BUTLER: Increasing the Availability of Low-Power Wireless Communication Protocols

Fabian Mager
TU Dresden
fabian.mager@tu-dresden.de

Andreas Biri
ETH Zurich
abiri@ethz.ch

Lothar Thiele
ETH Zurich
thiele@ethz.ch

Marco Zimmerling
University of Freiburg
zimmerling@cs.uni-freiburg.de

Abstract

Over the past years, various low-power wireless protocols based on synchronous transmissions (ST) have been developed to meet the high dependability requirements of emerging cyber-physical applications. For example, Wireless Paxos provides consensus, a key mechanism for building fault-tolerant systems through replication. However, Wireless Paxos and other ST-based protocols are themselves *not* fault-tolerant: They suffer from a single point of failure that fundamentally impairs the availability of the communication service in the presence of node crashes and network partitions.

We present BUTLER, a mechanism that allows removing the single point of failure in many ST-based protocols. BUTLER synchronizes all nodes in the network so that the communication process can be jointly started by *multiple* randomly chosen nodes rather than a single dedicated node. We analyze and formally prove the correctness of BUTLER and implement it on the state-of-the-art nRF52840 platform. Experiments on the FlockLab testbed demonstrate that BUTLER reliably synchronizes the network to within $\pm 3\,\mu s$ despite large initial offsets, unpredictable node failures, and network partitions. BUTLER's temporal overhead ranges well below 1 %. Because of this efficiency and effectiveness, our results further indicate that BUTLER can dramatically improve the availability of an existing ST-based protocol without any noticeable impact on the overall communication reliability and efficiency.

Categories and Subject Descriptors

C.2.1 [**Computer-communication Networks**]: Network Architecture and Design—*Wireless communication*; C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems

General Terms

Design, Experimentation, Reliability

Keywords

Wireless sensor networks, High availability, Fault tolerance, Multi-hop networks, Cyber-physical systems

1 Introduction

In recent years, wireless sensor networks (WSNs) have become an integral part of cyber-physical systems (CPS) and the Industrial Internet of Things with applications ranging from personalized medicine through infrastructure control to smart factories. WSNs offer unprecedented flexibility and cost efficiency in terms of installation, operation, and maintenance compared to wired communication systems [4, 16, 25]. However, as defined by the International Society of Automation, the *dependability* of the wireless communication service is essential in these critical application domains, where small disruptions can cause system outages involving huge financial losses or even catastrophic consequences [36].

In parallel to advances in hardware enabling more capable yet ultra-low-power microcontrollers, communication protocols have also evolved. With Glossy [8], *synchronous transmissions* (ST) became popular, and many different ST-based protocols emerged in the following years that greatly outperform the traditional link-based protocols [38]. For example, in the EWSN Dependability Competition [29], which attracted participants from industry and academia, teams with ST-based protocols consistently placed in the top three ranks. These new protocols can satisfy higher application requirements and make WSNs suitable even for demanding closed-loop control applications [22, 33]. One of the key advantages of many ST-based protocols is their topology-independent protocol logic, which provides unprecedented resilience and flexibility as required, for example, in highly dynamic application scenarios with mobile robots [4] or drone swarms [11].

Problem. From an application's perspective, a dependable communication service should transport messages reliably across the network *and* be available when needed to ensure efficient and timely message delivery. State-of-the-art ST-based protocols [38] provide a highly reliable and efficient message transport. Furthermore, protocols like Virtus [7] and Wireless Paxos [24] provide mechanisms (virtual synchrony and consensus) to build higher-layer fault-tolerant systems.

Despite these achievements, *these protocols are themselves not fault-tolerant*: already the failure of a single node can lead to the unavailability of the communication service in the entire network. The fundamental problem is that ST-based protocols require tight time synchronization, which is achieved by selecting a particular node, often named *initiator* (see, e.g., [8, 18, 13]), that provides a time reference. Typical faults in WSN deployments (e.g., due to software/hardware failures, fabrication problems, environmental factors, adversarial attacks, and battery depletion [15]) can cause the initiator and hence the communication service to fail. Moreover, the single initiator is also a problem when the network splits into different partitions, for example, because the node connectivity is affected by environmental factors such as obstacles or interference and by moving nodes (e.g., a swarm of drones splitting up in flight). In these situations, the single initiator is only part of one partition, and the nodes in all other partitions are no longer able to exchange any messages.

Contribution. To solve the availability problem, we present the design, theoretical analysis, and experimental validation of BUTLER, a lightweight and distributed synchronization mechanism. BUTLER enables ST-based protocols to distribute the role of the initiator across multiple nodes (randomly selected at runtime) so that all non-faulty nodes that are physically able to communicate can do so at the required time. To achieve this, BUTLER uses a fully decentralized and highly efficient mechanism where nodes probabilistically propose and distribute reference times using short messages that feature a natural order. This order allows nodes to quickly converge toward the same accurate reference time despite possible node failures, message losses, and network partitions. Afterward, communication can be initiated in a dependable way by multiple *synchronized* initiators, which is key to leveraging the efficiency and reliability of ST.

After describing the design of BUTLER in Sec. 3, we formally analyze and prove its correctness in Sec. 4. Sec. 5 presents an open-source implementation of BUTLER on the popular nRF52840 platform, which we use in Sec. 6 to evaluate performance and efficiency on the FlockLab testbed [34]. Our results demonstrate that BUTLER reliably synchronizes all nodes in the network to within $\pm 3\,\mu s$ despite large initial time offsets and unpredictable node failures. BUTLER achieves this while incurring only a minimal temporal overhead that ranges below 1 % in realistic scenarios. Moreover, experiments with Mixer [13], an existing ST-based communication protocol, show that communication performance in terms of latency and reliability significantly decreases when using multiple initiators. When instead extending the standard Mixer protocol with BUTLER to synchronize the initiators, our results indicate no performance degradation: latency and reliability are at least as good as for the original Mixer with a single initiator while providing superior availability.

In summary, this paper contributes the following:

- The design of BUTLER, a lightweight and distributed synchronization mechanism that pushes the availability of ST-based protocols to previously unseen heights.
- A rigorous theoretical analysis of BUTLER, including a formal proof of BUTLER's correctness.
- Real-world experiments that validate the theoretical analysis by demonstrating outstanding synchronization accuracy at minimal temporal overhead despite node failures.
- A case study demonstrating that BUTLER increases the availability of a state-of-the-art ST-based protocol without sacrificing overall communication performance.

2 Motivation and Background

Providing high availability in WSNs is an important yet unsolved research problem. The problem originates from the dependability requirements of emerging CPS applications. We discuss these requirements next, then review previous approaches toward providing dependability in WSNs, and, finally, state the problem. Sec. 7 discusses existing work that is most closely related to our specific contributions.

Application requirements and fault model. WSNs offer high flexibility and cost efficiency, making them a key building block for many current and future CPS, including mission- and safety-critical applications [4]. In addition to high performance demands, these applications require a dependable communication service that enables reliable data exchange and is available when needed despite certain failures [36].

We define a communication service to be *available* if messages can be exchanged at the required time (i.e., as requested by the application or a high-layer protocol) between all nodes in the network that are physically able to communicate with each other (i.e., when the signal-to-interference-plus-noise ration between these nodes is high enough to permit information transfer). Providing availability thus requires robustness and fault tolerance such that failures of individual nodes do not affect the availability of the communication service between the non-faulty nodes.

WSNs are deployed at scale and consist of many low-cost, resource-constrained embedded devices that can fail for various reasons [5]. For example, because of software and hardware faults or depleted batteries, nodes may suddenly stop working (i.e., fail-stop). Nodes may also recover from a failure and resume operation (i.e., crash recovery). In addition, the environment of the deployment also has a significant impact on the network. Due to obstacles, external interference, and node movement, the communication links are constantly changing, which leads to time-varying message losses and may split the network into several isolated partitions. While we consider all aforementioned types of failures (i.e., node crashes, message losses, and network partitions), we do not consider Byzantine (i.e., erratic or malicious) faults. That is, we assume that a node works according to its specification whenever it is operational, and modifications to messages during transmission can be reliably detected (e.g., using error detection codes such as a cyclic redundancy check (CRC)).

Dependability in WSNs. Early WSN protocols adopted link-oriented and routing-based communication techniques [10] to meet the requirements of uncritical applications (e.g., environmental monitoring [32]). Yet, already in 2003, Stankovic et al. noted the importance of real-time and dependability guarantees in WSNs to meet the requirements of more demanding applications, such as those involving control [31].

Motivated by the growing importance of such CPS applications, recent WSN protocols based on ST have been shown

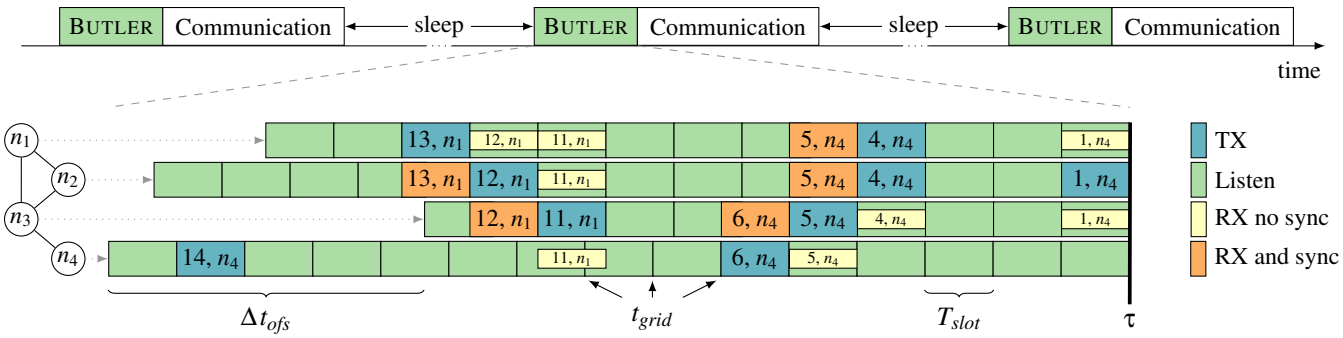


Figure 1: BUTLER is executed right before scheduled communication to synchronize the network. This removes the need for a single initiator in ST-based communication protocols, which increases their availability. *The example shows how 4 nodes (n_1 - n_4) with a maximum initial offset of Δt_{ofs} synchronize their local slot grids. The contents of the sync messages (T_τ and σ) are shown inside the slots, whereas T_τ is expressed in terms of the remaining number of slots until* BUTLER *terminates. Initially, n_1 - n_3 synchronize on the proposed reference time of n_1, but re-synchronize upon reception of the earlier reference time from n_4.*

to provide real-time guarantees [37] and certain dependability properties. For example, Virtus [7] provides atomic multicast and view management, while A^2 [3] and Wireless Paxos [24] provide distributed agreement and consensus. These dependability properties are fundamental to building fault-tolerant systems through redundancy and replication [28]. Providing these properties has been possible since ST allow the protocol logic to abstract away the complexity and dynamics of wireless networks. For instance, the temporary or long-term failure of individual wireless links can be smoothly handled by the spatio-temporal diversity of the ST technique [38].

Problem statement. However, ST-based protocols cannot deal with the failure of critical nodes, such as the node that initiates the communication process. Wireless Paxos, Virtus, A^2, and many other ST-based protocols (e.g., [8, 18, 13]) rely on one dedicated initiator that starts a packet exchange by transmitting first. This single initiator is a serious threat to the protocols' availability: a failure of this node prevents *any* communication in the network as all other nodes will keep waiting for an incoming transmission event that never occurs.

The reason for the single initiator is the need for an accurate time reference. ST require tight synchronization across nodes which must minimally satisfy the constraints set by the capture effect (e.g., 160 µs for IEEE 802.15.4) known as *capture window* [18]. If the time offsets of nodes exceed the capture window, communication becomes inefficient and unreliable.

To solve the availability problem, an ST-based protocol should ideally use a large set of multiple initiators, randomly and independently selected at runtime before every individual packet exchange. The problem, however, is that the nodes, and therefore the set of potential initiators for the next packet exchange, quickly get out of sync due to the inevitable clock drift between nodes. For example, the IEEE 802.15.4 standard [1] requires a clock drift of at most ± 40 ppm, which means that in the worst case, two initially perfectly synchronized nodes violate the capture window already after 2 s.

Based on these requirements, a scheme is needed that synchronizes the nodes to within the size of the capture window. Moreover, an effective synchronization mechanism must itself be fault-tolerant under the above-mentioned fault model, and be lightweight (i.e., low overhead) to avoid negatively affecting the overall communication performance.

3 Design

We introduce BUTLER, a synchronization mechanism that solves the problem outlined above to boost availability. Before presenting the details of BUTLER's design, we provide a high-level overview and state the scope of our work.

3.1 BUTLER Overview

BUTLER is a lightweight and distributed synchronization mechanism that is designed to directly integrate with existing communication protocols. It does not require periodic activity and is executed right before a scheduled communication round, as illustrated in the upper part of Fig. 1. BUTLER reliably establishes a common reference time τ among nodes despite node failures, message losses, and network partitions such that multiple initiators can start the communication protocol at the same time.

Because of clock drift, the nodes are initially unsynchronized and start BUTLER with different time offsets t_{ofs}, as shown in the lower part of Fig. 1. We assume a maximum initial offset Δt_{ofs} between the nodes, which can be calculated based on the current communication period and the known maximum clock drift. During the execution of BUTLER, the nodes probabilistically exchange *sync messages*. Each sync message is associated with a certain reference time. Using BUTLER, the nodes always synchronize to the earliest (minimum) reference time and propagate this reference time further in the network. Eventually, all nodes have the same reference time, which marks the end of BUTLER.

Scope. BUTLER is just one important piece of the puzzle to achieve high availability of the overall system. Specifically, BUTLER is meant to improve the availability of ST-based communication protocols that are responsible for the message exchange among nodes (Chaos [18], Mixer [13], and others [38]). These protocols, sometimes also referred to as *communication primitives*, can benefit from BUTLER if they are in principle able to support multiple initiators—any protocol that does not fulfill this requirement cannot provide availability! The necessity for multiple initiators rules out

protocols that can only realize a one-to-all message exchange in each execution (e.g., Glossy [8]) since there can only be one specific source node by design. Moreover, all protocols that build upon such protocols, e.g., the Low-Power Wireless Bus [6] or Crystal [14], are therefore inherently limited and cannot be made available with BUTLER. Higher-layer WSN protocols that also perform network management tasks at one dedicated node (e.g., scheduling) require additional mechanisms to provide availability by avoiding this single point of failure.

3.2 BUTLER in Detail

In the following, we will explain the structure and operation of BUTLER. Additionally, we will discuss certain design decisions and their impact.

Slot grid and sync message. In BUTLER, each node follows a local slot grid, as shown in Fig. 1, which is defined by the grid reference t_{grid} and the slot length T_{slot}. While T_{slot} is fixed and known to all nodes, each node uses t_{ofs}, the time at which it started BUTLER, as their initial t_{grid}. During the execution, nodes exchange sync messages to align their local slot grids and determine a common τ. Because the nodes align transmissions to their local slot grid, the receiving nodes will know the grid reference $\hat{t}_{grid}$ of the sender based on the receive timestamp. Each sync message is associated with a certain τ and contains two pieces of information: The duration T_τ, which is the time from $\hat{t}_{grid}$ until τ, and origin σ, the ID of the node that sent the particular τ for the first time. Note that sync messages with the same σ always describe the same τ. We use the size of a sync message to determine T_{slot}, because we want concurrent transmissions with the same σ to either overlap completely or not at all.

Operation. We explain BUTLER's operation based on the example in Fig. 1 and directly refer to the relevant lines in Algorithm 1. At the beginning, the nodes initialize the remaining BUTLER duration T_τ (i.e., the time until τ) as a multiple of T_{slot} (line 2). The grid reference t_{grid} and T_τ are used to determine the initial τ (line 4). Furthermore, all nodes start unsynchronized and set σ to 0 (line 5). BUTLER's main loop (line 7) is executed once per slot, at the end of which t_{grid} and T_τ are updated accordingly (lines 22-23). At $T_\tau = 0$, τ is reached and BUTLER terminates. In each slot, the nodes decide independently whether to transmit a sync message or listen. We start with the transmit decision and explain reception afterward.

The transmit decision is made with probability P_{TX} (line 9). If $\sigma = 0$, the node is still unsynchronized and will propose its own τ to the network, setting σ to its node ID (lines 10-11). The node then aligns the start of the transmission to the next slot ($t_{grid} + T_{slot}$) and sets the duration in the sync message to $T_\tau - T_{slot}$ (line 13). This is the case for the first transmission of n_1 and n_4 in Fig. 1.

In case a node decides not to transmit, it will listen for an incoming sync message. After a successful reception (line 16), the node computes the reference time $\hat{\tau}$ associated with the sync message (line 17). If $\sigma = 0$ or the received $\hat{\tau}$ is earlier than τ, the node synchronizes to $\hat{\tau}$ (lines 18-19). The rationale behind this is that the earliest reference time had the most time to propagate through the network and therefore is expected to reach the most nodes compared to other reference times. In Fig. 1, n_2 and n_3 synchronize to their first reception (RX and sync) because of $\sigma = 0$ and later re-synchronize due to $\hat{\tau} < \tau$. Synchronizing to a new reference time involves updating the local information with the information from the sync message and adjusting the local slot grid (line 25). To quickly spread the new τ, the node will always transmit in the next slot (line 26). In contrast, n_4 discards the first reception (RX no sync) because it is synchronized to an earlier reference time, hence the local slot grid remains unchanged as can be seen in Fig. 1. In the unlikely case that both $\hat{\tau}$ and τ are equal, the node synchronizes to the reference time of the sync message with the lower σ (lines 20-21).

Using Algorithm 1, BUTLER aligns the local slot grids of initially unsynchronized nodes to within T_{cap}, so that multiple nodes can reliably initiate the communication process at the same time in the subsequent communication round.

Algorithm 1 BUTLER

```
1:  procedure BUTLER_START ( )
2:      T_τ ← slots * T_slot                          ▷ Remaining duration
3:      t_grid ← t_ofs                                ▷ Slot grid reference
4:      τ ← t_grid + T_τ                              ▷ Reference time
5:      σ, tx ← 0

6:      ▷ Each loop iteration corresponds to one local slot ◁
7:      while T_τ > 0 do
8:          ▷ Make transmit decision ◁
9:          if tx or (P_TX > RANDOM()) then
10:             if σ = 0 then
11:                 σ ← n_i
12:             ▷ Transmission starts at t_grid + T_slot ◁
13:             TRANSMIT(T_τ − T_slot, σ)
14:             tx ← 0
15:         else
16:             if RECEIVE() ≠ 0 then
17:                 τ̂ = t̂_grid + T̂_τ   ▷ Received reference time
18:                 if (σ = 0) or (τ̂ < τ) then
19:                     SYNC ()
20:                 else if (τ̂ = τ) and (σ̂ < σ) then
21:                     SYNC ()
22:         t_grid ← t_grid + T_slot
23:         T_τ ← T_τ − T_slot

24: procedure SYNC ( )
25:     σ, T_τ, t_grid, τ ← σ̂, T̂_τ, t̂_grid, τ̂
26:     tx ← 1                          ▷ Relay new τ in the next slot
```

4 Analysis

After describing the design of BUTLER, we now theoretically analyze its synchronization behavior.

System model. We consider a system consisting of a set $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$ of N embedded devices (nodes). Each node n_i has a local clock that runs at a specific clock speed ν_i, which may vary from node to node due to imperfect clock sources (e.g., a crystal oscillator). The nodes have unique IDs and are equipped with half-duplex RF transceivers to transmit

Table 1: State transition matrix for node n_i in a network with N nodes. *Reference times represent the different states and are ordered from the latest (τ_1) to the earliest (τ_N). Each entry describes the transition probability from one state to another. The triangular form results from* BUTLER*'s behavior to synchronize only to earlier reference times.*

	To							
From	u	τ_1	τ_2	...	τ_i	...	τ_{N-1}	τ_N
u	$P_{RX}(\varnothing)$	$P_{RX}(\tau_1)$	$P_{RX}(\tau_2)$	...	P_{TX}	...	$P_{RX}(\tau_{N-1})$	$P_{RX}(\tau_N)$
τ_1	0	$P_{RX}(\varnothing)+P_{TX}$	$P_{RX}(\tau_2)$	...	0	...	$P_{RX}(\tau_{N-1})$	$P_{RX}(\tau_N)$
τ_2	0	0	$P_{RX}(\varnothing)+P_{TX}$	...	0	...	$P_{RX}(\tau_{N-1})$	$P_{RX}(\tau_N)$
...	0	0	0	...	0	...	$P_{RX}(\tau_{N-1})$	$P_{RX}(\tau_N)$
τ_i	0	0	0	0	$P_{RX}(\varnothing)+P_{TX}$	...	$P_{RX}(\tau_{N-1})$	$P_{RX}(\tau_N)$
...	0	0	0	0	0	...	$P_{RX}(\tau_{N-1})$	$P_{RX}(\tau_N)$
τ_{N-1}	0	0	0	0	0	0	$P_{RX}(\varnothing)+P_{TX}$	$P_{RX}(\tau_N)$
τ_N	0	0	0	0	0	0	0	1

and receive messages wirelessly. Communication over the shared wireless medium is unreliable, and the probability of successful packet reception is always below 1. Moreover, we assume that nodes do not have access to external synchronization sources such as GPS and must exclusively synchronize via communication. In general, multi-hop communication is needed to reach all nodes in the network because of the limited communication range. Due to environmental factors (e.g., interference), node faults, or node mobility, the network can split into partitions. We define a *network partition* as a subset of $\mathcal{N}$ where all nodes in the same partition can bidirectionally exchange information with each other over one or more hops.

4.1 Correctness of BUTLER

The goal of BUTLER is to achieve synchronicity among the nodes in the network such that multiple nodes can safely initiate the upcoming communication round (i.e., within the capture window T_{cap}). Therefore, BUTLER is *correct* if the maximum difference $\Delta\tau$ between the reference times of all nodes in the same network partition does not exceed T_{cap}.

Equal clock speeds. We begin with the simpler case, assuming all clocks run at the same speed, and prove the correctness of BUTLER for nodes in the same network partition.

LEMMA 1. *If all clock speeds are equal, then there is a unique total order of all reference times over the entire execution of* BUTLER.

PROOF. Reference times in BUTLER have a natural temporal order. If two reference times are equal, we select the reference time with the lower σ as the earlier one. However, nodes in BUTLER do not have a shared time base, so a sync message describes the reference time relative to its transmission time ($\tau = t_{grid} + T_\tau$). This relative duration is affected by the clock speed of the transmitting node, which would not be the case with absolute timestamps and a shared time base. Leveraging the assumption that all nodes have the same clock speed, the order of the reference times will be the same at any point during the execution of BUTLER. □

THEOREM 1. *If all clock speeds are equal,* BUTLER *is correct.*

PROOF. In BUTLER, the current reference time τ and its origin σ essentially describe the state τ_σ of a node. Based on Lemma 1 and without loss of generality, we assume the following (arbitrary) order among reference times $\tau_1 > \tau_2 > ... > \tau_N$, such that τ_N is the earliest. We can then create a corresponding state transition matrix for a node n_i, shown in Table 1, with the different states represented by the reference times. All nodes start in the unsynchronized state u, with $u > \tau_j$ for $1 \leq j \leq N$. The table entries describe the transition probabilities, e.g., n_i transitions from u to τ_2 with $P_{RX}(\tau_2)$, the probability of receiving τ_2 from any other node. These transition probabilities are highly dependent on the situation and continuously change based on factors such as network topology, environment, node behavior, as well as the state of its neighboring nodes. In BUTLER, nodes propose their own reference time only if they are in state u, i.e., they have not received any other reference time. Consequently, other nodes can reach τ_i only if n_i proposes it in the first place. Nodes remain in their current state if they either transmit (P_{TX}) or receive nothing ($P_{RX}(\varnothing)$), which includes receiving later reference times that are ignored.

In a network partition, all nodes can communicate with each other either directly or over multiple hops. Since the nodes decide randomly and independently if they transmit or listen, all transitions probabilities $P_{RX}(\tau_j)$ for proposed reference times τ_j in Table 1 are greater than 0. BUTLER ensures that transitions are only allowed toward earlier reference times, leading to the triangular form of the state transition matrix. Therefore, all nodes synchronize to the same reference time with high probability. Because we assume that all clock speeds are equal, it follows that the pairwise difference between the reference times of all nodes is $\Delta\tau = 0 < T_{cap}$, which proves correctness. □

Varying clock speeds. In the real world, the clock speeds of the nodes are imperfect and may vary within a certain range, specified by the frequency tolerance and stability properties of the clock source (e.g., a crystal oscillator). We can determine $\Delta\nu$, which is the maximum clock speed difference between two nodes based on the hardware specifications. For example, the IEEE 802.15.4 standard [1] requires $\Delta\nu = 80\,\text{ppm}$ ($\pm 40\,\text{ppm}$). We now extend the correctness proof for varying clock speeds by incorporating $\Delta\nu$.

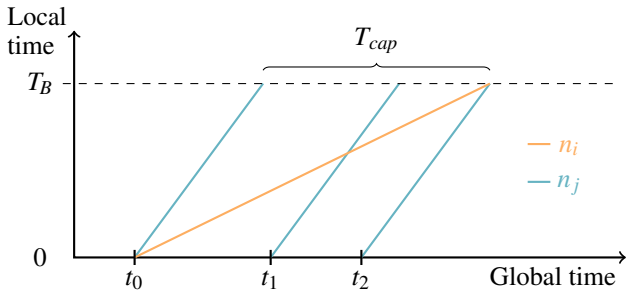


Figure 2: Local time vs. global time of nodes n_i and n_j with $\Delta\nu$ relative to each other. *The nodes are synchronized to different reference times. Three scenarios with different initial offsets for n_j are shown. Although the order of reference times can change during execution,* BUTLER *is still correct.*

LEMMA 2. *Correctness of* BUTLER *can only be ensured if the maximal execution duration $T_B \leq T_{cap}/\Delta\nu$.*

PROOF. We consider two nodes with $\Delta\nu$ relative to each other. Assuming that both nodes synchronize simultaneously to the same reference time, they are perfectly time-aligned at this point ($\Delta\tau = 0$). As time progresses, the local times of both nodes drift away from each other due to $\Delta\nu$, and it takes $T_{cap}/\Delta\nu$ time to have a difference of $\Delta\tau = T_{cap}$ between them. If BUTLER progresses further, correctness is violated, although both nodes are synchronized to the same reference time. Therefore, with varying clock speeds, it is necessary to limit the duration of BUTLER's execution to $T_B = T_{cap}/\Delta\nu$. □

THEOREM 2. BUTLER *is correct as long as $T_B \leq T_{cap}/\Delta\nu$.*

PROOF. With varying clock speeds, the order of the reference times can change during the execution of BUTLER. We will now prove BUTLER's correctness by analyzing the different situations that can occur with two reference times proposed by nodes n_i and n_j. Fig. 2 shows how the local times of n_i and n_j progress compared to the global time. We assume that both nodes have $\Delta\nu$ relative to each other, with n_i having the lowest and n_j having the highest clock speed. The nodes execute BUTLER for a duration of T_B according to the local time (y-axis), leading to different execution times concerning global time (x-axis). To visualize the different possible situations, we depict three scenarios for n_j, each with a different initial offset (t_0, t_1, and t_2) for the start of BUTLER, while n_i always starts at t_0.

If n_j starts BUTLER before t_0, τ_j will be earlier than τ_i for the entire BUTLER execution. Similarly, if n_j starts BUTLER after t_2, τ_i will always be earlier than τ_j. In these cases, all nodes eventually synchronize to the same reference time, and because of Lemma 2, $\Delta\tau$ will not exceed T_{cap}.

We now look at the case when n_j starts BUTLER between t_0 and t_2, for example at t_1. Since n_i starts BUTLER before n_j, τ_i is earlier than τ_j, and nodes would synchronize to τ_i upon reception. However, during the execution, the local time of n_j "overtakes" n_i (intersection), and the order of the reference times changes, i.e., $\tau_i > \tau_j$. Depending on the initial offset of n_j, this can happen at any time during the execution, leading to nodes possibly being synchronized to different reference times when BUTLER terminates. Nevertheless, because of Lemma 2, the difference $\Delta\tau$ between the reference times is less than or equal to T_{cap}.

As a result, either there is a unique reference time, or all chosen reference times differ by at most T_{cap}, implying BUTLER's correctness. □

4.2 Network partitions

BUTLER is a distributed synchronization mechanism with probabilistic transmit behavior that seamlessly supports network partitions. We assume that the network partitions can arbitrarily change between executions of BUTLER but that they remain stable while synchronization is ongoing, except that nodes can leave or fail at any point in time. This assumption is necessary to prevent a node with the earliest reference time from joining a new partition at the end of BUTLER, leaving no time for the other nodes in the partition to resynchronize. However, this is usually not a problem since the execution of BUTLER only takes a few tens of milliseconds (see Sec. 6.3).

With symmetrical communication links, and based on our definition of a network partition, all nodes that can communicate with each other must be in the same partition. Then, the presented proofs apply directly to each network partition. The state transition matrix in Table 1 would contain disjoint sets of states with one set per partition, and the transition probabilities between states of different sets would be 0 as no messages can be exchanged.

However, with asymmetrical links, nodes from one partition could receive a reference time τ_i from another node that is not in the same partition. If τ_i is earlier than all other reference times in the partition, all nodes will eventually synchronize to τ_i. Otherwise, if there exists an earlier reference time in the partition, then τ_i will be ignored. Therefore, the correctness of BUTLER is not affected by network partitions because it is irrelevant whether the node that proposed the reference time is part of the same partition.

4.3 Discussion

Limited time to converge. Our analysis shows an interesting area of tension between the theoretical proof of BUTLER's correctness and the practical challenge that the duration of BUTLER's execution is limited due to imperfect clocks ($\Delta\nu \neq 0$). Whether the maximum duration T_B of BUTLER (see Lemma 2) is sufficient for the nodes to converge on a single reference time depends on the network topology and environment, the node behavior, and the tolerance and stability of the clock source. In general, a network can have arbitrarily weak communication links so that the time to converge cannot be bounded. However, as we show in the evaluation in Sec. 6, these problems may be of low relevance in practice as the time to converge is several orders of magnitude lower than T_B. Among others, one reason is that the number of proposed reference times during the execution of BUTLER is low (as discussed in Sec. 5) compared to the overall number of nodes N, as most of the nodes will never propose their own reference time. Thus, the matrix in Table 1 will typically be sparse, which reduces the convergence time.

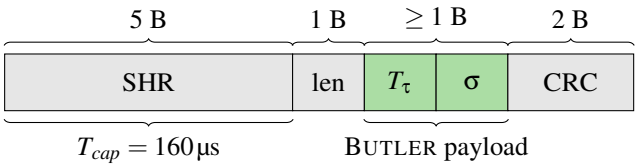


Figure 3: Packet structure of a sync message in BUTLER using the IEEE 802.15.4 physical layer.

Impact of interference. During the execution of BUTLER, the nodes will receive a reference time several times due to the random transmit behavior; thus, missing some messages, e.g., due to interference, can usually be compensated. In general, stronger interference leads to more message loss and increases the average time to converge but does not violate the correctness of BUTLER, which is independent of the receive probability. Note that a receive probability of 0 means the node is not connected. In practice, stronger interference can be proactively accounted for by deliberately extending the duration of BUTLER.

5 Implementation

We have implemented BUTLER on the popular Nordic nRF52840 platform using the IEEE 802.15.4 physical layer [1]. The code is published as open source at `https://gitlab.com/nes-lab/butler`.

Usage of BUTLER. It is straightforward to combine BUTLER with an existing ST-based communication protocol. BUTLER's API consists of the single function `butler_start(id)`, which takes the ID of the node as an argument. During the execution, BUTLER takes care of correctly handling all interrupts and should not be interfered with from the outside. Upon termination, the function returns the final reference time and origin shortly after reaching it. At this point, the nodes are synchronized and can start the next communication round. BUTLER does not require periodic or repeated execution and is scheduled on demand, provided the maximum initial offset Δt_{ofs} based on the communication period is known.

Sync message. Fig. 3 shows the structure of a *sync message* for the IEEE 802.15.4 physical layer. The synchronization header (SHR) is responsible for the capture window T_{cap} and contains the preamble and the start of frame delimiter. SHR and the length field (len) are mandatory parts of the communication standard. BUTLER adds the remaining duration T_τ and the origin σ as payload, whose space requirements are known at compile time but vary depending on the application and network size. Since the duration of BUTLER is initialized as a multiple of the slot length T_{slot} (line 2 in Algorithm 1), we can represent T_τ in the sync message more compactly as the number of remaining slots. To detect and filter out corrupted packets, we use a hardware-supported CRC.

From design to implementation. BUTLER operates in a slotted fashion (see Algorithm 1), where nodes decide to transmit or receive in every slot. The corresponding transmit probability P_{TX} will be low in practice (e.g., 2 % to 4 % in the evaluation), so nodes will often be receiving for multiple consecutive slots in a row. The nodes will continuously listen for incoming messages independent of slot boundaries and only align TX decisions to the slot grid. This implementation increases efficiency and avoids possible sync message misses at the slot boundaries, as sync messages can be received at any point in time.

In BUTLER's design, nodes switch instantly between RX and TX and vice versa, for example, after synchronizing to a new reference time. However, the radio hardware requires a turnaround time of 40 µs to execute this mode change. During the switch, the radio is deaf and cannot receive or transmit, effectively causing service downtime. One option to alleviate this issue would be to increase the slot length T_{slot} by the turnaround time, which would affect all slots. However, the number of slots in which a node synchronizes to a new reference time and is thereafter forced to transmit is only a fraction compared to the overall number of slots. Therefore, we instead opted to skip one slot when switching from RX to TX as it is more efficient to keep T_{slot} unchanged.

A crucial point in BUTLER is the computation of the reference time $\hat{\tau}$ from the received sync message (line 17 in Algorithm 1). The remaining duration $\hat{T}_\tau$ is part of the sync message and $\hat{t}_{grid}$ is determined based on the receive timestamp. This requires that the receive timestamp is equal to t_{grid} of the sender, except for negligible differences due to the time of flight of packets. We discovered that this is not the case on the nRF52840 platform and the receive timestamp is delayed by around 10 bit durations, depending on the data rate of the current radio mode. For example, using a data rate of 250 kbps and the IEEE 802.15.4 physical layer results in a receive timestamp delay of 40 µs, which has to be considered for the computation of $\hat{t}_{grid}$. As this delay splits equally between the TX and RX paths, we have to add 20 µs to the slot length T_{slot}, which is otherwise oriented at the size of the sync message (Fig. 3).

Variable transmit probabilities for efficiency. A probabilistic and independent transmit decision is essential to make BUTLER fault-tolerant and avoid single points of failure. As shown in Fig. 1, the local slot grids are initially unaligned, but increasingly synchronize as the execution progresses. At the beginning, the chances are high that concurrent transmissions overlap arbitrarily and violate the timing requirement T_{cap} of the capture effect, which leads to a reduced communication efficiency (i.e., a lower packet reception rate). Thus, P_{TX} should initially be chosen cautiously to reduce the number of concurrent transmissions. A side effect of a lower initial P_{TX} is that fewer reference times will be proposed, which decreases the overall convergence time. However, nodes already aligned to the same reference time can benefit from the capture effect since their transmissions start concurrently within T_{cap}. With an increasing number of aligned nodes, a higher P_{TX} improves the convergence to the final reference time. Therefore, the optimal value for P_{TX} varies over time and is network-specific. A similar challenge is faced in the Mixer protocol [13], where the transmit decisions depend on the local node density. We tested different topologies and found that choosing $P_{TX} = \frac{100\,\%}{N*2}$ until a node first transmits and doubling after that provides a conservative starting point for many topologies.

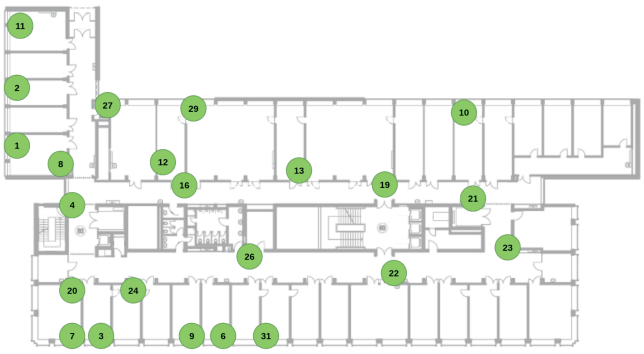


Figure 4: The FlockLab testbed with 23 nRF52840 devices.

6 Evaluation

Based on our implementation, we evaluate BUTLER in a real-world wireless testbed. We investigate BUTLER's behavior and confirm its correctness, together with measurements regarding performance and efficiency. Finally, we examine the interaction between BUTLER and Mixer[13], an ST-based communication protocol, and its impact on the communication performance. Our key findings are:

- *Correctness:* In our experiments, all nodes always synchronize to the same reference time at the end of BUTLER, validating BUTLER's correctness from the analysis.
- *Accuracy:* BUTLER synchronizes nodes to within $\pm 3\,\mu s$, which is well below the maximum tolerable time offset of 160 µs (i.e., size of the capture window).
- *Efficiency:* Thanks to BUTLER's efficient runtime execution, the temporal overhead of synchronizing the nodes is small and significantly below 1 % in most scenarios.
- *End-to-end performance:* BUTLER increases the availability of existing communication protocols without any negative impact on the communication performance.

6.1 Experimental Settings

All our experiments are executed on the FlockLab [34] testbed with 23 nodes deployed in an office environment as shown in Fig. 4. The experiments were conducted during the daytime and, thus, exposed to various sources of interference, e.g., WiFi. Our implementation uses the nRF52840 platform and the IEEE 802.15.4 physical layer. Using a transmit power of 8 dBm, the nodes form a network with 3-4 hops.

Since the clock drift of the nodes is unknown and also subject to change, we use the GPIO actuation capabilities of FlockLab to control the timing in our experiments. This also allows us to test larger initial offsets between the nodes without having to run excessively long experiments (i.e., with huge gaps between communication rounds). A final aspect is that by using the GPIO to purposely control the timing, we also increase the repeatability of our experiments. This way, each node realizes a random initial offset in the range $0 \leq t_{ofs} \leq 50\,ms$. The maximum offset $\Delta t_{ofs} = 50\,ms$ corresponds to a communication period of $\approx$10 min. To assess the robustness of BUTLER, we inject artificial node faults. Therefore, each node independently decides with a probability of 5 % not to participate in the next BUTLER execution. Furthermore, with this fault probability, the network sometimes splits into two partitions during the experiments, which permits an investigation of the behavior under network partitions.

In BUTLER, the sync information σ and T_τ require 1 B each, resulting in a packet size of 10 B (see Sec. 5) and a slot length T_{slot} of 335 µs. Using a few trial runs, we find that 250 slots (83.75 ms) are sufficient for BUTLER to synchronize all nodes on FlockLab for the considered range of initial offsets (see Sec. 6.3). We use two different values for the transmission probability (see Sec. 5), which are $P_{TX} = 2.2\,\%$ for the time until the first transmission, and $P_{TX} = 4.3\,\%$ afterward.

6.2 BUTLER in Action

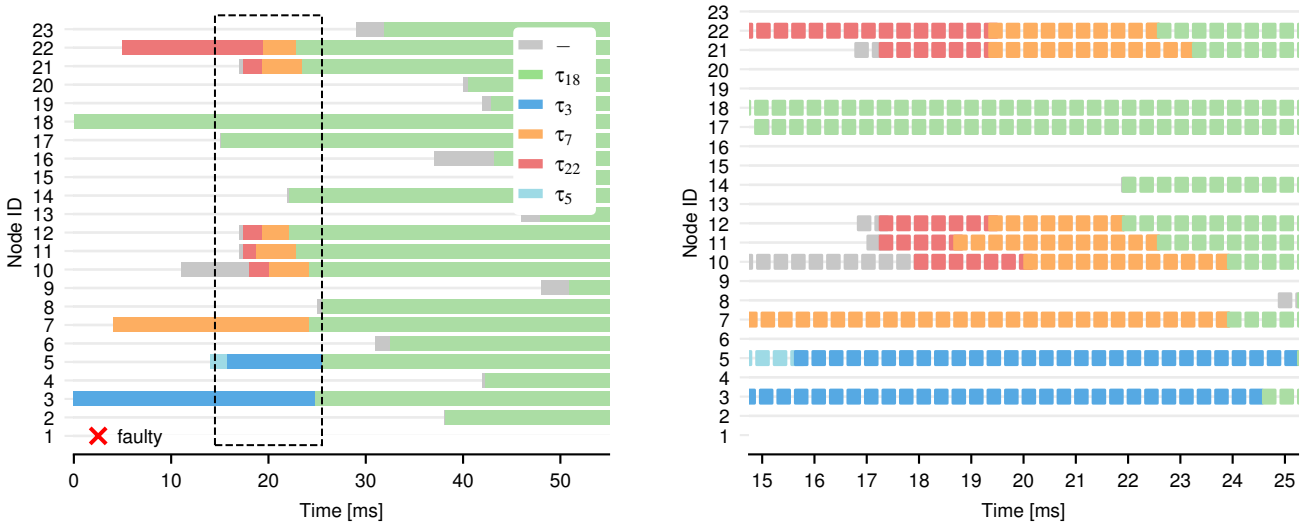
Before delving into the evaluation of BUTLER's performance, we look at its operation in a real low-power wireless network. Fig. 5 depicts the behavior of each of the 23 nodes on FlockLab during one representative execution of BUTLER. The beginning of a bar in Fig. 5a indicates when a node started the execution of BUTLER. The color of the bar indicates which proposed reference time a node currently follows. We can see that the nodes start BUTLER at different times with $\Delta t_{ofs} = 50\,ms$. Despite these significant initial offsets, BUTLER eventually makes all nodes follow the same reference time, as indicated by the green bars in the figure.

Diving a bit deeper into BUTLER's behavior in this particular run, we can see that most nodes initially have a gray bar. This means that these nodes did not propose their own reference time, but synchronized to the reference time of the first sync message they received. In contrast, nodes with a colored bar at the beginning did propose their own reference time at some point before receiving a sync message. Overall, there were 5 different reference times proposed in this execution, originating from nodes 3, 5, 7, 18, and 22. Node 18 started BUTLER first, a few microseconds before node 3, so the reference time of node 18 (green) is the earliest to which all other nodes eventually synchronize.

During the framed time interval between 15 ms to 25 ms, many nodes synchronize to new reference times. We zoom into this interval in Fig. 5b, where we can see the individual slots of BUTLER. Looking at the orange slots, we can see how the respective reference time propagates from hop to hop through parts of the network. At around 19 ms node 11 synchronizes to it. Two slots later, which is the implementation-specific delay due to the RX-TX turnaround time of the radio, node 11 transmits the new synchronization information (not shown in the figure) that is then received by the nodes 12, 21, and 22. Again two slots later, the orange reference time is further relayed to node 10. Eventually, however, the green reference time prevails as it is the earliest among all proposed reference times in this particular run.

The origin of the final reference time, i.e., the node which proposed it, is uniformly distributed among all nodes over all runs as shown in Fig. 6, underlining BUTLER's distributed nature. Moreover, the experiments show that the number of proposed reference times is related to the transmit probability P_{TX}. There are, on average, ≈ 4 different reference times.

The network splits into two partitions in a few runs due to multiple node faults. We find that the nodes correctly synchronize to the respective reference time in each partition. Composition into even more partitions would also work seamlessly without adjusting the protocol.



(a) Nodes start BUTLER at different times but eventually synchronize to the same reference time of node 18 at around 50 ms. Node 1 is faulty.

(b) Zoom into the marked area of the left plot, showing the individual slots. During this interval, the most synchronization events happen.

Figure 5: Time synchronization of nodes during the execution of BUTLER with colors indicating which reference time a node follows during execution. *Reference times in the legend (gray means unsynchronized) are sorted from earliest (τ_{18}) to latest (τ_5).*

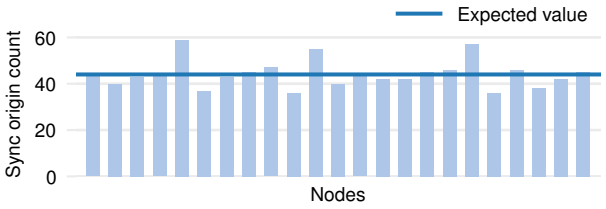


Figure 6: The number of times each node proposed the final reference time is uniformly distributed in BUTLER. *With 1016 runs, each node is expected to be the sync origin 44 times.*

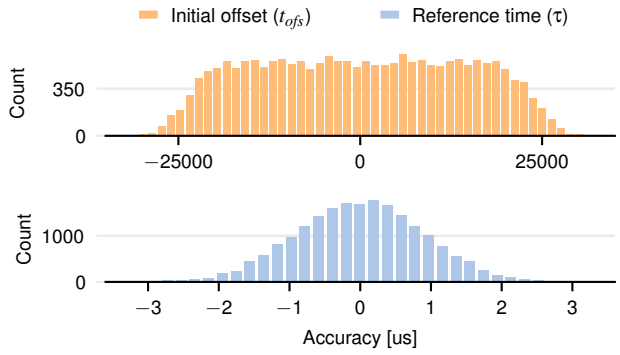


Figure 7: Synchronization accuracy before and after BUTLER. *Despite excessive initial offsets between the nodes (top),* BUTLER *synchronizes all nodes to within a few microseconds (bottom), which is far better than the required $T_{cap} = 160\,\mu s$.*

6.3 BUTLER's Performance and Efficiency

Accuracy. The main goal of BUTLER is to achieve synchronicity, such that the nodes in the network are time-aligned within the capture window T_{cap}. Our experiments show that BUTLER achieves this goal and synchronizes the nodes well below T_{cap}.

We use the GPIO tracing capabilities of FlockLab and mark the times when a node starts (t_{ofs}) and finishes (τ) BUTLER. The accuracy is measured as the difference between the observed values and the empirical mean in each BUTLER execution. Fig. 7 shows the results for around 900 executions. At the top, the distribution of the initial offsets t_{ofs} across all nodes is as expected, since each node picks a random initial offset between 0 and 50 ms. [1] In the lower plot we can see that despite the excessive initial offsets, most of the nodes achieve an accuracy of $\pm 2\,\mu s$, with at most $6\,\mu s$ between any two reference times. These values are well below the requirement of $T_{cap} = 160\,\mu s$ and validate the correctness of BUTLER, a necessary precondition for achieving high accuracy.

Efficiency. We evaluate the efficiency of BUTLER by measuring the time it takes to synchronize the network. To this end, we run experiments with varying maximum offsets (Δt_{ofs}), and nodes choose a random initial offset in the range of $0\,\text{ms} \leq t_{ofs} \leq \Delta t_{ofs}$. The time to synchronize the network starts with the first node entering BUTLER and ends when all nodes are synchronized to the same reference time, e.g., $\approx$51 ms in Fig. 5a.

Fig. 8 shows the distribution and mean of synchronization times for all nodes and all BUTLER executions ($\approx$ 1000) per experiment. We see that on FlockLab it takes on average about 8 ms to synchronize all nodes using BUTLER for $\Delta t_{ofs} =$

[1] The accuracy exceeds $-25\,000\,\mu s$ and $25\,000\,\mu s$ because it is based on the empirical mean of each BUTLER execution, which has some variation.

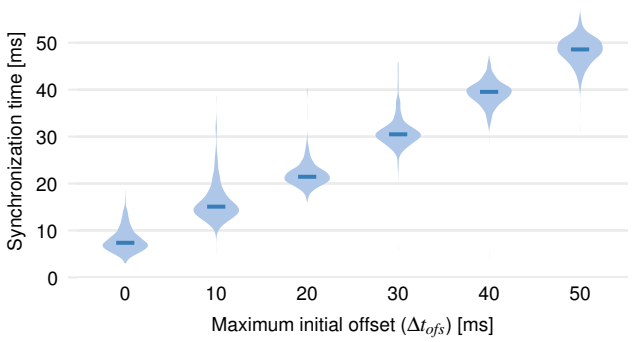


Figure 8: Time needed to synchronize all nodes for different initial offsets. BUTLER *reliably synchronizes all nodes within a few milliseconds and the synchronization time scales well with the clock drift.*

Table 2: Temporal overhead of BUTLER for different initial offsets and their corresponding communication periods.

Max. initial offset (Δt_{ofs})	Corresponding com. period	BUTLER duration	Temporal overhead
80 µs	1 s	33.58 ms	3.36 %
400 µs	5 s	33.9 ms	0.68 %
800 µs	10 s	34.3 ms	0.34 %
4.8 ms	1 min	38.3 ms	0.06 %
24 ms	5 min	57.5 ms	0.02 %
48 ms	10 min	81.5 ms	0.01 %

0 ms. With larger offsets, the synchronization time converges toward the respective Δt_{ofs} value. This is because many nodes are already synchronized to the final reference time before the last node starts BUTLER, as can be seen in Fig. 5a.

Duration and overhead. In general, the duration of BUTLER is the sum of two factors. One factor is the convergence time on the specific network topology, which can be experimentally explored at $\Delta t_{ofs} = 0$ ms and is ≈ 35 ms (100 slots) on FlockLab. The other factor is the maximum initial offset Δt_{ofs} to compensate for the accumulated clock drift since the last synchronization. Since the latter only depends on the communication period (assuming maximum clock drift $\Delta\nu$), the duration of BUTLER can be easily adjusted to any period during runtime; thus, BUTLER can be executed on demand. However, for very long communication periods, Δt_{ofs} can become large and exceed BUTLER's maximum duration T_B (Sec. 4.1). For instance, in our experiments $T_B = 2$ s, which corresponds to a communication period of ≈ 7 h. To support longer periods, BUTLER would have to be executed in-between to reset the accumulated clock drift.

In most scenarios, BUTLER only needs to be executed once per communication period, so we report its temporal overhead in relation to the period. Our results listed in Table 2 demonstrate that BUTLER is a lightweight mechanism with very little to negligible temporal overhead, enabling easy integration with communication protocols as BUTLER does not constrain their execution. Moreover, the temporal overhead decreases as the initial offset and associated communication period increase. For example, at a communication period of $\approx$3 s, the temporal overhead drops below 1 %, with only 0.01 % at a period of 10 min. However, the increased availability of BUTLER does come at the cost of increased energy consumption, as the execution time of BUTLER could otherwise be spent in sleep mode.

6.4 Reality Check: Making an Existing Protocol Available Through BUTLER

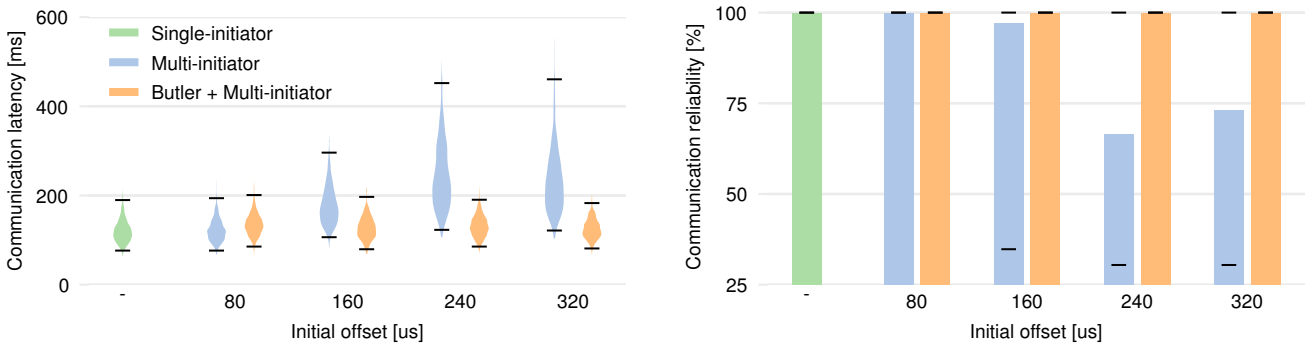
After investigating BUTLER's performance and efficiency in isolation, we now turn to the target use case where BUTLER is used to increase the availability of a (possibly existing) low-power wireless communication protocol.

Scenario and settings. As an example communication protocol, we use Mixer [13], which has recently been demonstrated to efficiently support distributed control scenarios [21]. However, Mixer provides no availability, often a key requirement in control applications, as it relies on a single initiator node that starts the many-to-many packet exchange. To overcome this problem and enable the use of multiple initiators in Mixer, we let BUTLER run before every communication round to synchronize the set of initiators. To be able to compare different settings and reproduce our results, we use a fixed set of two initiators located at opposite ends of the FlockLab testbed and also refrain from injecting artificial node failures.[2]

We compare the performance of Mixer with and without BUTLER for different initial offsets between the two initiators. Compared to the previous experiments, we use much smaller initial offsets here to show that multi-initiator Mixer requires help from BUTLER already at short communication periods. However, BUTLER can be used efficiently with Mixer irrespective of the communication period and resulting initial offset. For each setting, we conduct an experimental run that involves around 500 communication rounds. We also run the original Mixer with a single initiator as a baseline for the communication performance. In every communication round, each node initially has a 16 B message that it needs to share with all other nodes in the network during the Mixer round, so that eventually every node has all 23 messages. We consider two key metrics: *Latency*, which is the time it takes for a node to receive all messages in a round, and *reliability*, which is the fraction of received messages per round. Note that latency does not include BUTLER's execution time because BUTLER finishes before the scheduled communication round starts.

Results. Fig. 9a shows the latency distribution with single- and multi-initiator Mixer for different initial offsets; markers indicate the 1st and 99th percentiles. We can see that without BUTLER, the latency increases significantly by up to 2.8× for increasing initial offsets. When instead extending it with BUTLER, the latency of multi-initiator Mixer remains as low as for the original Mixer with a single initiator while providing higher availability. This is thanks to BUTLER's ability to accurately synchronize the initiators, which becomes an absolute necessity already for an initial offset of 160 µs to achieve

[2]We note that in order to maximize availability a larger set of initiator nodes randomly and dynamically chosen at runtime should be used, which is straightforwardly supported by our implementation.



(a) Communication latency. BUTLER *effectively addresses the clock drift between multiple initiators and enables the communication protocol to deliver the expected performance at significantly increased availability.*

(b) Communication reliability. *With* BUTLER*, the reliability remains unchanged above 99.9 %. Without* BUTLER*, multiple initiators lead to unreliable and unpredictable communication rounds.*

Figure 9: Communication performance of Mixer using multiple initiators with and without BUTLER.

high performance. Thus, the experiments also confirm our assumption about T_{cap} for correctness in Sec. 4.1.

Fig. 9b shows the reliability of single- and multi-initiator Mixer for different initial offsets when we limit the length of the multi-initiator Mixer rounds to the time needed by single-initiator Mixer (about 300 ms). This scenario is representative of typical constraints found in control applications, where interactions between distributed sensors and actuators must be completed within hard real-time deadlines to match the dynamics of physical processes [2]. Looking at Fig. 9b, we see without BUTLER the mean reliability decreases dramatically by up to 30 % as the initial offset increases. With BUTLER, the reliability remains unchanged and always above 99.9 %.

In summary, these results demonstrate that BUTLER effectively solves the problem of clock drift when using multiple initiators. BUTLER increases the availability of low-power wireless communication without sacrificing performance.

7 Related Work

BUTLER is the first work to address the availability problem of ST-based communication protocols. However, the underlying concept is closely related to the existing literature on time synchronization. Most time synchronization algorithms aim to provide an accurate, globally shared, and constantly available time base to all nodes in the network. While this is a powerful synchronization service that is essential for some applications, it needs to run periodically, and the associated overhead in terms of energy, time, and wireless bandwidth is very high. In fact, to distribute the initiator role among multiple nodes, which is what BUTLER aims for to increase availability, the nodes do not need a globally shared time base that is maintained for the entire lifetime of the system: all they need is to be able to perform a coordinated action [17]. This is also known as *synchronicity* [35] and can be achieved with less effort compared to full-fledged time synchronization.

Many time synchronization protocols, including TPSN [9], FTSP [23], Glossy [8], PulseSync [19], and TATS [20], use one dedicated node as a time reference for the entire network. Generally, these algorithms achieve excellent synchronization accuracy but are not fault-tolerant, which is a prerequisite for high availability. Furthermore, they often rely on topology information, causing instability in dynamic networks. The single node providing the reference time is also insufficient if the network splits into several isolated partitions. BUTLER overcomes these issues by adopting a fully decentralized approach that does not rely on topology information.

Over the years, several distributed synchronization protocols have been developed to mitigate centralization issues, such as RFA [35], DCTS [27], ATS [26], MTS [12], and MACTS [30]. These protocols do not rely on special nodes and are thus more robust and versatile than their centralized counterparts. The downside, however, is that these algorithms typically require a significant amount of time to synchronize the network, ranging from tens of seconds to multiple minutes. By contrast, BUTLER needs only tens of milliseconds to synchronize an entire network to within a few microseconds, thus substantially saving energy, time, and wireless bandwidth. The difference is that in most synchronization protocols, the nodes converge, e.g., by averaging the local clocks in an iterative process, which is needed to find a stable global time but requires a large number of messages to determine and account for the different clock drifts [30]. Furthermore, nodes that lose their state, e.g., due to a failure, potentially require all nodes to converge again. BUTLER does not adjust clock drift and uses the natural order of proposed reference times, requiring only a few messages to achieve the goal of synchronicity. BUTLER's short duration also simplifies the integration with communication protocols, which can be difficult with existing time synchronization protocols due to their significant overhead and the need for periodic executions.

8 Conclusion

With BUTLER, we have presented the design of a new dependable low-power wireless communication scheme that entirely eradicates any single point of failure. We first identify the common restriction that reliable time synchronization presents a prerequisite for ST-based protocols that could previously only be obtained using a dedicated node. Thereafter, we propose the design of a distributed synchronization mechanism that can extend existing protocols at negligible temporal overhead. We show through formal analysis that BUTLER permits tight time synchronization sufficient for ST despite node failures and network partitions. To verify our results, we

build a proof-of-concept implementation and extend an existing protocol with BUTLER so that its performance remains virtually unchanged while its last vulnerability is eliminated. We thereby demonstrate that a truly fault-tolerant low-power wireless protocol is feasible that can fulfill the high dependability requirements of present and emerging applications.

9 Acknowledgments

This work was supported in part by the German Research Foundation through SPP 1914 (grants ZI 1635/1-1), the Emmy Noether project NextIoT (grant ZI 1635/2-1), the Center for Advancing Electronics Dresden, and the Swiss National Science Foundation under NCCR Automation (grant agreement 51NF40_180545).

10 References

[1] IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pages 1–800, 2020.

[2] J. Åkerberg, M. Gidlund, and M. Björkman. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In *IEEE Int. Conf. on Industrial Informatics*, 2011.

[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*, 2017.

[4] D. Baumann, F. Mager, L. Thiele, M. Zimmerling, and S. Trimpe. Wireless control for smart manufacturing: Recent approaches and open challenges. *Proceedings of the IEEE*, 109(4):441–467, 2021.

[5] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 263–270, 1999.

[6] 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*, page 1–14, 2012.

[7] 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, 2013.

[8] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient Network Flooding and Time Synchronization with Glossy. In *ACM/IEEE Int. Conf. on Information Processing in Sensor Networks*, 2011.

[9] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, page 138–149, 2003.

[10] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *ACM Conference on Embedded Networked Sensor Systems*, 2009.

[11] S. Hayat, E. Yanmaz, and R. Muzaffar. Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Communications Surveys and Tutorials*, 18(4), 2016.

[12] J. He, P. Cheng, L. Shi, J. Chen, and Y. Sun. Time synchronization in WSNs: A maximum-value-based consensus approach. *IEEE Transactions on Automatic Control*, 59(3):660–675, 2014.

[13] C. Herrmann, F. Mager, and M. Zimmerling. Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks. In *ACM Conference on Embedded Networked Sensor Systems*, 2018.

[14] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza. Data prediction + synchronous transmissions = ultra-low power wireless sensor networks. In *ACM Conference on Embedded Network Sensor Systems*, 2016.

[15] P. Jiang. A new method for node fault detection in wireless sensor networks. *Sensors*, 9(2):1282–1294, 2009.

[16] A. A. Kumar S., K. Ovsthus, and L. M. Kristensen. An industrial perspective on wireless sensor networks – a survey of requirements, protocols, and challenges. *IEEE Communications Surveys Tutorials*, 16(3):1391–1412, 2014.

[17] B. Kusy, P. Dutta, P. Levis, M. Maróti, A. Lédeczi, and D. Culler. Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services. *International Journal of Ad Hoc and Ubiquitous Computing*, 1(4):239–251, 2006.

[18] 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*, 2013.

[19] C. Lenzen, P. Sommer, and R. Wattenhofer. Pulsesync: An efficient and scalable clock synchronization protocol. *IEEE/ACM Transactions on Networking*, 23(3):717–727, 2015.

[20] R. Lim, B. Maag, and L. Thiele. Time-of-flight aware time synchronization for wireless embedded systems. In *International Conference on Embedded Wireless Systems and Networks*, pages 149–158, 2016.

[21] F. Mager, D. Baumann, C. Herrmann, S. Trimpe, and M. Zimmerling. Scaling beyond bandwidth limitations: Wireless control with stability guarantees under overload. *ACM Trans. Cyber-Phys. Syst.*, Nov 2021.

[22] F. Mager, D. Baumann, R. Jacob, L. Thiele, S. Trimpe, and M. Zimmerling. Feedback control goes wireless: Guaranteed stability over low-power multi-hop networks. In *ACM/IEEE International Conference on Cyber-Physical Systems*, 2019.

[23] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conf. on Embedded Networked Sensor Systems*, page 39–49, 2004.

[24] V. Poirot, B. Al Nahas, and O. Landsiedel. Paxos made wireless: Consensus in the air. In *Proceedings of the 2019 Int. Conf. on Embedded Wireless Systems and Networks*, page 1–12. Junction Publishing, 2019.

[25] M. Raza, N. Aslam, H. Le-Minh, S. Hussain, Y. Cao, and N. M. Khan. A critical analysis of research potential, challenges, and future directives in industrial wireless sensor networks. *IEEE Communications Surveys Tutorials*, 20(1):39–95, 2018.

[26] L. Schenato and F. Fiorentin. Average TimeSynch: A consensus-based protocol for clock synchronization in wireless sensor networks. *Automatica*, 47(9):1878–1886, 2011.

[27] L. Schenato and G. Gamba. A distributed consensus protocol for clock synchronization in wireless sensor network. In *2007 46th IEEE Conference on Decision and Control*, pages 2289–2294, 2007.

[28] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, dec 1990.

[29] M. Schuß, C. A. Boano, M. Weber, and K. Römer. A competition to push the dependability of low-power wireless protocols to the edge. In *Int. Conf. on Embedded Wireless Systems and Networks*, Feb. 2017.

[30] F. Shi, X. Tuo, L. Ran, Z. Ren, and S. X. Yang. Fast convergence time synchronization in wireless sensor networks based on average consensus. *IEEE Trans. on Industrial Informatics*, 16(2):1120–1129, 2020.

[31] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou. Real-time communication and coordination in embedded sensor networks. *Proceedings of the IEEE*, 91(7):1002–1022, 2003.

[32] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *ACM Proceedings of the 3rd Int. Conf. on Embedded Networked Sensor Systems*, page 51–63, 2005.

[33] M. Trobinger, G. de Albuquerque Gleizer, T. Istomin, M. Mazo, A. L. Murphy, and G. P. Picco. The wireless control bus: Enabling efficient multi-hop event-triggered control with concurrent transmissions. *ACM Transactions on Cyber-Physical Systems*, 6(1), Nov 2021.

[34] R. Trüb, R. Da Forno, L. Sigrist, L. Mühlebach, A. Biri, J. Beutel, and L. Thiele. FlockLab 2: Multi-Modal Testing and Validation for Wireless IoT. In *3rd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things*. OpenReview.net, 2020.

[35] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *ACM Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, page 142–153, 2005.

[36] P. Zand, S. Chatterjea, K. Das, and P. Havinga. Wireless industrial monitoring and control networks: The journey so far and the road ahead. *Journal of Sensor and Actuator Networks*, 1(2):123–152, 2012.

[37] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele. Adaptive real-time communication for wireless cyber-physical systems. *ACM Transactions on Cyber-Physical Systems*, 1(2), Feb 2017.

[38] M. Zimmerling, L. Mottola, and S. Santini. Synchronous transmissions in low-power wireless: A survey of communication protocols and network services. *ACM Computing Surveys*, 53(6), Dec. 2020.