Hydra: Concurrent Coordination for Fault-tolerant Networking

Hydra: Concurrent Coordination for Fault-tolerant Networking

Andreas Biri
ETH Zurich
Zurich, Switzerland
abiri@ethz.ch

Reto Da Forno
ETH Zurich
Zurich, Switzerland
rdaforno@ethz.ch

Tobias Kuonen
ETH Zurich
Zurich, Switzerland
tkuonen@ethz.ch

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

Marco Zimmerling
University of Freiburg
Freiburg im Breisgau, Germany
zimmerling@cs.uni-freiburg.de

Lothar Thiele
ETH Zurich
Zurich, Switzerland
thiele@ethz.ch

ABSTRACT

Low-power wireless networks have the potential to enable applications that are of great importance to industry and society. However, existing network protocols do not meet the dependability requirements of many scenarios as the failure of a single node or link can completely disrupt communication and take significant time and energy to recover. This paper presents Hydra, a low-power wireless protocol that guarantees robust communication despite arbitrary node and link failures. Unlike most existing deterministic protocols, Hydra steers clear of centralized coordination to avoid a single point of failure. Instead, all nodes are equivalent in terms of protocol logic and configuration, performing coordination tasks such as synchronization and scheduling concurrently. This concept of *concurrent coordination* relies on a novel distributed consensus algorithm that yields provably unique decisions with low delay and energy overhead. In addition to a theoretical analysis, we evaluate Hydra in a multi-hop network of 23 nodes. Our experiments demonstrate that Hydra withstands random node failures without increasing coordination overhead and that it re-establishes efficient and reliable data exchange within seconds after a major disruption.

CCS CONCEPTS

• **Computer systems organization → Fault-tolerant network topologies**; *Availability*; *Sensor networks*; • **Networks** → *Network protocol design.*

KEYWORDS

Fault tolerance, consensus, network coordination, concurrent transmissions, WSN

ACM Reference Format:
Andreas Biri, Reto Da Forno, Tobias Kuonen, Fabian Mager, Marco Zimmerling, and Lothar Thiele. 2023. Hydra: Concurrent Coordination for Fault-tolerant Networking. In *The 22nd International Conference on Information Processing in Sensor Networks (IPSN '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3583120.3587047

IPSN '23, May 09–12, 2023, San Antonio, TX, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0118-4/23/05.
https://doi.org/10.1145/3583120.3587047

1 INTRODUCTION

The last few years have seen substantial innovations that have enabled wireless sensor networks (WSNs) with unprecedented dependability. For instance, novel protocols enable distributed battery-powered devices to reach an agreement [3], deliver messages in the desired order [17], and exchange data within hard real-time deadlines to meet the requirements of cyber-physical and Industrial Internet of Things applications [35]. To achieve these capabilities reliably and with broad applicability, the concept of concurrent transmissions [18, 54] has been instrumental in becoming highly resilient to external interference and network topology changes.

Problem. While such wireless systems are promising for enabling novel applications in pivotal scenarios, they cannot tolerate the failure of critical devices or key communication links. However, such failures are common as WSNs are frequently deployed in hostile or inaccessible scenarios where extreme weather [4], high ambient humidity [2, 47], the presence of living beings [8, 37], disasters [12], and non-line-of-sight conditions [13, 24] render the availability of devices and wireless links fragile and highly variable.

Current systems typically adopt a centralized design and suffer from a single point of failure: If the *network coordinator* that manages synchronization and scheduling fails or gets disconnected, the entire rest of the network also breaks down as reliable communication is no longer possible. This vulnerability stands in stark contrast to the required dependability of wireless communication in many applications where even minor system outages may involve significant financial costs and lead to long-term repercussions [47, 52]. Typical scenarios include autonomous drone swarms [27], early-warning systems in harsh conditions [6, 46], and networks operating under strong interference [9] or high dependability requirements [35]. Designating co-located devices as a failover for the primary network coordinator cannot solve the fundamental problem, as this approach is known to provide limited fault tolerance if failures are correlated, the common case in practice [48]. Modern WSNs should have the liberty to operate independently without requiring pre-assigned backups or expensive, redundant hardening.

Contribution. To address this problem, we introduce Hydra, the first fully distributed protocol that avoids centralized coordination to enable fault-tolerant low-power wireless networking. The fundamental paradigm underlying Hydra is that every node in the network is equivalent in terms of protocol logic and configuration, thus avoiding any entity that maintains a unique state or that serves a special role. The protocol adapts to the application's traffic demands while tolerating arbitrary message losses, temporary or permanent node failures, and sudden topology changes. Hydra uses distributed processing in tandem with communication primitives based on concurrent transmissions (*i*) to continuously track the set of nodes that is currently part of the network, (*ii*) to ensure that all nodes that concurrently compute and distribute a new schedule do so based on the same information, and (*iii*) to accurately time-synchronize the network to achieve high reliability and efficiency.

Concurrent coordination empowers Hydra to take full advantage of physical redundancy to yield outstanding fault tolerance and dependability for WSNs. Real-world experiments validate that Hydra's energy overhead almost matches that of LWB [16], a comparable centralized design. Hydra demonstrates robust, efficient data exchange while swiftly adapting to changing traffic demands despite arbitrary node failures and severe communication disruptions. In summary, this paper makes the following major contributions:

- We present Hydra, a communication protocol for low-power WSNs that excels in its fault tolerance through a novel distributed consensus and synchronization mechanism.
- We provide an algorithmic specification of Hydra and prove its properties: (*i*) freedom of harmful packet collisions, thus ensuring safe operation, (*ii*) adaptivity to changing traffic demands and dynamic network topologies, and (*iii*) persistent data exchange even under node and link failures.
- We implement Hydra on a microcontroller driving a Semtech SX1262 RF transceiver and provide the code as open source together with tools for reproducible fault injection [7].
- We evaluate Hydra on the FlockLab testbed [45]. Our experiments not only confirm its safety, adaptivity, and liveness but also demonstrate its fault tolerance at a negligible runtime overhead even if almost half of the nodes in the network fail.

After defining the problem space in Section 2, we provide an overview of Hydra in Section 3, a detailed protocol description in Section 4, and a formal analysis in Section 5. We implement Hydra to demonstrate its efficacy in realistic testbed experiments in Section 6, where we show that Hydra achieves fault tolerance despite a high degree of induced link errors and prolonged, adversarial node failures. Lastly, Section 7 discusses design trade-offs and limitations of Hydra.

2 PROBLEM AND CHALLENGES

We first introduce our failure model and derive the objectives and corresponding protocol requirements to achieve fault-tolerant networking before analyzing the challenges in Hydra's design space.

2.1 Objectives and Protocol Requirements

Failure model. To ensure general applicability, we assume that communication can fail in any phase of the protocol. This includes the possibility that a node can neither receive from nor send to any other node over prolonged periods. Packet loss may occur at any time and we do not make any assumption on the loss distribution or correlation between losses. We consider non-Byzantine failures, i.e., nodes adhere to their specifications and a packet either arrives correctly or is not received at all. The latter follows from corrupted packets being filtered using checksums. We further assume that a node may temporarily or permanently crash at any time.

Objectives. Motivated by cyber-physical and Industrial Internet of Things applications, we aim to design a protocol that provides reliable low-power wireless networking under the above-mentioned failure model. In particular, unlike state-of-the-art solutions, the protocol aims to avoid any single point of failure by design.

While tolerating faults, the protocol must also achieve the following objectives: (*i*) Because of the constrained nature of WSNs, efficient use of limited resources such as energy and bandwidth is vital. (*ii*) These resources should be flexibly allocated so that the network can react to changing environments and traffic demands. (*iii*) The protocol must be able to deliver packets to any node in the network despite dynamically changing, multi-hop network topologies. Data exchange needs to be highly reliable and predictable, and therefore packet collisions must be avoided.

Protocol requirements. Based on these objectives, we identify three requirements for a fault-tolerant low-power wireless protocol that must be satisfied under the failure model described above:

- *Safety:* Guaranteeing correct operation so that any two nodes never use the same time slot to exchange data, thereby avoiding packet collisions that impair reliability.
- *Adaptivity:* Enabling each non-faulty node to react to changes in traffic demands and environmental conditions, including the addition of nodes and the release of obsolete resources.
- *Liveness:* Ensuring that consistent data exchange is maintained between all non-faulty nodes that are physically able to communicate with each other (i.e., remaining network links suffice to transfer information).

2.2 Challenges and Trade-offs

All three requirements must be simultaneously satisfied, yet they are mutually conflicting. There are four fundamental challenges in meeting the objectives and protocol requirements outlined above.

First, one central network coordinator is a single point of failure. Unavailability of this node leads to unstable control loops endangering safety [35], inefficient resource usage due to a loss of adaptivity [48], or disrupted data flows violating liveness [16]. Using backup nodes that take over when the primary network coordinator fails is difficult as incompatible schedules during handover violate safety and liveness. Moreover, the fault tolerance of this approach is limited as correlated failures likely affect both the primary network coordinator and its backups [48], further complicating the handover procedure and endangering safety.

Second, instead of using a single leader, it is possible to increase fault tolerance by letting multiple nodes perform network coordination concurrently. However, the complexity of this approach increases as each of the participating nodes must base its decisions on the same inputs at the same time, thereby requiring consensus as soon as more than a single coordinator is involved. When information is missing, liveness and adaptivity are at risk.

Third, consensus on these protocol inputs is critical for safety but is particularly difficult to find when nodes become unreachable due to temporary or permanent faults. In this case, the set of nodes that have to agree must be flexible to maintain adaptivity.

Fourth, we either face reduced liveness if only nodes that obtained the latest inputs can communicate, or we violate safety if

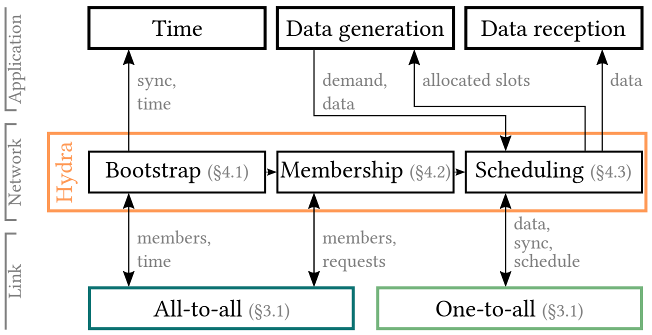

Figure 1: Hydra works on the network layer and interfaces with link-layer primitives to make fault tolerance available to the application. The three individual components of Hydra are explained in further detail in the respective subsections denoted in grey.

nodes with outdated decisions cause packet collisions. As an alternative to coordinated communication, both adaptivity and liveness are preserved with un-synchronized data exchange schemes, but packet collisions may violate safety.

In the design of Hydra, we have carefully explored this trade-off space, further discussed in Section 7.1. As we will show, Hydra involves all nodes in concurrent coordination of the network to maximize robustness and leverages the contention-free data floods to maintain redundant synchronization without requiring a fixed time reference. Furthermore, we choose to unconditionally provide safety for reliable, predictable performance and will hence be forced to occasionally delay adaptivity or restrict liveness.

3 PROTOCOL DESIGN

We introduce Hydra, a fault-tolerant low-power wireless network protocol. In Hydra, each node in the network is equivalent in terms of protocol logic and runtime configuration. As a result, network coordination tasks such as time synchronization and scheduling are distributed across all nodes and run concurrently. A formal analysis in Section 5 shows that Hydra provably satisfies safety, adaptivity, and liveness under arbitrary node and link failures. We first describe the protocol interfaces and provide an overview of Hydra and the mechanisms to achieve concurrent coordination.

3.1 Overview

Existing designs, presented in Section 8, have a single point of failure that limits their fault tolerance. While the crash of a centralized coordinator for scheduling and synchronization directly impairs the networking capabilities [16, 19], even partially decentralized solutions [15, 25, 26] rely on topology information like the routing tree of RPL-based protocols [49] for coordination that roots in a single device. However, modern communication primitives such as concurrent transmissions [54] permit reliable and topology-agnostic communication on the link layer if nodes are sufficiently synchronized. Based on this insight, Hydra adds a network layer that sits between the application and the link layer (see Figure 1) and guarantees fundamental properties through concurrent coordination. As detailed in Section 4, it includes novel concepts for bootstrapping the network, finding distributed consensus on network membership, and scheduling persistent communication.

Application layer. Hydra provides accurate network-wide time synchronization as well as a bi-directional communication service. The application submits its traffic demands to Hydra, which then takes care of facilitating corresponding communication resources.

Link layer. Hydra can be based on many communication primitives, as we will show in Section 5 that its safety is formally proven without relying on the primitives' reliability. However, concurrent transmissions have become the basis for countless robust and efficient multi-hop wireless protocols [18, 21, 54]. Network flooding permits predictable communication at high throughput and low latency, with its broadcast nature making it ideally suited to exploit physical redundancy in a system. Due to topology-agnostic message passing based on concurrent forwarding across hops, these primitives seamlessly handle network changes. In Hydra, we employ a combination of two classes of such communication primitives.

All-to-all primitives such as Chaos [29] and Mixer [21] exchange per-node information for network coordination. Nodes independently send and receive in consecutive contention slots and continually adapt packets so that information efficiently propagates through the network. However, they require tight synchronization as a prerequisite to receive packets based on the capture effect [31].

One-to-all floods like Glossy [18] reliably disseminate data packets. The initiator of the flood requires an exclusive time slot through scheduling for contention-free communication and serves as a unique time reference for (re-)synchronization.

By combining the ability of these all-to-all primitives to exchange scheduling information based solely on a shared time basis with the capability of fast network flooding to tightly synchronize a multi-hop network, Hydra creates a symbiosis on the link layer that empowers nodes to fully decentralize network coordination.

3.2 Hydra in a Nutshell

Hydra includes several dedicated mechanisms to fulfill its three protocol requirements. Next, we introduce the protocol structure and outline the algorithm presented in detail in Section 4.

Structure. Communication in Hydra occurs in rounds. As shown in Figure 2, each round consists of three phases: To ensure safety, a contention-free *data dissemination* (DD) phase permits nodes to reliably exchange data. To determine an adaptive schedule that allocates such exclusive time slots, nodes exchange their demand during the *schedule negotiation* (SN) phase. Lastly, the newly computed schedule is shared in the *schedule distribution* (SD) phase so nodes can independently communicate without permanently requiring each other's input and preserve liveness.

Hydra executes such rounds of period T for a network $\mathcal{N}$ of N nodes i with $\mathcal{N} = \{i \in \mathbb{N} \mid 1 \leq i \leq N\}$. An *epoch* consists of F rounds and defines the rate at which the schedule may change.

Data dissemination. During the DD phase, each node that knows a schedule initiates a one-to-all flood if it is the owner of the current slot and otherwise assists in propagating a packet by concurrently re-transmitting it after reception. Application data is exclusively

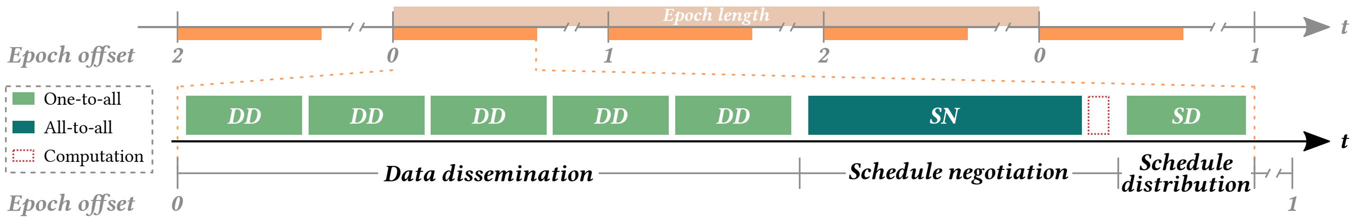

Figure 2: Each round in Hydra consists of three phases. During *data dissemination* (DD), nodes exchange application data according to a local schedule and synchronize with each other. During *schedule negotiation* (SN), requests are aggregated to compute the next schedule, which will be flooded during *schedule distribution* (SD) to ensure reliable progress. Throughout an epoch consisting of multiple rounds (e.g. three in this illustration), information regarding network membership and demand is accumulated to reach consensus and enable distributed scheduling.

disseminated in this phase, with each initiator serving as a time reference to re-synchronize without a centralized entity.

Schedule negotiation. Nodes may leave at any time due to volatile links or node failures and may (re-)join. Hydra permits a flexible set of nodes to form a network and determines a schedule that reflects the dynamically changing bandwidth demands of nodes.

To ensure that concurrent coordination finds consensus on the set of participating nodes and their bandwidth demands, network membership is determined dynamically during the SN phase. To this end, nodes maintain a local notion of whom they consider part of the current network as membership flags and exchange these flags in the SN phase. By awaiting and merging information from all expected nodes, a unique decision among all nodes of the current network can be guaranteed, as formally proven in Section 5.

To compute the next schedule, each node determines its request (e.g. the number of slots it desires) according to its current demand and known schedule. During the SN phase, nodes aggregate schedule requests and memberships flags from other nodes using all-to-all messages, as explained in detail in Section 4.2, and update the minimum and maximum schedule versions they have encountered. At the end, they have the ability to determine the next schedule if (*i*) they have obtained the complete set of information, meaning that they have received requests from all nodes that are part of the current network, i.e., whose membership flags are set after merging, and (*ii*) all these nodes have the same schedule version. The next schedule can then be computed using a deterministic algorithm based on the current schedule and this complete set of information.

Schedule distribution. During the SD phase, nodes that have computed the next schedule use a single, concurrent one-to-all flood at the end of an epoch to distribute it to others that have been unable to do so themselves or newly joined. In case some nodes have missed a previous update and still use an older schedule, the most recent schedule will be re-transmitted by updated nodes. A node knowing the next schedule, having either computed it or received it from others, will start using it after the end of the epoch.

3.3 Concepts to Ensure Fault Tolerance

To ensure that Hydra does not suffer from a single point of failure and simultaneously provides safety, adaptivity, and liveness, we develop three specialized components represented in Figure 1.

Ensuring safety: Distributed synchronization. To make sure that data exchange does not interfere, nodes must be reliably synchronized. For this purpose, the initiators of slots in the DD phase provide a unique time reference. Initial synchronization is achieved through a bootstrapping algorithm, introduced in Section 4.1, on a separate channel to prevent interference with an existing network.

Safety of the DD phase further requires that all used schedules are compatible, i.e., they do not assign slots to two different nodes. This is achieved by guaranteeing that (*i*) at most two successive schedules are present, (*ii*) successive schedules are compatible, and (*iii*) only a single, connected network operates at any point in time to avoid inter-network interference, as we will show in Section 5.

Ensuring adaptivity: Flexible network membership. If the schedule computation depends on input from all nodes, node failures block progress [3]. To permit nodes to join and leave the network anytime while safely adapting scheduling, Hydra requires consensus on a flexible set of nodes that form the members of the current network. By exchanging the local notions of reachable nodes in the SN phase, elaborated in Section 4.2, we provably guarantee that a schedule is only computed if the input from all nodes in the current network is obtained and that the result is unique.

In the case of link failures resulting in a missed schedule, future schedule computations fail due to a schedule version mismatch and progress would be permanently prohibited. To preserve adaptivity, nodes knowing the most recent schedule must detect this case and re-transmit it in the SD phase, giving other members the chance to catch up to the rest of the network so progress may resume.

Ensuring liveness: Local schedules. Synchronously changing schedules incurs the risk that nodes must halt communication when coordination is disturbed and consensus is unclear. To preserve liveness during such transitions between schedules, the conditions of Hydra's scheduling algorithm introduced in Section 4.3 ensure that two successive schedules cannot interfere. Therefore, nodes can asynchronously update without restricting anyone's ability to keep exchanging application data. To continuously maintain this compatibility between used schedules, the schedule versions ensure that a new schedule is only computed if all nodes have obtained the previous version. Therefore, a known schedule guarantees safe data exchange even if coordination is temporarily prohibited.

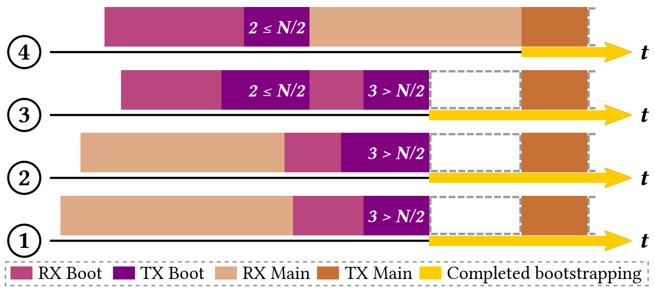

Figure 3: Bootstrapping nodes start independently and synchronize if a majority of the $N = 4$ nodes in this example exchange their information on the boot channel. As seen for node $i = 4$, a node can also join an existing network when listening on the main channel.

4 HYDRA IN DETAIL

Next, we take a closer look at Algorithm 1 and first show how a Hydra network is initially formed. Then, the concept of distributed consensus based on flexible network membership is presented. Lastly, we discuss how careful scheduling provides liveness and safety.

Notation. $\perp$ denotes a variable containing no information. The operator $\circ$ combines sets whose elements may have the value $\perp$ as follows: $\perp \circ \perp = \perp$ and $a \circ \perp = \perp \circ a = a \circ a = a$ for any value $a \neq \perp$. Combining $a \neq b$ with $a, b \neq \perp$ is undefined. The operator $\vee$ is the element-wise OR with $0 \vee 0 = 0$ and $1 \vee 0 = 0 \vee 1 = 1 \vee 1 = 1$.

4.1 Bootstrapping

After a reset or lost connectivity with a majority of nodes, a node must discover other nodes to synchronize or assemble a new network. Hydra's bootstrapping algorithm, detailed in our companion document [7], requires no pre-existing synchronization and terminates with the node being synchronized to a majority of nodes. It further sets the initial conditions for Algorithm 1 as introduced below, i.e., the current membership flags $M[j]$, the latest schedule version v, schedule S^v and epoch offset f, and values $\forall j \in [1, N] : C_e[j] \leftarrow 0$, $\forall j \in [1, N] : I_e[j] \leftarrow 0$, *updated* $\leftarrow 0$, and *retransmit* $\leftarrow 0$.

To not endanger the safety of a pre-existing network, bootstrapping occurs on a separate boot channel. To synchronize, each node randomly chooses whether it should listen on the main or the boot channel for a non-deterministic duration, depicted in Figure 3. On the main channel, it reboots if no existing network is discovered. On the boot channel, it floods a sync packet to which other listening nodes align. Nodes then aggregate information and sum the number of encounters. If a majority of nodes synchronizes, the newly founded network simultaneously switches to Algorithm 1. Note that bootstrapping is opportunistic and does not fulfill the requirements in Section 2.1. However, as it can be arbitrarily restarted without affecting the performance of Hydra's normal operation, it guarantees synchronization after termination and enables liveness.

4.2 Consensus on Network Membership

Hydra guarantees safety under any failure by design. To achieve such robustness despite nodes joining, leaving, or even failing, reliably deciding on the set of nodes whose *requests* r_j^v are considered for a new schedule is paramount. While quick convergence is desirable for adaptivity, the mechanism's key property is that its result is provably unique if consensus between a set of nodes is found.

Consensus. To make unique decisions for a flexible set of nodes, we introduce the *membership flags* M, which reflect a node's local notion of whom it considers part of the network. The flags remain constant throughout an epoch, i.e., $M[j] = 1$ if the node expects node j to be reachable as part of the current network and $M[j] = 0$ otherwise. Note that nodes may differ in their view of the current network and may have unequal membership flags.

The key to reaching consensus can be found in how the membership flags are used during the SN phase, which is illustrated in Figure 4. Instead of only gathering *schedule requests* R' from the nodes whose flags are set in M, each node also merges the information on network membership that it receives from others into the temporary membership flags M'. Information is only merged (line 21 in Algorithm 1) if both transmitter and receiver of a packet consider each other members of their network (line 16), as discussed below in more detail. This merging mechanism prevents the computation of a new schedule until a node has obtained all the required information (membership flags M and requests r_j^v) from all nodes it considers part of its network as well as their merged notion of the network (line 24). We call such a set where the corresponding request is known for each temporary membership flag, i.e., $\forall j \in [1, N] : M'[i] = 1 \Rightarrow R'[j] \neq \perp$, a *complete set of information.*

Merging criteria. A node only accepts information from another node if both consider one another part of the same network. Checking whether nodes expect each other is done by verifying that a node i is part of the temporary membership flags of j and vice versa (line 16). This acceptance check is crucial to prove the uniqueness of the complete set of information, presented in Section 5, and especially relevant for asymmetric links, as discussed in Section 6.5.

Updates. To update the membership flags and maintain adaptivity, we introduce the *connectivity counters* C_r and C_e which persist for a round and an epoch, respectively. Whenever a request r_j^v from node j is received, $C_r[j]$ is set (line 14). At the end of the SN phase, $C_e[j]$ is incremented if a request from node j has been received (line 31). At the end of an epoch, node j is added to the expected set of nodes in M if its request has been received (line 52), i.e., if $C_e[j] \geq C_{min}$ with $C_{min} = 1$. Otherwise, it is removed from the local membership flags and not expected anymore for consensus.

4.3 Schedule Computation

To ensure liveness, each node stores its current *schedule* S^v, where $S^v[k] \in \{\perp\} \cup [1, N]$ denotes the allocation of a slot $k \in [1, K]$ in the DD phase to a node. However, always transmitting according to a known schedule might endanger safety unless dedicated mechanisms prevent the collision of packets from Hydra nodes.

Versioning. The *schedule version* v uniquely identifies a schedule and denotes a node's current level of information. A request r_i^v for the next schedule must be unique for every schedule version v, i.e., it can only be adjusted when the schedule is updated. We use schedule versioning to control the pace of adaptivity and ensure that the entire network has been able to obtain the same scheduling

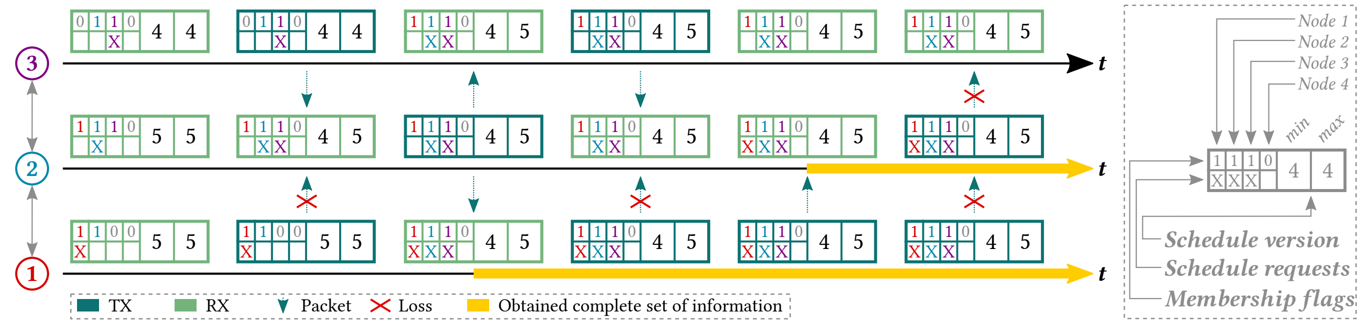

Figure 4: During the SN phase, membership flags and schedule requests are exchanged to obtain the complete set of information. In this example, node 4 failed and is not considered part of the network by nodes 1 - 3, which can reach consensus without requiring its input. At the end, nodes 1 and 2 have obtained the complete set of information, whereas node 3 is missing the request from node 1. While node 3 does not initially expect node 1, it correctly determines a lack of information after merging node 2's packet in the third slot. As node 3's schedule version 4 is outdated, nodes 1 and 2 cannot compute the next schedule and will re-transmit their schedule 5 in the following SD phase.

information before applying further changes to it based on new requests. Nodes independently acquire requests until they have a complete set of information (line 24), but can only compute a new schedule if all requests are based on the same schedule version (line 26). All nodes then apply a deterministic scheduling algorithm with the current schedule S^v and the complete set of information, i.e., the requests $R'[j] = r_j^v$ from all nodes with $M'[j] = 1$, as input (line 27). If a node detects that an old schedule version is still in use based on the minimum schedule version in the network, the most recent schedule will be re-transmitted (line 29) until all nodes have been able to catch up. Thereafter, adaptation may resume.

Compatibility. To ensure that successive schedules S^v and S^{v+1} are safe, i.e., they do not assign the same DD slot to different nodes, the following conditions must be satisfied. Suppose that we are given a schedule S^v based on previous requests and that new schedule requests R' lead to a different number of assigned slots. To update the schedule, slots of S^v that are not required anymore to reach the newly determined number of slots are released. If a node receives more slots in the next schedule, free slots in S^v are assigned to it. This means that a slot assigned in S^v can only be used by the same node in S^{v+1} or be released, and a free slot in S^v can be assigned to any node in S^{v+1} or remain free. Crucially, safety is guaranteed even if both schedules are used in parallel. Furthermore, this scheme can be used for any scheduling algorithm as transitions to an arbitrary slot assignment are possible in two consecutive steps [7].

Epochs. Nodes start using a new schedule at the beginning of an epoch at *epoch offset* $f = 0$. During an epoch of *epoch length* F, Hydra gathers information on a node's connectivity to update its local notion of the network for the next epoch. If node j's request r_j^v was received as collected in $C_e[j]$, it is included in the network by setting the membership flag $M[j]$ (line 52). Increasing F stabilizes network membership through a longer observation period, but delays the adaptation of the network membership and the schedule.

Expiration. Nodes that are not part of the network anymore must be prevented from using outdated schedules, as they would break the compatibility between schedules that is only guaranteed for two successive ones. Therefore, such schedules must expire.

Similar to the connectivity counters, we introduce the *interaction counter* I_e. $I_e[j] \leftarrow 1$ is set if either information from a node j in the network was received (line 18), or if a schedule was obtained (line 41). At the end of an epoch at $f = F$ (line 44), a node only keeps its schedule if it is mutually connected to a majority of nodes or if it has received the schedule in this epoch (line 47). This mechanism guarantees that a possibly outdated schedule expires (line 48) unless it either has been received in the current epoch, i.e. it is up-to-date, or if the node has ensured that it is still part of the network. Notice that a schedule can be kept even if no node has obtained the complete set of information. To verify that a node remains part of the current network, information is aggregated throughout the epoch. This duration can be tuned using F to ensure liveness despite a high failure rate. If this verification succeeds, the node's schedule cannot be outdated and can still be used without violating safety.

5 FORMAL ANALYSIS OF HYDRA

Consensus protocols are known to be notoriously complicated to understand and prove [28, 39]. However, due to the variety of states in distributed systems, it is essential that their properties are not only experimentally shown but are based on theoretical guarantees covering all conditions. Due to space constraints, we only sketch the proofs in this paper and provide complete formal proofs of safety, adaptivity, and liveness in our companion document [7]. In particular, adaptivity is omitted as the possibility to submit schedule requests (line 7) and to join and leave the current network (line 52) are inherent to and can be directly concluded from Hydra's design.

5.1 Safety

For Hydra's distributed consensus, we rely on the fact that for arbitrary initial membership flags M_k of a node k, i.e., independent of the set of nodes assumed to be reachable by k, the schedule requests R'_k used to compute the next schedule (line 27) are identical for nodes with a complete set of information. In short, nodes with a complete set of information compute the same schedule.

Algorithm 1: Behavior of node i after bootstrapping

```
   DD (data dissemination) phase
2      A node $i$ with $v > 0$ participates with its current schedule $S^v$ ;
   SN (schedule negotiation) phase
       Initialization
5          $v^{min} \leftarrow v$ ; $v^{max} \leftarrow v$ ;
           forall $j \in [1, N]$ do
7              $M'[j] \leftarrow M[j]$ ; $R'[j] \leftarrow \begin{cases} r_i^v & \text{if } j = i \\ \perp & \text{otherwise} \end{cases}$ ;
8              $C_r[j] \leftarrow 0$ ;
       Sending
10         send packet $p = (i, v^{min}, v^{max}, M', R')$ ;
       Receiving
           if (packet $q = (j_q, v_q^{min}, v_q^{max}, M_q, R_q)$ received) then
               forall $j \in [1, N] : R_q[j] \neq \perp$ do
14                 $C_r[j] \leftarrow 1$ ;
               ▷ Check if requests and flags should be merged ◁
16             if $((M'[j_q] = 1) \wedge (M_q[i] = 1))$ then
                   forall $j \in [1, N] : R_q[j] \neq \perp$ do
18                     $I_e[j] \leftarrow 1$ ;
19                 $v^{min} \leftarrow \min\{v^{min}, v_q^{min}\}$ ;
20                 $v^{max} \leftarrow \max\{v^{max}, v_q^{max}\}$ ;
21                 $R' \leftarrow R' \circ R_q$ ; $M' \leftarrow M' \vee M_q$ ;
       Final
           ▷ Check if node is complete and part of majority ◁
24         if $((\forall j \in [1, N] : M'[j] = 1 \Rightarrow R'[j] \neq \perp) \wedge$
               $(|\{j \in [1, N] : M'[j] = 1\}| > N/2)$ ) then
26             if $((v^{min} = v^{max}) \wedge (v > 0))$ then
27                 determine new schedule $S$ ; $updated \leftarrow 1$ ;
               else if $(v = v^{max})$ then
29                 $retransmit \leftarrow 1$ ;
           forall $j \in [1, N]$ do
31             $C_e[j] \leftarrow C_e[j] + C_r[j]$ ;
   SD (schedule distribution) phase
       ▷ Check if schedule can be distributed at end of epoch ◁
34     if $((f = F - 1) \wedge (updated = 1))$ then
35         $v \leftarrow v + 1$ ; $S^v \leftarrow S$ ; send packet $(v, S^v)$ ;
       else if $(retransmit = 1)$ then
37         send packet $(v, S^v)$ ; $retransmit \leftarrow 0$ ;
       else if (packet $q = (v_q, S_q)$ received) then
39         $v \leftarrow v_q$ ; $S^v \leftarrow S_q$ ;
           forall $j \in [1, N]$ do
41             $I_e[j] \leftarrow 1$ ;
   End of round
43     $f \leftarrow f + 1$ ;
44     if $(f = F)$ then
45         $updated \leftarrow 0$ ; $f \leftarrow 0$ ;
           ▷ Check if schedule must expire as not part of majority ◁
47         if $(\sum_{j \in [1,N]} I_e[j] \leq N/2)$ then
48             $v \leftarrow 0$ ;
           forall $j \in [1, N]$ do
               ▷ Check if information from a node was received ◁
51             if $((C_e[j] \geq C_{min}) \vee (j = i))$ then
52                 $M[j] \leftarrow 1$ ;
               else
54                 $M[j] \leftarrow 0$ ;
55             $C_e[j] \leftarrow 0$ ; $I_e[j] \leftarrow 0$ ;
```

Theorem 1 (Consensus on current network). *After the* SN *phase and independently of the initial membership flags M_k of any node $k \in [1, N]$, any two nodes $i, j \in [1, N]$ with a complete set of information either satisfy $R'_i = R'_j$ (identical requests) or $R'_i[k] \neq \perp \Rightarrow R'_j[k] = \perp$ for all $k \in [1, N]$ (requests from disjoint sets of nodes).*

Sketch of Proof. To prove the statement, we use a graph interpretation of the protocol where each node corresponds to a node $i \in V$ in a directed graph $G = (V, E)$. The initial membership flags M_i lead to the edges E of G: If $M_i[j] = 1$, then $(i, j) \in E$. Note that $(i, i) \in E$. We associate two sets with each node $i \in V$:

- $m_i \subseteq V$, where initially we have $m_i := \{j : (i, j) \in E\}$. This set corresponds to M'_i, i.e., we have $j \in m_i \Leftrightarrow M'_i[j] = 1$.
- $r_i \subseteq V$, where initially we have $r_i := \{i\}$. This set corresponds to R'_i, i.e., we have $j \in r_i \Leftrightarrow R'_i[j] \neq \perp$.

We perform arbitrary updates on these sets according to the protocol. In particular, node i may merge information with a node j if $j \in m_i \wedge i \in m_j$ (line 16). If so, the sets of node i are updated to $m_i \leftarrow m_i \cup m_j$ and $r_i \leftarrow r_i \cup r_j$. A node has a complete set of information if $m_i = r_i$. Using this graph representation, we now prove an equivalent statement independently of G and the number and order of merges: *Any two complete nodes $i, j \in V$ either satisfy $r_i = r_j$ or $r_i \cap r_j = \emptyset$.* To show this result, we apply three invariants:

- If $k \in r_i$, then $l \in m_i$ for all l with $(k, l) \in E$, i.e., if a node k is in the set r_i, then not only is this node in set m_i but also all its immediate successor nodes. This also leads to $r_i \subseteq m_i$.
- If $k \in m_i$, then there exists a directed path in G from i to k.
- If $k, l \in r_i$, then there exists a directed path in G from k to l.

Based on the above statements, we can conclude that if a node has the complete set of information ($m_i = r_i$), the nodes in m_i are strongly connected (each node $k \in m_i$ can reach any other node $l \in m_i$), and for any node in m_i, all its immediate successors are also in m_i. As a result, m_i of a complete node i forms a strongly connected component (SCC) of G. As all immediate successor nodes are already in m_i and therefore no node can be added, it is a *maximal* SCC. As such SCCs form a partition of all nodes [44], each node $i \in V$ can be in exactly one SCC, which proves the property. □

As the complete sets of information must be either identical or disjunct, and because only a single disjunct set can contain a majority of nodes required for schedule computation (line 24), the uniqueness of the newly computed schedule is guaranteed.

Based on the above result, the following two theorems show the safety of Hydra, i.e., no two nodes send different packets at the same time in the SD phase (Theorem 2) or DD phase (Theorem 3).

Theorem 2 (Safety of schedule distribution). *In the* SD *phase, no two nodes $i, j \in [1, N]$ transmit packets with different contents (v, S^v) (lines 35 and 37 of Algorithm 1).*

Sketch of Proof. Only nodes that have received the identical complete set of information satisfy either $updated = 1$ or $retransmit = 1$ (see proof of Theorem 1) and send packets in the SD phase. If the condition in line 26 is satisfied, all nodes in the current network have the same schedule version and the same set of schedule requests R'. As the scheduling algorithm is deterministic, they compute the same schedule S with the new schedule version and distribute it in line 35. An analogous argument holds for line 37. □

Theorem 3 (Safety of data dissemination). *In the* DD *phase, no two nodes $i, j \in [1, N]$ transmit different packets in the same* DD *slot (line 2 of Algorithm 1).*

Sketch of Proof. According to the scheduling conditions described in Section 4.3, two successive schedule versions can be used by any nodes $i, j \in [1, N]$ without violating safety. Note that nodes with $v = 0$ do not participate in the DD phase and are safe (line 2). Therefore, we need to show that at the beginning of the DD phase, nodes have either successive schedule versions or satisfy $v = 0$.

Suppose that two schedule versions v^* and $v^* - 1$ are in use at the start of a round. Let us first look at nodes within the current network, i.e., the SCC of G in the proof of Theorem 1. Due to multiple used schedule versions, the next schedule cannot be computed in line 27 and hence cannot be distributed in line 35. Only after all nodes with $v^* - 1$ have received version v^* in line 39 (and $v^* - 1$ is not used anymore), a new schedule $v^* + 1$ can be computed. At any time, there are at most two schedules in use which are successive.

A node that is not part of the current network may still know the outdated schedule version $v^* - 1$ if it did not receive any of the newer schedules distributed in the SD phases. The continued use of $v^* - 1$ is prevented by forcing $v \longleftarrow 0$ in line 48 as neither has a schedule been received in any SD phase of the current epoch nor is the node part of a majority network, i.e., an SCC of G with more than $N/2$ nodes. The latter is derived in line 47 from variables $I_e[j]$ which are only set in line 18 if the node has a bidirectional path to a node j in G. This is however limited to a minority set of nodes. □

As a result, nodes can independently and asynchronously update their schedules and remain assured that their transmissions will not collide with data packets from nodes using another schedule.

5.2 Liveness

The following Theorem 4 shows that nodes continue participating using the current schedule and can communicate data packets in the DD phase of every round even under broad failure conditions.

Theorem 4 (Liveness of data dissemination). *Every node participates in the* DD *phase unless both of the following conditions are satisfied: (i) It did not receive a schedule in any* SD *phase of the previous epoch and (ii) during the entire previous epoch, it only stored the requests from a minority of nodes.*

Sketch of Proof. A node does not participate in the DD phase if $v = 0$, i.e., if line 48 was executed. This only happens if a minority of nodes $j \in [1, N]$ satisfy $I_e[j] = 1$ (line 47). If the node has received a schedule, then $I_e[j] \longleftarrow 1$ for all nodes (line 41) and the condition is not satisfied. If the node stores the request of a node j, it sets $I_e[j] \longleftarrow 1$ (line 18). Therefore, if the node did store the requests from more than $N/2$ different nodes at least once during the epoch, the condition to set $v \longleftarrow 0$ is not satisfied either. □

Note that a schedule is typically computed and distributed by many nodes due to the high level of redundancy. Therefore, a node will only fail to receive a schedule in any round of an epoch under a significant amount of very specific and simultaneous node or link failures. And even in such a case, a node may still participate in the data exchange if it has received and stored requests from a majority of nodes at least once during the entire previous epoch.

6 EVALUATION

To validate Hydra's fault tolerance to node and link failures and demonstrate its ability to provide a reliable communication service, we test the protocol for different failure types and network sizes. We thereby investigate how the network (*i*) maintains safe data exchange, (*ii*) adapts to changing conditions, and (*iii*) keeps communication alive between non-faulty nodes. By comparing against LWB [16], a state-of-the-art wireless protocol using centralized scheduling, we highlight that concurrent coordination eradicates any single point of failure without noticeably increasing protocol overhead. Throughout our experiments, we never encountered any safety violation in thousands of Hydra rounds while scheduling under various conditions and adverse failures, experimentally verifying that our formal proof of safety also holds in practice.

6.1 Implementation

We implement Hydra on an STMicroelectronics STM32L433CC microcontroller driving a Semtech SX1262 RF transceiver [42]. The package is available as a target [5] on the FlockLab testbed [45], operates in the 868 MHz band, and provides a wide range of TX power from −9 to +22 dBm, making it highly versatile for changing environmental conditions. To minimize the impact of external influences, we use the GFSK modulation at 250 kbps. One-to-all floods are implemented using Glossy [18] with multiple consecutive transmissions [32], while the all-to-all exchange is based on Chaos [29] with randomized initial TX/RX decisions to commence without a network coordinator which usually triggers the exchange.

Research artifacts. The source code is publicly available as open source [7]. In addition, we provide tools to thoroughly and reproducibly test Hydra in different failure scenarios using a highly-flexible input method leveraging GPIO actuation. Analysis scripts automatically validate the correct execution of the communication primitives and protocol phases, including safety and synchronization checks. Our companion document [7] further includes the bootstrapping algorithm and the detailed formal proofs of Hydra.

6.2 Experimental Setup

Test scenario. To observe Hydra in practice including hardware failures and varying link characteristics, we utilize the FlockLab testbed [45] with 23 nodes evenly spread across the 65 × 30 m floor of an office building. We test networks of three sizes to investigate Hydra's scalability: a small network consisting of 5 nodes, a medium network of 12 nodes, and a larger network containing all 23 usable indoor nodes of the testbed. A TX power of +7 dBm results in an average of 1.24 hops for the small network, 1.57 hops for the medium network, and 1.70 hops for the larger network. During normal operation, we use a main frequency channel at 869.8875 MHz with an average one-to-all packet reception rate (PRR) of 99.72%, while bootstrapping occurs at 868.4375 MHz.

Fault injection. To reproducibly inject failures, we use the GPIO actuation feature of FlockLab [45] to precisely affect the protocol execution of individual nodes. Actuating the reset pin allows us to cause full hardware failure. Depending on the configuration, a second GPIO pin changes link characteristics by either enforcing that nodes drop packets with a given probability or completely disconnecting them through a shift to an isolated frequency band. This mechanism enables us to re-play a real scenario multiple times to investigate it in more detail, promotes the re-production of our results, and permits a fair comparison between competing protocols.

Protocol parameters. As a default configuration, we use a round period of $T = 3$ s and an epoch length $F = 3$. Each node requests 3 slots per DD phase which are fairly allocated to one of 80 DD slots, transmitting data packets of 20 B in each flood using three transmissions. 36 contention slots during the SN phase permit nodes to exchange membership flags and requests. Nodes leave the network after 2 epochs with $v = 0$ and revert to bootstrapping. A bootstrapping node listens for traffic of an existing network with a probability of 20% and otherwise attempts to establish a new network.

Protocol comparison. Since our analysis of the state of the art in Section 8 shows that no existing design satisfies all three requirements of safety, adaptivity, and liveness while tolerating node and link failures, we compare the concept of concurrent coordination to its closest relative based on a traditional centralized design.

We benchmark Hydra against LWB [16], a state-of-the-art protocol using concurrent transmissions, to represent a protocol with a single point of failure due to its centralized scheduling. Similar to Hydra's DD phase, LWB consists of a sequence of data slots where packets are flooded using Glossy [18]. A contention slot allows a node to transmit its demand to a host node as the network coordinator which centrally executes the scheduling algorithm and distributes the resulting schedule using two additional slots. In stark contrast to Hydra, nodes are only permitted to transmit if they have received the schedule directly preceding the round. To avoid that a single schedule miss prevents a node from participating, the schedule is transmitted both at the end of the previous round and at the beginning of the round for which it is valid. We use the same parameters for all slots as for Hydra's DD phase.

6.3 Hydra's Fault Tolerance in Action

Node failure. We first investigate how Hydra copes with node failures. During the entire experiment shown in Figure 5, we monitor the network PRR of fault-free nodes to examine how well the protocol adapts and whether it preserves liveness. The PRR is defined as the percentage of the total number of data packets received compared to the total number of data packets the nodes expect to receive according to their schedule. If no schedule is known, such as at the start, the PRR is defined to be 0%.

Initially, we bootstrap a network of 23 nodes and do not inject failures for the first 90 s. We find that Hydra enables nodes to quickly form a network and transmit packets according to their demands within 13 s on average by merging requests into a single packet, while LWB requires three times as long to send individual requests to the centralized scheduler with a mean scheduling delay of 39 s. Hydra nodes that first searched for pre-existing networks during bootstrapping and did not participate from the start (such as nodes 2 and 3) quickly join the formed network after discovery. Hydra's PRR remains high once all nodes joined and validates its liveness, while LWB suffers occasional drops from schedule misses.

At 90 s, we inject a first node failure at node 1, which causes a drop in PRR for both protocols. Hydra quickly adapts by excluding

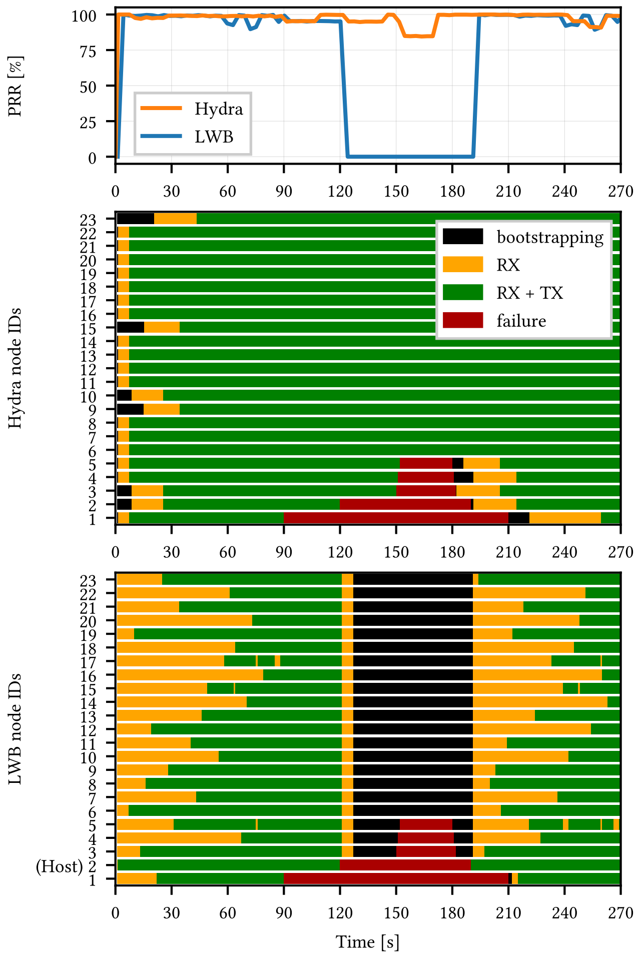

Figure 5: Hydra (middle) quickly bootstraps a network and starts exchanging scheduling information, with synchronized nodes listening according to the schedule ("RX") until their own request is granted. The network swiftly reaches distributed consensus, whereafter nodes efficiently transmit and receive data packets ("RX+TX") despite node failures. LWB (bottom) cannot maintain communication if the centralized scheduler at the host node 2 is unreachable.

node 1 after one epoch and determining a new schedule, thereby regaining efficiency as all assigned schedule slots are used. LWB on the other hand relies on a centralized timeout mechanism and delays the exclusion of node 1 until the network coordinator could confirm over multiple rounds that the node is permanently missing.

At 120 s, we inject a second node failure at node 2, which serves as the host node for LWB where the centralized scheduler is situated. Lacking a schedule, all LWB nodes must immediately cease transmissions to preserve safety as an old schedule might cause interference. As LWB's fixed failover leader only takes over after 2 min [16], communication collapses and nodes are forced to idle listen for a new leader. Hydra on the other hand has previously re-established efficient communication at 100% PRR and now only experiences another minor drop. Even three nodes failing almost simultaneously at 150 s do not endanger Hydra's liveness.

At 180 s, we reactivate nodes 3 - 5, which quickly join the existing Hydra network but must wait for the network coordinator to reappear for LWB. As soon as node 2 is restarted at 190 s, LWB recommences and gradually includes one node per round due to the contention-based relaying of requests to the centralized scheduler.

We find that Hydra successfully adapts to node failures and is able to quickly re-establish efficient communication independently of which nodes are affected. Liveness is ensured even if distributed consensus is temporarily delayed as nodes leave and join, and a coordinated transition between schedules ensures that safety is preserved. On the other hand, LWB experiences long delays until nodes could communicate their requests to the centralized scheduler. While LWB provides both adaptivity and safety, it cannot maintain liveness as it suffers from a single point of failure at the host node which orchestrates communication. This centralization of network coordination results in a catastrophic collapse of the entire network when the host node is not reachable.

Link failure. Next, we investigate Hydra's behavior when the network splits, a common scenario for devices moving in swarms [27]. As network separation results in partitions that might interfere and endanger safety, we artificially form two sets of 11 and 12 nodes. Initially, a network of 23 nodes establishes without injected failures as shown in Figure 6. After 45 s, we prevent all communication between the two sets and observe a brief drop in the PRR of the majority set of 12 nodes as half of the expected packets are missed.

After these nodes obtain the new complete set of information, visible in the top plot just before 60 s by the return to 100% of nodes being complete, they quickly determine a new schedule of version 5 as seen in the middle plot around 60 s. The schedule of the minority set of nodes already expires beforehand and their schedule version returns to 0. Finally, the minority again enters bootstrapping.

At 90 s, we reactivate communication between the two sets of nodes and observe that the minority set quickly rejoins the majority set. Around 105 s, the network determines a new schedule of version 6 which already includes a combination of the set of nodes.

We draw two major conclusions. First, we find that even if an arbitrary set of almost half the network is unreachable, the remaining nodes still continue operation and are not impeded in disseminating their data due to the failure of others. This demonstrates Hydra's high degree of fault tolerance. Second, the expiration of the schedule for the minority set of nodes, seen by the drop of the schedule version to 0 in the middle plot, is crucial for safety. If the minority kept using it, it could immediately start interfering with a new schedule determined by the majority of nodes when both sets of nodes are re-combined at 90 s. Only by asserting that no one but the majority of nodes is scheduled can the compatibility of used schedules be ensured and safety unconditionally guaranteed.

6.4 Distributed Scheduling Overhead

We have observed in Section 6.3 that distributed scheduling can greatly outperform centralized scheduling in case of node failures. However, concurrent coordination achieves its fault tolerance by relying on redundancy and therefore strives for every node in the network to obtain the required scheduling information. To investigate whether this increases runtime overhead, we execute both Hydra and LWB for 900 rounds and compare the average RX and TX duty cycles of their different phases for multiple network sizes in Figure 7. As mentioned in Section 6.2, LWB includes a data dissemination phase just like Hydra's DD phase. In addition, LWB uses a contention slot for nodes to relay requests to the centralized scheduler. This functionality corresponds to the SN phase of Hydra's distributed scheduler. Lastly, LWB's network coordinator transmits the new schedule once at the end of a round and re-transmits it at the beginning of the next round, while multiple Hydra nodes initiate a single schedule flood in the SD phase.

As data dissemination is equivalent for both Hydra and LWB with three packets flooded per node, the TX duty cycle does not differ for all network sizes. We find that Hydra's RX duty cycle for data exchange is slightly increased as Hydra starts to listen early in the first data slot of a round to compensate for potential clock drifts. Astonishingly, the scheduling overhead of both protocols is also

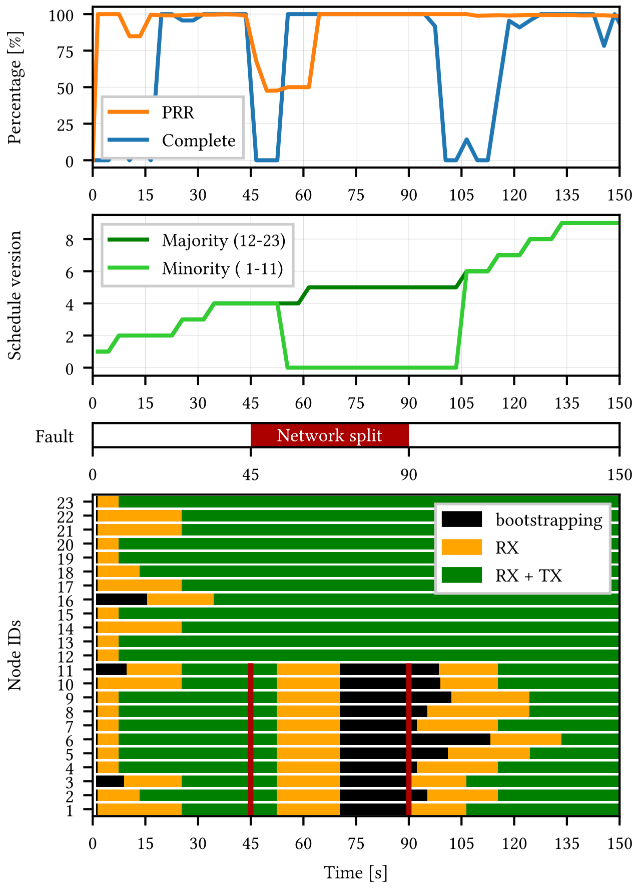

Figure 6: Splitting a network into a majority and a minority demonstrates that Hydra correctly forces the schedule of nodes being part of the minority to expire. To maintain safety, such nodes can only resume communication after re-connecting to the majority, whereafter a new complete set of information is swiftly obtained.

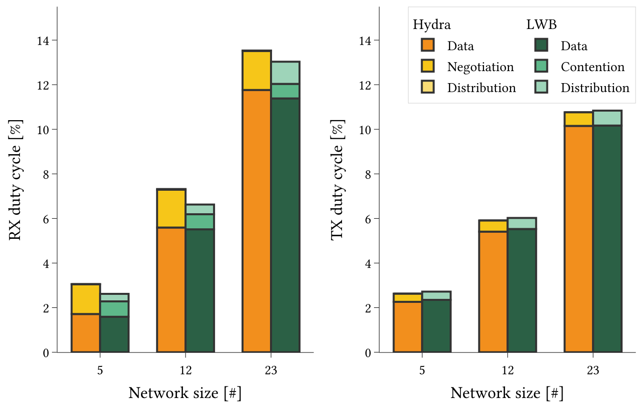

Figure 7: While data exchange is almost identical for both protocols as they rely on the same communication primitive, the difference in scheduling overhead is also negligible for larger networks.

almost identical, with the RX duty cycle rising by only 0.12% and the TX duty cycle dropping by −0.02% for a network of 23 nodes.

With the scheduling costs being almost equivalent for larger networks and smaller networks showing increases by less than a quarter despite over-provisioning for quick adaptivity (see Figure 9), we conclude that the overhead of distributed scheduling is negligible. While this result might initially be unexpected as the same data needs to be at all nodes instead of only one, it quickly becomes apparent that concurrent coordination also enables a reduction of the communication costs. First, as most nodes complete the SN phase successfully and no schedule needs to be distributed if requests remain constant, the SD overhead diminishes to only 0.02% RX duty cycle. Second, the constant merging of received information in the SN phase leads to relatively few transmissions required for the exchange of scheduling information. Third, because the SN phase consists of short contention slots, Hydra does not need to idle listen for long Glossy slots provisioned for multiple hops, while LWB's RX duty cycle increases with a larger network size as floods take longer to reach nodes further away.

6.5 Investigating Hydra in Detail

Epoch length. Next, we examine the epoch length F as a tool to increase network stability in conjunction with C_{min}, used in line 51. By default, $C_{min} = 1$ so nodes are included in M if their information is received at least once in an epoch. We now increase this threshold to $C_{min} = F$ and compare the behavior of networks with $F = 1$ and $F = 3$ upon link failures in Figure 8. To simulate packet loss, we force a network of nodes 1, 2, 4, and 5 to drop received packets from node 3 with a probability of 90% for the first 50 s. We find that a network with $F = 3$ successfully prevents such a node with unreliable links from joining and therefore preserves a high PRR. A network with $F = 1$ accepts node 3 even if it is only heard once, causing a PRR drop due to the included links with high loss.

At 50 s, we stop the artificial drop of packets and find that node 3 can quickly join the network with $F = 3$. At 100 s, we re-introduce link failures and observe that Hydra with $F = 3$ adjusts by excluding node 3 after it has been confirmed that its link quality has decreased.

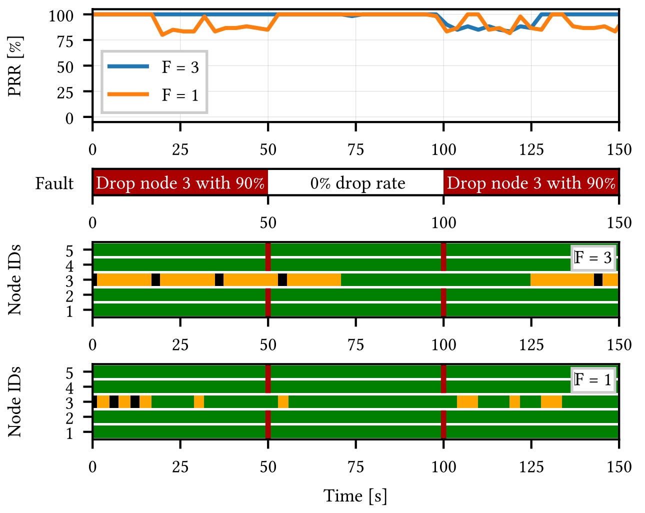

Figure 8: The epoch length F offers control of the speed of adaptivity and quality of included links. For both networks, 90% of the packets from node 3 are dropped for the first and last 50 s. In contrast to $F = 1$ (bottom), node 3 is only part of the network with $F = 3$ (middle) when it communicates reliably, which boosts PRR.

Thereafter, the PRR for the remaining network again reaches 100%. With $F = 1$, node 3 is permitted to stay to preserve liveness but incurs a reduced PRR for the rest of the network. We conclude that the epoch length is a valuable knob to increase network stability and can enable efficient networks by enforcing reliable links.

Asymmetric links. A critical case that may put safety at risk is the occurrence of asymmetric communication links, as a node might continue to consider itself part of the majority without them even being aware of the node. To prevent such imbalances, the merging criteria presented in Section 4.2 ensure that only nodes that mutually include each other in their membership flags can prevent schedule expiration. The effect of the merging criteria can be seen in Figure 8 for $F = 3$ at 125 s, where the exclusion of a node from the majority forces its schedule to expire and preserves safety.

Time synchronization. In contrast to traditional Chaos which is initiated by a network coordinator, Hydra needs another means of time synchronization. We find that we can successfully use the Glossy floods during the DD phase to synchronize transmissions of all 23 nodes during the contention-based transmissions of the SN phase to an average of 0.4 μs. With a maximum offset of 1.43 μs, this is more than sufficient for the capture effect requiring 160 μs [29].

Time to completion. Lastly, we test how quickly distributed consensus is found in Figure 9. We run 300 rounds for different network sizes and observe in which of the 36 contention slots of the SN phase the complete set of information is obtained. A network of 5 nodes succeeds for 99% of nodes in gathering the complete set with an average of 10 slots. For 12 nodes, 91% of nodes are complete at an average of 21 slots, and 23 nodes complete 88% of the time in a mean of 26 slots. As a single successful node is sufficient to determine the next schedule and distribute it to the rest of the nodes in the SD

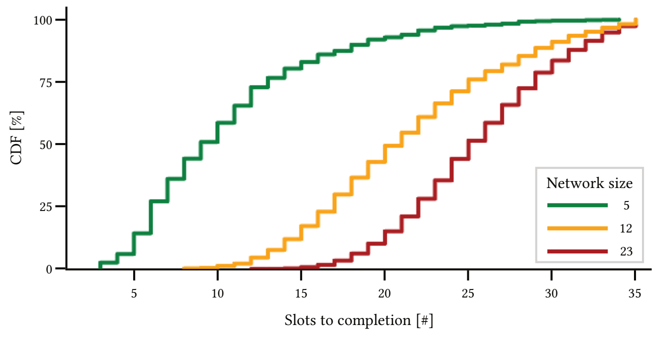

Figure 9: The average number of slots during the SN phase to obtain the complete set of information depends on the network size, with small networks requiring less overhead as they complete quickly.

phase, using fewer contention slots could substantially reduce the scheduling overhead, particularly for smaller networks.

7 DISCUSSION

7.1 Exploring the Trade-off Space

Trading off adaptivity and liveness. Hydra's three protocol requirements are conflicting. As a design choice, we unconditionally guarantee safety, as avoiding packet collisions is an indispensable precondition for reliable packet delivery and hence dependable communication. If we were to relax safety, then achieving adaptivity and liveness would be easily possible using best-effort strategies, e.g., by letting nodes send data purely according to current demand.

However, there is a trade-off between adaptivity and liveness. If the schedule were fully static, safety and liveness can be guaranteed as the schedule is always up-to-date. At the other extreme, adapting the network membership every round could result in a disproportionate loss of liveness for nodes briefly losing connectivity as schedules expire to preserve safety. With the epoch length F, Hydra offers control of this trade-off. While Hydra's design uses a fixed length, dynamically reducing this parameter at runtime based on current channel state information and past performance may boost adaptivity when environmental conditions remain stable.

Trading off complexity and robustness. While centralized network coordination constitutes a straightforward solution, it lacks robustness as demonstrated by the collapse in network communication in Figure 5 and must therefore be decentralized to provide fault tolerance. As soon as more than a single node is involved, the protocol complexity of consensus needs to be introduced independently of the number of participating coordinators. Therefore, while a hybrid approach with only a subset of nodes coordinating concurrently may seem appealing, reducing the fraction of involved nodes does not simplify the underlying problem. Furthermore, as the employed communication primitives based on concurrent transmissions result in all nodes indiscriminately obtaining all information, only the negligible local overhead of the schedule computation could be avoided. As nodes that do not participate in coordination must listen during the SD phase, we expect the resulting idle listening overhead to significantly surpass any potential computational savings.

7.2 Limitations and Extensions

Supporting flexible network sizes. With Hydra, we choose to prioritize safety over liveness. However, Hydra's consensus mechanism can also be employed for a fault-tolerant network where liveness should be maintained at the cost of potential packet collisions. This is of particular importance if co-existing network partitions should be supported, which Hydra deliberately prevents by enforcing that only a majority of nodes may persist. By removing the check on the number of included nodes (line 24) and the schedule expiration (line 48), such an extension is straightforward and would remedy a limitation on the minimum and maximum network sizes. However, an extra discovery mechanism to merge partitions is required to avoid a continuous decay into smaller sets of nodes.

Scaling to large networks. The use of all-to-all primitives such as Chaos [29] demands the allocation of fixed variables per potential node in each packet so that information can be merged. While this only amounts to 4 bits per node in our implementation for the membership flag and the request, the energy consumption scales proportionally with N. In addition to the packet size, we also see in Figure 9 that the length of the SN phase needs to be increased with N as more information has to be exchanged to obtain the complete set of information. While we find that an all-to-all primitive enables a network of $N = 23$ to bootstrap $2.9\times$ faster than a traditional design based on a single contention slot (Figure 5) and is therefore favorable if requests are changing frequently, a large and stable network may benefit from only exchanging information on demand.

8 RELATED WORK

Consensus in WSNs. LWB [16] ensures safety by limiting nodes to only transmitting when a schedule from a network coordinator has been received. It cyclically shifts through leaders upon failure, further dropping liveness in between. Virtus [17] provides atomic multicast with packet delivery ordering in addition to group membership but builds on LWB. Chaos [29] and Mixer [21] do not require explicit scheduling to exchange data among all nodes but depend on a static network coordinator to initiate the transfer and thus cannot ensure liveness. A^2 [3] and Wireless Paxos [40] are consensus protocols built on top of Chaos to take decisions on values proposed by nodes in the network, but rely on a static coordinator and hence a single point of failure. Furthermore, Hydra's specialized scope reaches asynchronous consensus without the overhead of distinct phases common in consensus protocols [28, 39]. BUTLER [36] enables the use of existing protocols based on concurrent transmissions without a unique time source once coarse synchronization could be established. wChain [50] focuses on fault-tolerant blockchain operations in wireless networks, but may not sustain leader failure. BCA [10] evaluates sensor data based on majority-consensus voting, but does not tolerate link failures.

Decentralized scheduling. TSCH [14] supports reliable multi-sink applications but requires fragile routing trees and a central network coordinator, which limits liveness and presents a single point of failure. Most TSCH schedulers, such as MASTER [19], are centralized and send all requests to a single entity, which then computes and distributes a schedule. Distributed schedulers, such as MSF [11] and DeTAS [1], leverage information provided by RPL [49], which requires a stable routing tree and hence increases latency upon link failures by up to 25 min [30], thereby forfeiting liveness.

By contrast, Orchestra [15] uses local schedules to perform synchronous communication through pseudo-random resource usage. However, it still builds upon RPL and relies on its tree structure for coordination and synchronization. ALICE [25] and A^3 [26] are autonomous scheduling schemes that adapt to arbitrary traffic patterns, but cannot avoid a single point of failure due to their reliance on RPL. TREE [30] tries to learn demands without RPL, and AUTOBAHN [20] bridges the gap between TSCH and concurrent transmissions. However, both require centralized coordination and therefore cannot tolerate arbitrary node failures.

Fighting interference. Significant effort has increased network robustness under external interference [22, 23, 33, 34, 38, 41, 43, 51, 53]. However, these protocols remain constrained by their requirement to reach a network coordinator for either scheduling or synchronization. We consider such efforts to optimize link robustness orthogonal to Hydra, which adds an extra layer of fault tolerance and guarantees fundamental properties despite failures.

9 CONCLUSION

With Hydra, we introduce a fault-tolerant WSN protocol that provides safety, adaptivity, and liveness for multi-hop networks. At its core is a distributed consensus algorithm which is formally proven to result in unique decisions and does not increase overhead compared to centralized schemes. We demonstrate that combined with a versioning-based scheduling mechanism, collisions with packets from other nodes running Hydra can be eradicated by design. By inducing node and link failures, we experimentally validate that *concurrent coordination* preserves efficient data exchange while remaining adaptive to changing conditions. We show that centralized approaches are unable to provide an equally reliable communication service, even though they build on the same robust communication primitives. While Hydra does not focus on mitigating failures, it addresses their impact on a higher layer by providing guarantees despite them. Hydra offers a unique ability to create networks where each node is equivalent in protocol logic and configuration, paving the way for a highly dependable Internet of Things.

ACKNOWLEDGMENTS

This research was supported by the Swiss National Science Foundation under NCCR Automation (grant agreement 51NF40_180545) and by the German Research Foundation through SPP 1914 (grant ZI 1635/1-1) and NextIoT (grant ZI 1635/2-1).

REFERENCES

[1] Nicola Accettura, Elvis Vogli, Maria Rita Palattella, Luigi Alfredo Grieco, Gennaro Boggia, and Mischa Dohler. 2015. Decentralized Traffic Aware Scheduling in 6TiSCH Networks: Design and Experimental Evaluation. *IEEE Internet of Things Journal* 2, 6 (2015), 455–470. https://doi.org/10.1109/JIOT.2015.2476915

[2] Mikhail Afanasov, Naveed Anwar Bhatti, Dennis Campagna, Giacomo Caslini, Fabio Massimo Centonze, Koustabh Dolui, Andrea Maioli, Erica Barone, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2020. Battery-Less Zero-Maintenance Embedded Sensing at the Mithræum of Circus Maximus. In *18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20)*. ACM, New York, NY, USA, 368–381. https://doi.org/10.1145/3384419.3430722

[3] Beshr Al Nahas, Simon Duquennoy, and Olaf Landsiedel. 2017. Network-Wide Consensus Utilizing the Capture Effect in Low-Power Wireless Networks. In *15th ACM Conference on Embedded Network Sensor Systems* (Delft, Netherlands) *(SenSys '17)*. ACM, New York, NY, USA, Article 1, 14 pages. https://doi.org/10.1145/3131672.3131685

[4] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, and Martin Vetterli. 2008. The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments. In *6th ACM Conference on Embedded Network Sensor Systems* (Raleigh, NC, USA) *(SenSys '08)*. ACM, New York, NY, USA, 43–56. https://doi.org/10.1145/1460412.1460418

[5] Jan Beutel, Roman Trüb, Reto Da Forno, Markus Wegmann, Tonio Gsell, Romain Jacob, Michael Keller, Felix Sutton, and Lothar Thiele. 2019. The Dual Processor Platform architecture: Demo abstract. In *18th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Montreal, QC, Canada) *(IPSN '19)*. ACM, New York, NY, USA, 335–336. https://doi.org/10.1145/3302506.3312481

[6] Andreas Biri, Reto Da Forno, Tonio Gsell, Tobias Gatschet, Jan Beutel, and Lothar Thiele. 2021. STeC: Exploiting Spatial and Temporal Correlation for Event-Based Communication in WSNs. In *19th ACM Conference on Embedded Networked Sensor Systems* (Coimbra, Portugal) *(SenSys '21)*. ACM, New York, NY, USA, 274–287. https://doi.org/10.1145/3485730.3485951

[7] Andreas Biri, Reto Da Forno, Tobias Kuonen, Fabian Mager, Marco Zimmerling, and Lothar Thiele. 2023. Hydra: Companion document and source code. https://gitlab.ethz.ch/tec/public/hydra

[8] Amelie Bonde, Jesse Codling, Kanittha Naruethep, Yiwen Dong, Wachirawich Siripaktanakon, Sripong Ariyadech, Akkarit Sangpetch, Orathai Sangpetch, Shijia Pan, Hae Young Noh, and Pei Zhang. 2021. PigNet: Failure-Tolerant Pig Activity Monitoring System Using Structural Vibration. In *20th IEEE International Conference on Information Processing in Sensor Networks* (Nashville, TN, USA) *(IPSN '21)*. ACM, New York, NY, USA, 328–340. https://doi.org/10.1145/3412382.3458902

[9] Hannah Brunner, Michael Stocker, Maximilian Schuh, Markus Schuß, Carlo Alberto Boano, and Kay Römer. 2022. Understanding and Mitigating the Impact of Wi-Fi 6E Interference on Ultra-Wideband Communications and Ranging. In *21th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Milan, Italy) *(IPSN '22)*. ACM, New York, NY, USA, 92–104. https://doi.org/10.1109/IPSN54338.2022.00015

[10] Jenghorng Chang and Fanpyn Liu. 2021. A Byzantine Sensing Network Based on Majority-Consensus Data Aggregation Mechanism. *Sensors* 21, 1 (2021), 17 pages. https://doi.org/10.3390/s21010248

[11] Tengfei Chang, Mališa Vučinić, Xavier Vilajosana, Simon Duquennoy, and Diego Roberto Dujovne. 2021. 6TiSCH Minimal Scheduling Function (MSF). RFC 9033. https://dl.acm.org/doi/10.17487/RFC9033

[12] Lili Chen, Jie Xiong, Xiaojiang Chen, Sunghoon Ivan Lee, Kai Chen, Dianhe Han, Dingyi Fang, Zhanyong Tang, and Zheng Wang. 2019. WideSee: Towards Wide-Area Contactless Wireless Sensing. In *17th ACM Conference on Embedded Networked Sensor Systems* (New York, NY, USA) *(SenSys '19)*. ACM, New York, NY, USA, 258–270. https://doi.org/10.1145/3356250.3360031

[13] Madeleine Daepp, Alex Cabral, Vaishnavi Ranganathan, Vikram Iyer, Scott Counts, Paul Johns, Asta Roseway, Charlie Catlett, Gavin Jancke, Darren Gehring, Chuck Needham, Curtis von Veh, Tracy Tran, Lex Story, Gabriele D'Amone, and Bichlien Nguyen. 2022. Eclipse: An End-to-End Platform for Low-Cost, Hyperlocal Environmental Sensing in Cities. In *21th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Milan, Italy) *(IPSN '22)*. ACM, New York, NY, USA, 28–40. https://doi.org/10.1109/IPSN54338.2022.00010

[14] Domenico De Guglielmo, Simone Brienza, and Giuseppe Anastasi. 2016. IEEE 802.15.4e: A survey. *Computer Communications* 88 (2016), 1–24. https://doi.org/10.1016/j.comcom.2016.05.004

[15] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. 2015. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In *13th ACM Conference on Embedded Networked Sensor Systems* (Seoul, South Korea) *(SenSys '15)*. ACM, New York, NY, USA, 337–350. https://doi.org/10.1145/2809695.2809714

[16] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. 2012. Low-Power Wireless Bus. In *10th ACM Conference on Embedded Network Sensor Systems* (Toronto, ON, Canada) *(SenSys '12)*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/2426656.2426658

[17] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. 2013. Virtual Synchrony Guarantees for Cyber-physical Systems. In *IEEE 32nd International Symposium on Reliable Distributed Systems* (Braga, Portugal) *(SRDS '13)*. IEEE, 20–30. https://doi.org/10.1109/SRDS.2013.11

[18] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. 2011. Efficient network flooding and time synchronization with Glossy. In *10th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Chicago, IL, USA) *(IPSN '11)*. ACM, New York, NY, USA, 73–84. https://ieeexplore.ieee.org/abstract/document/5779066

[19] Oliver Harms and Olaf Landsiedel. 2020. MASTER: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks. In *16th International Conference on Distributed Computing in Sensor Systems* (Marina del Rey, CA, USA) *(DCOSS '20)*. IEEE, 86–94. https://doi.org/10.1109/DCOSS49796.2020.00025

[20] Oliver Harms and Olaf Landsiedel. 2021. Opportunistic Routing and Synchronous Transmissions Meet TSCH. In *46th Conference on Local Computer Networks* (Edmonton, AB, Canada) *(LCN '22)*. IEEE, 107–114. https://doi.org/10.1109/LCN52139.2021.9524952

[21] Carsten Herrmann, Fabian Mager, and Marco Zimmerling. 2018. Mixer: Efficient Many-to-All Broadcast in Dynamic Wireless Mesh Networks. In *16th ACM Conference on Embedded Networked Sensor Systems* (Shenzhen, China) *(SenSys '18)*. ACM, New York, NY, USA, 145–158. https://doi.org/10.1145/3274783.3274849

[22] Timofei Istomin, Oana Iova, Gian Pietro Picco, and Csaba Kiraly. 2019. Route or Flood? Reliable and Efficient Support for Downward Traffic in RPL. *ACM Transactions on Sensor Networks* 16, 1, Article 1 (Oct. 2019), 41 pages. https://doi.org/10.1145/3355997

[23] Timofei Istomin, Matteo Trobinger, Amy Lynn Murphy, and Gian Pietro Picco. 2018. Interference-Resilient Ultra-Low Power Aperiodic Data Collection. In *17th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Porto, Portugal) *(IPSN '18)*. IEEE, 84–95. https://doi.org/10.1109/IPSN.2018.00015

[24] Dhananjay Jagtap and Pat Pannuto. 2021. Repurposing Cathodic Protection Systems as Reliable, in-Situ, Ambient Batteries for Sensor Networks. In *20th International Conference on Information Processing in Sensor Networks* (Nashville, TN, USA) *(IPSN '21)*. ACM, New York, NY, USA, 357–368. https://doi.org/10.1145/3412382.3458277

[25] Seohyang Kim, Hyung-Sin Kim, and Chongkwon Kim. 2019. ALICE: Autonomous Link-Based Cell Scheduling for TSCH. In *18th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Montreal, QC, Canada) *(IPSN '19)*. ACM, New York, NY, USA, 121–132. https://doi.org/10.1145/3302506.3310394

[26] Seohyang Kim, Hyung-Sin Kim, and Chong-kwon Kim. 2021. A3: Adaptive Autonomous Allocation of TSCH Slots. In *20th IEEE International Conference on Information Processing in Sensor Networks* (Nashville, TN, USA) *(IPSN '21)*. ACM, New York, NY, USA, 299–314. https://doi.org/10.1145/3412382.3458273

[27] Demeke Lakew, Umar Sa'ad, Nhu-Ngoc Dao, Woongsoo Na, and Sungrae Cho. 2020. Routing in Flying Ad Hoc Networks: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials* 22, 2 (2020), 1071–1120. https://doi.org/10.1109/COMST.2020.2982452

[28] Leslie Lamport. 2001. Paxos Made Simple. *ACM SIGACT News* 32, 4 (Dec. 2001), 51–58. https://www.microsoft.com/en-us/research/publication/paxos-made-simple/

[29] Olaf Landsiedel, Federico Ferrari, and Marco Zimmerling. 2013. Chaos: Versatile and Efficient All-to-All Data Sharing and in-Network Processing at Scale. In *11th ACM Conference on Embedded Networked Sensor Systems* (Roma, Italy) *(SenSys '13)*. ACM, New York, NY, USA, Article 1, 14 pages. https://doi.org/10.1145/2517351.2517358

[30] Tim Van Der Lee, Georgios Exarchakos, and Sonia Heemstra De Groot. 2020. Distributed Reliable and Energy-Efficient Scheduling for LR-WPANs. *ACM Transactions on Sensor Networks* 16, 4, Article 32 (Aug. 2020), 20 pages. https://doi.org/10.1145/3399805

[31] Krijn Leentvaar and Jan Flint. 1976. The capture effect in FM receivers. *IEEE Transactions on Communications* 24, 5 (May 1976), 531–539. https://doi.org/10.1109/TCOM.1976.1093327

[32] Roman Lim, Reto Da Forno, Felix Sutton, and Lothar Thiele. 2017. Competition: Robust Flooding using Back-to-Back Synchronous Transmissions with Channel-Hopping. In *International Conference on Embedded Wireless Systems and Networks* (Uppsala, Sweden) *(EWSN '17)*. Junction Publishing, USA, 270–271. https://dl.acm.org/doi/abs/10.5555/3108009.3108076

[33] Xiaoyuan Ma, Peilin Zhang, Xin Li, Weisheng Tang, Jianming Wei, and Oliver Theel. 2018. DeCoT: A Dependable Concurrent Transmission-Based Protocol for Wireless Sensor Networks. *IEEE Access* 6 (2018), 73130–73146. https://doi.org/10.1109/ACCESS.2018.2877692

[34] Xiaoyuan Ma, Peilin Zhang, Ye Liu, Carlo Alberto Boano, Hyung-Sin Kim, Jianming Wei, and Jun Huang. 2020. Harmony: Saving Concurrent Transmissions from Harsh RF Interference. In *IEEE Conference on Computer Communications* (Toronto, ON, Canada) *(INFOCOM '20)*. IEEE, 1024–1033. https://doi.org/10.1109/INFOCOM41043.2020.9155423

[35] Fabian Mager, Dominik Baumann, Romain Jacob, Lothar Thiele, Sebastian Trimpe, and Marco Zimmerling. 2019. Feedback Control Goes Wireless: Guaranteed Stability over Low-Power Multi-Hop Networks. In *10th ACM/IEEE International Conference on Cyber-Physical Systems* (Montreal, QC, Canada) *(ICCPS '19)*. ACM, New York, NY, USA, 97–108. https://doi.org/10.1145/3302509.3311046

[36] Fabian Mager, Andreas Biri, Lothar Thiele, and Marco Zimmerling. 2022. BUTLER: Increasing the Availability of Low-Power Wireless Communication Protocols. In *International Conference on Embedded Wireless Systems and Networks* (Linz, Austria) *(EWSN '22)*. ACM, New York, NY, USA, 108–119. https://dl.acm.org/doi/abs/10.5555/3578948.3578958

[37] Matthias Meyer, Timo Farei-Campagna, Akos Pasztor, Reto Da Forno, Tonio Gsell, Jérome Faillettaz, Andreas Vieli, Samuel Weber, Jan Beutel, and Lothar Thiele. 2019. Event-triggered Natural Hazard Monitoring with Convolutional Neural Networks on the Edge. In *18th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Montreal, QC, Canada) *(IPSN '19)*. IEEE, 73–84. https://doi.org/10.1145/3302506.3310390

[38] Venkata Modekurthy, Abusayeed Saifullah, and Sanjay Madria. 2019. DistributedHART: A Distributed Real-Time Scheduling System for WirelessHART Networks. In *IEEE Real-Time and Embedded Technology and Applications Symposium* (Montreal, QC, Canada) *(RTAS '19)*. IEEE, 216–227. https://doi.org/10.1109/RTAS.2019.00026

[39] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference* (Philadelphia, PA, USA) *(USENIX ATC '14)*. USENIX Association, USA, 305–319. https://dl.acm.org/doi/10.5555/2643634.2643666

[40] Valentin Poirot, Beshr Al Nahas, and Olaf Landsiedel. 2019. Paxos Made Wireless: Consensus in the Air. In *International Conference on Embedded Wireless Systems and Networks* (Beijing, China) *(EWSN '19)*. Junction Publishing, USA, 1–12. https://dl.acm.org/doi/abs/10.5555/3324320.3324322

[41] Valentin Poirot and Olaf Landsiedel. 2021. Dimmer: Self-Adaptive Network-Wide Flooding with Reinforcement Learning. In *IEEE 41st International Conference on Distributed Computing Systems* (Washington, DC, USA) *(ICDCS '21)*. IEEE, 293–303. https://doi.org/10.1109/ICDCS51616.2021.00036

[42] Semtech. Accessed: 2023-02-10. "Semtech SX1262 - LoRa Connect Long Range Low Power LoRa Transceiver". semtech.com/products/wireless-rf/lora-core/sx1262.

[43] Alberto Spina, Michael Breza, Naranker Dulay, and Julie McCann. 2020. XPC: Fast and Reliable Synchronous Transmission Protocols for 2-Phase Commit and 3-Phase Commit. In *International Conference on Embedded Wireless Systems and Networks* (Lyon, France) *(EWSN '20)*. Junction Publishing, USA, 73–84. https://dl.acm.org/doi/10.5555/3400306.3400316

[44] Robert Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160. https://doi.org/10.1137/0201010

[45] Roman Trüb, Reto Da Forno, Lukas Sigrist, Lorin Mühlebach, Andreas Biri, Jan Beutel, and Lothar Thiele. 2020. FlockLab 2: Multi-Modal Testing and Validation for Wireless IoT. In *3rd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things (CPS-IoTBench 2020)*. OpenReview.net, 7 pages. https://doi.org/10.3929/ethz-b-000442038

[46] Samuel Weber, Jan Beutel, Reto Da Forno, Alain Geiger, Stephan Gruber, Tonio Gsell, Andreas Hasler, Matthias Keller, Roman Lim, Philippe Limpach, Matthias Meyer, Igor Talzi, Lothar Thiele, Christian Tschudin, Andreas Vieli, Daniel Vonder Mühll, and Mustafa Yücel. 2019. A decade of detailed observations (2008–2018) in steep bedrock permafrost at the Matterhorn Hörnligrat (Zermatt, CH). *Earth System Science Data* 11, 3 (2019), 1203–1237. https://doi.org/10.5194/essd-11-1203-2019

[47] Daniel Winkler, Miguel Carreira-Perpinan, and Alberto Cerpa. 2018. Plug-and-Play Irrigation Control at Scale. In *17th ACM/IEEE International Conference on Information Processing in Sensor Networks* (Porto, Portugal) *(IPSN '18)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1109/IPSN.2018.00008

[48] Daniel Winkler and Alberto Cerpa. 2019. WISDOM: Watering Intelligently at Scale with Distributed Optimization and Modeling. In *17th Conference on Embedded Networked Sensor Systems* (New York, NY, USA) *(SenSys '19)*. ACM, New York, NY, USA, 219–231. https://doi.org/10.1145/3356250.3360023

[49] Tim Winter, Pascal Thubert, Anders Brandt, Jonathan Hui, Richard Kelsey, Philip Levis, Kris Pister, Rene Struik, Jean-Philippe Vasseur, and Roger Alexander. 2012. RPL: IPv6 routing protocol for low-power and lossy networks. RFC 6550. https://doi.org/10.17487/RFC6550

[50] Minghui Xu, Chunchi Liu, Yifei Zou, Feng Zhao, Jiguo Yu, and Xiuzhen Cheng. 2021. wChain: A Fast Fault-Tolerant Blockchain Protocol for Multihop Wireless Networks. *IEEE Transactions on Wireless Communications* 20, 10 (Oct. 2021), 6915–6926. https://doi.org/10.1109/TWC.2021.3078639

[51] Zihao Yu, Xin Na, Carlo Alberto Boano, Yuan He, Xiuzhen Guo, and Meng Jin. 2022. SmarTiSCH: An interference-aware engine for IEEE 802.15. 4e-based networks. In *21st ACM/IEEE Conference on Information Processing in Sensor Networks* (Milano, Italy) *(IPSN '22)*. IEEE, 350–362. https://doi.org/10.1109/IPSN54338.2022.00035

[52] Pouria Zand, Supriyo Chatterjea, Kallol Das, and Paul Havinga. 2012. Wireless Industrial Monitoring and Control Networks: The Journey So Far and the Road Ahead. *Journal of Sensor and Actuator Networks* 1, 2 (2012), 123–152. https://doi.org/10.3390/jsan1020123

[53] Tianyu Zhang, Tao Gong, Song Han, Qingxu Deng, and Xiaobo Sharon Hu. 2021. Fully Distributed Packet Scheduling Framework for Handling Disturbances in Lossy Real-Time Wireless Networks. *IEEE Transactions on Mobile Computing* 20, 2 (2021), 502–518. https://doi.org/10.1109/TMC.2019.2950913

[54] Marco Zimmerling, Luca Mottola, and Silvia Santini. 2020. Synchronous Transmissions in Low-Power Wireless: A Survey of Communication Protocols and Network Services. *Comput. Surveys* 53, 6, Article 121 (Dec. 2020), 39 pages. https://doi.org/10.1145/3410159