Low-Power Wireless Bus

Federico Ferrari* Marco Zimmerling* Luca Mottola† Lothar Thiele*
*Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
†Politecnico di Milano, Italy and Swedish Institute of Computer Science (SICS)
{ferrari, zimmerling, thiele}@tik.ee.ethz.ch luca.mottola@polimi.it

Abstract

We present the *Low-Power Wireless Bus (LWB)*, a communication protocol that supports several traffic patterns and mobile nodes immersed in static infrastructures. LWB turns a multi-hop low-power wireless network into an infrastructure similar to a shared bus, where all nodes are potential receivers of all data. It achieves this by mapping all traffic demands on fast network floods, and by globally scheduling every flood. As a result, LWB inherently supports one-to-many, many-to-one, and many-to-many traffic. LWB also keeps no topology-dependent state, making it more resilient to link changes due to interference, node failures, and mobility than prior approaches. We compare the same LWB prototype on four testbeds with seven state-of-the-art protocols and show that: (*i*) LWB performs comparably or significantly better in many-to-one scenarios, and adapts efficiently to varying traffic loads; (*ii*) LWB outperforms our baselines in many-to-many scenarios, at times by orders of magnitude; (*iii*) external interference and node failures affect LWB's performance only marginally; (*iv*) LWB supports mobile nodes acting as sources, sinks, or both without performance loss.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Design, Experimentation, Performance

Keywords

Shared bus, network flooding, time-triggered protocols, centralized scheduling, mobility, wireless sensor networks

1 Introduction

Low-power wireless networks are gaining momentum beyond early data collection applications. For instance, recent deployments demonstrate the feasibility of closed-loop control [7] and increasingly employ mixed installations of static and mobile devices [9, 15]. These applications are characterized by a blend of traffic patterns, such as many-to-many communication for collecting sensor data at multiple sinks and one-to-many communication for disseminating control commands [7]. They also often feature end-to-end interactions across static and mobile nodes [9, 15].

In contrast to the diverse needs of emerging applications, current communication protocols support specific traffic patterns (*e.g.*, one-to-many [30], many-to-one [18] or many-to-many [37]) in distinct scenarios (*e.g.*, static networks [18] or with sink mobility [36]). This forces designers to form ad-hoc protocol ensembles to satisfy the application demands, which may entail adapting existing implementations [7] or developing custom protocols in absence of suitable off-the-shelf solutions [9]. As a result, multiple protocols that were designed in isolation need to operate concurrently. This is often detrimental to system performance [10], and causes protocol interactions that are difficult to cope with [45].

To address this problem, we present the *Low-Power Wireless Bus (LWB)*, a simple yet efficient communication protocol that provides unified support for several traffic patterns and mobile nodes immersed in static infrastructures. Our design revolves around three cornerstones:

1. We exclusively use fast *network floods* for communication. This turns a multi-hop low-power wireless network into a network infrastructure similar to a shared bus.
2. We adopt a *time-triggered operation* to arbitrate access to the shared bus. Nodes are time-synchronized and access the bus according to a *global communication schedule* computed online based on the current traffic demands.
3. We compute the communication schedule *centrally* at a dedicated *host* node. The host periodically distributes the schedule to all nodes to coordinate the bus operation.

To support our design, we use Glossy [17] as the underlying flooding mechanism. Glossy provides high flooding reliability with minimal latencies, offering a foundation for 1. and 3., and accurate global time synchronization at no additional cost [17], which we leverage as a stepping stone for 2. Glossy also maintains no topology-dependent network state, which spares state reconfigurations when topologies change.

As a result, LWB simplifies the networking architecture of low-power wireless systems by replacing the standard network stack with a single-layer solution that:

SenSys'12, November 6–9, 2012, Toronto, ON, Canada.

- *Supports multiple traffic patterns.* The exclusive use of Glossy network floods makes all nodes in the network potential receivers of all data. LWB leverages this opportunity to support both many-to-one and many-to-many traffic, besides the one-to-many pattern provided by Glossy itself. This occurs without changes to the protocol logic, and straightforwardly enables scenarios where, for example, multiple sinks are opportunistically deployed [44].
- *Is resilient to topology changes.* Different from most existing solutions, the network state kept at a LWB node is independent of the network topology and thus resilient to any such change. No state reconfigurations are indeed required to keep up with changing topologies, which reduces LWB's control overhead to a minimum. This provides efficient support to deal with link fluctuations, most notably due to node failures [4] and interference [32].
- *Supports node mobility.* As an extreme form of topology change, LWB encompasses also mobile nodes, acting as sources, sinks, or both, without any changes to the protocol logic. This applies to scenarios where, for example, mobile nodes interact with a fixed infrastructure [9].

The LWB protocol, described throughout Secs. 2 to 4, renders our design concrete and complements it with mechanisms to: (*i*) ensure a fair allocation of bandwidth across all traffic demands; (*ii*) support nodes dynamically joining and leaving the system (*e.g.*, due to node failures or disconnections); and (*iii*) resume communication after a host failure, thus overcoming single point of failure problems.

Using the same prototype, we evaluate in Secs. 5 to 10 LWB's performance on four testbeds that range from 26 to 260 nodes, including a testbed with nodes attached to robots for repeatable mobility experiments. For comparison, we consider seven combinations of state-of-the-art routing and link-layer protocols: BCP [36] over CSMA; CTP [18] over A-MAC [14], LPL [38], and CSMA; Muster [37] over LPL and CSMA; and Dozer [6]. Based on 256 independent runs over a total duration of 838 hours, we find that:

- In many-to-one scenarios, LWB performs comparably to Dozer and outperforms CTP in data yield and radio duty cycle; for example, LWB sustains traffic demands of 5 packets per second from 259 sources with almost 100 % data yield, a situation where CTP+LPL collapses.
- In the same scenarios, LWB adapts promptly and efficiently to varying traffic demands; for example, when the aggregate traffic load suddenly increases from 54 to 460 packets per minute, LWB keeps data yield close to 100 %, whereas CTP+LPL and Dozer are significantly affected.
- In many-to-many scenarios, LWB outperforms Muster regardless of the number of sources or sinks, providing higher data yield than Muster+CSMA at a fraction of the radio duty cycle of Muster+LPL at all wake-up intervals.
- Under external interference and multiple concurrent node failures, LWB's performance is only marginally affected, whereas CTP and Dozer require routing state reconfigurations that cause significant performance loss.
- In the presence of mobile nodes, LWB outperforms BCP and CTP at no additional energy costs, delivering more than 99 % of the packets at very low radio duty cycles regardless of whether sources, sinks, or both are roaming.

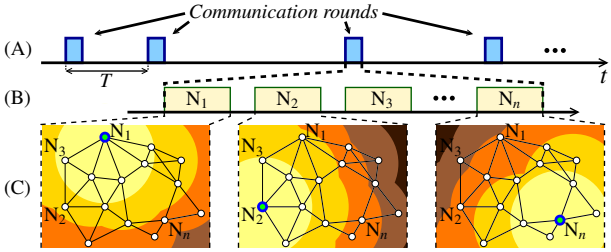


Figure 1. Time-triggered operation in LWB. *Protocol operation is confined within communication rounds that repeat with a possibly varying round period T (A); each round consists of a possibly varying number of non-overlapping slots (B); each slot corresponds to a distinct Glossy flood (C).*

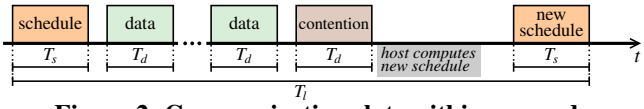


Figure 2. Communication slots within a round.

Our results demonstrate that LWB is more versatile than existing communication protocols, and performs comparably or significantly better than the state of the art in all scenarios we tested. As such, LWB is directly applicable to a broad spectrum of low-power wireless applications, from data collection [8, 31] to control [7] and mobile scenarios [9, 15].

Under specific operating conditions such as linear topologies that span several tens of hops [23] and applications with mostly aperiodic traffic [3] LWB's efficiency decreases and dedicated solutions may perform better. We discuss the limitations of LWB in Sec. 11 by illustrating its scaling properties and the dependence of a few protocol parameters on the network diameter. We also present alternative scheduling policies to satisfy different application requirements. We review related work in Sec. 12 and conclude in Sec. 13.

2 Overview

The LWB protocol completely replaces the standard network stack, sitting between radio driver and application.

LWB in a nutshell. LWB maps all communication on fast Glossy floods. A single flood serves to send a packet from one node to all other nodes. To avoid collisions among different floods, LWB adopts a time-triggered operation: nodes communicate according to a *global communication schedule* that determines when a node is allowed to initiate a flood.

LWB exploits Glossy's accurate global time synchronization. The protocol operation is confined within *communication rounds*. As shown in Fig. 1 (A), rounds repeat with a possibly varying *round period T*, computed at the host based on the current traffic demands. Nodes keep their radios off between two rounds to save energy. Every round consists of a possibly varying number of non-overlapping *communication slots*, as shown in Fig. 1 (B). In each slot, at most one node puts a message on the bus (initiates a flood), whereas all other nodes read the message from the bus (receive and relay the flood), as shown in Fig. 1 (C). All nodes participate in every flood, thus LWB inherently achieves load balancing.

Fig. 2 shows the different communication slots within a round. A round starts with a slot allocated to the host for distributing the communication schedule. The schedule in-

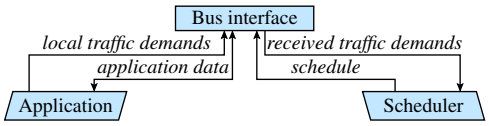


Figure 3. Conceptual architecture of a LWB node. *The scheduler is present at all nodes but active only at the host.*

cludes the round period T and the mapping of individual nodes to the following data slots, if any. A *contention slot* without a preassigned node follows; all nodes can contend in this slot, for example, to inform the host of their traffic demands. Based on the received traffic demands, the host computes the schedule for the next round—with a possibly updated round period and mapping of nodes to data slots—and transmits the new schedule at the end of the round.

Application interface. The application interacts with LWB in two ways, as shown in Fig. 3. First, LWB offers operations to place application messages in the outgoing queue for eventual transmission, and to receive incoming messages. Because of flooding-based communication, all nodes potentially receive all messages. At a sender node, the application specifies the intended recipients as a parameter to the send operation; at a receiver, LWB delivers a received message to the application only if the node is an intended recipient.

Second, LWB provides functions the application uses to notify LWB of changes in the traffic demands of a node. Targeting applications that feature mostly periodic traffic, LWB accepts traffic demands in the form of *periodic streams* of packets, defined by an *inter-packet interval (IPI)* and a *starting time*. The application can dynamically change the traffic demands, creating new streams or stopping existing ones.

When a new traffic demand arises, the application issues a stream add request, specifying IPI and starting time. The latter may lie in the past if, for example, the application needs to transmit a local backlog of packets. When a traffic demand ceases to exist, the application issues a stream remove request to cancel the stream. The application may issue multiple stream add requests from the same node (*e.g.*, if different sensors produce readings at different rates) and may individually remove streams.

Next, Sec. 3 describes the LWB protocol operation, while Sec. 4 focuses on the scheduler, which is a stand-alone component present at all LWB nodes but active only at the host.

3 Protocol Operation

To illustrate the protocol operation, we use sample executions of our LWB prototype whose configuration parameters and implementation details are described in Sec. 5. We split the illustration according to the different phases an execution evolves into. These phases are purely for illustration purposes and do not correspond to distinct modes of protocol operation. Rather, the mechanisms we illustrate next blend together and co-exist in a single protocol logic.

We start by illustrating in Sec. 3.1 the LWB operation in steady-state conditions; that is, when the host is informed of all traffic demands and these do not change over time. Next, we describe in Sec. 3.2 the bootstrapping phase that leads to such steady state. Finally, we describe in Sec. 3.3 how LWB adapts to further reduce overhead should steady-state conditions endure for a given time. We discuss protocol optimizations in Sec. 3.4, mechanisms to handle communication and node failures in Sec. 3.5, and host failures in Sec. 3.6.

The scenario we consider for the non-failure case is a multi-hop network of six source nodes and one sink. All source nodes have a stream with IPI $= 6$ s and starting time $t = 0$ s. For simplicity, the sink acts also as the host.

3.1 Steady-State Conditions

Intuition. In steady state, nodes are time-synchronized and the host is aware of all traffic demands. The aggregated traffic demand in the scenario we consider amounts to 6 packets per second. Say the round period T is 1 second and nodes are already informed of that. One way to schedule such traffic demands is to allocate 6 data slots, one per source node, in one round every six. Other schedules are feasible, possibly with different round periods; here we simply illustrate a specific instance of our general scheduling strategy (see Sec. 4).

Steady traffic demands. The middle part of Fig. 4 shows how the above materializes in a real LWB execution. Once steady-state conditions are reached at $t = 12$ s, the host distributes a schedule including 6 data slots, one per source. Nodes periodically turn their radios on during communication rounds according to the round period $T = 1$ s. Based on the schedule, each source accesses the bus during its allocated data slot and initiates a Glossy flood. All other nodes turn their radios on during every communication slot to relay the data packets. In addition, the sink also delivers the packets to the application. These operations repeat every 6 seconds, as shown in the middle part of Fig. 4.

The five rounds in between include no data slots. As described next, these seemingly redundant rounds are used to possibly receive further stream requests at the host. Should the host not receive new stream requests, rounds without data slots eventually disappear, as illustrated in Sec. 3.3.

3.2 Bootstrapping

Intuition. To reach steady-state conditions, we need to: (*i*) time-synchronize all nodes with the host and inform them of the current round period T, and (*ii*) communicate the current traffic demands to the host. Nodes boot with their radios turned on and use the very first schedule transmission to synchronize initially. Afterward, nodes may use the contention slot to communicate their traffic demands. Nodes can simultaneously access the bus during this slot, so communication may be unreliable when different Glossy floods overlap. We use a simple acknowledgement scheme to confirm that the host successfully received the traffic demands. As these are progressively received at the host, the scheduling of data slots intertwines with newly received stream requests.

Initial synchronization. At $t = 0$ s in Fig. 4, only the host is part of the bus operation; all other nodes have their radios turned on. At $t = 2$ s, the host transmits the schedule for the first time. This includes no data slots: only the schedule itself and the contention slot. Upon receiving the first schedule, Glossy time-synchronizes the nodes with the host, and nodes learn about the round period T. This allows them to start duty cycling their radios and to effectively join the bus operation.

Communicating traffic demands. During the early rounds in Fig. 4, multiple source nodes use the contention slot con-

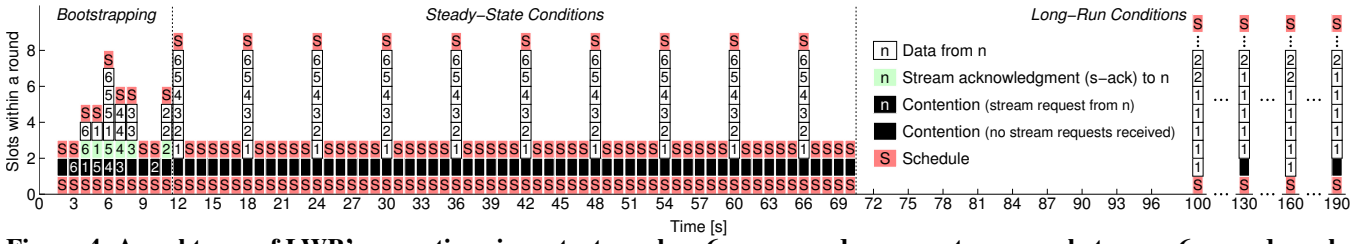


Figure 4. A real trace of LWB's operation since startup when 6 source nodes generate one packet every 6 seconds each. *Nodes start with their radios turned on. Upon receiving the schedule for the first time at $t = 2$ s, nodes time-synchronize with the host and start duty cycling their radios. At $t = 11$ s, the host received all stream requests in the preceding contention slots and starts allocating 6 data slots, one per source, in one round every six. At $t = 70$ s, since no more stream request were recently received, the scheduler extends the round period T from 1 second to 30 seconds to reduce energy costs, and allocates 5 data slots to every source in the following rounds; for the same reason, it allocates a contention slot only every other round.*

currently to transmit their stream requests. In most cases, the host receives one stream request due to capture effects [29]. This happens, for example, at $t = 3$ s in Fig. 4, when the host receives a stream add request from node 6.

Based on this request, the host allocates two additional slots in the next round at $t = 4$ s: one to itself to transmit a *stream acknowledgment* (s-ack), and one to node 6 to transmit the data packet it generated at $t = 0$ s. If no s-ack were received, node 6 would exponentially back off for some rounds before retransmitting the request. This reduces the number of contending requests in subsequent rounds, eventually increasing the chances of successful transmission.

Building up to steady state. At startup, the scheduler sets the round period T to the shortest possible to offer more contention slots, speeding up the initial joining of nodes. Operations similar to the ones above repeat for node 1 at $t = 4$ s (add request) and $t = 5$ s (s-ack and data). At $t = 6$ s, the host allocates one data slot each to nodes 6 and 1 for transmitting their second data packets. It also allocates two data slots to node 5 in response to a stream add request received in the previous round: node 5 has two packets already generated (at $t = 0$ s, 6 s) and not yet transmitted. A similar processing occurs in the following rounds for nodes 4, 3, and 2.

Meanwhile, nodes are kept synchronized by the periodic transmission of schedule packets. At $t = 11$ s, the host received all stream add requests and is thus aware of all traffic demands: the steady-state phase commences. Nevertheless, the host still includes one contention slot in each schedule for possible further requests. If new stream requests arrive later, the processing is exactly as the one described here.

3.3 Long-Run Conditions

Intuition. If steady-state conditions endure for a given time, the application has likely converged to a stable traffic pattern and load. This is the case in many scenarios we target [3, 7, 34, 43], where periodic streams of data are initiated at startup and live on for the entire execution. This means that new stream requests are unlikely to arrive. In such situations, LWB minimizes control overhead by changing the schedule. Specifically, we set the round period such that: (*i*) LWB still provides enough bandwidth, and (*ii*) schedule transmissions occur sufficiently often to keep nodes synchronized.

Reducing overhead. At $t = 70$ s in Fig. 4, the host detects that no new stream requests were recently received. It infers that the traffic demands are stable and increases the round period T from 1 second to 30 seconds. This value is based on the current traffic demands and the scheduling policy described in Sec. 4. As a result, the following rounds occur every 30 seconds starting from $t = 100$ s. Increasing the round period reduces overhead, because it spares communication rounds with no data slots. At each round, the host indeed allocates $T/\text{IPI} = 5$ data slots to every source node, corresponding to the number of data packets generated between consecutive rounds. LWB takes care of buffering these packets at the source nodes until a data slot is available.

3.4 Optimizations

We complement the LWB operation just described with optimizations that further reduce the overhead and improve system responsiveness to changes in the traffic demands.

Schedule transmissions. Fig. 4 already shows that schedules are transmitted twice in a round. In principle, this is not necessary: the schedule transmitted at the beginning of a round would suffice to keep nodes synchronized and instruct them on when to access the bus. However, if the host changes the round period T as a result of computing the schedule for the next round, it would like to promptly communicate the new T to the nodes, so they can immediately switch to the new period (*e.g.*, to save energy if the new round period is larger, as is the case at $t = 70$ s in Fig. 4). We make this possible by transmitting the schedule for the next round already at the end of the current round. This improves responsiveness, because it makes nodes adapt earlier to new round periods.

Scheduling contention slots. Under stable traffic conditions similar to those in Sec. 3.3, new stream requests arrive rarely. Besides increasing the round period T, the host schedules contention slots according to a different period, independent of T. The right part of Fig. 4 indeed already shows that contention slots appear only once every minute. This optimization further reduces the overhead, especially when rounds unfold quickly to satisfy high, but stable, traffic demands.

Piggybacking stream requests. With long round periods, as in the right part of Fig. 4, contention slots occur rarely, and the optimization above makes them occur even more sparingly. Nevertheless, many nodes may compete in the contention slot, and at most one at a time succeeds. These factors increase the latency in communicating changed traffic demands to the host. To ameliorate the problem, we let nodes piggyback stream requests on data packets if they are already assigned a data slot. This improves responsiveness,

as gives nodes more chances to send stream requests, and also reduces the pressure on the contention slot.

3.5 Node and Communication Failures

LWB needs to deal with node and communication failures, and nodes that spontaneously disconnect from the network (*e.g.*, because of mobility). Host failures, instead, require special countermeasures, as discussed in Sec. 3.6.

Node failures and disconnections. If a node fails or disconnects from the network, its active streams are eventually reclaimed. LWB uses a simple counter-based scheme to detect such situations at the host. If the host does not receive any packet within a certain number of consecutive rounds from a stream s, it removes s from the set of active streams. The threshold for removal is set as a protocol parameter. As a result, the scheduler stops allocating data slots to stream s, which saves bandwidth and energy.

This policy allows LWB to effectively detect situations where, for example, multiple nodes fail concurrently, as we show in Sec. 9.2. Due to Glossy's high reliability, our simple scheme is quite resilient to false positives: it is very unlikely that Glossy does not deliver data for a number of consecutive rounds while a node is still running and connected.

Communication failures. Lost stream requests are not an issue: our acknowledgment scheme ensures that all requests eventually reach the host. Problems may, however, arise if schedule packets are lost. These are critical to keep nodes synchronized and to instruct them on when to access the bus.

To address this problem, LWB applies two policies. First, a node is allowed to participate in a communication round *only if* it received the schedule for the current round. Otherwise, it turns the radio off and keeps quiet for the remaining part of the round. Second, to compensate for a possibly higher synchronization error after a missed schedule packet, a node increases its guard times and wakes up slightly before the beginning of the next round. Moreover, if a node misses the schedule packet for a given number of consecutive rounds, it continuously listens until it receives again a new schedule. The specific threshold for turning the radio on is a protocol parameter.

Because of Glossy's high reliability, situations like those above happen very rarely, and usually indicate that a node is disconnected from the network or that the host has failed.

3.6 Host Failures

We address host failures by deploying the scheduler on *all* nodes, and by complementing LWB with mechanisms to dynamically enable or disable the scheduler at specific nodes according to a given policy. We describe next a simple failover policy that avoids multiple hosts being active on the same channel (*e.g.*, when networks merge after partition).

Failover policy. Nodes detect a failure of the host based on the *complete* absence of communication within a specified time interval T_{hf}. If neither schedule nor data packets are received within T_{hf}, it is very likely that also other nodes are not receiving schedules and thus that the host has failed.

We hardcode into all LWB nodes a circular ordered list of $\langle channel, host\_id \rangle$ pairs that maps a set of communication channels to an appointed host for each channel. Upon detecting a host failure, a node switches to the channel next in the list. If the node is the appointed host for the channel, it activates the scheduler and starts distributing (empty) schedules. Otherwise, the node turns the radio on and listens; if schedule packets eventually arrive, it joins the LWB running on the new channel. In both cases, if no communication from other nodes is ever heard within another time interval T_{hf}, the node switches to the next channel and the procedure repeats.

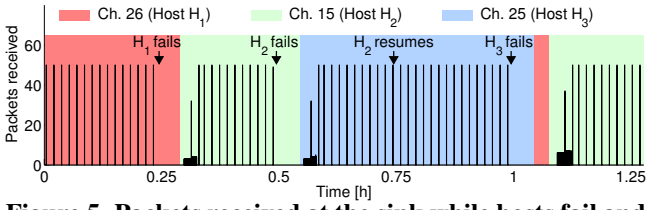


Figure 5. Packets received at the sink while hosts fail and resume. *A few minutes after detecting a host failure, communication resumes on a different channel with a new host.*

Our simple failover policy makes LWB remain functional despite repeated host failures. We note, however, that it is possible that after a network partition several buses operate on different channels and never merge again. More sophisticated failover policies are in our immediate research agenda, possibly based on self-stabilizing leader election [11].

Sample execution. We exemplify the functioning of our policy by inducing host failures in a real-world experiment. We use a multi-hop network of one sink and 50 sources that generate packets with IPI = 1 min. We set $T_{hf} = 2$ min, and use IEEE 802.15.4 channels 26, 15, and 25 with hosts H_1, H_2, and H_3. Initially, nodes use channel 26 and H_1 is the host.

Fig. 5 shows the number of packets received at the sink over time. When host H_1 fails at $t = 0.25$ h, communication successfully resumes after $T_{hf} = 2$ min on the next channel in the list and with H_2 as the new host. All nodes join again, and the source nodes issue stream add requests, considering also packets that were generated while they were disconnected from the bus. The new host eventually activates all previous streams and communication resumes as before the failure of H_1. The same events occur after H_2 fails at $t = 0.5$ h: nodes switch to channel 25 and H_3 is the new host.

When H_2 recovers at $t = 0.75$ h, there is no visible impact on the bus operation. This is because nodes are operating on a different channel: after T_{hf} without receiving any stream request on channel 15, H_2 switches to channel 25 where it joins the ongoing LWB operation with H_3 as the host. Finally, after the failure of H_3 at $t = 1$ h, communication resumes after $2 \times T_{hf} = 4$ min. Host H_1 for channel 26 has indeed not recovered yet, and a second timeout expires before nodes switch to channel 15 where H_2 is the new host.

4 Scheduler

The scheduler running at the host orchestrates communication over the bus by computing the communication schedule. This involves (*i*) determining the round period, and (*ii*) allocating data slots to streams. We describe next a scheduling policy to minimize energy costs for low-power applications that can tolerate end-to-end latencies of a few seconds. We use this policy in Secs. 5–10 to evaluate LWB. Sec. 11 presents alternative scheduling policies that trade smaller end-to-end latencies for slightly higher energy costs.

4.1 Determining the Round Period

Several trade-offs are involved in determining the round period T. It must be set sufficiently small to provide enough bandwidth for all traffic demands. However, the faster the rounds unfold, the higher is the energy overhead for distributing the schedule. We choose one specific design point: minimize the energy overhead under steady traffic conditions while satisfying all traffic demands whenever possible.

In addition, a LWB implementation on real devices imposes three constraints: (*i*) a lower bound $T_{\min}$ ensures that T is longer than the total *duration of a round* T_l (see Fig. 2), the latter being an implementation-dependent constant; (*ii*) the round period T must also not exceed $T_{\max}$, to ensure that nodes update their Glossy synchronization state sufficiently often; (*iii*) platform-dependent restrictions on the packet size also determine an upper bound $d_{\max}$ on the number of data slots that the scheduler can map in a single schedule packet, and thus on the number of data slots it can allocate per round.

Based on the above considerations, the scheduler computes the round period T as follows. To satisfy all traffic demands, it should allocate $R_{\text{tot}} = \sum_{s=1}^{N}(1/\text{IPI}_s)$ data slots per time unit, corresponding to the rate of data slots required by all N existing streams. To minimize the energy overhead for distributing the schedule, the scheduler should use the minimum number of rounds; that is, it should allocate *all* possible $d_{\max}$ data slots every round. The round period T_{opt} that minimizes energy while satisfying all traffic demands is thus:

$$T_{\text{opt}} = \frac{d_{\max}}{R_{\text{tot}}} = \frac{d_{\max}}{\sum_{s=1}^{N}(1/\text{IPI}_s)} \tag{1}$$

Shorter round periods can also satisfy all traffic demands, but entail more rounds and thus higher energy overhead. Longer round periods, instead, cannot satisfy all traffic demands.

Due to constraints (*i*) and (*ii*) above, the scheduler bounds T_{opt} in (1) within $T_{\min}$ and $T_{\max}$ and sets it to the largest previous multiple of 1 second before updating the round period T. If $T_{\text{opt}} < T_{\min}$, the number of available data slots is insufficient for the current traffic demands: the network is *saturated*. When the scheduler detects saturation, it embeds this information into the schedule packet, allowing source nodes to take appropriate actions if needed, such as temporarily storing data in external memory to prevent queue overflows.

We complement the solution above with a simple policy to promptly react to varying traffic demands. If stream requests were recently received (*e.g.*, in the last minute), the scheduler sets T to $T_{\min}$ regardless of the current traffic demands, and allocates a contention slot in every round in anticipation of further stream requests. Such conditions occur, for example, when bootstrapping a network, as shown in Sec. 3.2, or when a subset of nodes wishes to send data at higher rates, as we show in Sec. 7.3.

4.2 Allocating Data Slots to Streams

The scheduler allocates data slots to maximize fairness across all streams according to Jain's fairness index [21], a widely used metric in the literature [40, 46]. Other metrics or algorithms can be applied by modifying the scheduler.

To compute Jain's fairness index, we denote with a_s the number of data slots the scheduler *allocates* to stream s during a round, and with $r_s = T/\text{IPI}_s$ the number of data slots

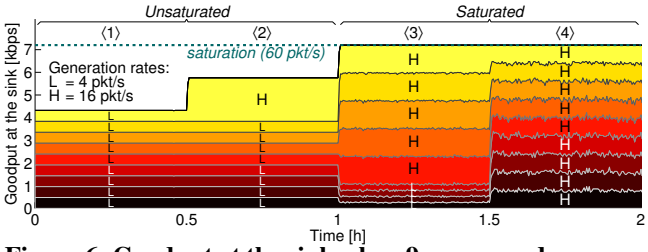


Figure 6. Goodput at the sink when 9 source nodes generate varying amount of traffic. *LWB always allocates slots fairly, based on traffic demands and available bandwidth.*

stream s *demands* every round. The *allocation* to stream s is thus $x_s = \min(a_s/r_s, 1)$ [21]. Using the allocations of all N streams, Jain's fairness index is defined as:

$$f(x) = \frac{\left(\sum_{s=1}^{N} x_s\right)^2}{N \cdot \sum_{s=1}^{N} x_s^2} \tag{2}$$

A fairness index of 1 indicates that the scheduler is equally fair to all streams; smaller values indicate less fairness.

As illustrated next, our slot allocation scheme achieves a fairness index of 1 in the long run, because it is in general not possible to achieve fairness in individual rounds. For instance, a fair allocation in individual rounds may in principle require allocating a non-integer number of slots to streams.

In the following, we use an experiment on a multi-hop network of 10 nodes to exemplify how the scheduler allocates data slots in both the unsaturated and saturated case. One node acts as host and sink. Each of the other 9 nodes generates one stream of 15-byte packets, either at low rate $\text{L} = 4\,\text{pkt/s}$ or at high rate $\text{H} = 16\,\text{pkt/s}$. These generation rates follow patterns of four phases ⟨1⟩–⟨4⟩ in Fig. 6, where different nodes generate data at different rates. In our LWB prototype, we set $T_{\min} = 1\,\text{s}$, $T_{\max} = 30\,\text{s}$, and $d_{\max} = 60$ slots.

Unsaturated network. When the network is not saturated, the scheduler can allocate sufficient data slots to satisfy all traffic demands, that is, $a_s = r_s$ for all streams s. This also implies that the slot allocation is eventually fair across all streams: $f(x) = 1$, because $x_s = 1$ for all streams s.

In the example experiment, the network is not saturated in phases ⟨1⟩ and ⟨2⟩. In phase ⟨1⟩, all 9 streams generate packets at low rate L: using (1), the scheduler computes $T_{\text{opt}}\langle 1\rangle = 1.67\,\text{s} > T_{\min}$. In phase ⟨2⟩, one node generates data at higher rate H, and $T_{\text{opt}}\langle 2\rangle = 1.25\,\text{s}$ is still greater than $T_{\min}$. The scheduler sets $T = T_{\min} = 1\,\text{s}$ in both phases, and allocates data slots as follows.

Phase ⟨1⟩: all streams demand 4 data slots every round and the scheduler satisfies this demand by allocating in total 36 slots, 4 to each stream. Source nodes indeed contribute equally to the total goodput at the sink, as shown in Fig. 6.

Phase ⟨2⟩: one source node increases its data rate to H. The scheduler allocates every round 16 slots to this node and 4 to the other nodes. In total, 48 slots are allocated, and all traffic demands are satisfied.

Saturated network. If the network is saturated, the scheduler sets the round period T to the lower bound $T_{\min}$ but cannot satisfy all traffic demands. This means $a_s < r_s$ for at least one stream s. The scheduler allocates then a_s data slots to each stream s such that the provided bandwidth is maxi-

Term	Description	Value
$T_{\min}$	Minimum round period	1 s
$T_{\max}$	Maximum round period	30 s
$d_{\max}$	Max. data slots per round	60
T_s	Length of a schedule slot	15 ms
T_d	Length of a data slot	10 ms

Table 1. LWB parameters.

Protocol	Code Footprint
LWB	22 kB
CTP+{CSMA, LPL, A-MAC}	{26, 28, 27} kB
Dozer	38 kB
Muster+{CSMA, LPL}	{35, 37} kB
BCP+CSMA	23 kB

Table 2. Code footprints.

Name	Nodes	TX Power	Diameter
TWIST	90	-7 dBm	3 hops
KANSEI	260	-20 dBm	4 hops
CONETIT	26 (5 mobile)	-25 dBm	3 hops
FLOCKLAB	55	0 dBm	5 hops

Table 3. Testbeds.

mized and the allocation is eventually fair across all streams:

$$\text{allocate } a_s \text{ slots such that } \sum_{s=1}^{N} a_s = d_{\max} \text{ and } f(x) = 1 \quad (3)$$

It can be shown that allocating $a_s = T_{\text{opt}}/\text{IPI}_s$ data slots to each stream s is a solution to (3). Intuitively, this entails allocating slots to streams proportionally to their data rates. This allocation is fair because $x_s = a_s/r_s = T_{\text{opt}}/T_{\min}$ is constant across all streams s, and hence $f(x) = 1$ according to (2).

The network in our example experiment is saturated during phases $\langle 3 \rangle$ and $\langle 4 \rangle$ in Fig. 6, because the corresponding T_{opt} are smaller than $T_{\min}$. The scheduler sets $T = T_{\min}$ and allocates data slots to streams as follows.

Phase $\langle 3 \rangle$: 5 streams generate packets with rate H and 4 with rate L, leading to $T_{\text{opt}}\langle 3 \rangle = 0.625$ s. Streams demand altogether 96 slots per second, but the scheduler can allocate at most $d_{\max} = 60$ slots. To achieve fairness, it allocates slots proportionally to their data rates; that is, on average, $0.625 \times 16 = 10$ slots to each of the 5 streams with rate H and $0.625 \times 4 = 2.5$ slots to each of the 4 streams with rate L, against a demand of 16 slots and 4 slots, respectively.

Phase $\langle 4 \rangle$: all streams generate data at rate H, and the network is saturated: $T_{\text{opt}}\langle 4 \rangle = 0.417$ s. The scheduler allocates on average $a_s = 6.67$ slots to every stream, against a demand of $r_s = 16$ slots. The source nodes equally contribute to the goodput at the sink, as shown in Fig. 6.

Worth noticing is that the slot allocation in the saturated case is fair because it applies the same allocation x_s to all streams s: $x_s = 10/16 = 2.5/4$ in phase $\langle 3 \rangle$ and $x_s = 6.67/16$ in phase $\langle 4 \rangle$. Fig. 6 indeed shows that during both these phases nodes with the same rate have the same goodput, and the four nodes with rate L in phase $\langle 3 \rangle$ have together the same goodput as one node with rate H, for $\text{L} = \text{H}/4$.

5 Evaluation Methodology

Before presenting experimental results, we describe the metrics, protocols, and testbeds we use to evaluate LWB.

Metrics. We consider two key performance metrics commonly used for evaluating low-power wireless communication protocols [6, 18, 36, 37]: (*i*) *data yield*, defined as the fraction of application data packets successfully received at the sink(s) over those sent; and (*ii*) *radio duty cycle*, computed as the fraction of time a node keeps the radio on. The former is an indication of the level of service provided to applications in delivering sensed data, whereas the latter provides a measure of a protocol's energy efficiency.

To determine data yield and radio duty cycle, we embed packet sequence numbers and radio timings into data packets. We measure the radio duty cycle in software, using Contiki's power profiler and a similar approach in TinyOS. For each experimental setting and protocol, we compute these metrics based on three independent runs and report per-node or network-wide averages and the 5th and 95th percentiles.

Protocols. We implement our LWB prototype on top of the Contiki operating system, targeting the TelosB platform. We set the LWB configuration parameters as in Table 1. Our choice for T_d and T_s is further discussed in Sec. 11.2. We compare our LWB prototype with seven different combinations of routing and link-layer protocols. Each in its own setting, these protocols represent the current state of the art:

- The Collection Tree Protocol (CTP) [18] is a staple reference for many-to-one scenarios. We run CTP over a non duty-cycled Carrier Sense Multiple Access (CSMA) layer, the Low-Power Listening (LPL) [38] layer, and A-MAC [14]. The CSMA case serves as a baseline for CTP's data yield performance, since it provides the highest network capacity. The LPL setting matches the configuration used by Gnawali et al. [18]. A-MAC is a receiver-initiated link layer shown to outperform LPL when running CTP in specific settings [14].
- Dozer is a state-of-the-art TDMA-based collection protocol for periodic, low-rate many-to-one scenarios [6]. It achieves ultra-low radio duty cycles of 0.07–0.34 % in real deployments [22]. To compare Dozer with LWB, we port the original TinyNode/SX1211 implementation to the TelosB platform and the CC2420 radio. Our results confirm that our port performs comparably to the original implementation under similar testbed settings.
- Muster is one of the few protocols for many-to-many communication tested on real nodes [37]. We run Muster atop LPL and CSMA. The LPL setting matches the configuration used by Mottola et al. [37], and the CSMA case serves as a baseline for Muster's data yield performance.
- The Backpressure Collection Protocol (BCP) represents the state of the art in data collection at a single mobile sink [36]. Results indeed suggest that BCP outperforms recent mobile sink routing protocols in terms of data yield [28, 36, 41]. We run BCP atop CSMA, which matches the configuration used by Moeller et al. [36].

Table 2 lists the code footprints of all protocol configurations we consider in our evaluation, including test applications.

We use an application payload of 15 bytes in all experiments. Unless otherwise stated, we neglect the bootstrapping phase not to bias our results, and start measuring after 10 min with LWB, after 0.5 h with CTP and Muster, after 2 h with Dozer, and after 15 min with BCP, giving each protocol enough time to discover the network and stabilize. We evaluate the bootstrapping performance separately in Sec. 6.

Testbeds. We use four testbeds: TWIST [20], KANSEI [16], the CONET integrated testbed (CONETIT) [1], and an extended instance of FLOCKLAB [33]. All testbeds feature TelosB nodes but differ along several dimensions as shown in Table 3, including number of nodes, density, diameter, and node mobility. KANSEI is the largest testbed we were able to

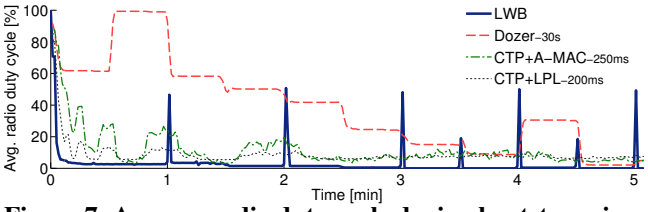


Figure 7. Average radio duty cycle during bootstrapping. *LWB lets nodes join fast initially, and saves energy by quickly adapting the round period as is visible from the peaks.*

gain access to. The 5 mobile nodes in CONETIT are attached to robots, allowing for repeatable mobility patterns.

The network diameter in Table 3 is based on the *physical* topology, matching LWB's perception; the maximum route stretch with other protocols is typically larger. Using a received signal strength indicator (RSSI) scanner, we find that on FLOCKLAB channel 20 is most exposed to WiFi traffic. We use this channel in Sec. 9.1 to assess a protocol's vulnerability to external interference, whereas we use channel 26 in all other experiments to minimize WiFi interference.

6 Bootstrapping

Bootstrapping can be a critical phase in real deployments, because nodes may already spend a considerable amount of energy merely on commencing communication. By examining this facet of LWB, we find that:

Finding 1. *LWB bootstraps quickly and efficiently, while distributing energy costs equally among nodes.*

Scenario. We consider a many-to-one scenario on TWIST, where 89 sources start generating packets with IPI = 1 min. Nodes log every second their current radio duty cycle into the local flash memory, and dump these logs over the serial port after 30 min. We run tests with LWB, Dozer, and CTP over A-MAC and LPL. We use the default 30 s beacon interval in Dozer [6], and set the wake-up intervals of A-MAC and LPL to 250 ms and 200 ms, which provide a good trade-off between data yield and energy consumption in this setting.

Results. We consider the systems fully bootstrapped when all 89 source nodes delivered at least one packet to the sink. We find that LWB and CTP (independently of the link layer) bootstrap in 2 min, whereas Dozer requires more than 18 min. Our results also indicate that LWB bootstraps most energy-efficiently: during the first 30 min of operation, nodes have an average radio on-time of 27 s with LWB against 128 s with Dozer, 130 s with CTP+A-MAC, and 131 s with CTP+LPL.

Fig. 7 shows a fine-grained analysis of energy costs by plotting the average radio duty cycle across all nodes over the first 5 min of operation. The high energy efficiency of LWB is due to the scheduler setting the round period to $T_{\text{min}} = 1$ s on startup, which allows nodes to quickly time-synchronize and start duty cycling their radios after a few seconds; and increasing the round period to $T = 30$ s and allocating fewer contention slots after roughly 2.5 min when all stream add requests are served, which further reduces energy costs. The initial synchronization is instead very expensive in Dozer, because of its fixed beaconing period of 30 s.

CTP's adaptive beaconing [18] ameliorates the problem, but still requires nodes to transmit broadcasts frequently during the first seconds. Broadcasts are costly over link layers like LPL and A-MAC, as visible from the peaks with increasing period in Fig. 7. In Dozer, the synchronization between parent and children in the tree compounds the problem, because nodes need to keep the radio on for a full beacon period to discover their neighbors. This scanning phase is visible in the step-wise pattern in Fig. 7, particularly between 30 s and 1 min where the average radio duty cycle is 100 %.

Finally, we find that Dozer and CTP distribute the energy load unevenly among nodes. After 30 min, the difference between the maximum and minimum radio on-time of a node is 236 s with Dozer, 192 s with CTP, and less than 27 s with LWB. This may later cause a network partition due to faster battery depletion at nodes nearby the sink [39]. The absence of a routing hierarchy makes LWB immune to this problem.

7 Many-to-One Communication

In this section, we investigate the performance of LWB in many-to-one scenarios under varying traffic loads, which represent a significant fraction of existing low-power wireless applications [3, 31, 34, 43]. Our results indicate that:

Finding 2. *In many-to-one scenarios, LWB operates reliably and efficiently under a wide range of traffic loads, and promptly adapts when the traffic demands change over time.*

A key aspect to understand the following performance results is that radio activity in LWB is exclusively driven by the global communication schedule. This spares nodes from periodically waking up merely for probing the channel, as in contention-based protocols like LPL or A-MAC. Placing this observation in perspective: with the energy budget A-MAC requires solely for probing the channel every 500 ms, LWB supports 50 streams with IPI = 1 min to a single sink.

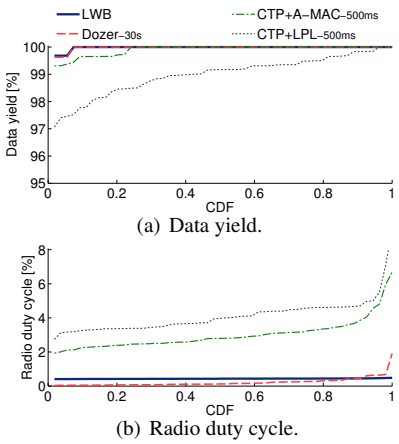
7.1 Light Traffic

We first look at a common scenario for low-power wireless sensor networks: periodic low-rate data collection. This is typical of environmental monitoring, where high data yield and energy efficiency are paramount [34, 43].

Scenario. We use FLOCKLAB and let 54 sources generate packets with IPI = 2 min for 4 h. We compare LWB with Dozer and CTP over A-MAC and LPL. Given the light traffic load and stable network conditions in this scenario, we use Dozer's default 30 s beacon interval, and set the wake-up interval in A-MAC and LPL to 500 ms.

Results. Fig. 8 plots CDFs of per-node data yield and per-node radio duty cycle. Fig. 8(a) shows that all protocols but CTP+LPL deliver more than 99 % of the packets from all sources. In particular, LWB and Dozer exhibit a very high and almost identical average data yield of 99.98 %. Because of their synchronized operation, these protocols perform at their best under stable network conditions, and better than contention-based protocols like A-MAC and LPL.

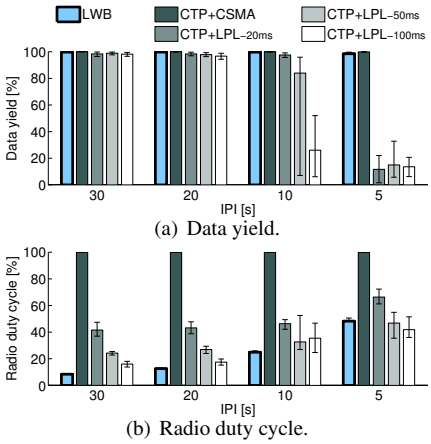
Fig. 8(b) indeed shows that LWB and Dozer achieve low average radio duty cycles of 0.43 % and 0.23 %, respectively. LWB's efficiency is due to the little control overhead to distribute schedules and allocate contention slots, which accounts for only 0.05 % of a node's radio duty cycle. Moreover, we again observe that tree-based protocols like Dozer and CTP bias the routing load towards the sink. For example, radio duty cycles with Dozer range between 0.04 %



(a) Data yield.

(b) Radio duty cycle.

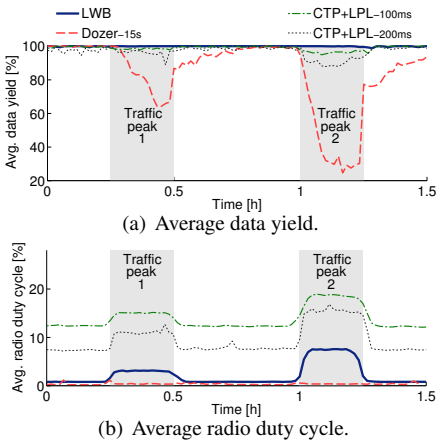
Figure 8. Per-node performance at light traffic. *Synchronized protocols outperform contention-based protocols at stable network conditions.*



(a) Data yield.

(b) Radio duty cycle.

Figure 9. Performance at heavy traffic from 259 nodes. *LWB consistently provides a higher network capacity and is more energy-efficient than CTP+LPL.*



(a) Average data yield.

(b) Average radio duty cycle.

Figure 10. Performance as the traffic demands change. *LWB balances energy costs and network capacity by adapting round period and data slot allocation.*

and 1.91 %, whereas LWB distributes the energy load more evenly, achieving per-node radio duty cycles of 0.41–0.48 %.

7.2 Heavy Traffic

We next consider hundreds of nodes generating relatively heavy network traffic. Applications such as data center monitoring exhibit similar aggregate traffic loads [31].

Scenario. We use the 260 nodes available on KANSEI. All nodes but the sink act as sources and generate packets with the same fixed IPI for 4 h. We test four different IPIs: 30 s, 20 s, 10 s, and 5 s, and compare LWB with CTP over CSMA and LPL, using wake-up intervals of 100 ms, 50 ms, and 20 ms for the latter.[1] We exclude Dozer, as it is not designed for such heavy traffic: constraints on the maximum packet queue size and an increased risk of collisions [6] cause significant packet loss at higher traffic loads (see also Sec. 7.3).

Results. Fig. 9 plots data yield and radio duty cycle for different IPIs. Bars show network-wide averages; error bars indicate the 5th and 95th percentiles. We see from Fig. 9(a) that LWB and CTP+CSMA achieve a data yield close to 100 % across all IPIs. LWB delivers a goodput of 6.1 kbps at IPI = 5 s: its synchronized operation provides sufficient bandwidth to cope with these high traffic demands. By contrast, CTP+LPL collapses at IPI = 5 s even with the shortest wake-up interval: the bandwidth provided by LPL is insufficient, leading to congestion and more than 80 % packet loss.

Fig. 9(b) exposes the trade-off between energy costs and network capacity in CTP+LPL. A longer LPL wake-up interval may save energy, but reduces the available bandwidth. In a network of 259 source nodes, LWB constantly provides a higher network capacity and is more energy-efficient than CTP+LPL. This holds particularly for nodes in the vicinity of the sink, which have the highest radio duty cycles with CTP+LPL as they carry the highest loads. Overall, LWB requires only 2.50 ms of radio-on time to deliver a single application packet, whereas CTP+LPL needs *four* times as much.

[1] We omit inconsistent results with CTP+A-MAC. Because the current A-MAC implementation does not support multiple channels when using broadcasts, many probes collide with data packets at heavy and fluctuating traffic, which affects A-MAC's performance significantly in these scenarios.

7.3 Fluctuating Traffic

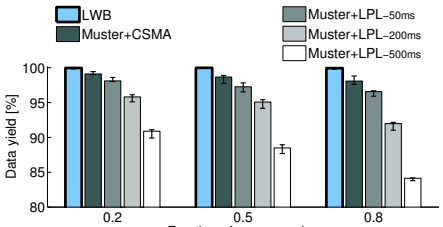
In this experiment, we evaluate the performance of LWB when the traffic demands change over time, which is characteristic of applications that adjust the data rates in response to external stimuli [3].

Scenario. We use 54 source nodes on FLOCKLAB that generate packets with IPI = 60 s for 1.5 h. During two periods of 15 min each, 14 spatially close nodes switch to IPI = 5 s (traffic peak 1) and IPI = 2 s (traffic peak 2), respectively. We compare LWB with Dozer and CTP over LPL[1]. In Dozer, we halve the beacon interval to 15 s and triple the queue size to 60 packets to help performance during traffic peaks. We test 100 ms and 200 ms as LPL wake-up intervals.

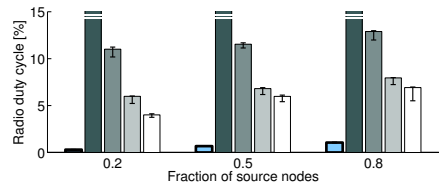
Results. Fig. 10 plots data yield and radio duty cycle over time, averaged over all source nodes. Fig. 10(a) shows that data yield with LWB is always close to 100 %, even during the two traffic peaks: LWB promptly reacts to the changed traffic demands and adapts the round period T. For example, the scheduler sets $T = 30$ s when nodes generate packets with IPI = 60 s, but reduces it to $T = 7$ s during peak 2. Fig. 10(b) shows that the radio duty cycle rises from 0.8 % to 7.8 % during traffic peak 2, but returns to 0.8 % once the peak is over.

Dozer and CTP+LPL lack such adaptability. Dozer's data yield is almost 100 % before peak 1, but drops severely when the traffic load increases, below 30 % during peak 2. Because Dozer sets no limit on packet retransmissions, lost packets are entirely due to queue overflows. Numerous queue overflows occur also with the largest queue size we could fit in RAM (220 packets). With CTP+LPL the drop in data yield is less severe, and depends on the LPL wake-up interval.

More generally, Dozer's beacon interval and LPL's wake-up interval are fixed and set before operation, based on the desired trade-off between energy efficiency and network capacity. For instance, the better data yield with 100 ms wake-up interval comes at the price of higher energy costs, paid also during low-traffic periods when a longer wake-up interval would suffice. Finding suitable parameters for these protocols is indeed a challenge, and additional complexity is often required to adapt them at runtime [49].

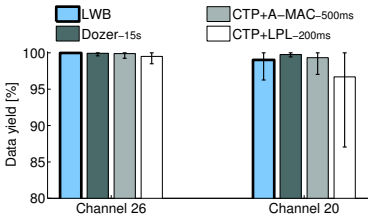


(a) Data yield.

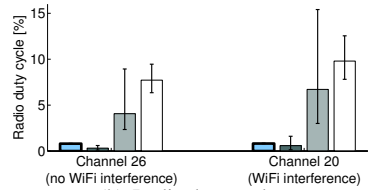


(b) Radio duty cycle.

Figure 11. Performance with 8 sinks and varying fractions of sources. *LWB performs as in single-sink scenarios, since all nodes receive all data.*

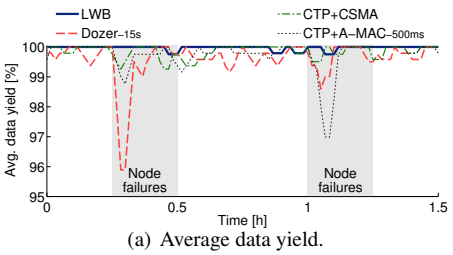


(a) Data yield.

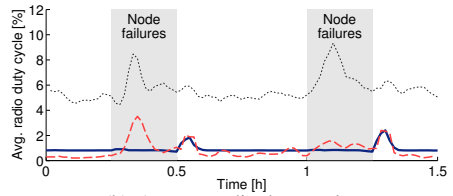


(b) Radio duty cycle.

Figure 12. Performance with and without WiFi interference. *LWB is more resilient to WiFi interference than other protocols.*



(a) Average data yield.



(b) Average radio duty cycle.

Figure 13. Average performance while 8 nodes concurrently fail. *When nodes recover, LWB reduces the round period to make them quickly join the bus operation.*

8 Many-to-Many Communication

We assess LWB's performance in many-to-many scenarios. These arise, for example, in control applications, where multiple sources feed different control loops running at multiple actuators [7]. We observe that:

Finding 3. *LWB efficiently supports many-to-many communication without any changes to the protocol logic.*

Scenario. Out of the 90 nodes available on TWIST, we randomly pick 8 as sinks and a fraction of 0.2, 0.5, or 0.8 of the total as sources. These generate packets with IPI = 1 min for 4 h. We use the same LWB implementation and parameter settings as in Sec. 7. We compare LWB with Muster [37], a state-of-the-art routing protocol for many-to-many communication. We run Muster atop the CSMA and LPL link layers, using three different wake-up intervals for the latter: 500 ms, 200 ms, and 50 ms.

Results. Fig. 11 shows that LWB consistently outperforms Muster in data yield and radio duty cycle. The average data yield across all sinks and sources with LWB, shown in Fig. 11(a), is always above 99.94 %. In contrast, with Muster over CSMA, data yield starts at 99.01 % and drops to 97.98 % as the fraction of source nodes increases. Muster performs route maintenance on a source-sink basis; more sources (or sinks) translate into higher control overhead and hence higher packet loss due to collisions. This behavior is even more evident with LPL, as it provides less bandwidth.

The trends in radio duty cycle, shown in Fig. 11(b), confirm the trade-off between reliability and energy already observed in Sec. 7. With LWB, the average radio duty cycle is 0.31–1.06 %. The highest data yield with Muster+LPL corresponds to an average radio duty cycle of 10.14–12.57 %. Compared with Dozer and CTP, however, Muster distributes the load more evenly, as indicated by the 5th and 95th percentiles. This is due to a load-balancing mechanism added on top of Muster's normal protocol operation [37]. By contrast, LWB achieves network-wide load balancing by design, because all nodes participate in every flood.

9 Topology Changes

Low-power wireless communication protocols must be robust against topology changes caused by external interference [32] and node failures [4]. This sections investigates the resilience of LWB to these changes and reveals that:

Finding 4. *Thanks to the absence of topology-dependent state, LWB operates efficiently also in the presence of topology changes due to external interference and node failures.*

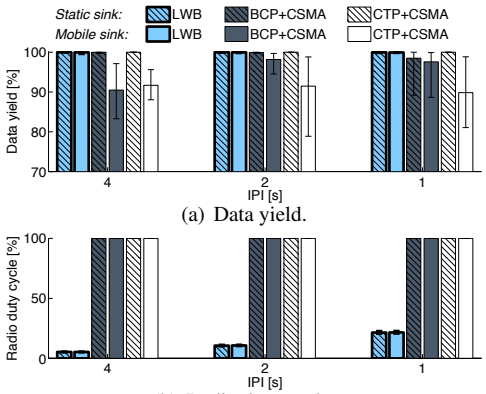
Dozer and CTP, like most existing protocols, rely on periodic broadcasts to keep routing and synchronization state up-to-date, which is extremely costly atop contention-based link layers [39]. The high efficiency of Glossy keeps the energy cost of LWB's little control traffic (*e.g.*, to distribute the schedule and time-synchronize the nodes) to a minimum.

9.1 External Interference

WiFi interference can significantly degrade the performance of low-power wireless protocols [14, 32, 42].

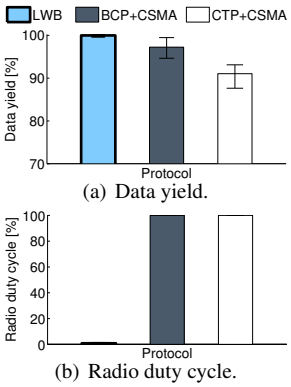
Scenario. We run 3-hour experiments on FLOCKLAB during working hours, letting 54 sources generate packets with IPI = 1 min. We first use channel 26, which is most immune to WiFi [42], and then channel 20, which is most affected by WiFi in FLOCKLAB. We compare LWB with Dozer and CTP over A-MAC and LPL. We set the beacon interval in Dozer to 15 s to improve reactiveness to topology changes. The wake-up intervals of A-MAC and LPL are set to 500 ms and 200 ms, providing a good trade-off between data yield and energy consumption at this traffic load.

Results. Fig. 12 plots data yield and radio duty cycle, with and without WiFi interference. Fig. 12(a) shows that all protocols but CTP+LPL maintain high data yield also with WiFi interference, averaging above 99 %. LWB shows also no noticeable impact on radio duty cycle, shown in Fig. 12(b). In contrast, the radio duty cycles increase considerably with Dozer and CTP; for example, the 95th percentile with Dozer rises from 0.60 % to 1.61 %. These protocols must adapt the routing tree to varying channel conditions, leading to higher radio activity, whereas LWB is immune to the problem.



(a) Data yield.
(b) Radio duty cycle.

Figure 14. Performance with a mobile sink. *LWB performs as if the sink was static.*



(a) Data yield.
(b) Radio duty cycle.

Figure 15. Performance with a mobile sink and 4 mobile source nodes.

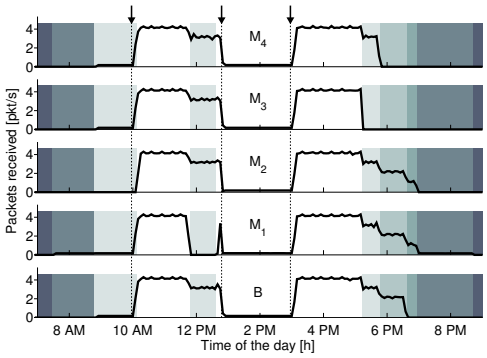


Figure 16. Packets received by 5 mobile nodes in a real-world trial. Arrows indicate when the button is pressed on B; areas are lighter when more mobile nodes are active.

9.2 Node Failures

Real deployments must deal with temporary node disconnections and persistent outages [4]. In the following, we evaluate how effectively LWB adapts to these situations.

Scenario. We run experiments on FLOCKLAB for 1.5 h, letting 54 source nodes generate packets with IPI = 1 min. We adopt a similar scenario as Gnawali et al. [18]: after 15 min, we turn off 8 nodes in the vicinity of the sink. We turn them on again after 15 min, and repeat the off-on pattern after 30 min. We compare LWB with Dozer and CTP over CSMA and A-MAC. Because the traffic load is the same as in the previous experiment, we set again Dozer's beacon interval to 15 s and A-MAC's wake-up interval to 500 ms.

Results. Fig. 13 plots data yield and radio duty cycle over time, averaged over all functional nodes. In the first 15 min, all protocols deliver more than 99 % of the packets and have a stable radio duty cycle. When 8 nodes are turned off, we observe no noticeable change in LWB's data yield, shown in Fig. 13(a), since its route-free operation renders state reconfigurations unnecessary. The only effect is that the host realizes that it receives no packets from streams generated by the 8 failed nodes and eventually removes these streams.

The removal of inactive streams reduces the number of allocated data slots, slightly improving the average radio duty cycle, from 0.83 % to 0.72 %. When the failed nodes recover, they turn the radio on to synchronize again, causing the short increases in radio duty cycle in Fig. 13(b). As soon as the scheduler receives the first stream request, it reduces the round period to $T_{\min} = 1$ s to make nodes join faster. In less than 20 s all 8 nodes are again part of the bus, and after 1 min the scheduler sets the round period back to 30 s.

By contrast, Dozer's average data yield drops below 96 % when nodes are removed, and a slight dip is also visible with CTP. Due to the tree topology in these protocols, the failing nodes close to the sink forward most packets. After a failure, both protocols update the routing tree, which generates more control traffic and thus higher radio activity, as visible from the increase in radio duty cycle. This process is also prone to temporary inconsistencies such as routing loops. These factors all concur to packet losses. With no routing state to update, LWB keeps delivering packets reliably without an increases in energy costs, as long as the network is connected.

10 Mobility

We evaluate LWB's performance in the presence of mobile nodes, a scenario that proved challenging [13, 26, 28]. At the same time, emerging real-world applications increasingly rely on the ability to attach sensor nodes to mobile entities [5, 9, 15]. Our experimental results indicate that:

Finding 5. *LWB supports mobile nodes acting as sinks, sources, or both without any changes to the protocol logic and performance loss compared with the static network case.*

10.1 Mobile Sink

We first consider data delivery to a mobile sink, a setting frequently found in participatory sensing applications [5].

Scenario. We program one robot in CONETIT to move at a constant speed of 1 m/s (approximately human walking speed) along a predefined zigzag trajectory that starts at one corner of the testbed area and ends at the opposite corner. Every run lasts 30 min. During the movement, the node attached to the robot is within the neighborhood of any other static node at least once. The other 4 robots remain at their default locations and act together with the 21 static nodes as sources, using an IPI of 4 s, 2 s, or 1 s in different runs.

We compare LWB with BCP+CSMA and CTP+CSMA, using default parameter settings. CTP is not designed for mobile scenarios, but we consider it nevertheless to understand how the state of the art for static networks performs when the sink is mobile. We use no duty-cycled link layer: most existing protocols supporting mobile nodes do not target energy efficiency [36, 28], and would possibly require modifications to existing link layers to do so [36]. We also perform several runs with a static sink to obtain a baseline.

Results. Fig. 14 plots the results for different IPIs. Fig. 14(a) shows that the same LWB prototype used so far achieves an average data yield above 99.94 % also when the sink moves. This is because LWB keeps no topology-dependent state: as long as the network is connected, LWB is oblivious to topology changes. In contrast, BCP and CTP deliver fewer packets when the sink moves: they need to constantly reconfigure state that depends on a node's current neighbors and connectivity. Although the degree of mobility is fairly limited, this already suffices to affect the performance of both protocols.

A deeper look reveals that with sink mobility CTP deliv-

ers consistently around 90 % of the packets almost regardless of data rates. Instead, BCP's data yield peaks at 98 % with IPI = 2 s. We conjecture that the optimal parameter setting in BCP is sensitive to traffic load. In contrast, LWB is equally efficient in both static and mobile networks with the same configuration, also in terms of radio duty cycle: Fig. 14(b) shows that LWB's performance in a mobile setting is similar to that in static networks regardless of the traffic load.

10.2 Mobile Sources and Mobile Sink

We look at a typical setting where mobile sources deliver data to a mobile sink via a stationary infrastructure [24].

Scenario. Four robots on CONETIT act as mobile sources, generating packets with IPI = 1 s; one robot acts as the mobile sink. The robot trajectories and experiment duration are as in Sec. 10.1. The remaining 21 static nodes generate no packets and form a stationary relay backbone. We compare LWB with BCP+CSMA and CTP+CSMA. Although similar solutions are employed in settings akin to ours [12], these protocols are not expressly designed for mobile sources. Unfortunately, we could not gain access to a reliable implementation of an alternative baseline conceived for such scenarios.

Results. Fig. 15 depicts the results. Overall, the performance is consistent with the mobile sink case discussed above. To leverage mobile sources through a static infrastructure, LWB requires no changes to the protocol logic. Specifically, LWB achieves an average performance of 99.98 % in data yield and 0.84 % in radio duty cycle. The latter figure is lower than in Sec. 10.1, because now only four nodes generate data.

10.3 Real-World Trial

Finally, we run a week-long experiment at ETH to study LWB in a longer-term setting involving many-to-many and one-to-many traffic, changes in traffic demands and active nodes, and mobile nodes acting as sources and sinks. Such setting would currently require two network-layer protocols (*e.g.*, Muster and Trickle [30]) atop a link-layer protocol. LWB provides all required features in a single protocol logic.

Scenario. We use 5 battery-powered nodes carried by people on 7 consecutive days during working hours. People roam around FLOCKLAB, which acts as a static infrastructure. People carrying the nodes entail less structured movement than in the previous experiments, mimicking mobility of real-world applications [9]. All mobile nodes act as both sources and sinks, generating packets with IPI = 5 min. All nodes in FLOCKLAB also generate packets at the same rate.

To induce changes in the traffic demands and set of active nodes, people switch their node off when they leave (*e.g.*, after work). They switch their node on again when they come back, eventually reconnecting the node to the bus. One mobile node, named B, plays a special role: the person can press its user button to trigger a second high-rate stream at IPI = 1 s. When the other 4 mobile nodes M_1–M_4 recognize this, they generate such high-rate stream as well. When the person presses the button again, B cancels the high-rate stream and so do M_1–M_4.

Results. Fig. 16 shows a 14 h-excerpt of our measurements throughout the week. At about 10 AM, node B triggers the high-rate stream. All mobile nodes are running at this time besides M_2, which is off the very moment the traffic peak begins. It also starts generating high-rate packets as soon as it becomes active. This corresponds to the reception of slightly more than 4 pkt/s, one from each of the other 4 mobile nodes plus the low-rate traffic, as shown in Fig. 16. As a consequence of the traffic increase, the scheduler reduces the round period T from $T_{\max} = 30$ s to 11 s. The radio duty cycle accordingly rises from 0.18 % to about 3.78 %.

At around 12 PM, node M_1 is turned off. As a result, the throughput at the other mobile nodes lowers to about 3 pkt/s and their radio duty cycle decreases correspondingly: the host detects that M_1 disconnected and then reclaims its active streams. Close to 1 PM, node M_1 restarts, but right after that node B removes all high-rate streams. This manifests in the short-lived peak shown in Fig. 16 right before 1 PM. Nodes only generate packets at low rate afterwards, and the radio duty cycle of all mobile nodes drops again to 0.18 %.

At around 3 PM, node B triggers again the high-rate streams. Both throughput and radio duty cycle increase similarly as before at all mobile nodes. After 5 PM, people start leaving: node M_3 is the first to be switched off, followed by M_4, B, M_2, and finally M_1 right before 9 PM. As a result, LWB progressively adapts its operation, and both throughput and radio duty cycle decrease in a step-wise fashion.

11 Discussion

This section illustrates LWB's scalability as the number of streams increases, the impact of the network diameter on LWB's efficiency and a few protocol parameters, and alternative scheduling policies to reduce end-to-end latency.

11.1 Scalability Properties

Memory and computation time. The number of streams determines the computation and memory overhead at the host. The worst-case computation time in our experiments is 49 ms with 259 streams (see Sec. 7.2). Memory scales linearly with the number of streams; our LWB prototype uses 13 bytes per stream. Nevertheless, memory and computation costs are paid only at the current host, and proved to be affordable with several hundreds of streams on TelosB nodes.

Bandwidth. Bandwidth provisioning also scales linearly with the number of active streams. Depending on their number and IPIs, different solutions may perform better. In a sense, we hit this point in the KANSEI experiments in Sec. 7.2, where CTP+CSMA slightly outperforms LWB in data yield at IPI = 5 s. This, however, comes at the price of 100 % radio duty cycle: a possible, yet rarely affordable design choice in real-world applications.

11.2 Impact of Network Diameter

The time taken for a Glossy flood to cover the entire network depends on the network diameter [17]. Because LWB uses only Glossy floods for communication, its efficiency decreases in deep networks that span several tens of hops, and other approaches may perform better [23].

In particular, the length of data (T_d) and schedule (T_s) slots must be sufficient for a Glossy flood to cover the entire network. Our setting in the evaluation is sufficient for networks whose physical topology spans at most 7 hops [17]. However, networks may be longer and it may be difficult to determine in advance the network diameter. In these situations, LWB users need to conservatively increase T_d and T_s.

Protocol	Data Yield	Radio Duty Cycle	Latency
LWB	99.98 %	1.40 %	11.13 s
LWB-low-latency	99.83 %	1.44 %	1.19 s
LWB-fixed-period	99.99 %	1.94 %	1.23 s
Dozer-30 s	98.42 %	0.19 %	31.82 s
CTP+A-MAC-500 ms	99.80 %	4.16 %	1.73 s
CTP+LPL-200 ms	98.97 %	6.99 %	0.42 s

Table 4. Average performance of three LWB scheduling policies versus Dozer and CTP over A-MAC and LPL.

We study how this affects LWB's performance through 3-hour experiments on FLOCKLAB with 54 sources that generate packets with IPI = 1 min. Besides the default parameter setting, we test a configuration called *LWB-long-slots* that doubles the values for T_d and T_s to support network diameters of up to 14 hops. We find that the average radio duty cycle increases only by 0.02 %: using Glossy, nodes typically turn off their radios before the end of a slot already with the default parameter setting. Data yield is marginally affected: LWB-long-slots delivers 99.98 % of data against 99.97 %.

Longer slots, however, translate into fewer available slots per round, and thus into an overall decrease in bandwidth. For example, based on (1), we conclude that the default parameter setting would support $N = 300$ streams generating packets with IPI = 5 s or higher without saturating the network, whereas LWB-long-slots would sustain at most IPI = 10 s from the same number of streams. In the applications we target, however, this bandwidth still largely suffices.

11.3 Alternative Scheduling Policies

The scheduling policy in Sec. 4 aims at energy savings while still being responsive to changes in the network. To do so, it trades packet latency for energy efficiency. Although this choice seems appropriate for the applications we target, others may afford to spend some energy for smaller latencies.

We provide two simple alternative scheduling policies that aim primarily at minimizing packet latency. The first policy, *LWB-low-latency*, adapts the round period T such that the next round occurs immediately after the generation of new packets. The second policy, *LWB-fixed-period*, fixes $T = T_{\min}$. We assess their performance based on 3 h-experiments on TWIST, where all nodes but a sink generate data at IPI = 1 min; nodes use transmit power -15 dBm. In addition to the usual performance metrics, we measure end-to-end latency by timestamping packets at the source. We compare LWB with Dozer and CTP over A-MAC and LPL.

Table 4 shows the results. We see that the two alternative policies achieve average packet latencies in the order of 1 s, similar to CTP+A-MAC and CTP+LPL. This comes at a marginal increase in energy costs for LWB-low-latency, whereas LWB-fixed-period shows a larger increase due to the overhead for distributing the schedule every $T_{\min} = 1$ s.

Worth noticing is that in LWB the logic to trade performance requirements resides at a single place, the scheduler, whereas most other protocols may require non-trivial modifications in various places. As a result, users can easily implement custom LWB schedulers using well-defined interfaces.

12 Related Work

Flooding has long been considered too expensive for regular communication in low-power wireless networks. Nevertheless, a few protocols exploit the robustness of flooding for routing while reducing collisions and energy costs, mostly by adding random delays before or by completely suppressing retransmissions [35, 47, 48]. Different from LWB, these protocols keep substantial topology-dependent state, which increases their control overhead and sensitivity to link changes.

Completely contrary to LWB's flooding-based approach, the Broadcast-Free Collection Protocol (BFC) [39] avoids costly broadcasts in the presence of duty-cycled link layers altogether. Targeting low-rate data collection at a single sink, BFC achieves significant energy savings in comparison with CTP even under poor connectivity conditions, which comes at the price of higher delays when forming the collection tree initially. By contrast, LWB is applicable to a wider range of scenarios and bootstraps as fast as CTP.

At the network layer, routing protocols construct multi-hop paths based on some cost metric [2]. Efficient solutions exist that tackle the single-sink [18], multi-sink [37], mobile sink [28, 36], and mobile sources [19] case. In addition, Trickle-based protocols provide reliable network-wide data dissemination [30]. At the link layer, the many protocols available can be divided into contention-based and TDMA-based [27]. The former, sender-initiated [38] or receiver-initiated [14], better tolerate topology changes, whereas the latter enable higher energy savings. LWB instead replaces the standard network stack with a single-layer solution. Our experimental results demonstrate that LWB supports efficient and reliable many-to-one, one-to-many, and many-to-many traffic in both static and mobile scenarios.

LWB's design is inspired by prior work on fieldbus-based communication protocols [25]. Intended for distributed real-time control applications, these protocols primarily focus on providing predictable transmissions and guaranteed timeliness. Different from these protocols, LWB must cope with the unreliability of low-power wireless communications and the resource limitations of the employed devices, particularly in terms of bandwidth, energy, and memory.

13 Conclusions

Many emerging low-power wireless applications feature multiple traffic patterns and mobile nodes in addition to static ones, but existing communication protocols support only individual traffic patterns in dedicated network settings. LWB overcomes this limitation by using exclusively Glossy floods for communication, thereby making all nodes in the network potential receivers of all data. As a result, LWB inherently supports one-to-many, many-to-one, and many-to-many traffic. Since LWB also keeps no topology-dependent state, it is more resilient to external interference and node failures than prior approaches, and seamlessly caters for node mobility without any performance loss. Our experimental results confirm LWB's versatility and superior performance across a variety of scenarios. LWB thus provides a unified solution for a broad spectrum of applications, ranging from traditional data collection to emerging control and mobile scenarios.

Acknowledgments. We thank Olaf Landsiedel, Olga Saukh, and Thiemo Voigt for feedback on early versions of this paper, Roman Lim for sharing his expertise in Dozer, Alberto De San Bernabé Clemente for help in using CONETIT, and

the anonymous reviewers for helpful comments. This work was supported by Nano-Tera.ch, NCCR-MICS under SNSF grant #5005-67322, the Cooperating Objects Network of Excellence under contract #EU-FP7-2007-2-224053, and programme IDEAS-ERC, Project EU-227977-SMScom.

14 References

[1] CONET integrated testbed. https://conet.us.es/cms/.

[2] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Elsevier Ad Hoc Networks*, 3(3), 2005.

[3] A. Arora et al. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Elsevier Computer Networks*, 46(5), 2004.

[4] J. Beutel et al. PermaDAQ: A scientific instrument for precision sensing and data recovery under extreme conditions. In *8th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '09)*.

[5] J. Burke et al. Participatory sensing. In *1st Workshop on World-Sensor-Web (WSW '06)*.

[6] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low power data gathering in sensor networks. In *6th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '07)*.

[7] M. Ceriotti et al. Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels. In *10th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '11)*.

[8] M. Ceriotti et al. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *8th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '09)*.

[9] O. Chipara et al. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In *8th ACM Conf. on Embedded Networked Sensor Systems (SenSys '10)*.

[10] J. Choi, M. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In *7th ACM Conf. on Embedded Networked Sensor Systems (SenSys '09)*.

[11] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. *IEEE Trans. Parallel Distrib. Syst.*, 8(4), 1997.

[12] A. Dunkels et al. The announcement layer: Beacon coordination for the sensornet stack. In *8th European Conf. on Wireless Sensor Networks (EWSN '11)*.

[13] P. Dutta and D. Culler. Mobility changes everything in low-power wireless sensornets. In *12th USENIX Workshop on Hot Topics in Operating Systems (HotOS XII)*, 2009.

[14] P. Dutta et al. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *8th ACM Conf. on Embedded Networked Sensor Systems (SenSys '10)*.

[15] V. Dyo et al. Evolution and sustainability of a wildlife monitoring sensor network. In *8th ACM Conf. on Embedded Networked Sensor Systems (SenSys '10)*.

[16] E. Ertin et al. Kansei: A testbed for sensing at scale. In *5th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '06)*.

[17] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *10th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '11)*.

[18] O. Gnawali et al. Collection tree protocol. In *7th ACM Conf. on Embedded Networked Sensor Systems (SenSys '09)*.

[19] A. Gonga, O. Landsiedel, and M. Johansson. MobiSense: Power-efficient micro-mobility in wireless sensor networks. In *7th IEEE Intl. Conf. on Distributed Computing in Sensor Systems (DCOSS '11)*.

[20] V. Handziski et al. TWIST: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *2nd ACM Intl. Workshop on Multi-hop Ad Hoc Networks (REALMAN '06)*.

[21] R. Jain, D.-M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report 301, DEC, 1984.

[22] M. Keller et al. Comparative performance analysis of the PermaDozer protocol in diverse deployments. In *6th Intl. Workshop on Practical Issues in Building Sensor Network Applications (SenseApp '11)*.

[23] S. Kim et al. Health monitoring of civil infrastructures using wireless sensor networks. In *6th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '07)*.

[24] J. Ko, C. Lu, M. Srivastava, J. Stankovic., A. Terzis, and M. Welsh. Wireless sensor networks for healthcare. *Proc. IEEE*, 2010.

[25] H. Kopetz and G. Grünsteidl. TTP – a time-triggered protocol for fault-tolerant real-time systems. In *23rd Intl. Symp. on Fault-Tolerant Computing (FTCS-23)*, 1993.

[26] B. Kusy et al. Predictive QoS routing to mobile sinks in wireless sensor networks. In *8th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '09)*.

[27] K. Langendoen. Medium access control in wireless sensor networks. In *Medium Access Control in Wireless Networks*. Nova Science Publishers, 2008.

[28] J. W. Lee, B. Kusy, T. Azim, B. Shihada, and P. Levis. Whirlpool routing for mobility. In *11th ACM Intl. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc '10)*.

[29] K. Leentvaar and J. Flint. The capture effect in FM receivers. *IEEE Trans. Commun.*, 24(5), 1976.

[30] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *1st USENIX Symp. on Networked Systems Design and Implementation (NSDI '04)*.

[31] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. RACNet: A high-fidelity data center sensing network. In *7th ACM Conf. on Embedded Networked Sensor Systems (SenSys '09)*.

[32] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis. Surviving Wi-Fi interference in low power ZigBee networks. In *8th ACM Conf. on Embedded Networked Sensor Systems (SenSys '10)*.

[33] R. Lim et al. Demo abstract: Distributed and synchronized measurements with FlockLab. In *10th ACM Conf. on Embedded Networked Sensor Systems (SenSys '12)*.

[34] A. Mainwaring et al. Wireless sensor networks for habitat monitoring. In *1st ACM Intl. Workshop on Wireless Sensor Networks and Applications (WSNA '02)*.

[35] M. Maroti. Directed flood-routing framework for wireless sensor networks. In *5th ACM/IFIP/USENIX Intl. Middleware Conf.*, 2004.

[36] S. Moeller et al. Routing without routes: The backpressure collection protocol. In *9th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '10)*.

[37] L. Mottola and G. P. Picco. MUSTER: Adaptive energy-aware multi-sink routing in wireless sensor networks. *IEEE Trans. Mobile Comput.*, 10(12), 2011.

[38] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys '04)*.

[39] D. Puccinelli, M. Zuniga, S. Giordano, and P. Marron. Broadcast-free collection protocol. In *10th ACM Conf. on Embedded Networked Sensor Systems (SenSys '12)*.

[40] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: A hybrid MAC for wireless sensor networks. In *3rd ACM Conf. on Embedded Networked Sensor Systems (SenSys '05)*.

[41] T. Schoellhammer, B. Greenstein, and D. Estrin. Hyper: A routing protocol to support mobile users of sensor networks. Technical report, UCLA, 2006.

[42] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. An empirical study of low-power wireless. *ACM Trans. on Sens. Netw.*, 6(2), 2010.

[43] G. Tolle et al. A macroscope in the redwoods. In *3rd ACM Conf. on Embedded Networked Sensor Systems (SenSys '05)*.

[44] Z. Vincze, V. Rolland, and A. Vidacs. Deploying multiple sinks in multi-hop wireless sensor networks. In *IEEE Intl. Conf. on Pervasive Services (ICPS '07)*.

[45] M. Wachs et al. Visibility: A new metric for protocol design. In *5th ACM Conf. on Embedded Networked Sensor Systems (SenSys '07)*.

[46] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *18th ACM Symp. on Operating Systems Principles (SOSP '01)*.

[47] F. Ye, G. Zhong, S. Lu, and L. Zhang. A robust data delivery protocol for large scale sensor networks. In *2nd Workshop on Information Processing in Sensor Networks (IPSN '03)*.

[48] Y. Zhang and M. Fromherz. Constrained flooding: A robust and efficient routing framework for wireless sensor networks. In *20th IEEE Intl. Conf. on Advanced Information Networking and Applications (AINA '06)*.

[49] M. Zimmerling et al. pTunes: Runtime parameter adaptation for low-power MAC protocols. In *11th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN '12)*.