2019 A^2 latencies for commit-only transactions (Figure 16a), and is faster in the case of network-wide aborts (Figure 16b).

Hybrid also has the lowest latency among all implementations for 3PC (see Figure 16). Similarly to 2PC, 3PC-*Hybrid* matches A^2's 2019 implementation for transaction commits (Figure 16c) and provides improvement over aborts (Figure 16d). Please note that this comparison made using published results only, and does not account for variations in the network interference or the states of the nodes at different times of the experiments. A more robust comparison is left for future work.

5 Limitations and Further Work

The limitations of this work are typical of those in this field. It is difficult to control the radio environment of a remote WSN testbed. We hold that our experiments with the injection of radio interference do tell us something useful about the resilience of the hybrid use of Glossy and Chaos. A more controlled environment could have given us more precise results.

Hybrid also suffers from the same issues of scale shared by Glossy and Chaos. Glossy needs an individual network-wide flood for each node. Chaos uses a control frame that contains one bit for each node in the network. Both of these limit the size of the network on which each can be used.

5 Jamming Nodes	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	23.01	1274.33	**P1**: 77.07 [G] **P2**: 83.22 [G]	**P1**: 8.13 **P2**: 7.33
2PC Chaos	0.00	1061.64	**P1**: 56.47 [C] **P2**: 68.99 [C]	**P1**: 9.13 **P2**: 7.10
2PC *Hybrid*	14.94	1291.59	**P1**: 68.35 [C] **P2**: 75.47 [C]	**P1**: 8.87 **P2**: 7.41
3PC *Hybrid*	13.79	1506.73	**P1**: 63.49 [C] **P2**: 79.80 [C] **P3**: 77.68 [C]	**P1**: 8.74 **P2**: 7.52 **P3**: 6.67
6 Jamming Nodes	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	2.66	1192.78	**P1**: 72.24 [G] **P2**: 83.31 [G]	**P1**: 8.87 **P2**: 7.61
2PC Chaos	0.00	1041.92	**P1**: 53.16 [C] **P2**: 63.65 [C]	**P1**: 9.27 **P2**: 7.67
2PC *Hybrid*	1.56	1169.43	**P1**: 68.55 [C] **P2**: 78.62 [C]	**P1**: 8.78 **P2**: 7.11
3PC *Hybrid*	1.94	1277.95	**P1**: 67.54 [C] **P2**: 76.36 [C] **P3**: 78.38 [C]	**P1**: 9.14 **P2**: 7.64 **P3**: 7.14
7 Jamming Nodes	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	1.15	1231.18	**P1**: 66.15 [G] **P2**: 76.99 [G]	**P1**: 9.62 **P2**: 8.35
2PC Chaos	0.00	1054.03	**P1**: 46.10 [C] **P2**: 60.00 [C]	**P1**: 9.80 **P2**: 10.00
2PC *Hybrid*	0.39	1230.55	**P1**: 61.19 [C] **P2**: 74.45 [C]	**P1**: 9.59 **P2**: 8.30
3PC *Hybrid*	0.80	1293.27	**P1**: 59.80 [C] **P2**: 73.67 [C] **P3**: 76.30 [C]	**P1**: 9.66 **P2**: 8.82 **P3**: 8.52
8 Jamming Nodes	**Reliability (%)**	**Latency (ms)**	**1st Tx Coverage (%)**	**Avg. Retr.**
2PC Glossy	0.00	1252.55	**P1**: 58.79 [G] **P2**: 76.40 [G]	**P1**: 9.74 **P2**: 7.00
2PC Chaos	0.00	1048.42	**P1**: 41.44 [C] **P2**: 37.33 [C]	**P1**: 9.82 **P2**: nan
2PC *Hybrid*	0.00	1336.88	**P1**: 51.40 [C] **P2**: 68.26 [C]	**P1**: 9.75 **P2**: 8.00
3PC *Hybrid*	0.00	1300.07	**P1**: 50.49 [C] **P2**: 0.00 [C] **P3**: 0.00 [C]	**P1**: 10.00 **P2**: 10.00 **P3**: 10.00

Table 4: Comparisons of XPC protocols with multiple jamming nodes (Microwave Interference).

For further work, we would like to find a way to determine and adapt to the best Chaos slot duration at runtime based on the network conditions. We would also like to extend the use of XPC and the *Hybrid* ST approach to see what further communication protocols could be supported, or what new ones could be developed. We would also like to explore a way to incorporate the use of multiple channels to increase resilience.

6 Conclusion

In this paper, we present X-Phase Commit (XPC) for the implementation of atomic commit protocols using Synchronous Transmissions and *Hybrid*. We describe the design of XPC and reference implementations of two-phase commit and three-phase commit using Glossy and Chaos. We also present *Hybrid*, a way to use both Glossy and Chaos to provide fast and reliable flooding. We used XPC and our reference implementations to assess the latency and reliability of Glossy, Chaos, and *Hybrid* for both the two-phase commit and three-phase commit transactional protocols.

Our testbed evaluation showed that *Hybrid* enabled by XPC has lower latency than Glossy on its own, and is more reliable than Chaos on its own. We evaluated Glossy, Chaos, and *Hybrid* with increasing levels of network radio interference and saw that under low to moderate interference, *Hybrid* was as reliable as Glossy but with lower latency. The XPC library is available online at https://gitlab.doc.ic.ac.uk/xpc.

7 Acknowledgments

The authors would like to thank Luca Mottola and all of the reviewers for their help improving this work.

8 References

[1] Advanticsys. Telosb mote, 2019.

[2] M. Afanasov, L. Mottola, and C. Ghezzi. Software adaptation in wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 12(4):18, 2018.

[3] B. Al Nahas, S. Duquennoy, and O. Landsiedel. Network-wide consensus utilizing the capture effect in low-power wireless networks. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, pages 1:1–1:14. ACM, 2017.

[4] B. Al Nahas and O. Landsiedel. Competition: Aggressive synchronous transmissions with in-network processing for dependable all-to-all communication. In *EWSN*, pages 209–210, 2018.

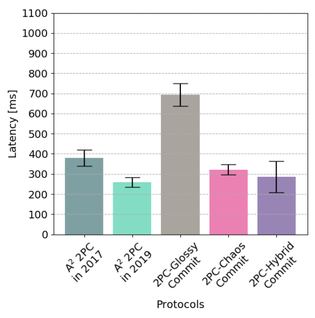
[5] M. Baddeley, U. Raza, A. Stanoev, G. Oikonomou, R. Nejabati, M. Sooriyabandara, and D. Simeonidou. Atomic-sdn: Is synchronous flooding the solution to software-defined networking in iot? *IEEE Access*, 2019.

[6] A. Baggio. Wireless sensor networks in precision agriculture. In *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN 2005), Stockholm, Sweden*, pages 1567–1576. Citeseer, 2005.

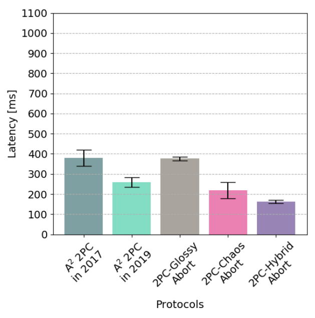
[7] D. Baumann, F. Mager, R. Jacob, L. Thiele, M. Zimmerling, and S. Trimpe. Fast feedback control over multi-hop wireless networks with mode changes and stability guarantees. *ACM Transactions on Cyber-Physical Systems*, 4(2):18, 2019.

[8] C. A. Boano, M. Schuß, and K. Römer. Ewsn dependability competition: Experiences and lessons learned. *IEEE Internet of Things Newsletter*, 2017.

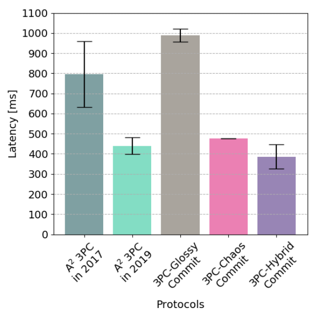
[9] C. A. Boano, T. Voigt, C. Noda, K. Romer, and M. Zuniga. Jamlab: Augmenting sensornet testbeds with realistic and controlled interfer-



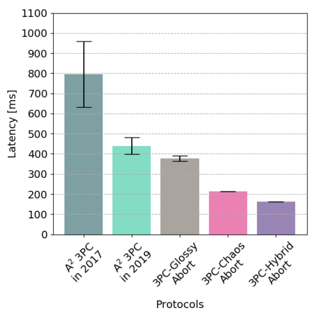
(a) 2PC Commit Comparison



(b) 2PC Abort Comparison



(c) 3PC Commit Comparison



(d) 3PC Abort Comparison

Figure 16: Comparison with A^2 implementations over all XPC versions of 3PC.

ence generation. In *Proceedings of the 10th ACM/IEEE IPSN*, pages 175–186, 04 2011.

[10] M. Breza, I. Tomic, and J. McCann. Failures from the environment, a report on the first failsafe workshop. *ACM SIGCOMM Computer Communication Review*, 48(2):40–45, 2018.

[11] T. Chang, T. Watteyne, X. Vilajosana, and P. H. Gomes. Constructive interference in 802.15. 4: A tutorial. *IEEE Communications Surveys & Tutorials*, 21(1):217–237, 2018.

[12] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 269–282. USENIX Association, 2013.

[13] W. Du, J. C. Liando, H. Zhang, and M. Li. Pando: Fountain-enabled fast data dissemination with constructive interference. *IEEE/ACM Trans. Netw.*, 25(2):820–833, Apr. 2017.

[14] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 1–14. ACM, 2012.

[15] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Virtual synchrony guarantees for cyber-physical systems. In *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 20–30. IEEE, 2013.

[16] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 73–84, 2011.

[17] C. Gezer, C. Buratti, and R. Verdone. Capture effect in ieee 802.15. 4 networks: Modelling and experimentation. In *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*, pages 204–209. IEEE, 2010.

[18] J. Gray and L. Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160, Mar. 2006.

[19] C. Herrmann, F. Mager, and M. Zimmerling. Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, pages 145–158. ACM, 2018.

[20] R. Jacob, J. Bächli, R. Da Forno, and L. Thiele. Synchronous transmissions made easy: Design your network stack with baloo. In *16th International Conference on Embedded Wireless Systems and Networks (EWSN 2019)*, 2019.

[21] A. Jhumka and L. Mottola. On consistent neighborhood views in wireless sensor networks. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 199–208. IEEE, 2009.

[22] S. Kartakis, E. Abraham, and J. A. McCann. Waterbox: A testbed for monitoring and controlling smart water networks. In *Proceedings of the 1st ACM International Workshop on Cyber-Physical Systems for Smart Water Networks*, page 8. ACM, 2015.

[23] R. Kolcun, D. Boyle, and J. A. McCann. Efficient in-network processing for a hardware-heterogeneous iot. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 93–101. ACM, 2016.

[24] A. Kurniawan. Basic contiki-ng programming. In *Practical Contiki-NG*, pages 47–66. Springer, 2018.

[25] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, 2013.

[26] M. T. Lazarescu and L. Lavagno. Wireless sensor networks. *Handbook of Hardware/Software Codesign*, pages 1–42, 2017.

[27] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. of the 1st USENIX/ACM Symp. on Networked Systems Design and Implementation*, volume 25, 2004.

[28] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *2013 ACM/IEEE IPSN*, IPSN '13, pages 153–165, 2013.

[29] N. A. Lynch. *Distributed algorithms*. Elsevier, 1996.

[30] L. Mottola, G. P. Picco, F. J. Opperman, J. Eriksson, N. Finne, H. Fuchs, A. Gaglione, S. Karnouskos, P. Montero, N. Oertel, et al. makesense: Simplifying the integration of wireless sensor networks into business processes. *IEEE Transactions on Software Engineering*, 2017.

[31] O. S. Oubbati, M. Atiquzzaman, P. Lorenz, M. H. Tareque, and M. S. Hossain. Routing in flying ad hoc networks: Survey, constraints, and future challenge perspectives. *IEEE Access*, 7:81057–81105, 2019.

[32] V. Poirot, B. Al Nahas, and O. Landsiedel. Paxos made wireless: Consensus in the air. In *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, EWSN '19, pages 1–12, 2019.

[33] S. Raza, M. Faheem, and M. Guenes. Industrial wireless sensor and actuator networks in industry 4.0: Exploring requirements, protocols, and challenges a mac survey. *International Journal of Communication Systems*, 32(15):e4074, 2019.

[34] T. Sekimoto and J. Puente. A satellite time-division multiple-access experiment. *IEEE Transactions on Communication Technology*, 16(4):581–588, 1968.

[35] D. Skeen. Nonblocking commit protocols. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pages 133–142. ACM, 1981.

[36] F. Stajano, N. Hoult, I. Wassell, P. Bennett, C. Middleton, and K. Soga. Smart bridges, smart tunnels: Transforming wireless sensor networks from research prototypes into robust engineering infrastructure. *Ad Hoc Networks*, 8(8):872–888, 2010.

[37] I. Tomić, M. J. Breza, G. Jackson, L. Bhatia, and J. A. McCann. Design and evaluation of jamming resilient cyber-physical systems. In *2018 IEEE Cybermatics Congress*, pages 687–694. IEEE, 2018.

[38] U. Wetzker, I. Splitt, M. Zimmerling, C. A. Boano, and K. Römer. Troubleshooting Wireless Coexistence Problems in the Industrial Internet of Things. In *Proceedings of the* 14th *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 98–109. IEEE, Aug. 2016.