A Low-Cost Low-Power LoRa Mesh Network for Large-Scale Environmental Sensing

Dixin Wu *Student Member, IEEE*
Jörg Liebeherr *Fellow, IEEE*

***Abstract*—Sustainability and climate monitoring efforts create a need for long-term in-situ sensing of large geographic areas. However, environmental monitoring in remote areas of developing countries remains impeded by a lack of low-cost, scalable IoT solutions. Whereas IoT systems for in-situ sensing abound, they mostly are either low-cost or suitable for large areas, but not both. In this paper, we present a low-cost low-power network solution for in-situ sensing of areas up to hundreds of square kilometers. Taking advantage of LoRa technology, we develop a self-organizing mesh network that can be scaled to a hundred and more nodes. Scalability is achieved by developing methods that mitigate packet collisions during data collection. We present a protocol, called CottonCandy, with which nodes self-organize in a spanning-tree network topology in a distributed fashion. A power profile on a custom-built circuit board shows that CottonCandy nodes can run thousands of duty cycles on 2 AA batteries, sufficient to operate for years in many applications. Using off-the-shelf components, the cost of a CottonCandy node is less than US-$ 15. Evaluations by simulation show that CottonCandy networks with 100 nodes achieve a packet delivery ratio of >90%. Measurements of an outdoor deployment with 15 nodes corroborate the high packet delivery ratio in a real-life setting.**

***Index Terms*—Low-Power Wide-Area Network (LPWAN), LoRa MAC Layer, LoRa Mesh Network, Network Protocols.**

I. Introduction

In-situ sensing of environmental factors plays an important role in sustainability efforts. For instance, by monitoring soil conditions and evaporation on their fields, farmers can precisely tailor watering schedules, thereby preserving scarce water resources. Making such efforts practicable in rural areas of developing nations requires low-cost communication subsystems that can collect sensor data across large areas with low power requirements. Wireless sensor networks (WSNs) and the Internet-of-Things (IoT) have been embraced by earth and environmental sciences [1]–[3], however, most WSNs and IoT systems are either low-cost or can cover large areas, but not both. Low-cost IoT technologies, such as BLE and Zigbee, have a limited range [4, Table 2], while cellular radios, such as GSM, NB-IOT, and LTE-M, cover large distances, but incur high costs.

The emergence of low-power wide-area networks (LPWANs), such as LoRa and Sigfox, enables remote monitoring for applications where a long transmission range and long battery life are preferred over high throughput [4]. Both operate in unlicensed ISM bands and have communication ranges of several kilometers [5]. Both LoRa and SigFox radios are based on proprietary technologies. Unlike Sigfox, which is only available as a subscription service, LoRa is complemented by an open-source protocol stack, called LoRaWAN, for building LPWANs. With LoRaWAN, sensor nodes send their data directly to a LoRaWAN gateway [6], thus forming a star network topology. As consequences of this, not only must each LoRa radio be within communication range of a gateway, the performance of communication also degrades with increasing numbers of connected radios [7].

The limitations of LoRaWAN can be overcome by organizing LoRa radios in a mesh network [8]–[12]. A mesh network extends communication ranges via multi-hop forwarding, where LoRa nodes participate in relaying data between other LoRa nodes. Since there is no requirement that all nodes are within range of a gateway, mesh networks can cover larger areas. Costs can be further reduced by replacing LoRaWAN gateways by simpler (and less costly) gateway devices that connect to the Internet using standard IoT protocols [13]. Given the low cost of LoRa radios, LoRa mesh networks can significantly reduce the cost of large sensor deployments. However, scaling such networks to hundreds of nodes while maintaining low power consumption is challenging. A major problem of a multi-hop design for LPWANs is the incurred power consumption for packet relaying, as forwarding of packets between nodes drains batteries. Moreover, multi-hop relaying toward the gateway in large networks incurs a high risk of packet collisions. The latter can be mitigated by resorting to time-division multiplexing [10], [14] or assigned frequency bands [15], which, however, limit the scalability of the LPWAN.

In this paper, we present a LoRa multi-hop mesh network solution, called *CottonCandy*, that can be scaled to a hundred and more sensor nodes. CottonCandy is designed for time-driven in-situ sensing applications [16], where nodes report data to a data sink in coordinated duty cycles. Nodes conserve energy by hibernating in a low-power state between duty cycles. Since it is designed to support long-term measurements of environmental data on a large scale, CottonCandy does not attempt reliable data delivery. Rather than recovering packet losses through retransmissions, the media access protocol of CottonCandy seeks to reduce the occurrences of packet losses by adapting to network conditions.

A unique feature of CottonCandy, compared to existing

D. Wu is with the University of Toronto and Spero Analytics. J. Liebeherr is with the Department of Electrical and Computer Engineering, University of Toronto. The work is supported in part by the University of Toronto Centre for Global Engineering (CGEN).

LoRa mesh networks, is that the network is self-organizing, in the sense that the network forms without the need for central coordination or management [17]. Advantages of self-organization are that nodes can be dynamically added to an existing network and that the network can adapt to node failures. CottonCandy nodes form a rooted spanning tree network topology, where the root is a gateway device with access to the Internet. The hierarchical structure of a spanning tree enables simple, recursive protocols for both uplink and downlink communications.

The contributions of this paper are summarized as follows:

1) We devise collision mitigation methods that result in a high delivery ratio on multi-hop routes.
2) We present a self-organizing protocol that enables nodes to synchronize their duty cycles, to dynamically join the spanning tree network, and to participate in a request-based data collection of sensor data.
3) We design a prototype CottonCandy node with a unit cost below US-$ 15. A detailed power profile shows that the prototype can operate for thousands of data cycles on 2 AA batteries. Assuming that environmental data is collected a few times per day, this ensures a battery lifetime in excess of one year.
4) We validate the performance claims of our protocol via simulation and outdoor measurement experiments. Simulations show that a CottonCandy network, even with 100 nodes, achieves a packet delivery ratio above 90%. The findings are confirmed in measurements of a deployed network with 15 nodes.

The proposed self-organizing protocol targets sensing application where data is reported in regular time intervals, ranging from several minutes to several days. The protocol does not provide a solution for time-critical sensing applications, such as detection wildfires, flash floods, or mudslides.

The remainder of this paper is organized as follows. In Sec. II, we discuss related work. In Sec. III, we present methods for collision mitigation in multi-hop LoRa networks. In Sec. IV, we describe the CottonCandy protocol. The CottonCandy prototype is discussed in Sec. V. We present a performance evaluations using simulations in Sec. VI, and outdoor measurement experiments in Sec. VII. We present brief conclusions in Sec. VIII.

II. Background and Related Work

In this section, we discuss aspects of the LoRa physical layer relevant to this paper and related work on media access control (MAC) protocols for WSNs and LoRa-based LPWANs. We refer to available surveys for more detailed discussions [15], [18]–[21].

LoRa PHY: LoRa PHY is a proprietary physical layer developed by Semtech that operates in unlicensed ISM bands, e.g., 915 MHz in North America and 868 MHz in Europe. LoRa uses variants of chirp spread spectrum modulation and forward error correction. The range of a LoRa link is often reported as 5–15 km in open space [22], but can be much shorter in practical settings [23], [24]. The bit rate of LoRa radios depends on spreading factor, channel bandwidth and coding rate, which are all configurable system parameters. A higher spreading factor uses more chirps to modulate a single bit and enhances the reliability of transmission at a lower bit rate. Without a MAC protocol, LoRa communication is subject to collisions when two or more incoming packets overlap in both time and frequency. The number of available LoRa channels depends on the frequency band and the configured channel bandwidth. With a typical channel separation in LoRa of 1.5 times the channel bandwidth [25] and a channel bandwidth of 125 kHz, the North American frequency band for LoRa of 902 MHz–928 MHz supports up to 139 different channels.[1]

Classification of MAC Protocols for WSNs: MAC protocols for WSNs were studied extensively in the early 2000s. While the specifics of those protocols are not applicable to LoRa, proposed classifications of WSN MAC protocols help to gain perspective on LoRa MAC protocols. One classification distinguishes between asynchronous, synchronous, frame-slotted, and multi-channel protocols [28]. Asynchronous MAC protocols establish communication links on demand. Here, receivers wake up periodically and start receiving if they hear a bit signature, e.g., a preamble [29], [30]. Alternatively, receivers can proactively send beacon messages to initiate communication with senders who are waiting to transmit [31], [32]. In contrast, with synchronous MAC protocols nodes wake up and communicate for certain active times, the so-called duty cycles [33]–[35]. This eliminates the need to establish communication on demand, but may incur additional overhead for clock synchronization. Frame-slotted and multi-channel mechanisms are often used for collision mitigation. Most frame-slotted MAC protocols implement time-division multiple access (TDMA) on synchronous WSNs. Alternatively, network throughput can be improved by concurrent transmissions on orthogonal frequency channels using multi-channel MAC protocols. Another useful classification in [16] distinguishes between time-driven WSNs involving duty cycles, event-driven WSNs where sensor nodes alert the sink node upon the occurrence of certain events, and query-driven WSNs, where sensor nodes send their data only upon receiving a query from the sink node. Viewed in terms of these classifications, CottonCandy belongs to the class of time-driven WSNs with a synchronous MAC protocol.

MAC Protocols for LoRa-based LPWANs: In the star topology of LoRaWAN, nodes transmit data to a gateway in an asynchronous, ALOHA-like fashion. Even though LoRaWAN gateways support concurrent reception of multiple packets, the number of nodes that can be supported without performance degradation is limited [36]. A geographically dispersed LPWAN deployment therefore requires multiple LoRaWAN gateways, which, due to the high cost of commercial LoRaWAN gateways defeats the goal of a low-cost network solution. Public LoRaWAN infrastructures are an increasingly popular alternative to private gateways [37], [38], however, they are generally unavailable in remote regions of developing countries. Many recent studies on LoRaWAN improve the

[1] In Europe, the frequency allocation policy in its 868 MHz ISM band depends on the type of device, with 35 available channels for non-specific devices [26]. In India, the available number of channels is 10 [27].

capacity of LoRaWAN gateways via physical-layer coding, e.g., Pyramid [39], CoLoRa [40], mLoRa [41], OCT [42], Choir [43]. Other works replace the ALOHA protocol by CSMA [44], TDMA [45] and lightweight scheduling [46]. However, the challenge of limited scalability remains unsolved due to the inherent drawbacks of a star topology.

The limitations of LoRaWAN have inspired the design of multi-hop LoRa LPWANs. Some studies [12], [47] propose multi-hop forwarding between LoRaWAN gateways so that only one gateway requires Internet access. This, however, still relies on multiple LoRaWAN gateways. Among single-gateway LoRa LPWAN solutions, some protocols involve multiple intermediate nodes in the relaying of a single data packet [9], [48], which harvests the benefit of concurrent transmissions at the cost of higher energy consumption. In [11], nodes are organized in a rooted tree topology, where the root of the tree queries one node at a time. Having no sleep schedule, nodes must listen to queries at all times, which is impractical for battery-powered deployments. Several recent LoRa-based LPWANs embraced TDMA to mitigate collisions. In [14] LoRaWAN is extended with synchronous mesh sub-networks that employ TDMA, where sub-networks are limited to only five nodes each. Another solution [10] centrally computes a global TDMA schedule and routing paths. This requires a new TDMA schedule and new routing tables each time a node is added or removed.

III. Collision Mitigation Methods

The viability of a large-scale multi-hop LoRa mesh network depends largely on the degree to which its protocols can avoid or mitigate packet collisions. The root cause of packet collisions are transmissions with overlapping frequencies and air times. Time-division or frequency-division multiplexing can dramatically reduce collisions, however, at the cost of reduced scalability, limited flexibility, and a need for centralized coordination or configuration. For a self-organizing architecture that seeks to scale to hundreds of nodes, resorting to preassigned or centrally coordinated time slots or frequencies is not an option. Instead, we develop distributed solutions for network formation, channel assignment and access, and data collection that jointly achieve a low rate of packet collisions. In Sec. IV, our solutions will be integrated into a self-organizing network protocol.

We will evaluate the proposed mitigation methods using simulations on ns-3 [49]. We adopt the physical-layer LoRa model from [50], which assumes a log-distance path loss model. Unless specified otherwise, network parameters are as listed in Table I. We simulate networks with 100 nodes that are placed on 10×10 grids with an additional node, the root of the spanning tree, at the center of the grid. By modifying the horizontal and vertical distance between neighboring nodes d, we can vary the node density ρ and the area covered by the network. Table II relates these quantities for a 10×10 grid for the range of values evaluated in this paper.

Throughout the paper we consider data collection in a rooted spanning tree network topology. The root of the tree is a special node that serves the role of a gateway, which offers connectivity to the Internet. (Methods and protocols for connecting to the Internet are briefly discussed in Sec. V, but are not a subject of this paper.) Each node, with exception of the root, is associated with a *parent* node that offers a path to the root. Each node, including the root, may provide a path to the root for other nodes, which are called the *children* of the node. Nodes with a common parent are referred to as *siblings*, and nodes that have a parent-child relationship are referred to as *neighbors*. The *descendants* of a node are all nodes whose path to the root passes through the node.

TABLE I
Physical-layer parameters used in simulations.

PHY Parameters	Value
Loss Exponent (γ)	3.76
Reference Loss (PL_0)	7.7
Reference Distance (d_0)	1 m
LoRa Spreading Factor (SF)	7
LoRa Channel Bandwidth (BW)	125 kHz
LoRa Coding Rate (CR)	4/5
LoRa Sensitivity (SF7 and BW125)	-123 dBm [25]

TABLE II
Network density in terms of coverage area and grid distance.

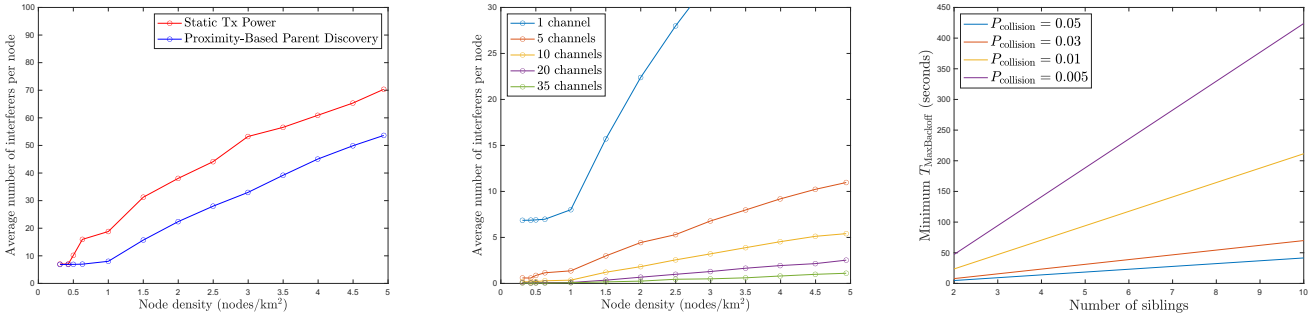
Node density ρ (nodes/km^2)	0.3	0.5	1.0	1.5	2	3	5
Distance d (km)	2	1.57	1.11	0.91	0.79	0.64	0.50
Area (km^2)	324	200	100	67	50	33	20

A. Proximity-Based Parent Discovery

Nodes located inside the transmission range of each other are subject to interference, if they share the same channel or have overlapping frequencies. We refer to a node that experiences interference during a transmission as a *victim* and to nodes that are sources of interference as *interferers*. A node can reduce its likelihood of becoming an interferer to other nodes by reducing its transmission power, thereby effectively decreasing its transmission range. This motivates a network formation strategy where newly joining nodes try to find a parent in their proximity.

A proximity-based parent discovery can be realized by incrementally increasing the transmission power of newly joining nodes. Once a parent node is found, the transmission power used to find the parent is applied in all future uplink transmissions to the parent. The adaptive transmission power is not applied to packets sent to the children of a node, since these packets must be able to reach multiple recipients located at different distances from the sender.

We evaluate the benefits of proximity-based parent discovery in simulations of a grid arrangement of 100 nodes and compare it to a baseline with static transmission power. We assume that all nodes transmit on the same LoRa channel. The baseline method transmits at $P_{\text{tx}} = 17$ dBm, the highest transmission power for continuous operation on SX1276 LoRa transceivers [25], which yields a transmission range of 3.3 km with log-distance path loss and without antenna gains. Nodes using the proximity-based discovery start at $P_{\text{tx}} = 8$ dBm, which results in a transmission range of approximately 1.9 km. The value of P_{tx} is incremented after each failed discovery



(a) Reducing interferers through proximity-based parent discovery.

(b) Reducing interferers through multi-channel communication.

(c) Required backoff range for target collision rates.

Fig. 1. Impact of collision mitigation methods.

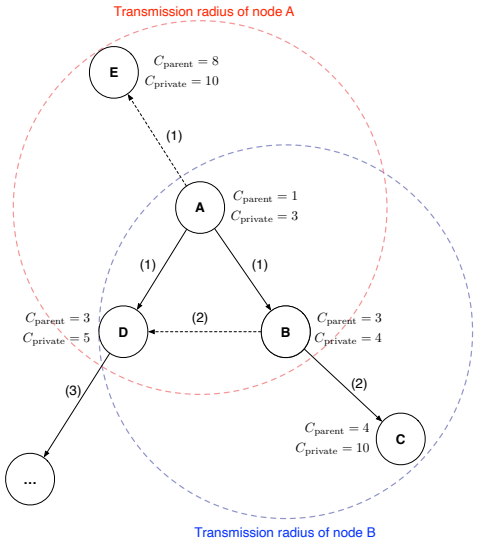


Fig. 2. Channel selection through channel announcements.

until it reaches 17 dBm. Fig. 1a presents the resulting average number of interferers as a function of the density of nodes. We observe that the proximity-based method has fewer interferers than a method with static transmission power. The benefits diminish when the network density is low.

We note that the described proximity-based parent discovery creates deeper spanning trees, which, in turn, increases overall power consumption due to an increased need for relaying packets. In Sec. VI, we evaluate the tradeoff between reduced collisions and increased power requirements.

B. Multi-Channel Communication

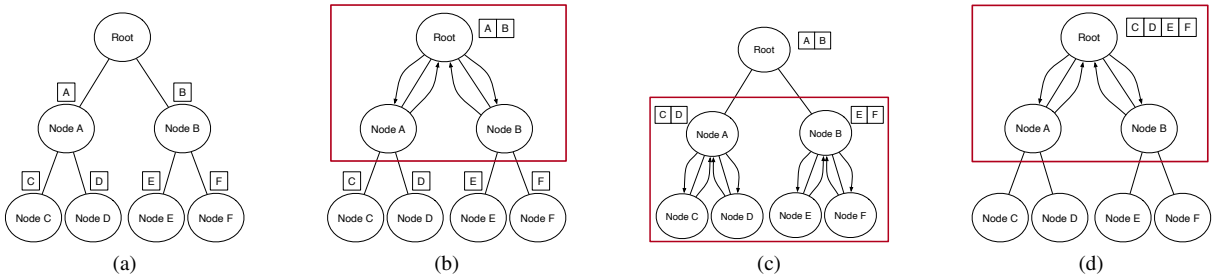
Since packet transmissions on non-overlapping LoRa channels are trivially collision-free, packet collisions can be avoided in most cases by having nodes transmit on different channels. This is exploited in LoRaWAN, where nodes select a random channel before sending data to a gateway, and further refined in [51].

We take advantage of channel diversity by having each node allocate a channel, referred to as its *private channel*, for communication with its children. Every node selects a private channel and notifies its children in the spanning tree about its selection. With one exception (to be explained later), all communications between a parent node and its children occur over the private channel of the parent node. Since most nodes in a spanning tree have the roles of both parent and child, each node performs transmissions on two private channels: (1) the node's private channel for transmissions to its children, and (2) the parent's private channel for transmissions to its parent. This requires that nodes frequently switch between their own private channels and those of their respective parent nodes. Since, in LoRa, configuring the channel frequency takes less than the typical symbol time, the overhead of channel switching in LoRa is negligible.

Nodes select their private channels in a distributed fashion. While a switch to a different private channel can be communicated using the current private channel, bootstrapping private channel information, e.g., when a node joins a network, requires a separate process. We therefore reserve a common *public channel* that is used by all nodes to transmit so-called *channel announcements*, which contain the private channel of a node and that of its parent.

Channel announcements are sent only at the start of a duty cycle and are only sent downstream in the spanning tree. The root node is the first to transmit a channel announcement. Upon receiving a channel announcement from its parent, a node selects its own private channel and broadcasts its own channel announcement. Concurrent channel announcements are avoided by having nodes insert a random backoff delay before transmitting their channel announcements. When selecting a private channel, each node avoids the channels contained in received channel announcements. If that is not possible, it avoids channels that have been announced most recently and were sent most frequently.

Fig. 2 illustrates the transmission of channel announcements between a group of nodes. Solid lines represent channel announcements between neighbors in the spanning tree, and dotted lines indicate channel announcements between non-neighbors. Dashed circles indicate the transmission ranges of channel announcements. The order of transmissions is indicated by labels, as (1), (2), (3). First, a channel announcement sent by node A is received by nodes B, D and E. Since nodes B



(a) (b) (c) (d)

Fig. 3. Recursive data collection. (a) Each node (other than the root) has a data item indicated by a square. (b) The root requests data from nodes A and B. (c) Nodes A and B request data from nodes C–F. (d) In the next round of requests, the root collects the data items of nodes C–F from node A and B.

and D both have node A as parent, they now select their own private channels, and announce it, together with the channel of their parent. Next, nodes B and D send announcements. Due to a random backoff delay, the announcements are likely sent at different times. In the figure, node B sends its announcement before node D. Since node D receives the channel announcement of node B before sending its own announcement, it considers B's channel selection when selecting its own private channel. Lastly, the third announcement is sent by node D.

The multi-channel communication strategy also complements proximity-based parent discovery. Essentially, a new node listens on the public channel to learn about other nodes from their channel announcements, which subsequently become candidates as parent of the new node. The new node then proceeds with proximity-based parent discovery using the private channel of a parent candidate. In this fashion, message exchanges for joining the network are protected against interference by other nodes.

We measure the impact of multi-channel communication using ns-3 simulations of 100-node grid networks. Fig. 1b shows the average number of interferers per node with respect to node density and the number of available channels. Generally, using more channels reduces the number of interferers, especially when the geographical density of nodes is high.

C. Adaptive Random Backoff Delay

Packet collisions due to interference may occur for transmissions of siblings to their parent on the private channel of the parent, for any transmission on the public channel, and for transmissions with a channel conflict. In these situations, we can reduce the likelihood of a collision by adding a random backoff delay before a transmission.

Assuming fixed-sized packets and identical settings for transmissions at each node, given the air time of a LoRa packet T_{air} for a transmission at time t, any transmission attempt on the same channel by an interferer in the time interval $[t - T_{\text{air}}, t + T_{\text{air}}]$ creates a packet collision. One generally assumes that the packets of both victim and interferers are lost in a collision. In practice, one of the packets may survive due to the capture effect [52].

Our objective is to use random backoff delays to achieve a target packet collision rate for transmissions by siblings. Here, collisions occur (1) when siblings send data to their parent on the private channel of the parent, and (2) when siblings send channel announcements (see Subsec. III-B) to their respective children on the public channel. We can ensure that siblings start their backoff timer at the same time by having parents prompt their children to upload or forward data (see Sec. III-D).

Note that an additional source of packet collisions are overlapping transmissions of channel announcements on the public channel by nodes that do not have a common parent. Here, random backoff delays also reduce collisions. However, since accounting for the set of interferers for this collision type requires global knowledge, we do not account for them for determining the range of random backoffs.

Consider two siblings with a common parent that have data for transmission either upstream on the private channel or downstream on the public channel. With a backoff delay drawn from a uniform distribution $U(0, T_{\text{MaxBackoff}})$, the probability of overlapping packets is given by

$$P_{\text{overlap}} = \frac{2T_{\text{air}}}{T_{\text{MaxBackoff}}} .$$

Among n siblings, a node delivers a packet successfully only if the packet does not overlap with transmissions from the other $n-1$ siblings, which results in a probability of a collision given by

$$P_{\text{collision}} = 1 - (1 - P_{\text{overlap}})^{n-1} .$$

We employ an adaptive random backoff delay, where a parent determines the value of $T_{\text{MaxBackoff}}$ such that the collision probability meets a target value. Specifically, for node with $n-1$ siblings and a target of $P_{\text{collision}}$, we set

$$T_{\text{MaxBackoff}} = \frac{2T_{\text{air}}}{1 - \sqrt[n-1]{1 - P_{\text{collision}}}} . \tag{1}$$

By having parent nodes compute $T_{\text{MaxBackoff}}$ for their children and including the value in their channel announcements, nodes do not need to know the number of their siblings. Fig. 1c shows the minimum $T_{\text{MaxBackoff}}$ required to achieve a collision probability of 0.05, 0.03, 0.01 and 0.005 for 64-byte packets. The figure illustrates that, even with a small number of siblings, achieving a collision probability less than 1% requires backoff delays up to several minutes. Since excessive backoff delays cut into the power budget, we set the target $P_{\text{collision}} = 0.05$ and limit the number of children to three for each parent node, thereby keeping $T_{\text{MaxBackoff}}$ below 10 seconds.

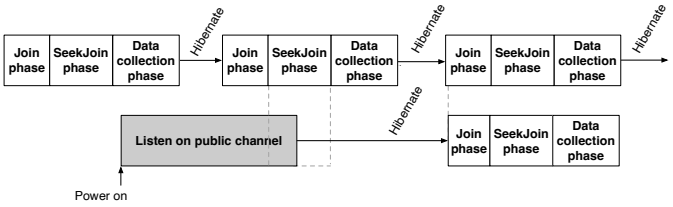


Fig. 4. CottonCandy duty cycles.

D. Request-Based Data Collection

During the collection of sensor data, nodes in the spanning tree forward data items obtained from their own sensors or the sensors of their descendants toward the root. The amount of data items that a node must forward to its parents is proportional to the number of its descendants. Hence, in larger networks data may accumulate at nodes close to the root, which increases the risk of congestion and packet collisions when sibling nodes transmit their data to their parent. Congestion during data collection can be prevented by having parent nodes prompt their children to upload data. Data collection can then proceed in a recursive fashion, where, starting at the root, nodes request data items from their children. Fig. 3 illustrates the data collection process. In the figure, data items of nodes are indicated by squares. Starting at the root, a node fetches data from its children by sending a request. Upon receiving a request, a child sends its data item to its parent and then issues a request for data items to its own children. The transmission of requests to children continues as long as there is data to be collected. The explicit requests for data act as a flow control mechanism, which prevents nodes close to the root from becoming overwhelmed with data items from downstream nodes.

Since siblings receive requests for data simultaneously, nodes can start their backoff timers for uploading data at the same time. The same holds for downstream transmission of channel announcements. Siblings receive the same channel announcement message sent by their parent, and therefore, can start their backoff timer for sending their own channel announcement at the same time.

IV. CottonCandy Protocol

CottonCandy is a distributed protocol that organizes LoRa nodes in a rooted spanning tree with a designated gateway node as the root, and which periodically moves sensor data from all nodes in the spanning tree toward the root. CottonCandy permits dynamic additions of nodes to the network, and it adapts to node departures or failures. By adopting the collision mitigation methods from Sec. III, CottonCandy keeps the rate of packet collisions low.

CottonCandy operates in synchronized duty cycles [53] where all nodes, including the root, start their duty cycles at about the same time. Between duty cycles, nodes hibernate in a low-power state, where they do not receive messages. The start time of the next duty cycle is disseminated by the root in the preceding duty cycle. CottonCandy is intended for applications where the elapsed time between duty cycles

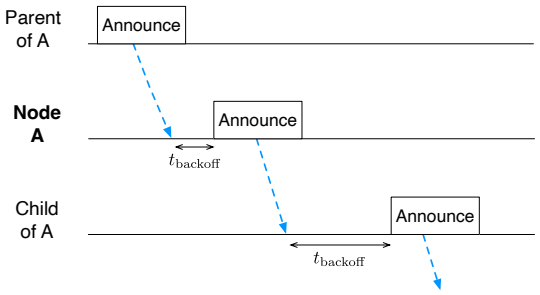


Fig. 5. Transmissions in the *SeekJoin* phase. Transmissions on the public channel are indicated by dashed arrows, with the arrow pointing to the receiver.

ranges between several minutes and several days, that is, nodes spend the majority of time in hibernation.

Duty cycles consist of three phases, which are managed individually by each node. The first two phases, *Join* and *SeekJoin*, are used for network formation and are relatively short with a fixed duration. They are followed by the variable-length *Data collection* phase, which takes up the remainder of the duty cycle. Fig. 4 shows the sequence of phases in a duty cycle. The first row shows the sequence of phases for nodes that are members of the spanning tree. We will refer to these nodes as *in-network nodes*. In the *Join* and *Data collection* phases, nodes communicate with each other only over private channels. In the *SeekJoin* phase, in-network nodes transmit messages to their children on the *public channel*. A newly joining node, shown in the second row of Fig. 4, must wait for messages that in-network nodes send during their *SeekJoin* phase These messages enable a new node to synchronize with the start of the next duty cycle, where it seeks to join the network in the *Join* phase.

Even though the phases of all in-network nodes are roughly synchronized, CottonCandy does not work with absolute time and therefore does not require synchronized clocks. Each node is assumed to have a real-time clock that drifts by no more than a few seconds between duty cycles.

A. Network Formation

Nodes join a CottonCandy network dynamically without central management or configuration. They do so by finding an in-network node and selecting it as parent node in the spanning tree. A node that wants to join a CottonCandy network faces two difficulties. First, since nodes in the spanning tree are in hibernation most of the time, new nodes can only be added during a duty cycle. Thus, joining nodes must know when duty cycles commence. Second, since we avoid the use of the public channel for upstream communication in the spanning tree, newly joining nodes must learn the private channel of a prospective parent node before they can send a message to this node. Both difficulties are solved in the *SeekJoin* phase of a duty cycle, as we will discuss next.

In the *SeekJoin* phase, starting at the root, nodes in the spanning tree send an *Announce* message on the public channel. These messages realize the 'channel announcements' discussed in Sec. III-A, but also have other functions. As shown in Fig. 5, when node A receives an *Announce* message from its parent, it sends an *Announce* message of its own

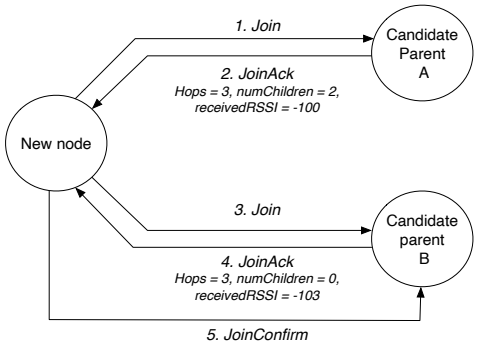


Fig. 6. Message exchange when a new node joins a CottonCandy network. Parent candidates A and B have the same number of hops to the root, but B has fewer children. So, B is selected as parent node.

after a backoff of length t_{backoff}, which is selected uniform random from the interval $[0, T_{\mathrm{MaxBackoff}}]$, where $T_{\mathrm{MaxBackoff}}$ is computed from Eq. (1). The children of node A proceed in the same fashion. To protect against losses of *Announce* messages, the root may send several such messages. After a node has sent the *Announce* message(s), its *SeekJoin* phase ends and the node hibernates until the start of the *Data collection* phase.

An *Announce* message sent by a node carries information on the private channels of the sending node and its parent node, the value $T_{\mathrm{MaxBackoff}}$ of the maximum backoff interval to be used by its children, and the time until the start of the next duty cycle T_{NextDC}. Note that by having the parent compute and disseminate the value of $T_{\mathrm{MaxBackoff}}$ for its children, children do not need to know the number of their siblings which appears in Eq. (1). The length of T_{NextDC} is set so that all nodes start the next duty cycle at the same time. The root sets T_{NextDC} to a configured value, which can be modified between duty cycles. All other nodes decrease the received value of T_{NextDC} by their backoff time t_{backoff} when forwarding an *Announce* message.

Network formation is initiated by a root node. When a root node is powered on, it immediately starts a duty cycle. Even without another node present, the root runs through duty cycles with the sequence of phases shown in Fig. 4. We next describe the interactions of a node that wants to join the CottonCandy network, referred to as 'new node.' Every node (other than the root) acts as a new node after it is powered on. As illustrated in the bottom figure in Fig. 4, a new node continuously listens to the public channel to receive *Announce* messages sent by CottonCandy nodes in the *SeekJoin* phase. A node may need to wait for a duration up to the hibernation period between duty cycles before it can observe such a message. Every node from which the new node receives an *Announce* message is a candidate to become its parent node in the spanning tree. After receiving the first *Announce* message, the new node waits for a few seconds to receive up to two more *Announce* messages. Then, the node hibernates to synchronize with the next duty cycle of the CottonCandy network.

In the *Join* phase of the next duty cycle, the new node sends a *Join* message to each parent candidate. To mitigate collisions in case multiple nodes send their *Join* messages to the same parent, every new node applies a random backoff before sending a *Join* message. Unlike the adaptive $T_{\mathrm{MaxBackoff}}$ for *Announce* messages, the random backoff for *Join* messages uses a static value of $T_{\mathrm{MaxBackoff}} = 1\,\mathrm{s}$. With the information in the *Announce* messages, the new node knows when to contact the parent candidates and which channels to use. Starting with the sending of *Join* messages, the exchange of a new node with its parent candidates follows a three-way handshake, shown in Fig. 6, where the third part of the handshake finalizes the parent selection.

In the *Join* phase of a duty cycle, all in-network nodes listen on their private channel for *Join* messages. If a node receives a *Join* message it determines whether it is willing to take on another child node, and, if so, replies with a *JoinAck* message. The reply contains the number of hops of the node to the root, its number of children, and the measured RSSI value of the received *Join* message. A new node selects its parent based on link quality, hop count to the root, and the parent's current number of children. Link quality is determined as the minimum of the measured RSSI value of an incoming *JoinAck* message and the RSSI value contained in the *JoinAck* message, thus reflecting both directions of communication. If the link quality is below a threshold value, the candidate is discarded. The threshold is a configurable parameter with default value 8 dBm above the minimum sensitivity of a LoRa radio. The threshold for link quality is skipped, once the new node has reached the maximum transmission power in the parent discovery (see Sec. III-A) and has not yet found an adequate parent. For the remaining candidates, the new node selects the node with the smallest hop count. In case of a tie, it favors the candidate with fewest children. For candidates with identical values for both metrics, it selects as parent the node with the best link quality as a final tiebreaker. After selecting its best parent candidate, the new node sends a *JoinConfirm* message to the selected parent.[2] After this, the new node is part of the CottonCandy network and can now accept children of its own. Fig. 6 presents an example of a message exchange when a new node joins. Nodes A and B are parent candidates with the same number of hops to the root. Since node B has fewer children, the new node selects it as its parent.

The process for joining a node may fail for several reasons, e.g., *Join* or *JoinAck* messages may incur transmission errors or the link quality for all parent candidates falls below the threshold. If a new node cannot link up with a parent, it returns to listening on the public channel for *Announce* messages. As described in Sec. III-A, in the next attempt, the node increases its transmission power when sending its *Join* messages.

The above discussion presumes that a parent keeps track of the status of its children. If a parent does not receive any messages from a child in multiple consecutive duty cycles, several scenarios are possible: (1) the child has failed (e.g., insufficient battery), (2) the child was desynchronized with the network and is searching for a new parent, or (3) packets from the child have been lost due to collisions or poor link quality. In all cases, the parent presumes the child lost, removes it from its list of children, and recomputes the value of $T_{\mathrm{MaxBackoff}}$

[2]Even if the *JoinConfirm* message does not arrive at the selected parent node, the parent will accept the new node as a new child and will forward data items received from this node.

for its remaining children. We note that in the third situation, the presumed lost child may still be attached to the parent.

B. Data Collection Phase

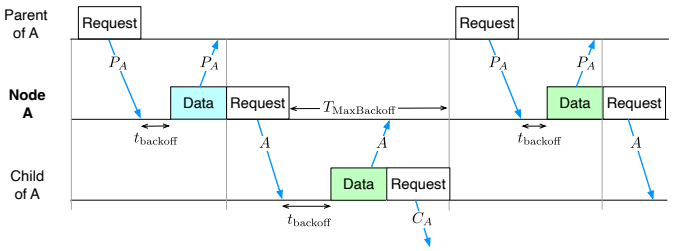
In the *Data collection* phase, the in-network nodes forward data items from their own sensors and the sensors of their descendants toward the root.[3] As discussed in Sec. III-D and illustrated in Fig. 3d, data collection proceeds in a recursive fashion starting at the root, where nodes request data items from their children.

All messages in the *Data collection* phase are sent over private channels, where messages between a parent and a child use the private channel of the parent. To avoid overlapping transmissions by siblings to their parent, children delay their transmission by a random backoff t_{backoff} selected uniformly at random from the interval $[0, T_{\text{MaxBackoff}}]$, where the value of $T_{\text{MaxBackoff}}$ was sent by the parent in its most recent *Announce* message.

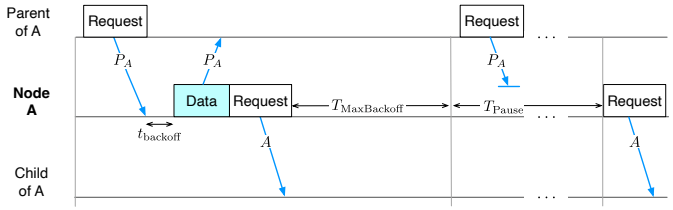
Fig. 7a illustrates details of the data exchange relative to a node A. When node A receives a request from its parent on the parent's private channel, it sends its data item after a random backoff t_{backoff}. Following the transmission, node A switches to its own private channel and transmits a request to its children. The depicted child node proceeds in the same fashion as node A, that is, it transmits its data item to its parent (node A) after a backoff and then issues a request on its private channel. After the transmission of its request, node A listens on its private channel for a duration of $T_{\text{MaxBackoff}}$ and waits for data items from its children. Then, it switches to the private channel of its parent, P_A, and waits for the next request message. When the request arrives it sends its data item as before (if one is available). A node that received multiple data items from its children concatenates the data items into a single message, up to a maximum message size. Once the data has been passed to its parent, node A issues another request to its children. Fig. 7a illustrates that a node sends a new request after it has delivered a message to its parent node. The root, which has no parent, issues a new request after it has processed the data items received from its children. Processing a data item can be as simple as writing it to permanent storage, but generally consists of transmitting the data item to an edge or cloud server.

If a node has requested data from its children but has not received a data item from any of its children for a time period of $T_{\text{MaxBackoff}}$, it enters a low-power state, referred to as *Pause*, which lasts T_{Pause}, after which it sends another request for data items to its children. The purpose of the *Pause* state is that descendants are given time to collect data that will be ready once the node exits the low-power state. Refer to Fig. 7b for an illustration, where the difference to Fig. 7a is that node A does not receive data from its child node following

[3]If there are multiple sensor readings per duty cycle or sensors operate asynchronously, e.g., a rain gauge, a node may need to wake up from hibernation between duty cycles. Buffering data between duty cycles is sufficient to ensure that no sensor readings are lost. Reading of sensors and processing of sensor data are orthogonal to the CottonCandy protocol, since the target sensing applications of climate change and agriculture are generally not time-critical.



(a) Node A and a child of node A report data items.



(b) Child of node A does not have data item to report.

Fig. 7. Data collection for node A (Transmissions are indicated by arrows, which are labeled with the channel. A, P_A, C_A, respectively, indicate the private channel of node A, A's parent, and A's child.).

its first request. While in the *Pause* state, node A does not listen to any channel. As shown in Fig. 7b, node A therefore does not receive the second request from its parent.

The described process has the data collection at each node operating in rounds, where in each round a node sends a *Request* message to its children. A new round is started after receiving data items from its children or after having ended a pause period.

If a node has entered the *Pause* state several times in a row without receiving a data item from any of its children it assumes that its descendants have no more data items to deliver. It then terminates the current duty cycle and hibernates in a low-power state until the start of the next duty cycle. Generally, leaf nodes in the spanning tree are the first to end data collection, and the root is the last. The maximum number of consecutive unanswered requests $R_{\max}$ is configurable.

Several situations exist where a node listens on its parent's channel for *Request* messages without success: (1) the parent node has failed, (2) *Request* messages from the parent encounter transmission errors, or (3) the parent has ended its *Data collection* phase and no longer issues requests. To prevent that a node listens indefinitely, each node has a timeout value T_{end}, which bounds the duration of a duty cycle. A timer for the timeout value is started at the begin of a *Data collection* phase. We use $T_{\text{end}} = 15$ minutes if the time between duty cycles is 20 minutes or more.

To participate in the data collection, a node must know T_{NextDC}, the time until the next duty cycle. As discussed, this information is included in *Announce* messages. Due to the importance of this information, T_{NextDC} is also included in each *Request* message. A node that misses all updates on the schedule of the next data collection cycle becomes desynchronized with the rest of the network. Such a node must rejoin the CottonCandy network as a new node.

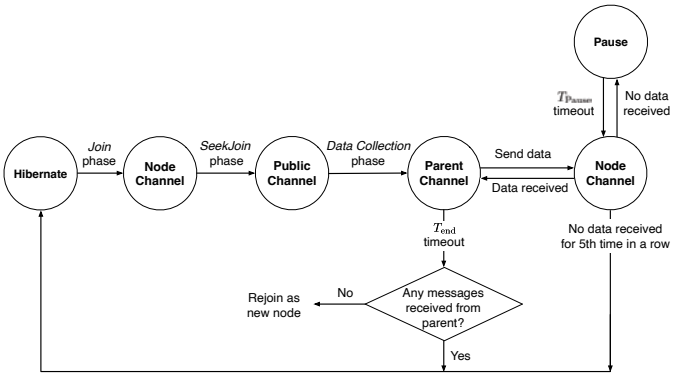


Fig. 8. Channel transitions in a duty cycle. 'Node channel' and 'parent channel,' respectively, refer to the private channel of a node and its parent.

C. Channel Transitions

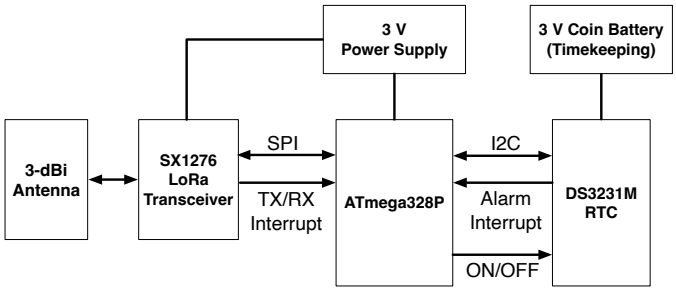
In the course of a duty cycle, a CottonCandy node switches between states where it listens to the public channel, its own private channel, and the private channel of its parent. In the hibernation phase between duty cycles and in the *Pause* state during the *Data collection* phase, a node does not listen to any channel.

Fig. 8 illustrates how a node transitions between channels. At the start of a duty cycle, a node returns from hibernation and enters the *Join* phase where it listens to its private channel for *Join* messages from new nodes. This is followed by the *SeekJoin* phase where a node sends and receives *Announce* messages on the public channel. In the *Data collection* phase, a node switches between its own private channel, its parent's channel, and no channel at all. Each time the node joins its private channel it issues a *Request* message. At each node, the return to its private channel can be viewed as a new round of data collection. The node ends the duty cycle either after a T_{end} timeout or if it has not received messages several rounds in a row. Then, a node hibernates until the start of the next duty cycle. If a node has not received any messages from its parent in the current duty cycle after the T_{end} timeout, it assumes it is no longer synchronized with the CottonCandy duty cycles and tries to rejoin as a new node.

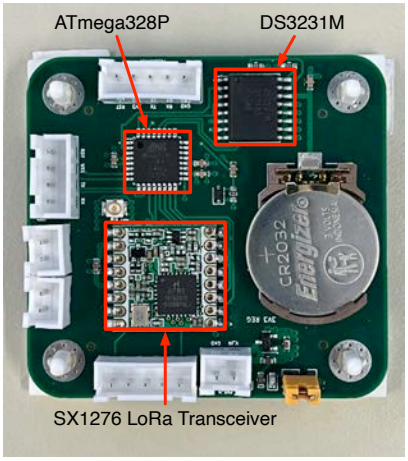
The channel transitions of the root are slightly different from Fig. 8 in that the parent channel is skipped, and, in case of a T_{end} timeout, the root always proceeds to hibernate until the next duty cycle.

V. Hardware Platform

We have implemented the CottonCandy protocol on a microcontroller platform and designed a printed circuit board for a CottonCandy node. To demonstrate that CottonCandy achieves the goal of a low-cost system with a small footprint, we exclusively used low-end commodity components. The protocols runs on an ATmega328P microcontroller [54] with only 32 kB flash memory and 2 kB RAM. The ATmega328P is connected to a Semtech SX1276 LoRa transceiver equipped with a 3 dBi external antenna. A DS3231M real-time clock (RTC) is used for timekeeping with a daily error less than 0.432 seconds [55]. The RTC enables the microcontroller



(a) System design.



(b) Custom-built circuit board. The white connectors are for attaching a battery and sensors, and for installing software.

Fig. 9. CottonCandy node.

and LoRa transceiver to enter a deep sleep mode between duty cycles. At the start of a duty cycle, the RTC wakes up the microcontroller, which in turn switches on the LoRa transceiver. The hardware platform operates on 2 AA batteries with 3V input voltage. The RTC additionally draws power from an on-board coin battery.

Fig. 9a provides a sketch of the system design, and Fig. 9b presents the custom-built circuit board of a CottonCandy node. The total cost the depicted CottonCandy node is less than US-$ 15.

The selection of a low-end microcontroller imposes a limit on security features. Currently, CottonCandy protects packet integrity with a 4-byte cipher-based message authentication codes (CMAC) based on AES-128 [56] and a 16 byte shared key. Support of confidentiality is not viable on the ATmega328P platform.

A. Power Profile

We next present measurements of the power requirements of a CottonCandy node. The power consumption of individual nodes varies depending on their position in the spanning tree topology. Leaf nodes consume far less energy than the direct children of the root node, due to the need of the latter for forwarding data items from all descendants. Nonetheless, as long as nodes use the same underlying hardware platform, the power consumption at each stage of the protocol is consistent across all nodes. This allows us to profile the energy usage

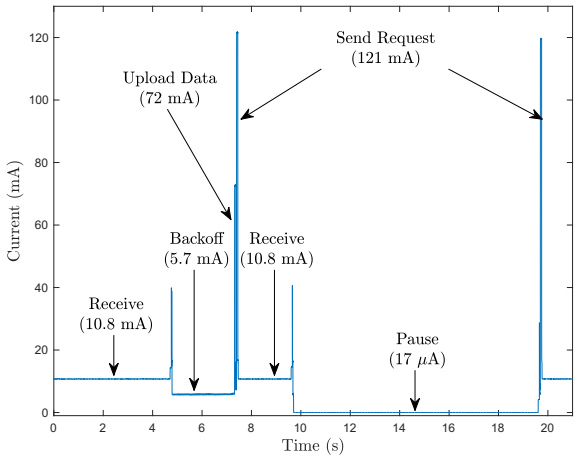


Fig. 10. Current draw of a CottonCandy node in the *Data collection* phase.

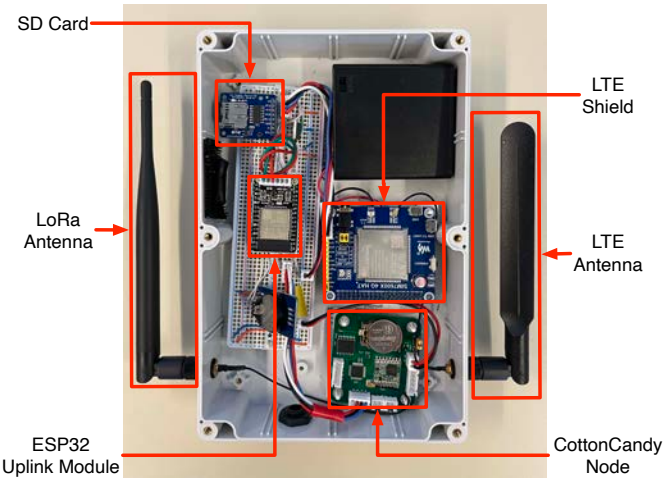


Fig. 11. Gateway of a CottonCandy network (a battery pack is installed above the LTE shield).

during different phases of the operations (computing, receiving data, transmitting data) as well as during hibernations. Given a duty cycle, the total number of transmissions and time spent in reception mode, we can provide an accurate estimate of the battery life of a CottonCandy node.

The energy consumption of a CottonCandy node is profiled using a Nordic Power Profiler Kit II (PPK2) with 1000 samples/s. The accuracy of the measurements have been verified using a professional grade SDM3065X digital multimeter.

The tested node is the only child of a root node and does not have further children. Fig. 10 depicts the energy consumption of the node over a duration of 25 s. The measured activity corresponds to that of node A in the timeline diagram in Figure 7b. Until around $t = 5$ s, where the current draw is constant at approximately 11 mA, the node waits for a *Request* message from its parent (which, in this case, is the root). During this time, the microcontroller is in deep sleep mode, while the LoRa transceiver is active. The arrival of a *Request* message, at around $t = 5$ s, wakes up the microcontroller and triggers a small spike up to 40 mA, which is due to packet processing. The node then performs a random backoff followed by a transmission with an instantaneous current draw of 72 mA. The transmission contains the data item of the node sent on the private channel of the parent. Shortly thereafter is a another transmission, specifically, the transmission of a *Request* on the node's private channel, which peaks at 121 mA. The difference in current draw for the two transmissions is due to proximity-based parent discovery, which reduced the power for transmissions to the node's parent. After the second transmission, the node switches to a receive mode on its private channel where it waits for data from its child. As before, when waiting for a message the microcontroller is set to deep sleep mode, yielding a current draw of around 11 mA. When no data is received after a timeout of $T_{\text{MaxBackoff}} = 3$ s, the microcontroller performs some processing and then enters a hibernation mode with a duration of $T_{\text{Pause}} = 10$ s, where the current draw drops to 17 μA. Here, both the microcontroller and transceiver are in deep sleep mode.

The power profile in Fig. 10 contains all levels of current draws that exist in our implementation. In the next section, we will use the power profile in simulations to obtain the power consumption in a duty cycle for a network with 100 nodes.

B. Gateway

As discussed earlier, the root node of a CottonCandy network is associated with a gateway that provides connectivity to the Internet. The access modality and protocols used for uploading sensor data to the Internet is independent of CottonCandy. Fig 11 shows the hardware of a gateway for a CottonCandy network. The gateway has a modular design that contains two microcontrollers connected via a serial interface: (1) an ATmega328P running as a CottonCandy root node, and (2) an ESP32 responsible for Internet access. The ATmega328P runs the same code as any other CottonCandy node. The ability to run as root is encoded in the address of the node. The ESP32 manages an uplink to the Internet via an onboard Wifi radio and an attached cellular shield. A satellite uplink with an Iridium satellite modem is work in progress. The gateway exchanges data with a cloud system using the MQTT protocol [57] over TLS [58]. We refer to [59] for details of the design and implementation of the gateway.

VI. Simulation Experiments

We next evaluate the CottonCandy protocol in *ns-3* simulations for networks with 100 nodes. The simulations use the physical-layer parameters from Table I, which match the characteristics and configuration of SX1276 LoRa transceivers of our hardware platform. Different from the simulations in Sec. III, the simulations in this section run a full implementation of the CottonCandy protocol, with protocol parameters as given in Table III.

The simulations evaluate (1) network formation time, which measures the elapsed time to establish a CottonCandy network, (2) packet delivery ratio (PDR), defined as the fraction of transmitted data items that are successfully delivered from an originating node to the root, (3) the relative benefits of the collision mitigation methods from Sec. III, and (4) the energy consumption in a duty cycle.

TABLE III
COTTONCANDY PARAMETERS USED IN SIMULATIONS.

Parameters	Value
Initial Tx power for parent discovery	8 dBm
Maximum Tx power for parent discovery	17 dBm
Minimum link quality for parent discovery	-115 dBm
Number of channels	20
Maximum number of children	3
Maximum packet size	64 Bytes
Target max. collision prob. between siblings ($P_{\text{collision}}$)	0.05
Timer between duty cycles (T_{NextDC})	1 hour
Duration of *Join* phase (T_{Join})	6 s
Duration of *SeekJoin* phase (T_{SeekJoin})	120 s
Duration of Pause (T_{Pause})	10 s
Maximum duration of *Data collection* phase (T_{end})	900 s
Max. number of consecutive Pauses, non-leaf node ($R_{\max}$)	5
Max. number of consecutive Pauses, leaf node ($R_{\max}$)	2

As in Sec. III, the simulations distribute 100 nodes uniformly spaced on either a 10×10 grid or uniformly random on a disk, with an additional root node located in the center. The node density (ρ) is varied by modifying the size of the 10×10 grid or the disk. In the simulations, all nodes, including the root, are turned on simultaneously and then execute 10 000 duty cycles. Due to the self-organizing nature of CottonCandy, each simulation run creates a different spanning tree topology. Even with the same spanning tree, repeated runs render different results due to the random backoffs before a transmission. To ensure consistency of results, the presented graphs will show the results from at least five simulation runs.

A. Duration of Network Formation

This experiment measures the time needed to form a spanning tree network with 100 nodes. Since, with CottonCandy, nodes are added only during the *Join* phase of the duty cycle, the time is measured in terms of the number of duty cycles required to complete the network formation. As stated, we assume that all nodes start up at the same time. Also, we assume that there are no node failures. Fig. 12 shows the duty cycles consumed until all 100 nodes have a parent in the spanning tree, as a function of the node density. Each data point presents the results for one simulation run, with five simulation runs for each setting. For node densities above 1 node/km^2, network formation takes less than 25 duty cycles on average. Observe that the outcomes for the grid and uniform disk settings are similar, if not identical. When nodes are distributed more sparsely, the network formation time increases exponentially. This is due to proximity-based parent discovery, where nodes slowly increment their transmission power upon a failed attempt to find a parent. For the uniform random disk, no results are reported for node densities below 1 node/km^2. Here, even at the maximum transmission power, some nodes are unable to find a parent thereby preventing the network from fully forming.

Observe that the network formation times do not decrease monotonically with increasing node density, but increase when the node density exceeds 3 nodes/km^2. At these density levels, joining of nodes is delayed because of packet collisions, mostly of *Announce* messages sent on the public channel and *Join* messages sent to a common parent candidate.

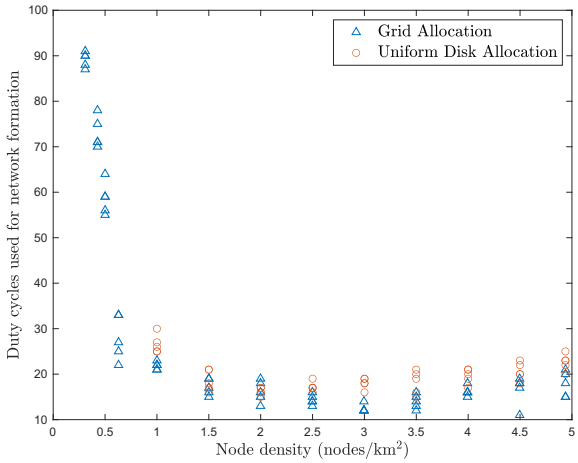
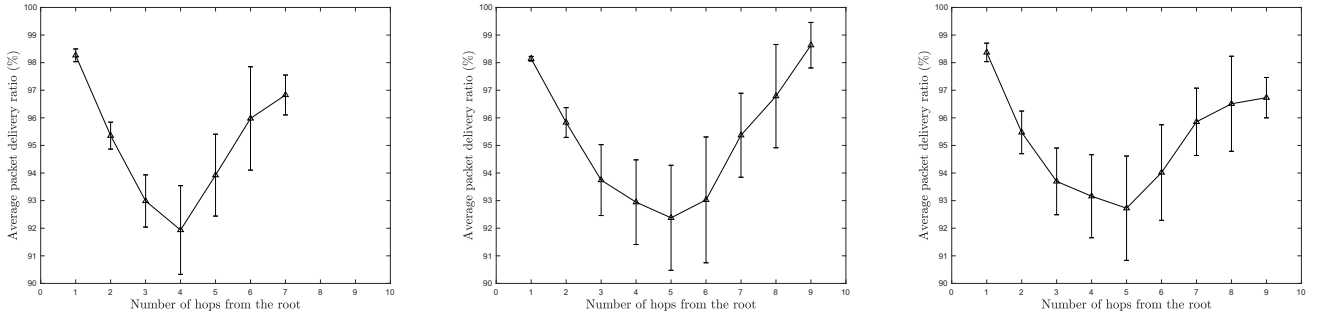


Fig. 12. Duration of network formation as a function of node density.

B. Packet Delivery Ratio

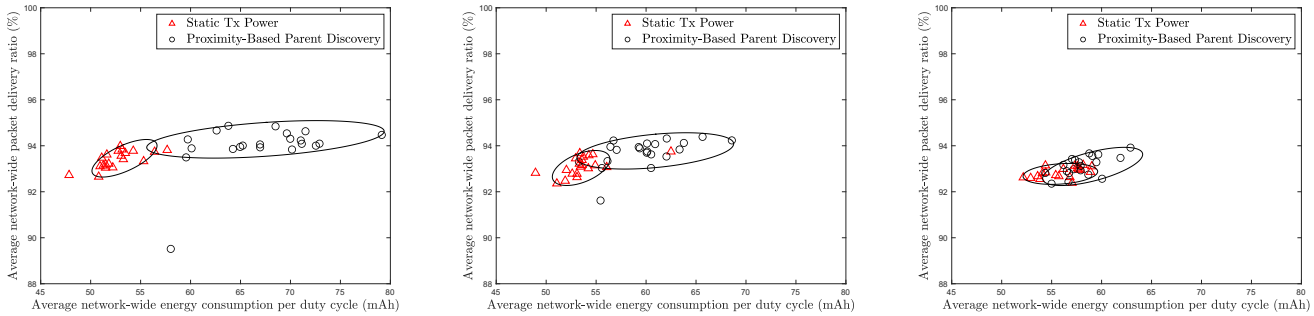
In this experiment, we measure the fraction of data items that are delivered successfully to the root in terms of the PDR of the corresponding messages. The PDR reflects how well CottonCandy mitigates packet collisions. We present the PDR as a function of the distance to the root, measured in terms of number of hops. We assume that in each duty cycle each CottonCandy node (with exception of the root) has one data item that will be sent toward the root during the *Data collection* phase. We arrange 100 nodes in a 10×10 grid or uniformly random on a disk. For the grid arrangements, we use node densities $\rho = 4$ and 0.3 nodes/km^2. For the random placement, we deploy nodes in a disk with a radius of 5 km, resulting in a node density of 1.2 nodes/km^2. The three networks cover an area of about 25, 324, and 78.5 km^2, respectively. measurements of the PDR commence after all nodes have joined the network. In each duty cycle, every node prepares an 8-byte data item, which, with protocol headers, results in a packet length of 15 Bytes.

Fig. 13 shows the average and the standard deviation of the PDR with respect to distance from the root for three node placements. First, note that the average and standard deviation of the PDR are above 90% in all simulations. Second, note that all plots result in U-shaped curves. The lowest PDR is observed for nodes located within 3–6 hops from the root, which comprises the majority of nodes. Due to the operation of the data collection in rounds, data items from nodes with the same distance to the root are in transmission at about the same time. Therefore, the transmissions originating from nodes in the range 3–6 hops create a 'rush hour' in the network, which encounters the highest degree of interference in the entire *Data collection* phase. Nodes that are delivered to the root before or after the rush hour, that is nodes with smaller or larger number of hops to the root, encounter less interference, which results in a higher delivery ratio.



(a) Grid placement (25 km^2). (b) Grid placement (324 km^2). (c) Uniform random disk placement (78.5 km^2).

Fig. 13. Average packet delivery ratio for networks with 100 nodes. The error bars represent the standard deviation.



(a) $\rho = 1.2$ nodes/km^2 (80 km^2). (b) $\rho = 2$ nodes/km^2 (50 km^2). (c) $\rho = 4$ nodes/km^2 (25 km^2).

Fig. 14. Trade-off of the proximity-based parent discovery with different node densities.

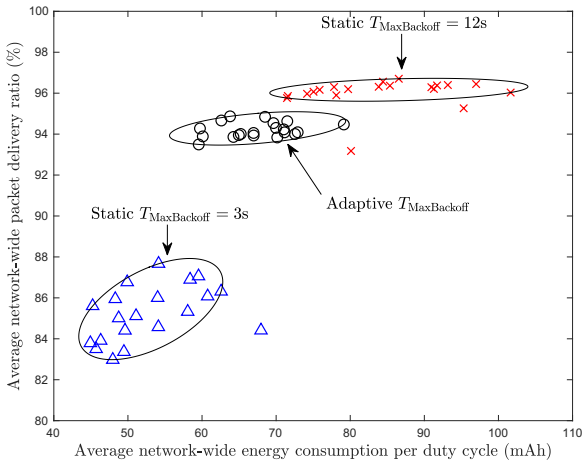


Fig. 15. Evaluation of the adaptive random backoff (80 km^2).

C. Evaluation of Collision Mitigation Techniques

Some of the collision mitigation techniques from Sec. III present trade-offs between PDR and energy consumption. For example, proximity-based parent discovery reduces the number of interferers at each node, but creates longer paths to the root, which requires more relaying of data items, which, in turn, increases energy consumption. Also, extending backoff intervals reduces packet collisions, but prolongs the data collision phase, which again increases energy consumption. We next present simulations that study both trade-offs.

The first group of simulations uses a 10×10 grid placement with a node density of 1.2 nodes/km^2. For these densities, Fig. 1a showed that proximity-based parent discovery has a clear advantage in terms of interferers over neighbor discovery with a static transmission power. In Fig. 14 we depict the average PDR and average power consumption of CottonCandy nodes with and without proximity-based parent discovery. With the latter, nodes use a static transmission power of 17 dBm. Each data point represents a simulation run. We enclose data points from repeated simulations by a minimum volume ellipse computed with the Khachiyan algorithm, with outliers excluded. We observe that the improved PDR of proximity-based parent discovery comes at the cost of higher energy consumption, especially in sparse networks. This is not surprising, since collisions in sparse networks are effectively reduced by the other mitigation strategies. The data in Fig. 14 suggest that the benefits of proximity-based parent discovery are rather limited.

We also explore the trade-off between PDR and energy consumption for random backoff delays. The simulations are performed in the same 10×10 grid as before with a node density of 1.2 nodes/km^2, covering an area of around 80 km^2. We compare our adaptive random backoff delay with random backoffs with a static upper bound of $T_{\text{MaxBackoff}} = 3$ s and $T_{\text{MaxBackoff}} = 12$ s. The results are shown in Fig. 15, in terms of average PDR and average energy consumption. As expected, $T_{\text{MaxBackoff}} = 3$ s has the lowest PDR and lowest energy consumption. Setting $T_{\text{MaxBackoff}} = 12$ s improves the PDR by approximately 10% but consumes 50% more energy on average. The adaptive $T_{\text{MaxBackoff}}$ achieves a similar gain in the packet delivery, but with only about 30% more energy use on average. Note that, the adaptive random backoff, which is

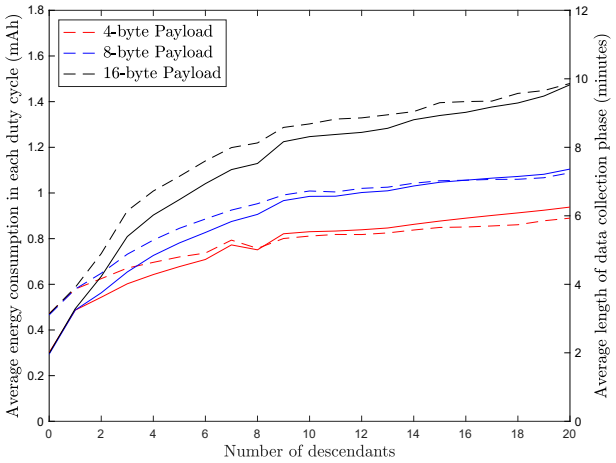


Fig. 16. Duration of *Data collection* phase (solid line) and energy consumption (dashed line).

based on Eq. (1), approximately achieves the targeted PDR of $(1 - P_{\text{collision}}) \cdot 100$.

D. Data Collection Phase and Energy Consumption

In this experiment we evaluate the energy consumption of CottonCandy nodes in a network with a 10×10 grid placement at a node density of 1.2 nodes/km^2. The energy consumption in CottonCandy largely depends the number of descendants for which data items must be relayed. We therefore tally the energy consumption of a node as a function of the number of descendants. Fig. 16 shows the average energy consumption in a duty cycle, measured over all phases of a duty cycle. The energy usage is computed using the power profile from Sec. V-A. We also plot the average duration of the *Data collection* phase. Since most network operations are performed in this phase, we observe that the energy consumption closely tracks the length of the *Data collection* phase. The results are shown in Fig. 16 for different sizes of delivered data items. Both energy consumption and the *Data collection* phase increase with the number of descendants. Leaf nodes (which do not have any descendants) are usually the first to end the *Data collection* phase, after approximately 2 minutes on average. Note that payload size has a big impact on energy consumption and duration of duty cycles. The experiment demonstrates that, using AA batteries with a typical capacity of 3000 mAh each, a CottonCandy node can operate for several thousand data cycles on 2 AA batteries.

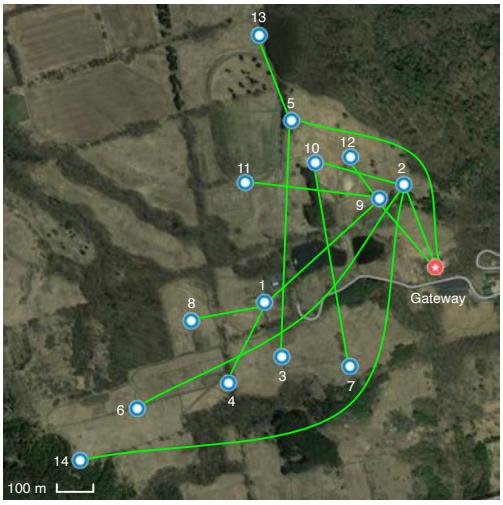
VII. Evaluation of an Outdoor Deployment

We next present measurements from an outdoor deployment of a CottonCandy network at the Koffler Scientific Reserve in King Township, Ontario. The deployed network has 14 CottonCandy nodes (Fig. 17a) and one gateway (Fig. 17b) in a 70 hectare open terrain. All devices are encased in waterproof IP65 enclosures with externally mounted antennas, and mounted approximately one meter above ground at a distance of 100–300 meters to each other.

The CottonCandy configuration mostly uses the parameters in Table III. Since the network is relatively small and to

(a) CottonCandy node.

(b) Gateway.



(c) Network topology.

Fig. 17. CottonCandy deployment at the Koffler Scientific Reserve.

accelerate the collection of data, we configure the frequency of duty cycles to 10 minutes, and adjust parameters T_{end} and T_{SeekJoin} accordingly. Data items of nodes have an 8-byte payload, in addition to a 4-byte CMAC.

Fig. 17c shows the placement of nodes and the spanning tree topology of CottonCandy when measurements were taken. Three nodes (nodes 2, 5, and 9) are children of the gateway. Due to hills, depressions, ponds, and woodland in the overall terrain, geographical proximity of nodes is not a good predictor for signal strength. In fact, only few nodes have an unobstructed line of sight to each other.

The measurements reported here are taken over a 10-day period and cover 1,500 duty cycles. The network topology, shown in Fig. 17c, did not change during this period. Fig. 18 presents the PDR values for all non-root nodes. All nodes have a PDR of 90% or higher, which is consistent with our simulation results. Nodes close to the gateway (1 hop) have the highest PDR. Note that nodes with a distance of 3 hops from the root outperform their parents. For example, the PDR of node 8 is above 92%, while the PDR of its parent, node 1, successfully delivered only 89.5% of its data items. This is consistent with the simulation results in Fig. 13.

In a separate experiment, we tested the fault recovery of

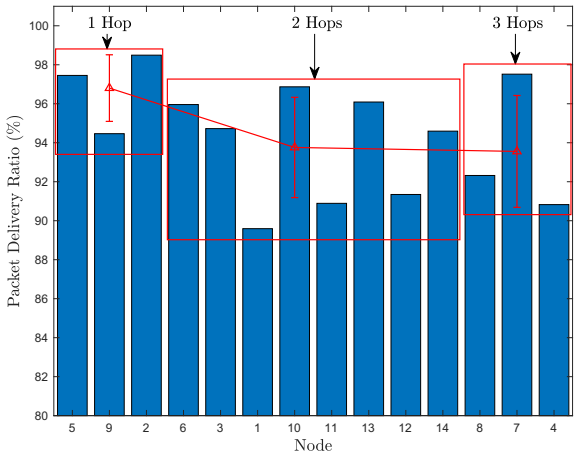


Fig. 18. PDR of outdoor measurements. Nodes are grouped by the hop count to the root. Error bars show the average and standard deviation for each group.

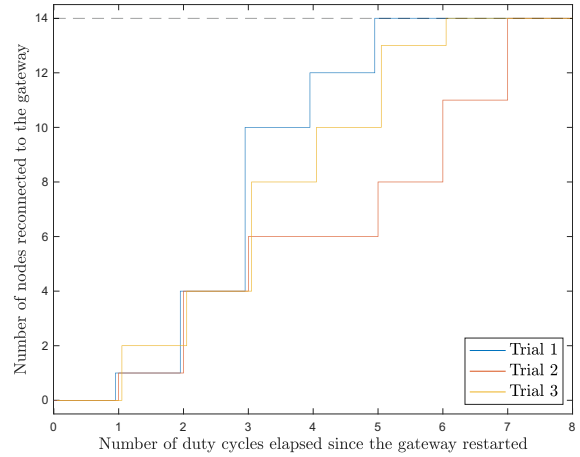


Fig. 19. Network recovery time after the gateway restarted.

CottonCandy. During the operation of the 15-node network, the gateway is turned off for a few consecutive duty cycles, so that all non-root nodes become desynchronized and rejoin the network as a new node. When the gateway is restarted, the nodes can reconnect to the network. This is repeated three times. Fig. 19 shows the number of nodes that have rejoined the network at a given number of duty cycles after the gateway is restarted. In all three trials, all 14 non-root nodes are able to rejoin the spanning tree within 5–7 duty cycles.

VIII. Conclusion

We have presented a self-organizing LoRa multi-hop mesh network for low-cost environmental sensing of large areas. By aggressively exploiting collision mitigation techniques, we were able to demonstrate scalability to at least one hundred nodes. The mitigation methods were integrated in the presented CottonCandy protocol, which dynamically adapts to node additions and failures. The protocol was implemented on a custom-built circuit board, at a cost of less than US-$ 15 per node using off-the-shelf components. Using a detailed power profile, we showed that CottonCandy nodes can operate for years on two AA batteries. We evaluated the CottonCandy protocol in simulations of 100-node networks and outdoor measurements of a deployed CottonCandy network. Both simulations and measurement experiments showed that CottonCandy achieves a packet delivery ratio above 90%. In future work, we will further explore the scalability limits of CottonCandy. An ongoing effort to equip a CottonCandy gateway with a satellite module will enable us to deploy CottonCandy in remote locations without a terrestrial communications infrastructure, e.g., the sub-Arctic and Arctic regions of northern Canada. The current version of CottonCandy only supports data integrity, but does not incorporate additional security measures. Support of data privacy and protection against malicious attacks requires more powerful microcontrollers. Power-saving techniques are another priority. Since significant portion of energy consumption of CottonCandy nodes results from waiting for incoming packets, we will explore a duty-cycle driven receiving mode, where transceivers periodically search for LoRa preambles and return to standby mode if no packet is detected.

References

[1] J. K. Hart and K. Martinez, "Environmental sensor networks: A revolution in the earth system science?" *Earth-Sci. Reviews*, vol. 78, no. 3-4, pp. 177–191, 2006.

[2] S. O. Olatinwo and T.-H. Joubert, "Energy efficient solutions in wireless sensor systems for water quality monitoring: A review," *IEEE Sensors J.*, vol. 19, no. 5, pp. 1596–1625, 2018.

[3] P. Corke *et al.*, "Environmental wireless sensor networks," *Proc. IEEE*, vol. 98, no. 11, pp. 1903–1917, 2010.

[4] F. Montori *et al.*, "Machine-to-machine wireless communication technologies for the internet of things: Taxonomy, comparison and open issues," *Pervasive Mob. Comput.*, vol. 50, pp. 56–81, 2018.

[5] K. Mekki *et al.*, "A comparative study of LPWAN technologies for large-scale IoT deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, 2019.

[6] L. Feltrin *et al.*, "LoRaWAN: Evaluation of link-and system-level performance," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2249–2258, 2018.

[7] F. Adelantado *et al.*, "Understanding the limits of LoRaWAN," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 34–40, 2017.

[8] J. R. Cotrim and J. H. Kleinschmidt, "LoRaWAN mesh networks: A review and classification of multihop communication," *Sensors*, vol. 20, no. 15, p. 4273, 2020.

[9] C. Liao *et al.*, "Multi-hop LoRa networks enabled by concurrent transmission," *IEEE Access*, vol. 5, pp. 21 430–21 446, 2017.

[10] X. Jiang *et al.*, "Hybrid low-power wide-area mesh network for IoT applications," *IEEE Internet Things J.*, vol. 8, no. 2, pp. 901–915, 2021.

[11] H. Lee and K. Ke, "Monitoring of large-area IoT sensors using a LoRa wireless mesh network system: Design and evaluation," *IEEE Trans. Instrum. Meas.*, vol. 67, no. 9, pp. 2177–2187, 2018.

[12] D. Lundell *et al.*, "A routing protocol for LoRa mesh networks," in *Proc. 19th IEEE WoWMoM*, 2018, pp. 14–19.

[13] A. Al-Fuqaha *et al.*, "Internet of Things: a survey on enabling technologies, protocols, and applications," *IEEE Commun. Surv. Tuts.*, vol. 17, no. 4, pp. 2347–2376, 2015.

[14] C. Ebi *et al.*, "Synchronous LoRa mesh network to monitor processes in underground infrastructure," *IEEE Access*, vol. 7, pp. 57 663–57 677, 2019.

[15] C. Li and Z. Cao, "LoRa networking techniques for large-scale and long-term IoT: A down-to-top survey," *ACM Comput. Surv.*, vol. 55, no. 3, pp. 1–36, 2022.

[16] G. Barrenetxea *et al.*, "SensorScope: Out-of-the-box environmental monitoring," in *Proc. 7th Int. Conf. on Information Processing in Sensor Networks*, 2008, pp. 332–343.

[17] J. Liebeherr, M. Valipour, and T. Y. Zhao, "Elements of application-layer internetworking for adaptive self-organizing networks," *Proc. IEEE*, vol. 107, no. 4, pp. 797–818, 2019.

[18] Z. Sun, H. Yang, K. Liu, Z. Yin, Z. Li, and W. Xu, "Recent advances in LoRa: A comprehensive survey," *ACM Trans. Sen. Netw.*, vol. 18, no. 4, pp. 67:1–67:44, 2022.

[19] S. Sundaram *et al.*, "A survey on LoRa networking: Research problems, current solutions, and open issues," *IEEE Commun. Surv. Tuts.*, vol. 22, no. 1, pp. 371–388, 2020.

[20] P. Huang *et al.*, "The evolution of MAC protocols in wireless sensor networks: A survey," *IEEE Commun. Surv. Tuts.*, vol. 15, no. 1, pp. 101–120, 2012.

[21] I. Demirkol, C. Ersoy, and F. Alagoz, "MAC protocols for wireless sensor networks: a survey," *IEEE Commun. Mag.*, vol. 44, no. 4, pp. 115–121, 2006.

[22] Semtech Corporation, "What are LoRa and LoRaWAN?" 2019.

[23] J. Petäjäjärvi *et al.*, "On the coverage of LPWANs: range evaluation and channel attenuation model for LoRa technology," in *Proc. 14th Int. Conf. on ITS Telecommunications*, 2015, pp. 55–59.

[24] M. R. Seye *et al.*, "A study of LoRa coverage: Range evaluation and channel attenuation model," in *Proc. 1st Int. Conf. on Smart Cities and Communities*, 2018, pp. 1–4.

[25] Semtech Corporation, *SX1276/77/78/79 Low Power Long Range Transceiver Data Sheet*, Camarillo, CA, USA, 2020. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-core/sx1276#download-resources

[26] *ERC Recommendation 70-03 Relating to the use of short-range devices*, European Conference of Postal and Telecommunications Administrations, 1997. [Online]. Available: https://docdb.cept.org/download/25c41779-cd6e/Rec7003e.pdf

[27] *National Frequency Allocation Plan*, Department of Telecommunications, Ministry of Communication, Government of India, 2018. [Online]. Available: https://dot.gov.in/sites/default/files/NFAP%202018.pdf?download=1

[28] P. Huang *et al.*, "The evolution of MAC protocols in wireless sensor networks: A survey," *IEEE Commun. Surv. Tuts.*, vol. 15, no. 1, pp. 101–120, 2013.

[29] J. L. Hill and D. E. Culler, "Mica: A wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, 2002.

[30] A. El-Hoiydi, "Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks," in *Proc. IEEE ICC*, 2002, pp. 3418–3423.

[31] E. A. Lin, J. M. Rabaey, and A. Wolisz, "Power-efficient rendez-vous schemes for dense wireless sensor networks," in *Proc. IEEE ICC*, 2004, pp. 3769–3776.

[32] R. Musaloiu-Elefteri, C. M. Liang, and A. Terzis, "Koala: Ultra-low power data retrieval in wireless sensor networks," in *Proc. 7th Int. Conf. on Information Processing in Sensor Networks*, 2008, pp. 421–432.

[33] W. Ye, J. S. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proc. IEEE INFOCOM*, 2002, pp. 1567–1576.

[34] T. van Dam and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proc. ACM SenSys*, 2003, pp. 171–180.

[35] S. Du, A. K. Saha, and D. B. Johnson, "RMAC: A routing-enhanced duty-cycle MAC protocol for wireless sensor networks," in *Proc. IEEE INFOCOM*, 2007, pp. 1478–1486.

[36] F. Adelantado *et al.*, "Understanding the limits of LoRaWAN," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 34–40, 2017.

[37] "The Things network," The Things Network, 2019. [Online]. Available: https://www.thethingsnetwork.org/

[38] A. Haleem *et al.*, "Helium: A decentralized wireless network," Helium Systems Inc., Release 0.4.2, 2018. [Online]. Available: http://whitepaper.helium.com/

[39] Z. Xu, P. Xie, and J. Wang, "Pyramid: Real-time LoRa collision decoding with peak tracking," in *Proc. IEEE INFOCOM*, 2021, pp. 1–9.

[40] S. Tong, Z. Xu, and J. Wang, "CoLoRa: Enabling multi-packet reception in LoRa," in *Proc. IEEE INFOCOM*, 2020, pp. 2303–2311.

[41] X. Wang *et al.*, "mLoRa: A multi-packet reception protocol in LoRa networks," in *Proc. IEEE ICNP*, 2019, pp. 1–11.

[42] Z. Wang *et al.*, "Online concurrent transmissions at LoRa gateway," in *Proc. IEEE INFOCOM*, 2020, pp. 2331–2340.

[43] R. Eletreby *et al.*, "Empowering low-power wide area networks in urban settings," in *Proc. ACM Sigcomm*, 2017, pp. 309–321.

[44] A. Gamage *et al.*, "LMAC: efficient carrier-sense multiple access for LoRa," in *Proc. ACM MobiCom*, 2020, pp. 43:1–43:13.

[45] D. Zorbas *et al.*, "TS-LoRa: Time-slotted LoRaWAN for the industrial internet of things," *Comput. Commun.*, vol. 153, pp. 1–10, 2020.

[46] B. Reynders *et al.*, "Improving reliability and scalability of LoRaWANs through lightweight scheduling," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1830–1842, 2018.

[47] M. H. Dwijaksara, W. S. Jeon, and D. G. Jeong, "Multihop gateway-to-gateway communication protocol for LoRa networks," in *Proc. IEEE ICIT*, 2019, pp. 949–954.

[48] M. C. Bor, J. Vidler, and U. Roedig, "LoRa for the internet of things," in *Proc. Int. Conf. on Embedded Wireless Systems and Networks*, 2016, pp. 361–366.

[49] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Springer Berlin Heidelberg, 2010, pp. 15–34.

[50] D. Magrin, M. Capuzzo, and A. Zanella, "A thorough study of LoRaWAN performance under different parameter settings," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 116–127, 2020.

[51] Y. Yu *et al.*, "Adaptive multi-channels allocation in LoRa networks," *IEEE Access*, vol. 8, pp. 214 177–214 189, 2020.

[52] V. Toro-Betancur *et al.*, "Modeling communication reliability in LoRa networks with device-level accuracy," in *Proc. IEEE INFOCOM*, 2021, pp. 1–10.

[53] O. Yang and W. Heinzelman, "Modeling and performance analysis for duty-cycled MAC protocols with applications to S-MAC and X-MAC," *IEEE Trans. on Mobile Comput.*, vol. 11, no. 6, pp. 905–921, 2012.

[54] Atmel Corporation, *ATmega328P Data Sheet*, San Jose, CA, USA, 2015. [Online]. Available: https://www.microchip.com/en-us/product/ATmega328P#document-table

[55] Maxim Integrated, *DS3231M Data Sheet*, San Jose, CA, USA, 2015. [Online]. Available: https://www.maximintegrated.com/en/products/analog/real-time-clocks/DS3231M.html#tech-docs

[56] National Institute and Technology of Standards, "Advanced encryption standard," *NIST FIPS PUB 197*, 2001.

[57] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-SA publish/subscribe protocol for wireless sensor networks," in *Proc. 3rd Int. Conf. on Communication Systems Software and Middleware and Workshops (COMSWARE)*, 2008, pp. 791–798.

[58] E. Rescorla, "The transport layer security (TLS) protocol version 1.3." *IETF RFC 8446*, pp. 1–160, Aug. 2018.

[59] A. Bogdan, "Designing an end-to-end internet-of-things system for large scale environmental monitoring," University of Toronto, Division of Engineering Science, B.A.Sc. Thesis, Apr. 2022.

Dixin Wu (S'17) received the MASc degree in Electrical and Computer Engineering at the University of Toronto in 2023. He is currently with the University of Toronto and Spero Analytics where he commercializes sensor network technologies for large-scale environmental sensing.

Jörg Liebeherr (S'88, M'92, SM'03, F'08) received the Ph.D. degree in Computer Science from the Georgia Institute of Technology in 1991. He was on the faculty of the Department of Computer Science at the University of Virginia from 1992–2005. Since Fall 2005, he is with the University of Toronto as Professor of Electrical and Computer Engineering.