Paxos Made Wireless: Consensus in the Air

Citation for the original published paper (version of record):
Poirot, V., Al Nahas, B., Landsiedel, O. (2019). Paxos Made Wireless: Consensus in the Air. Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks: 1-12. http://dx.doi.org/10.5555/3324320.3324322

Paxos Made Wireless: Consensus in the Air

Valentin Poirot[†], Beshr Al Nahas[†], Olaf Landsiedel[‡†]
[†] Chalmers University of Technology, Sweden
[‡] Kiel University, Germany
{poirotv, beshr}@chalmers.se, ol@informatik.uni-kiel.de

Abstract

Many applications in low-power wireless networks require complex coordination between their members. Swarms of robots or sensors and actuators in industrial closed-loop control need to coordinate within short periods of time to execute tasks. Failing to agree on a common decision can cause substantial consequences, like system failures and threats to human life. Such applications require consensus algorithms to enable coordination. While consensus has been studied for wired networks decades ago, with, for example, Paxos and Raft, it remains an open problem in multi-hop low-power wireless networks due to the limited resources available and the high cost of established solutions.

This paper presents *Wireless Paxos*, a fault-tolerant, network-wide consensus primitive for low-power wireless networks. It is a new flavor of Paxos, the most-used consensus protocol today, and is specifically designed to tackle the challenges of low-power wireless networks. By building on top of concurrent transmissions, it provides low-latency, high reliability, and guarantees on the consensus. Our results show that Wireless Paxos requires only 289 ms to complete a consensus between 188 nodes in testbed experiments. Furthermore, we show that Wireless Paxos stays consistent even when injecting controlled failures.

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

Consensus, Paxos, Multi-Paxos, Concurrent transmissions, Wireless sensor networks, IoT

1 Introduction

Context. Many applications in low-power wireless networks need to reach an agreement among themselves before an action can be performed. Mission critical systems are one typical example of such applications, since conflicting commands can have important and possibly harmful consequences. For instance, a swarm of Unmanned Aerial Vehicles (UAVs) must agree on a common destination [34]; while centrally computed transmission schedules in wireless sensor networks (WSNs) have to be agreed on and distributed by the nodes in the network [38].

However, not all faults can be avoided in wireless networks. Message loss and devices running out of battery are common failures seen in deployments. Thus, an agreement ensures that at most one action is chosen, even if some devices cannot participate. UAVs operating on limited batteries must agree on a common trajectory, even if some UAV ran out of power along the way; and an updated schedule must be agreed upon and used in a WSN, even if some nodes disappeared since the last agreement.

Efficient and highly reliable dissemination protocols have been proposed in the literature [12, 18]. However, they do not provide the same guarantees ensured by an agreement. A Glossy initiator cannot detect network segmentation, for instance, and would be unaware that its command was received by a small subset of the network only. In a swarm of UAVs, it is preferable that as many drones as possible continue on their agreed trajectory, rather than only the few that managed to receive the disseminated command. Reaching a consensus is therefore primordial to ensure the correct and optimal behavior of such applications.

Consensus refers to the process of reaching an agreement. A consensus is achieved once participants agree on a single, common decision, from a set of initial values. Consensus is challenging in the presence of failures (node crashes, message losses, network partitions, etc.). It is even proven that consensus is impossible in a fully asynchronous setting [19], where one node might never be able to communicate.

Many solutions to the consensus problem have been proposed in the literature [25, 33, 36]. Paxos was one of the first protocols to provide consensus [25, 26]. It is (non-Byzantine) fault-tolerant and proven to be correct: Paxos will lead to a correct consensus as long as a majority of nodes are participating. Due to the properties of Paxos, all nodes will eventually learn the correct value as long as a majority

accepted the decision. Paxos is often used in an extended and optimized form, Multi-Paxos [25], which allows nodes to agree on a continuous stream of values and enables state machine replication. For example, UAVs can continuously coordinate their next destination with Multi-Paxos.

Today, Paxos and Multi-Paxos have become the default protocols to ensure consistent data replication within datacenters. They are used in many modern deployments, for instance Google's Chubby locking mechanism [8] and their globally distributed database Spanner [10], Microsoft's datacenter management Autopilot [22], and IBM's data-store Spinnaker [39].

Challenges. The complexity of Paxos and its many required interactions pose key challenges in low-power wireless networks. Devices in WSNs have strong resource-constraints in terms of bandwidth, energy, and memory. Radios are, for example, commonly duty-cycled to save energy [6, 13, 37]. In contrast, Paxos requires many message exchanges and a high bandwidth to reach consensus.

In addition, links are highly dynamic and unreliable in low-power wireless communication [41]. Paxos is resilient to these network faults by design, but many of its implementations use end-to-end routing, which induces an overhead to the consensus. Paxos has been initially designed for wired networks, and is therefore heavily influenced by its unicast structure. Later work shows that Paxos can be partially executed with multicast [5], or by introducing an additional logical ring to reduce communications [32]. However, these approaches rely on unicast for parts of the algorithm.

In contrast, low-power wireless networks are broadcast-oriented networks where each transmission can be received by all neighboring nodes. Executing unicast-based schemes in wireless networks usually induces higher costs, especially in multi-hop networks. Moreover, multi-hop networks also provide opportunities for data-aggregation and computation of intermediate results, which are not part of Paxos' design rationale.

Approach. In this paper, we bring fault-tolerant consensus to low-power wireless networks. We propose *Wireless Paxos*, a new flavor of Paxos fitted to the characteristics of low-power wireless networking: we show that Paxos can be transformed from a unicast (or multicast) scheme to a many-to-many scheme, which can be efficiently executed in low-power wireless networks. We co-design the consensus algorithm along with the lower layers of the network stack to greatly improve the latency of consensus and have a tighter control on the transmission policy. The overall result is a broadcast-driven consensus primitive using in-network processing to compute intermediate results in Paxos. Our solution builds on top of Synchrotron [2], a kernel for concurrent transmissions inspired by Chaos [29], providing a basis for highly reliable and low-latency networking in low-power wireless with support for in-network processing.

Contributions. This paper makes the following contributions:

- By distributing parts of the proposer logic, we show that Paxos can be expressed as a many-to-many communication scheme, rather than a partially multicast scheme;
- We present *Wireless Paxos*, a new flavor of Paxos specifically designed to address the challenges of low-power wireless networks, and *Wireless Multi-Paxos*, an optimized extension of Wireless Paxos for continuous streams of agreed values for constrained devices;
- Both primitives are ready to use by any application as an open-source library on GitHub[1];
- We implement and evaluate our contributions on two testbeds, composed of 27 and 188 nodes, and compare our results to solutions from the literature.

The remainder of the paper is organized as follows. Sec. 2 introduces consensus, Paxos, and concurrent transmissions. Sec. 3 gives an overview of our design. Next, Sec. 4 dives into Wireless Paxos and Sec. 5 evaluates our contributions. We discuss related work in Sec. 6 and conclude in Sec. 7.

2 Background

This section introduces the necessary background on consensus and concurrent transmissions. We begin with an overview of consensus. Then, we present Paxos and Multi-Paxos. Finally, we introduce concurrent transmissions and Synchrotron.

2.1 Agreement and Consensus

In distributed systems, a consensus refers to the problem of reaching agreement among a set of participants. A consensus is complete if, in the end, nodes agree on the same decision. To be correct, a consensus algorithm must fulfill certain properties: the final value must be valid, i.e., it was proposed at the beginning of the algorithm (Validity); each node must eventually make a decision (Termination); and at most one value can be agreed upon, i.e., the result of the agreement must be consistent among the participants (Agreement) [11].

Dissemination is no consensus. Dissemination allows a node to share a value with the entire network, often (but not always) in a best-effort manner. As such, dissemination does not – *and is not meant to* – solve consensus. For example, Glossy [18] provides high reliability, and a unique value is present if at most one flood initiator is in the system. However, even if a node missing a value is aware of the failed flood, it will never be able to recover the correct command.

2 and 3-Phase Commit. Common algorithms for agreement (but not consensus) are 2-Phase Commit (2PC) and 3-Phase Commit (3PC) [20, 40]. Both algorithms are used to solve the problem of commit, i.e., whether a transaction should be executed by all nodes, or none. 2PC works by first requesting and collecting votes from all nodes in the network, and then disseminating the result of the decision; while 3PC adds an intermediary phase to dissociate the decision from the commit. Both protocols handle failures by aborting or blocking, i.e., by delaying the decision to maintain consistency. However, 3PC might lead to inconsistencies.

Fault-tolerant consensus. Paxos provides consensus. While many values can be initially proposed, it ensures that at most one value is chosen. Eventually, all nodes will learn

[1] Available at: https://www.github.com/iot-chalmers/wireless-paxos

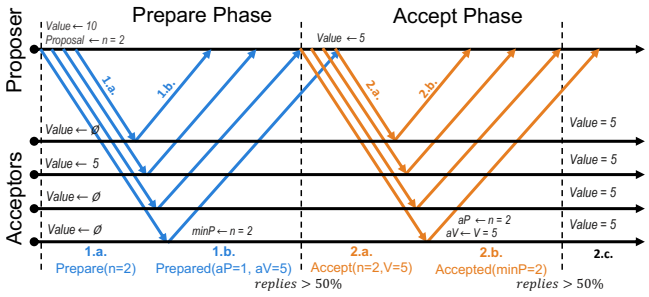


Figure 1. Executing Paxos. *A proposer wants to propose the value V=10 to four acceptors. It sends a Prepare request to all acceptors* ***(1.a)*** *with proposal number n=2. An acceptor replies with a Prepared message* ***(1.b)*** *with the most recently accepted proposal, here proposal aP=1 with value aV=5. Once a majority of replies are received by the proposer, it adopts the highest value received, and sends an Accept request* ***(2.a)****. Upon reception, an acceptor accepts the value* ***(2.b)*** *and replies with an Accepted message and the highest proposal received so far. After a majority of Accept messages, the value is chosen.*

the decision if progress is not impeded. To do so, Paxos relies on a majority of responses to make the decision, rather than from the entire network, thus handling (non-Byzantine) failures (e.g., message losses, network segmentation, node crashes).

2.2 The Paxos Basics

Paxos is a fault-tolerant protocol for consensus. It assumes an asynchronous, non-Byzantine system with crash-recovery, i.e., it handles both process crash and recovery (a persistent storage is needed), but not misbehaving nodes or transient faults; delayed or dropped messages, but not corrupted messages; and network segmentation. The protocol guarantees that, if a majority of nodes runs for long enough without failures, all running processes will agree on the same proposed value. For example, the value can be the destination point for UAVs, or the network configuration in WSNs.

Roles. A node can act as three different roles: *Proposer*, *Acceptor*, and *Learner*. Nodes can implement more than one role. Proposers propose a value to agree on and act as coordinators for the protocol's execution. Acceptors, unlike in 2PC, don't "vote", but act, in a very informal manner, as the system's "fault-tolerant memory": they reply to proposers requests by accepting proposals. Learners do not participate in the consensus: they only learn which value has been chosen by the acceptors once a consensus is met. Unlike 2PC and 3PC, where at most one coordinator must be present, Paxos can tolerate the presence of multiple proposers, at the cost of impeding the progress of the agreement.

Execution. The protocol consists of two phases: the *Prepare* phase and the *Accept* phase, as depicted in Fig. 1. The protocol is executed as follows:

1. **Prepare Phase**

a. Any proposer starts the protocol by choosing a value V to agree upon and a unique proposal number n (i.e., no other proposer can choose the same proposal number n). A *Prepare(n)* request is sent to every acceptor.

b. Upon reception of a *Prepare(n)* request, an acceptor will save the proposal number n if and only if it has never heard any higher proposal number *minProposal* before; i.e., if $n > minProposal$, then $minProposal \leftarrow n$. The acceptor only replies to the request if the precedent condition is met, meaning that the node is promising not to reply to any request with a lower proposal number anymore. The acceptor returns both the last proposal *accepted* by that process, noted *acceptedProposal*, if any has been accepted so far, and the corresponding value, noted *acceptedValue*.

2. **Accept Phase**

a. Upon hearing from a majority of acceptors, the proposer *adopts* the value with the highest proposal number, such as $V \leftarrow acceptedValue$, if any has been received. This condition ensures that at most one value can be chosen by the system. The proposer switches to the Accept phase and sends an *Accept(n,V)* request to all acceptors.

b. Upon receiving an *Accept(n,V)*, an acceptor *accepts* the value V if and only if the proposal number n is higher or equal to the proposal number the process has prepared for, namely *minProposal*. If the condition is true, the acceptor saves the proposal number n as its highest proposal number heard and as its accepted proposal, and the value V as its accepted value, i.e., if $n \geq minProposal$, then $minProposal \leftarrow n$, $acceptedProposal \leftarrow n$ and $acceptedValue \leftarrow V$. Regardless of the result of the condition, the acceptor replies to the request with the highest proposal heard (*minProposal*).

c. Upon receiving at least one reply with $minProposal > n$, the proposer knows that its value has been *rejected*. This also means at least one other proposer is present, and the process can either restart the protocol with a higher proposal number n to compete or let the other proposer win. If the proposer received a majority of replies and no rejection, the value is *chosen*.

The proposer can therefore inform the learners that a value has been chosen by the consensus algorithm. Using *minProposal* ensures that only the most recent proposal can be accepted and the data returned at step 1.b. ensures that at most one value can be chosen.

2.3 Multi-Paxos

Using the protocol described above leads to the agreement and dissemination of a single value. Due to the properties of the protocol, any additional execution will lead to the same value being adopted. Being able to agree on a sequence of values is a desirable property that Paxos, in its simple form, cannot satisfy. We can, however, run several *rounds* of Paxos to achieve this result. A multi-round version of Paxos is called *Multi-Paxos*. More importantly, Multi-Paxos allows state machine replication, i.e., replicating operations across processes such as all replicas are identical. For example, the

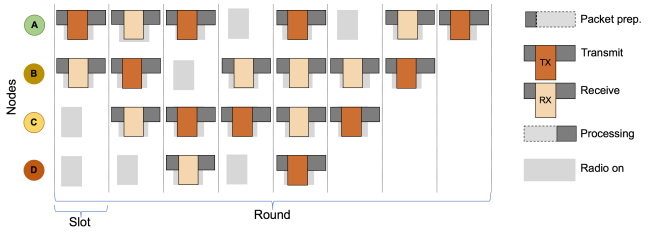


Figure 2. Synchrotron Overview: *Synchrotron schedules flooding rounds for network-wide dissemination, collection or aggregation, as requested by the application. A round is composed of consecutive slots, during which a node can either transmit, receive, or sleep, and processes the data following a per-application logic.*

stream of values can represent intermediary waypoints towards the final destination for UAVs, or the evolution of the network configuration over time for WSNs. For details on Multi-Paxos, we refer the reader to the original paper [25].

Executing multiple rounds. A Paxos execution is now identified by its *round number*. Many rounds can be executed simultaneously or sequentially. In the former case, messages of multiple rounds can be merged into one unique message in order to save network resources.

Using a unique proposer. In many applications, nodes are stable enough to keep a unique proposer for a relatively long period of time. In doing so, the Paxos execution can be simplified to its sole Accept phase, the Prepare phase being used only to prepare acceptors to listen to that specific proposer. The Prepare phase is executed at the beginning of Multi-Paxos or after a crash of the proposer, and successive rounds execute only the Accept phase for value adoption. The protocol does not break in the presence of multiple proposers since acceptors will only accept the highest proposal and inform lower proposals of their rejection.

Executing Multi-Paxos. Fig. 1 shows the execution of Paxos. With Multi-Paxos, the Prepare phase (blue) is executed once at the beginning, and is followed by many Accept phases (orange), until the proposer fails.

2.4 Synchrotron

Due to the broadcast nature of wireless communications, concurrent transmissions of nearby nodes inherently interfere with each other. However, when such transmissions are precisely timed, one of them can be received, nonetheless. We briefly introduce the concept of capture effect, and present Synchrotron, the communication primitive used in this work.

Capture effect. Nodes overhearing concurrent transmissions of different data can receive the strongest signal under certain conditions; this is known as the capture effect [30]. To achieve capture at the receiver with IEEE 802.15.4 radios, the strongest signal must arrive no later than 160 μs after the first signal and be 3 dBm stronger than the sum of all other signals [15].

Synchrotron. Agreement in the Air (A^2) [2] extends the concepts of Chaos, a primitive for all-to-all aggregation [29], to network-wide agreement. A^2 introduces Synchrotron, a new transmission kernel using high-precision synchronization. Like Chaos, Synchrotron operates periodically within rounds, which we refer to as "flooding" rounds to distinguish from Paxos rounds. A (flooding) round is composed of slots, in which nodes concurrently transmit different data, while the reception relies on the capture effect. In-network processing is applied after a successful reception and the result of the computation is then transmitted during the following slot, until all progress flags are set, denoting the participation of the different nodes. Fig. 2 presents an overview of Synchrotron's inner working.

Our solution reuses the Synchrotron layer of A^2, and provides fault-tolerant consensus with Paxos, while A^2 provides agreement with 2&3PC.

3 Design Rationale

In this section, we discuss why Paxos is too expensive for low-power wireless networks. We then observe that, in fact, Paxos does not require unicast communications but can be represented as a many-to-many scheme, which makes it suitable for low-power wireless networks. Finally, we explain how the broadcast-oriented wireless medium provides opportunities to develop an efficient and low-latency version of Paxos.

3.1 Cost of Paxos

The goal of Paxos is to provide a solution to the consensus problem that is resilient against both node failures and network faults. It does so by relying on the responses of a majority of nodes only, and by providing semantics that force all nodes to agree on the exact same decision. It is proven to be correct and has become one of the default protocols for consistent data replication within data-centers.

Paxos is an expensive protocol to run, especially in resource-constrained wireless networks. To achieve consensus, Paxos requires $2N+2$ messages, N being the number of nodes. For example, the public and multi-hop deployment of Euratech [1] is composed of 188 nodes. Paxos would therefore need to transmit 378 messages to achieve a consensus, usually by relying on end-to-end routing.

While data-centers networks can easily sustain this cost, it becomes impractical in low-power wireless networks with strong resource constraints, as bandwidth and memory are limited, and nodes must keep their duty cycle to a minimum to save energy.

3.2 Paxos Beyond Unicast?

We observe that, indeed, unicasts are not needed when Paxos is executed outside of wired networks.

Paxos with multicast. A phase of Paxos can be divided into two steps: a proposer sharing its request with acceptors, and acceptors responding to the proposer. The first step can be assimilated to a broadcast, or a *dissemination*. Using unicast to disseminate a request is expensive, and several approaches use multicast communication instead (e.g., [5, 32]).

The second step can be assimilated to a *collection*, where each response might be different. Multicast cannot be used here, as Paxos requires many acceptors to communicate towards one proposer. To solve this, Ring Paxos [32] builds a logical ring composed of acceptors. An accept response traverses the ring if no higher proposal is present, and the

proposal is chosen once a response traversed back to the proposer. By doing so, Ring Paxos minimizes the number of unicasts needed, but heavily relies on a stable topology.

Many-to-many. We make the observation that the majority of acceptors does not need to be statically defined in advance (e.g., by constructing a ring), but instead can be composed on the fly. By exploiting the broadcast nature of the wireless medium, acceptors (locally) broadcast their response, and *aggregate* responses heard from their neighbors. Eventually, aggregation leads to a majority of responses combined. Additionally, we explain in §4.1 that a proposer is not dependent on each response individually, but rather on the application of the maximum function over those responses. We can therefore distribute this logic to all acceptors, thus completely removing the need for unicast.

The execution of Paxos therefore becomes a disseminate-and-aggregate scheme (many-to-many), which ideally maps to the concept of Chaos [29]. Fig. 3 shows how an execution of Wireless Paxos looks like when using concurrent transmissions and an aggregation function, in contrast to the traditional execution of Paxos (Fig. 1). In conclusion, it is possible to express Paxos as a many-to-many scheme, which can be efficiently executed in low-power wireless networks.

3.3 Basic Idea: Wireless Paxos

Based on §3.2, we have a mapping between Paxos and Chaos, a protocol for many-to-many communication that is highly reliable and low-latency. Therefore, it is possible to design an efficient version of Paxos for low-power wireless networks. We *co-design* Paxos with the lower layers of the network stack to provide network-wide consensus at low-latency. Specifically, we base the design of Wireless Paxos on three principles:

- **Broadcast-Oriented Communication:** We take advantage of the broadcast properties of the wireless medium to disseminate requests and collect responses efficiently from all nodes.
- **Concurrent Transmissions:** Like Chaos and A^2, we build on top of concurrent transmissions to provide low-latency and high reliability.
- **Local Computing and Aggregation:** We distribute the decision logic of the proposer to all nodes through an aggregation function to convert Paxos to a many-to-many scheme.

4 Designing Wireless Paxos

In this section, we dive deep into the design of Wireless Paxos. We begin by breaking down our solution. Next, we extend our concepts to its Wireless Multi-Paxos counter-part. Finally, we explain key mechanisms of our work.

4.1 Wireless Paxos

We present the design of Wireless Paxos, its phases and the effect of flooding on the protocol.

Failure model. Wireless Paxos uses a partially-synchronous communication-model and a non-Byzantine failure-model with crash recovery (see §2.2). Partial synchrony is given by the use of Synchrotron.

Flooding rounds and slots. As Wireless Paxos builds on Synchrotron, it follows a similar nomenclature, as depicted in Fig. 2. Wireless Paxos is executed within "flooding" rounds. A round is divided into slots. At each slot, a node either transmits, tries to receive a message, or sleeps. The node then has time for computation before the next slot starts.

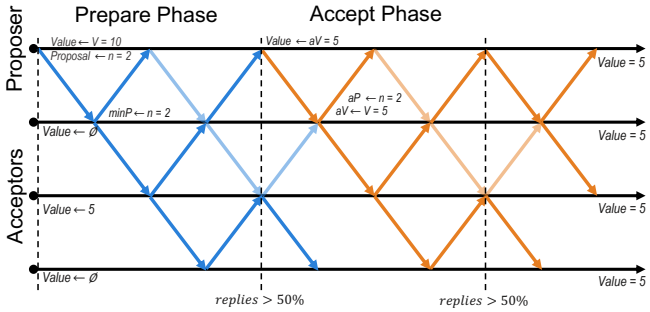


Figure 3. Executing Wireless Paxos. *The proposer broadcasts a Prepare request. Upon reception, acceptors add their response to the message and retransmit it concurrently. Once the proposer receives a majority of participation, it switches the phase and sends an Accept request. Messages are propagated in the network and the value is adopted.*

Prepare phase. In the original protocol, a proposer broadcasts its proposal number to a majority of acceptors, which reply with their accepted proposal and value if any. The proposer then selects the result with the highest proposal number and reuses the value associated.

In Wireless Paxos, the proposer starts to disseminate a prepare request with its proposal number and two additional fields for the acceptors to fill in. Upon reception, an acceptor applies the maximum function between its accepted proposal and the one received. If the local variable is higher, it replaces both the proposal and the value by the ones in memory. In addition, an acceptor always indicates its participation to the message by setting a corresponding bit in the flag field of Synchrotron.

The phase finishes as soon as the proposer receives back its proposal with at least more than half of the flags set. Applying locally the maximum function will always lead to the highest proposal number of the majority because the maximum is a commutative and idempotent function.

Accept phase. We follow a similar principle for the Accept phase. In the original protocol, the proposer broadcasts the value to agree on, as well as its proposal number. The acceptors reply with the highest proposal they prepared for. If the proposer receives a reply with any proposal higher than its own, the proposal is rejected.

In our implementation, the proposer broadcasts its proposal number, the value to agree on and an additional empty field for the responses. Upon reception, an acceptor returns the highest proposal it heard of. The proposer can therefore learn if it lost the consensus, i.e., there is a proposal higher than its own, or that the value was chosen once that a majority participated in the agreement. Again, we use the maximum function here. While it is not mandatory for the proposer to learn which proposal is the highest, this information can be used in the case of competition between multiple

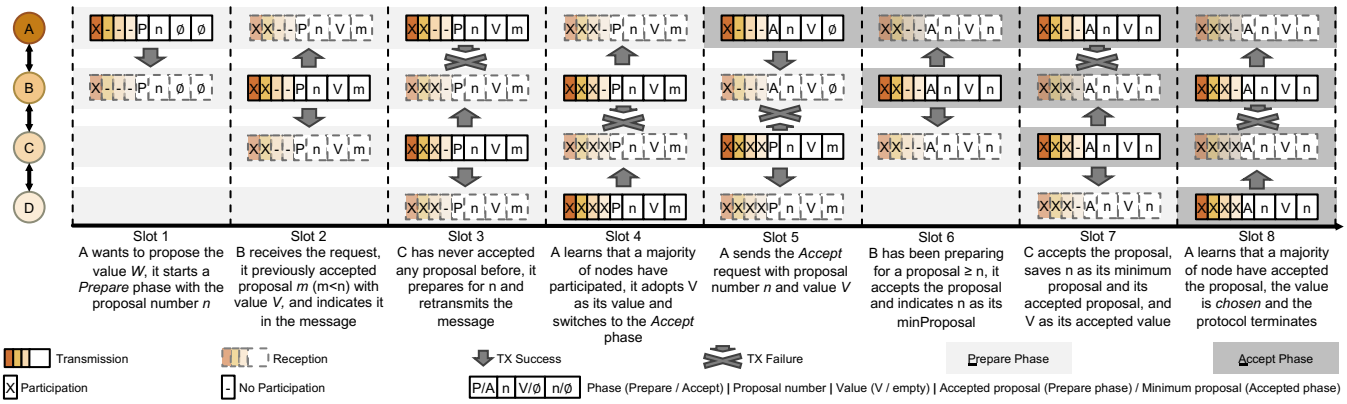


Figure 4. Wireless Paxos in action. *All nodes act as an acceptor and node A acts additionally as a proposer. Node A wants an agreement on value W. It initiates at slot 1 a Prepare phase with proposal number n. The value and proposal fields are left empty. At slot 2, B informs the network that it has previously accepted value V with proposal number m. At slot 5, A learns that a majority of nodes have participated, it adopts value V as its own and switches to Accept phase in slot 6. At slot 7, B informs that the highest proposal it prepared for is n, and accepts the value V. At slot 8, A learns that a majority of nodes have accepted the value since no higher proposal than n was reported, the value V is therefore chosen by the network. RX failures happen due to concurrent transmissions. Transmissions continue afterwards until all nodes receive all the flags set.*

proposers.

Optional dissemination. Paxos guarantees that a majority of nodes, but not all, are aware of a value being proposed (but not chosen). It also guarantees that no other value can be shared in the network afterwards. The proposer must contact the learners to disseminate the final decision. Since our design is based on flooding, Wireless Paxos provides a built-in dissemination of the decision.

Any node receiving an Accept phase with a majority of participation and no higher proposal included knows that the value is chosen, and can *learn* this value as the final decision. In addition, the network-wide flooding of Synchrotron will force all nodes available to hear about the decision, therefore providing a network-wide dissemination of the result with high probability.

A typical execution. Fig. 4 represents an execution of the Paxos primitive with four nodes. All nodes act as acceptors while A is the only node with a proposer behavior. A starts the flooding round with the objective of agreeing on value K. However, node B has accepted a proposal with value V in the past. This example shows how Paxos avoids inconsistencies by adopting previously accepted values.

4.2 Wireless Multi-Paxos

Next, we design a primitive that provides the functionality of Multi-Paxos for agreement on a continuous stream of values and state machine replication. The optimization of Multi-Paxos over Paxos results in a lower duty-cycle, as we now only execute the Accept phase most of the time. We highlight selected mechanisms of Wireless Multi-Paxos.

Bounded memory. The original Multi-Paxos specification requires both unbounded memory and message size. Like other implementations, we relax those requirements and limit the memory space available. A memory buffer, or log, is used to save the last values agreed upon, with a predefined *log size*. A *message length* is fixed to accommodate multiple values in one flooding round. A large log size allows nodes to recover from past failures.

Multi-Paxos allows multiple (Paxos) rounds to be executed at once, by aggregating all the requests into one message. Our design also allows to agree on multiple values at once, but requires that those rounds are consecutive in order to reduce the amount of data transmitted (see §2.3).

Prepare-phase specifics. Multi-Paxos requires proposers to learn the outcome of all previous rounds since the start of the system. The goal is to avoid inconsistencies and allow nodes with missing values to recover, or insert a special no-operation token to maintain state machine replication if no value was chosen. Wireless Multi-Paxos provides the same behavior. Proposers must learn all previous values in memory before agreeing on any new value.

However, a proposer will not be able to learn all previous values at once if the *message length* is smaller than the log of values. We use an *iterative learning process*, that allows the proposer to iterate back-and-forth between the prepare and accept phase until all previous values have been learned.

A new proposer starts the prepare phase to learn old values, disseminates them with the accept phase, and iterates back to the prepare phase to learn the next values, until all values have been learned.

Ordering messages. With Wireless Paxos, it is easy to compare two messages to decide which one is newer. The tuple $(proposal, phase)$ is sufficient, since a higher proposal is always newer, and the Accept phase is a newer message if both requests have the same proposal number.

However, with Wireless Multi-Paxos, the comparison of the phase does not hold anymore. We extend it with the tuple $(proposal, round, phase)$. Since the round number increases with the iteration process, messages are correctly ordered once again.

Leader lease. For the optimization to hold, Multi-Paxos

Table 1. Estimating the Cost of Feedback in Euratech with 188 nodes. *By repeating the flood multiple times, reliability is improved but no feedback is available (Repeated Glossy). Each node can report its status with a new flood (Glossy with Feedback) at a very high cost. Wireless Multi-Paxos provides consensus at a lower cost.*

Protocol	Repeated Glossy	Glossy with Feedback	Wireless Multi-Paxos
Latency [ms]	100	3760	500

Table 2. Statistics and parameters of testbeds used in the evaluation. *They represent a very dense and a low-density deployment, respectively. Both are collocated with Wi-Fi and Bluetooth deployments.*

Testbed	*Size* [#]	*Coord.* ID	*Dens.* [#]	*Diam.* [hops]	*Chann.* [#]	*TX Pw.* [dBm]
Euratech	188	3	106	2	16	0
Flocklab	27	3	7	4	2	0

requires that at most one proposer is present in the system for a prolonged period of time. The literature refers to this proposer as the *leader*. To avoid unnecessary competition between self-proclaimed leaders, we implement a *leader lease*. Each node implicitly acknowledges the proposer with the highest proposal as the leader, and promises not to compete until it believes the leader has crashed.

Once it believes the leader has crashed, a node throws a coin before proclaiming itself as the new leader.

4.3 System Details

In this section, we list key mechanisms at play in Wireless Paxos that do not depend on a specific phase or protocol.

Proposer and acceptor. In Wireless Paxos, all nodes in the network act as acceptors. In addition, any node can act as proposer. If this is the case, the node will first execute the acceptor logic, and then the proposer within the same slot. Due to the properties of the protocol, all nodes are also learners, at no additional computing cost.

Proposal cohabitation. We explain in §4.1 how to order messages. Paxos requires acceptors to discard any request older than the *minProposal* request. We supplement this requirement by transmitting the newest message every time an older request is received. We therefore reduce the risk and cost of propagating outdated data.

Phase cohabitation. Due to the flooding mechanism of Synchrotron, both phases will co-exist during a transitional period. We reduce this transition period by two means: **(a)** The proposer (and all acceptors) will always transmit the new phase if they receive an older phase transmission; and **(b)** All other nodes will reduce their transmission rate if they receive a prepare phase message with a majority of flags and resume at the normal rate once an accept phase is received.

4.4 Design Discussions

In this section, we explain why building consensus with Glossy is more costly than Wireless Paxos.

Cost of Feedback. Glossy offers an ultra-low latency and highly reliable ($> 99.99\%$) dissemination. It means that, under normal conditions, most of the nodes in the network will receive the value. It is however not guaranteed, since Glossy does not use explicit feedback. For example, an initiator will never detect a network segmentation, or high message losses, or node failures. An initiator only knows if at least one node received the flood (due to the semantics of Glossy), and nodes implicitly detect missed floods, but cannot recover the value. Repeating multiple times the flood increases further reliability, but still doesn't provide additional guarantees. To provide explicit feedback, each node would be required to start its own flood to report its status (e.g., see [17]).

Small example. For example, in the Euratech testbed, with $N = 188$ nodes, a flood takes 20 ms. Table 1 gives a back-of-the-envelope calculation of the cost of feedback with Glossy. Repeating 5 times a flood takes 100 ms. Receiving feedback from all nodes requires 3760 ms. In contrast, we show in §5.3 that 193 ms are needed with Wireless Multi-Paxos to get a majority of replies, and 500 ms for all nodes to hear about all other nodes.

4.5 On the Correctness of Wireless Paxos

In this section, we give a short and informal walk-through of why Wireless Paxos does not break the correctness of Paxos (see [26] for the original proofs). The main difference between Paxos and its wireless counterpart lies in the *maximum* function being distributed from the proposer to all nodes. While Paxos requires a majority of replies, Wireless Paxos requires an aggregate from these replies.

As both the *maximum* and the *(set) union* functions — used to compute the number of replies — are commutative and idempotent, the local execution by all nodes of both functions is equivalent to the execution of the functions by the proposer alone. As such, Wireless Paxos maintains the safety properties of Paxos.

5 Evaluation

Paxos is a notoriously complex protocol to implement, and even tougher to evaluate [8, 36]. Wireless Paxos is no different from that perspective. Proving that an implementation of more than a thousand lines is correct is often an extremely delicate and cumbersome task. We instead decide to use an extensive testing approach (like [8]), during which we evaluate our system on physical deployments of different topology and density.

We begin by discussing our setup. We then follow a bottom-up approach, and present how a node's state evolves during a round. Then, we compare both the cost and fault resilience of Wireless Paxos with related network-wide primitives.

5.1 Evaluation Setup

In this section, we lay out how we implement Wireless Paxos and what scenarios are executed for the evaluation. We then list the different metrics and testbeds used.

Table 3. Slot length of each protocol. *Due to their complexity, WPaxos and WMulti-Paxos require more computation time.*

Protocol	Glossy, 2&3PC	WPaxos	WMulti-Paxos
Slot Length	4 ms	5 ms	6 ms

Implementation. We implement Wireless Paxos in C, on top of Synchrotron [2], for the Contiki OS [14]. We target wireless sensor nodes equipped with a low-power radio such as TelosB and WSN430 platforms which feature a 16bit MSP430 CPU @ 4 MHz, 10 kB of RAM, 48 kB of firmware storage and a CC2420 [42] radio compatible with 802.15.4.

Testbeds. We use two publicly available testbeds for our evaluation: Flocklab [31] and the Euratech deployment of FIT-IoT Lab [1]. Flocklab is composed of 27 nodes while Euratech contained up to 214 nodes, with around 188 active during our evaluation. Table 2 summarizes properties of both deployments. Due to the closure of the Euratech testbed, some parts of the evaluation are carried in Flocklab only.

Scenarios. We evaluate the following applications:

- The Wireless Paxos primitive (*WPaxos*): one proposer starts a consensus on a 1-byte data item, leading to a total payload of 31 bytes in Euratech and 10 bytes in Flocklab. Both phases are executed at every flooding round. We refer the reader to [29] for the effect of varying payload size over the system;
- The Wireless Multi-Paxos primitive (*WMulti-Paxos*): one proposer starts a series of consensus on 1-byte data items. Each flooding round corresponds to one value. Acceptors keep a log of the last four values they agreed on. The Prepare phase is executed during the first flooding round and is skipped afterward;
- WPaxos with multiple proposers: a pre-defined number of proposers are competing at each flooding round on different data items;
- Consistency: At each slot, a node has a pre-defined probability to enter in failure mode. A failed node stops to use its radio. At the end of a flooding round, we compare each node's state to detect any inconsistency in the consensus.

Metrics. We focus on the following indicators:

- Progress and State: the progress of a node represents the number of participants that node heard of. Its state represents the phase of the protocol (prepare, accept);
- Latency is refined into two notions: the *Paxos latency*, representing the time when a value is chosen, and the *full completion latency*, representing the time when the lower layer Synchrotron converged (see §4.1 and §5.3);
- Radio-on time: the total time the radio is active during a flooding round. It is used as a proxy for the energy consumed during a round;
- Consistency: from a network-wide perspective, it represents if nodes agree on the same final value.

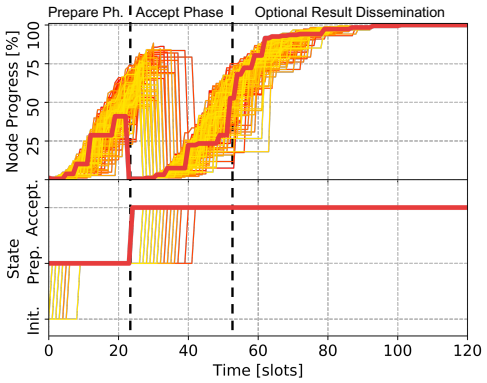


Figure 5. A snapshot of a typical WPaxos round. *The upper figure represents the progress of each node (proposer is the thick red line) while the lower represents their state. It takes 21 slots for the proposer to receive a majority of replies and switch to the Accept phase. It takes an additional 35 slots for the Accept phase to complete with a majority of flags. Results converged after 115 slots and the flood finishes.*

Slot Length. Different protocols have different complexity, and require a different amount of execution time. Table 3 contains the different slot length used during the evaluation. Due to their complexity, both WPaxos and WMulti-Paxos require additional computation time compared to the literature. Note that the hardware used features a 4 MHz CPU. The computation time would be reduced on newer hardware with a faster CPU.

5.2 Dissecting Wireless Paxos

We evaluate how a consensus is handled with Wireless Paxos. We first analyze a representative instance of the protocol through the different node states. Then, we compare the execution of WPaxos and WMulti-Paxos.

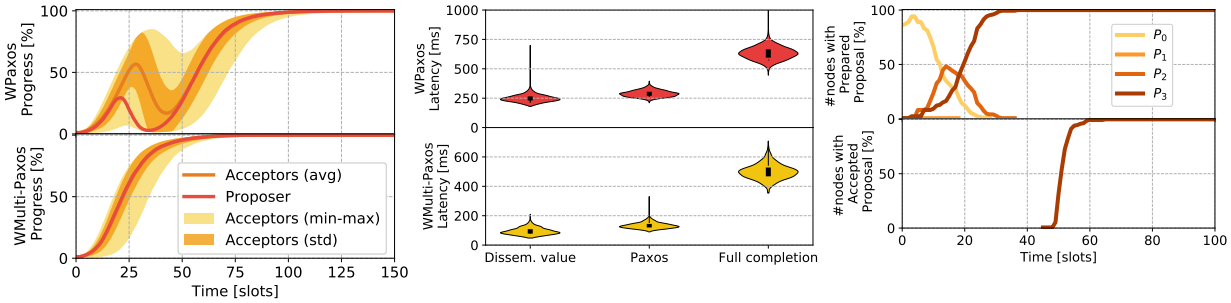
Basic Round. Fig. 5 depicts a representative execution of Wireless Paxos in Euratech with 188 nodes. All nodes start in the initial, empty state.

At slot 0, the proposer (represented by a red thick line) starts the round with a Prepare request. Since the proposer also acts as an acceptor, the node transitions into the Prepared state. Due to the high density of Euratech, it takes around 10 slots (50 ms), for all nodes to hear the request and enter the prepared state.

After 21 slots (105 ms), the proposer detects that a majority of nodes participated and replied with a promise. It starts the accept phase, resets back the progress to 0 and switches into the Accepted state. Due to the prepare phase still spreading, a transition period of roughly 20 slots (100 ms) is necessary for the acceptors to learn about the new phase. At slot 56 (280 ms), the proposer learns that a majority of the nodes accepted the value (referred as *Paxos latency*). The value is therefore *chosen* and the consensus succeeded.

The results naturally converge and at slot 115 (575 ms), the 188 nodes learned that the entire network agreed on the value (*full completion latency*).

Note. It takes exactly the same time if another value is



(a) Comparing WPaxos and WMulti-Paxos. **(b) Different Definitions of Consensus.** **(c) Multiple Proposers Competing (WPaxos).**

Figure 6. Executing Wireless Paxos (WPaxos) and Wireless Multi-Paxos (WMulti-Paxos) in Euratech with 188 nodes. ***(a)*** *WMulti-Paxos requires only one phase and finishes in roughly 83 slots, while WPaxos requires around 127 slots for its two phases.*
(b) *In Euratech, 248 ms and 94 ms are necessary for a dissemination to all 188 nodes for WPaxos and WMulti-Paxos respectively. 289 ms and 133 ms are needed for the proposer to hear a majority, and it takes 633 ms and 500 ms for a network-wide knowledge of the result.*
(c) *A line represents the number of nodes that locally prepared or accepted a proposal, but not the progress seen by a proposer. Proposals are initially competing but quickly ruled out by the highest proposal, and only P_3 sees a majority of replies.*

present in the system. The proposer would simply replace its own value by the one reported at the end of the first phase, and disseminate it during the accept phase.

WPaxos and WMulti-Paxos. Paxos is not an efficient protocol. The prepare phase is necessary only if multiple proposers are present or the node just started as proposer. The first phase is therefore superfluous the rest of the time. Multi-Paxos builds on that knowledge and executes the prepare phase only once (cf. §2.3 and §4.2).

Fig. 6a compares the average *full completion latency*, i.e., time until Synchrotron completion, for WPaxos and WMulti-Paxos with 188 nodes. The red line represents the proposer's progress, while the orange line represents the average progress of the acceptors. The dark-orange area represents the standard deviation of the acceptors' progress, and the light-orange area the minimum and maximum progress, i.e., the slowest and fastest acceptor respectively.

It takes by average 127 slots (633 ms), for WPaxos to complete (from a Synchrotron perspective), while it takes only 83 slots (500 ms) for WMulti-Paxos. Removing the first phase improves the latency although WMulti-Paxos requires more computation time (cf. Table 3).

5.3 Paxos and Primitive Latencies

Paxos defines a consensus complete once a majority of accept responses have been received by the proposer. However, the lower layer Synchrotron does not stop communication at that point, but continues until all nodes have converged (i.e., until all nodes see all flags set, see §4.1). We measure the latency from Paxos and Wireless Paxos perspectives.

Metrics. We select three definitions of latency: **(a)** *Disseminated value*: similar to Glossy, corresponds to the time required for all nodes to hear the value the first time; **(b)** *Paxos*: following Paxos definition, a correct consensus requires that the proposer receives a majority of replies during the accept phase; **(c)** *Full completion*: similar to the *Max* primitive of Chaos [29], the latency is defined as the time required for all nodes to hear from all other nodes.

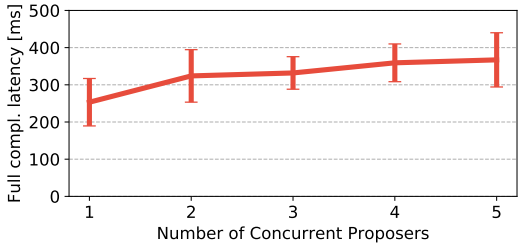


Figure 7. Cost of Multiple Proposers in Flocklab. *The competition between proposers causes a latency overhead. After 3 concurrent proposers, the overhead stops growing.*

Results. Fig. 6b shows the reported latencies. Simply disseminating the value is fast, it takes WPaxos only 248 ms to share the result to all nodes. Collecting feedback is more expensive, and it takes 289 ms for the proposer to learn that at least a majority replied to its Accept request (Paxos definition of consensus). Collecting the flags from all the nodes induces a large overhead, with a mean latency of 633 ms. On the other hand, WPaxos thus ensures that all nodes received the decision, beyond the majority ensured by Paxos.

WMulti-Paxos is faster. It takes 94 ms for the dissemination, 133 ms to hear from a majority and 500 ms to receive all flags.

5.4 Influence of Multiple Proposers

In this section, we study the effect of having more than one proposer in the system.

Scenario. Traditional agreement protocols require at most one node to act as a leader, and fail if multiple are present. Paxos, on the other hand, can deal with the presence of more than one proposer in the system. We run several hundred WPaxos rounds while increasing the number of proposers in the system. Proposers are chosen following a

uniform distribution, and send a prepare request as soon as a lower proposal is received. A proposer stops competing once a packet with a higher proposal number is received.

In a round. Fig. 6c depicts the competition between four proposers in the highly dense deployment of Euratech. P_0 represents the proposal of the flood initiator, while P_1 to P_3 are from randomly chosen proposers. As the request floods the network, many acceptors prepare for P_0. The other proposals start competing very early but their dispersion is slower due to the collisions during transmission. Intermediate proposals P_1 and P_2 cannot gain enough momentum to collect a majority of replies.

While both P_0 and P_3 are heard by a majority of acceptors, only P_3 manages to collect enough responses and "wins" the competition. After roughly 50 slots (250 ms), the proposer of P_3 starts disseminating the accept phase. The proposal is accepted as it propagates through the network.

Latency. Fig. 7 characterizes the effect of competition in terms of full completion latency in Flocklab, although the Paxos latency follows the same behavior. The presence of multiple proposers induces an overhead of up to 100 ms in Flocklab.

As the number of concurrent proposers grows, the overhead starts to plateau. Acceptors quickly discard lower proposals to force the spread of the highest proposal, and competitors are quickly ruled out of the system. In addition, a proposer will not compete if its own proposal is lower than the proposal received.

With WMulti-Paxos, the overhead is only present when the first phase is executed and with similar results. Any consecutive round is executed as usual as long as proposers do not compete again.

5.5 Comparing the Cost of Primitives

We evaluate the cost of running the WPaxos primitive, and compare it with dissemination (Glossy [18]) and agreement primitives (2&3PC from A^2 [2]), in terms of latency and radio-on time. Since the lower layer Synchrotron continues to communicate after a value is chosen, we use the full completion latency (see §5.3) as metric.

Scenario. We compare five applications: **(a)** Glossy Mode (Glossy), a one-to-all dissemination mode without flags; **(b)** Two-Phase Commit (2PC), an all-to-all agreement protocol with flags and votes introduced to A^2; **(c)** Three-Phase Commit (3PC), similar to 2PC but with an additional phase, also introduced by A^2; **(d)** the Wireless Paxos primitive; and **(e)** the Wireless Multi-Paxos primitive.

Experiments are run both for Flocklab with 27 nodes and Euratech with 188 nodes. All applications are executed for 2500 rounds in Flocklab, and for a thousand rounds in Euratech. Glossy, 2PC and 3PC are executed only for several hundred rounds in Euratech due to the final closure of the testbed. The results reported here are nonetheless consistent with the results reported by A^2 [2]. The slot length used are summarized in Table 3.

Results. Fig. 8 summarizes the comparison in terms of full completion latency, both in slots and milliseconds, and in terms of radio-on time. First, we observe a cost increase with the number of nodes. The increase is however not proportional, as Euratech is seven times larger than Flocklab but induces an increase of roughly 2.5× only. This is mainly due to the difference in density and topology of the deployments.

We now take a look at the performance of each application. Glossy is the fastest since it does not require feedback, but does not provide the same guarantees as consensus (see §2.1). WMulti-Paxos has the second lowest cost, since only one phase is executed. WPaxos shows a cost of roughly 1.5× the cost of WMulti-Paxos in terms of slots. This is due to the fact that the primitive has two phases, but the first one requires half the nodes only. 2PC is roughly 2× more expensive than WMulti-Paxos in terms of slots, while 3PC is roughly 3× more costly. Again, it translates to the two and three phases of 2&3PC, respectively.

Due to the different slot length (cf. Table 3), the improvements of WMulti-Paxos are less visible when considering the latency in milliseconds. WPaxos takes 1.3× the time of WMulti-Paxos, while 2PC takes 1.4× the time in Euratech and 3PC takes 2.4× the time of WMulti-Paxos. Again, we point out that these results are due to the hardware used (4 MHz CPU). Modern hardware would provide improved results, closer to the slot latency.

Note that in Flocklab, WPaxos and 2PC presents a similar latency, even if WPaxos requires fewer slots. For small deployments, the length of the slot has a bigger effect than for dense deployments.

Finally, the radio-on time represents how long the radio was active (either transmitting or receiving), and is used as a proxy for energy consumption. Once again, WMulti-Paxos and WPaxos require fewer transmissions than their counterparts 2PC and 3PC, since they rely on majorities.

5.6 Primitives Consistency

In this section, we evaluate the consistency of the different primitives under injected failures.

Scenario. Paxos, by design, is tolerant against failures, and solve them by using majorities, while 2PC delays the decision (consistency over availability). We compare how the different consensus primitives are affected by node failures.

We run Wireless Paxos on Flocklab for 900 rounds with different failure rates. At each slot, each node can fail following a given probability (from 0 to 4×10^{-5}), i.e., it stops communicating for the duration of the round. This failure model is similar to a network segmentation or a crash-recovery where the node saves the result of the consensus in a stable storage.

A round is considered consistent if all nodes have the same value in the end. In WPaxos, a consensus is also consistent if at least a majority share the same decision, even if some node missed the value. We refer to those cases as *MAJ-consistent*. A system can also abort a decision in 2PC and 3PC. The result is consistent if all nodes have aborted. A blocked round means no decision has been chosen yet. Other cases are inconsistent.

Results. Fig. 9 shows the result of consensus under failure. Because WMulti-Paxos executes the accept phase only, the consensus is faster and less prone to failure. WPaxos, with its two phases, is slightly less resilient. As the failure rate increases, some nodes are missing the final value. The consensus remains nonetheless correct and nodes can eventually learn the decision, keeping the system consistent.

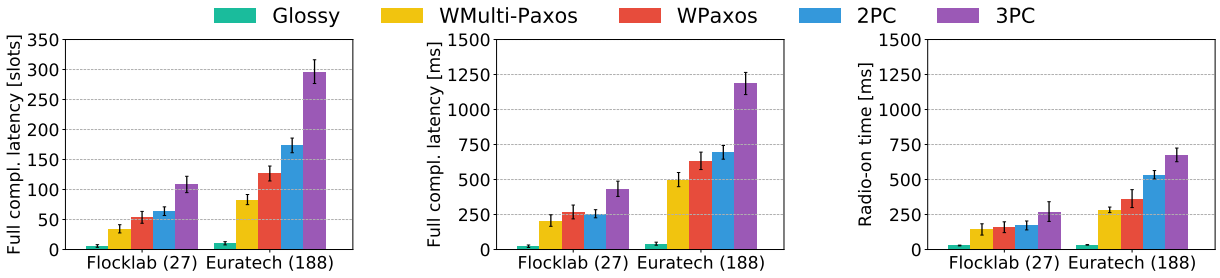


Figure 8. Comparing the cost of different primitives. *Glossy solves dissemination, WPaxos and WMulti-Paxos solve consensus while 2PC and 3PC (here from A^2 [2]) solve commit (agreement). The cost increases with the complexity (i.e., the number of phases). The latency is measured for the network-wide dissemination of the final decision.*

2PC blocks if at least one node is missing the decision. Strong consistency is maintained at the cost of availability. 3PC requires more communication, and is thus more prone to faults. Some rounds were inconsistent (some nodes aborted while others committed). However, 3PC is non-blocking by design.

6 Related Work

Concurrent Transmissions. Glossy [18] is one of the pioneer works in the field of concurrent transmissions. In Glossy, nodes synchronously transmit the same packet, allowing constructively interfering signals to be received. Glossy thus provides highly reliable and low latency dissemination. LWB [16] and Crystal [23] use Glossy to provide data dissemination and collection from all nodes by scheduling and executing floods sequentially.

By relying on the capture effect instead of constructive interference, Chaos [29] and Mixer [21] relax the tight synchronization condition and can transmit *different* data concurrently. Chaos uses in-network processing to provide data collection and aggregation by applying an aggregation function locally over the received data, while Mixer uses network coding to provide many-to-all communication.

Consensus. In distributed systems, agreement and consensus have been extensively studied for several decades already. Well-known solutions include 2PC [20] and 3PC [40]. Other solutions, like PBFT [7], provide a solution in the presence of Byzantine faults. Paxos is a general (non-Byzantine) fault-tolerant solution to the consensus problem [25, 26]. Paxos has been extended and further optimized, for example with Fast Paxos [27], Cheap Paxos [28] or Ring Paxos [32]. Raft [36] is an alternative to Paxos, designed to be more comprehensive.

Consensus in WSNs. Consensus has been studied in opportunistic and ad-hoc networks [3, 9]. However, most approaches focus on single-hop networks [4, 43]. Consensus with Byzantine failures in single-hop networks has also been studied [35].

Most multi-hop consensus solutions rely on routing. Köpke proposes an adapted 2PC for WSNs [24], while Borran et al. extends Paxos with a new communication layer for opportunistic networks [5]. Borran's solution relies on the MAC layer of 802.11 and builds a tree to collect and route responses. Furthermore, unicast and acknowledgments are used for collecting responses. In contrast, this work co-designs Paxos with the lower layers of the network stack to provide an efficient and low-latency consensus primitive for low-power wireless networks. We do not rely on any routing, but utilize concurrent transmissions to communicate in multi-hop networks.

A^2 provides an implementation of 2&3PC using concurrent transmissions [2]. Wireless Paxos reuses the transmission kernel introduced by A^2, but the consensus primitives differ. A^2 handles failures by delaying the decision (consistency over availability), while Paxos handles failures through majorities.

Finally, VIRTUS [17] brings virtual synchrony to low-power wireless networks. Virtual synchrony provides atomic multicast, i.e., it allows to deliver messages in order to all members of a group, or none. Virtual synchrony and Multi-Paxos are two ways to create state machine replication in distributed systems. Both solutions provide guarantees on the consistency of delivered messages. We argue in §4.4 why Glossy-based schemes (like VIRTUS) induce higher costs than Wireless Paxos.

7 Conclusion

This paper presents Wireless Paxos, a fault-tolerant, network-wide consensus primitive that builds on top of concurrent transmissions to offer low-latency and reliable consensus in low-power wireless networks. We argue that although established consensus protocols like Paxos offer many benefits like fault-tolerance, correctness, and consistency guarantees, their designs, based on unicast communications, make them unfit for low-power wireless deployments. Wireless Paxos fills this gap by showing that Paxos can be expressed as a many-to-many communication scheme, and by co-designing the consensus primitive along with the lower layers of the network stack and concurrent transmissions to offer highly reliable and low-latency consensus. We experimentally demonstrate that Wireless Paxos **(a)** guarantees that at most one value can be agreed upon, **(b)** provides consensus between 188 nodes in a testbed in 289 ms, and **(c)** stays consistent under injected failures.

Acknowledgments

We would like to thank the anonymous reviewers, as well as Simon Duquennoy, Marco Zimmerling, and Carlo Alberto Boano, for their valuable comments and suggestions to improve the paper. This work was supported by the Swedish

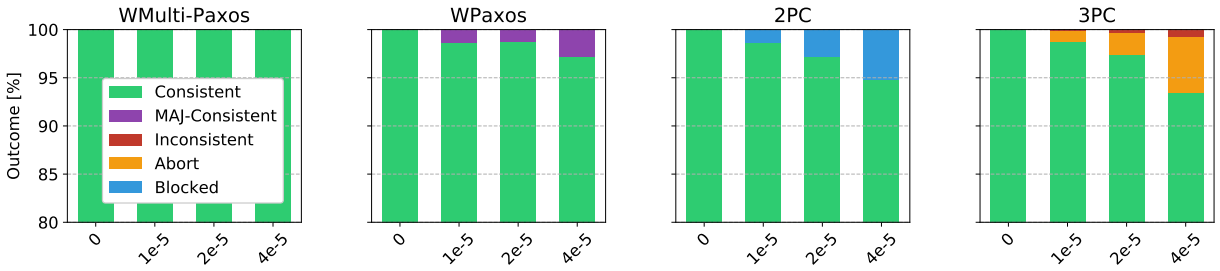


Figure 9. Consensus consistency under injected failure. *WPaxos handles failures with majorities and semantics, while 2PC is blocking (consistency over availability), and 3PC is non-blocking but sometimes inconsistent.*

Foundation for Strategic Research (SSF) through the project LoWi, reference FFL15-0062.

8 References

[1] C. Adjih, E. Baccelli, E. Fleury, and G. Harter et al. FIT IoT-LAB: A large scale open experimental IoT testbed. *IEEE WF-IoT*, 2015.

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

[3] A. Benchi and P. Launay. Solving Consensus in Opportunistic Networks. In *ICDCN*, 2015.

[4] C. A. Boano, M. A. Zúñiga, K. Römer, and T. Voigt. JAG: Reliable and predictable wireless agreement under external radio interference. In *IEEE RTSS*, 2012.

[5] F. Borran, R. Prakash, and A. Schiper. Extending paxos/lastvoting with an adequate communication layer for wireless ad hoc networks. In *IEEE SRDS*, 2008.

[6] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks. In *ACM SenSys*, 2006.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *UNESIX OSDI*, 1999.

[8] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: An engineering perspective. In *ACM PODC*, 2007.

[9] G. Chockler, M. Demirbas, S. Gilbert, C. Newport, and T. Nolte. Consensus and collision detectors in wireless ad hoc networks. In *ACM PODC*, 2005.

[10] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, and C. Frost et al. Spanner: Google's globally-distributed database. In *USENIX OSDI*, 2012.

[11] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems (3rd Ed.): Concepts and Design*. Addison-Wesley Longman Publ., 2001.

[12] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *USENIX NSDI*, 2013.

[13] A. Dunkels. The contikimac radio duty cycling protocol. Technical report, SICS, 2012.

[14] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *IEEE LCN*, 2004.

[15] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *ACM SenSys*, 2010.

[16] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *ACM SenSys*, 2012.

[17] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Virtual synchrony guarantees for cyber-physical systems. In *IEEE SRDS*, 2013.

[18] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *ACM/IEEE IPSN*, 2011.

[19] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 1985.

[20] J. Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, 1978.

[21] C. Herrmann, F. Mager, and M. Zimmerling. Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks. In *ACM SenSys*, 2018.

[22] M. Isard. Autopilot: Automatic data center management. Technical report, Microsoft, 2007.

[23] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza. Data prediction + synchronous transmissions = ultra-low power wireless sensor networks. In *ACM SenSys*, 2016.

[24] A. Köpke. *Engineering a communication protocol stack to support consensus in sensor networks*. PhD thesis, TU Berlin, 2012.

[25] L. Lamport. The part-time parliament. *ACM TOCS*, 16(2), 1998.

[26] L. Lamport. Paxos made simple. *SIGACT*, 32, 2001.

[27] L. Lamport. Fast Paxos. *Distributed Computing*, 19(2), 2006.

[28] L. Lamport and M. Massa. Cheap Paxos. In *IEEE/IFIP DSN*, 2004.

[29] 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.

[30] K. Leentvaar and J. Flint. The capture effect in FM receivers. *IEEE Trans. on Communications*, 1976.

[31] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, and P. Sommer et al. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *ACM/IEEE IPSN*, 2013.

[32] P. J. Marandi, M. Primi, N. Schiper, and F. Pedone. Ring paxos: A high-throughput atomic broadcast protocol. In *IEEE/IFIP DSN*, 2010.

[33] J. P. Martin and L. Alvisi. Fast byzantine consensus. *IEEE Trans. on Dependable and Secure Computing*, 3(3), 2006.

[34] I. Maza, F. Caballero, J. Capitán, J. R. Martínez-de-Dios, and A. Ollero. Experimental results in multi-UAV coordination for disaster management and civil security applications. *J. of Intelligent & Robotic Systems*, 61(1), 2011.

[35] H. Moniz, N. F. Neves, and M. Correia. Byzantine fault-tolerant consensus in wireless ad hoc networks. *IEEE Trans. on Mobile Computing*, 12(12), 2013.

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

[37] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *ACM SenSys*, 2004.

[38] W.-B. Pöttner, H. Seidel, J. Brown, U. Roedig, and L. Wolf. Constructing schedules for time-critical data delivery in wireless sensor networks. *ACM TOSN*, 10(3), 2014.

[39] J. Rao, E. J. Shekita, and S. Tata. Using Paxos to build a scalable, consistent, and highly available datastore. *VLDB Endowment*, 4(4), 2011.

[40] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Trans. on Soft. Eng.*, SE-9(3), 1983.

[41] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis. The β-factor: Measuring wireless link burstiness. In *ACM SenSys*, 2008.

[42] Texas Instruments. Chipcon CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver, 2006.

[43] Q. Wang, X. Vilajosana, and T. Watteyne. 6TiSCH Operation Sublayer Protocol (6P). IETF draft-ietf-6tisch-6top-protocol-12, IETF, 2018.