Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

FLEET: When time-bounded communication meets high energy-efficiency

CHAYAN SARKAR[1], (Member, IEEE), R. VENKATESHA PRASAD[2], (Senior Member, IEEE), AND KOEN LANGENDOEN[2], (Senior Member, IEEE).

[1]TCS Research & Innovation, Kolkata - 700160, India, (e-mail: sarkar.chayan@tcs.com)
[2]Delft University of Technology, 2628 CD, Delft, The Netherlands (e-mail: r.r.venkateshaprasad@tudelft.nl, koen.langendoen@tudelft.nl)

Corresponding author: Chayan Sarkar (e-mail: chayan@ieee.org).

ABSTRACT With the advent of low-cost, embedded sensor-actuator devices, the applications of cyber-physical systems have spread multi-fold in domains like infrastructure, manufacturing, automation, etc. Wireless sensor-actuator networks (WSANs) act as the backbone for applications in these domains. Typical WSAN deployments focus on energy-efficiency (in-turn lifetime) as replacing batteries is labor intensive and expensive. However, many CPS applications require highly-reliable data delivery with strict time bounds. Unfortunately, the classical approach of scheduling/prioritizing flows for bounded time communication is hard to implement with energy constrained embedded devices.

In this work, we present FLEET, a communication primitive that guarantees timely data delivery with (i) low latency by scheduling a maximum number of end-to-end flows within a short time span; (ii) highly energy-efficient networking; and (iii) reliable data delivery. Using a smart parallelization technique, FLEET achieves simultaneous transmissions while guaranteeing data delivery. This reduces the average duty-cycle of the nodes and makes it more energy-efficient than many state-of-the-art protocols. By combining multiple routing strategies, FLEET not only simplifies the schedulability problem but also accommodates more flows within a time span reducing delay considerably. Overall, with respect to the state of the art, FLEET offers a delay and duty cycling reduction by 2.2 and 2.8 times, respectively.

INDEX TERMS capture effect, clustering, energy-efficient, low-latency, wireless sensor networks

I. INTRODUCTION

TRADITIONALLY, Wireless Sensor Networks (WSNs) have been deployed for collecting data over extended periods of time [1]. In recent years, due to their ease of deployment and management, Wireless Sensor-Actuator Networks (WSANs) have become an integral part of many smart-* applications in the domain of Internet of Things (IoT) or cyber-physical systems (CPS) at large. However, many CPS applications have stringent requirements, unlike traditional WSAN. More precisely, in CPS, it is important to have: (i) as low latency as possible; (ii) as high reliability as possible and above all (iii) as low energy consumption as possible since the devices are battery operated.

Figure 1 illustrates a generic CPS scenario consisting of a number of sensor and actuator nodes and a controller. The system works in a sense-decide-actuate cycle, where sensors detect and report the events, the controller takes the decision and sends the actuation commands based on the event, and the actuators act according to the commands. Even though this cycle is the primary aspect of any CPS, the underlying communication protocol makes it possible. The stringent requirements of CPS make it difficult to simply adopt the existing protocols in the WSN domain. Let us take the process industry as an example, where there are more than 40 M sensor and actuator devices installed worldwide, operating feedback control loops in the time scale of tens of milliseconds [2], [3]. Since these devices are typically battery operated, there is an equally strong requirement on energy efficiency to keep the operational and maintenance costs down. As classic WSN protocols trade off performance for energy efficiency, there is a need for a new set of bounded-time communication protocols that strike a (different) bal-

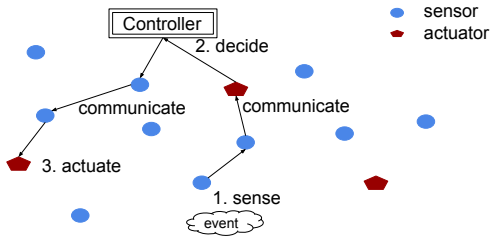


FIGURE 1. Communication plays a crucial role in the sense-decide-actuate cycle of a CPS.

ance between latency and energy consumption while meeting the demands of CPS applications.

A. PROBLEM DESCRIPTION

Crafting an energy-efficient, low-latency, highly-reliable communication primitive is not easy for several reasons, which are listed below.

Duty-cycling dilemma: Duty cycling is the standard approach to reduce energy consumption by periodically putting sensor devices to sleep, thereby extending their lifetime up to years compared to days/weeks [4]. To keep the design complexity low, many WSN-specific MAC protocols apply duty cycling in an asynchronous fashion. This, however, seriously affects end-to-end latency as packets will be delayed at each hop through the network to wait for the next node to wake up.

Rendezvous predicament: To craft an efficient route (i.e., a staggered set of slots) between a source (sensor or sink) and a destination (sink or actuator) node the network topology must be known *apriori*. Collecting, and keeping an up-to-date status of link availability between neighboring nodes is quite a challenge as environmental factors can have detrimental effects on packet reception rates, and link quality may fluctuate heavily. Worse, scheduling multiple flows (i.e., source-destination pairs) is known to be an NP-hard problem [5], and it becomes even more difficult when each flow needs to be completed within a tight time-bound [6].

Periodic v/s. event-driven tussle: Another issue is that most of the existing bounded-time protocols are aimed at periodic traffic (flows) where sensor nodes report their data at regular intervals [7], [8]. However, many cyber-physical system applications involve event-driven scenarios where the sensors may read out periodically, but data is reported only when a significant event is detected [2]. In order to provide a fixed delay bound for event-data delivery, a trivial solution would be to decide schedules similar to that of periodic traffic, i.e., a dedicated set of slots at regular intervals to the nodes that are involved in routing the data packet from each source. However, as only the source nodes know whether the data would be sent or not, all the intermediate nodes would wait in receiving mode in their respective receiving slots (idle listening) unnecessarily, which leads to wastage of enormous amount of energy. Indeed listening to the channel consumes a bit more energy than transmission. An alternative approach is to assign a few open slots to handle event-based traffic, where a node contend within the open slots when it detects an event. Naturally, when many events are to be reported and few slots are available, the data packets may repeatedly collide with each other without being able to deliver, let alone on time. This has forced protocol designers to overprovision by a large margin.

B. APPROACH

In this paper we describe FLEET (**F**lat **L**atency **E**nergy **E**fficient **T**ransmission), a bounded delay communication protocol that effectively tackles the above-mentioned scheduling issues and also avoids overprovisioning. FLEET combines synchronous, slot-based communication at the link level using advanced flooding and clustering at the network level to reduce energy wastage. Important techniques used in FLEET are listed below.

Clustering: To limit over provisioning and enhance efficiency, FLEET employs a hierarchical approach in which nodes report to cluster heads, who aggregate data from all members into a single packet, which is forwarded (flooded) to the controller. This reduces the number of data packets within the network as only the aggregated packets need to travel multiple hops, which in turn reduces the number of slots in the global schedule, i.e., across the clusters.

Capture effect: Data collection within a cluster is short ranged, which allows for spatial reuse of slots when seen from the network perspective. Because of the capture effect [9], which dictates that a node will be able to receive the packet sent from the closest source with high probability, FLEET opportunistically orchestrates all clusters to operate in parallel and thus speeding up the data collection process considerably. Here retransmissions are supported to enhance reliability for safety-critical applications.

Constructive interference: FLEET employs constructive interference (CI) to quickly flood the aggregated data through the network. We capitalize on the success of Glossy [10] and eliminate the need for hop-by-hop routing. As each cluster head is directed to flood its message in turns –in one slot– the flow scheduling problem is effectively eliminated and at the same time sending packets back-to-back reduces radio transceiver on-time, thus, energy usage.

To demonstrate the feasibility of FLEET's novel design leveraging the benefits of clustering, capture and CI, we implemented it on the Contiki [11] operating system and tested it on two testbeds (Indriya [12] and FlockLab [13]). FLEET was demonstrated to achieve up to 2.2 and 2 times lower delay, while consuming up to 2.8 and 3.8 times lesser energy when compared with the state-of-the-art low-power wireless bus (LWB) [14] protocol, which is a flow-scheduling protocol on top of Glossy.

C. CONTRIBUTIONS

The main contribution of this work is in designing a new communication protocol that encompasses the best practices in WSN protocol design, stitch them together, and adapt them

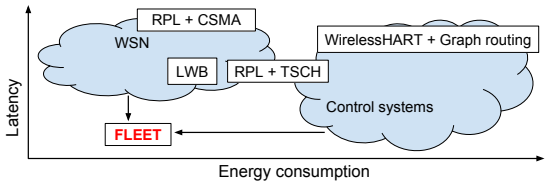


FIGURE 2. Positioning of FLEET in the design space of energy-efficient and bounded-time protocols.

in a way that two contrasting requirements are met simultaneously (Fig. 2). Our FLEET protocol tries to optimize the delay, reliability as well as energy and time. The salient features are summarized below:

- Due to its low overhead, FLEET supports both periodic and event-driven traffic (through aggregation) flexibly, while energy waste for event-driven traffic is kept minimal.
- FLEET provides bounded communication latency without explicitly solving the complex flow-scheduling problem. We prove that the total latency in FLEET is bounded by $(p\overline{n})$ and $O(n)$ slots, where the total number of flows in the network is n.
- To the best of our knowledge, FLEET is the first to employ on-the-fly clustering for time-bounded IoT applications.
- Using efficient spatial slot reuse, FLEET ensures that a large number of flows can be delivered within a given short time period. In other words, more flows can meet their deadline compared to conventional (sequential) communication.

The novelty of this work does not lie in the individual features, but for a holistic approach, i.e., it improves the state-of-the-art on every aspect at the same time. To the best of our knowledge, this is the first work that comprehensively brings multiple aspects and binds them into a useful and highly efficient protocol [1]. Further, we have also evaluated the FLEET protocol on two testbeds.

II. RELATED WORK

FLEET builds upon the foundations laid by energy-efficient communication developed for WSNs and the low-latency protocols developed for wireless sensor-actuator networks (WSANs). We will succinctly review the most relevant work from both these areas as the amount of literature is vast. Moreover, as much of FLEET's efficiency is due to the use of clustering, we will also briefly address that topic.

Energy-efficient communication: Communication protocols for WSNs are geared towards duty-cycling at the MAC layer [16]–[18], and data collection at the routing layer [19]–[21]. The former generally sacrifice latency and predictability for a reduction in energy consumption, while the latter is too specific in routing traffic only to one (or few) edge nodes to be used for IoT applications. Even recent protocols, like RPL [22] and ORPL [23], that do support any-to-any routing perform poorly in a real-time context [24].

[1]Part of this work is included in the dissertation of one of the authors [15].

Bounded latency communication: WirelessHART is the most prominent time-bound MAC protocol for WSANs [25]. The standard defines a TDMA structure, but leaves open the scheduling of the slots [26]. In practice, WirelessHART is often combined with source routing [27] or graph routing [28]. Source routing is straight forward to implement but struggles with changing link qualities as routes need to be recomputed. Graph routing is more robust as it includes alternative routes in the schedule but pays its price in terms of latency and energy consumption [28].

To ensure time-bounded data delivery, end-to-end node scheduling is exercised hand-in-hand with routing. As node scheduling is an NP-hard problem [5], a number of heuristic solutions have been proposed in the literature. For example, a mathematical model for joint routing and link scheduling is proposed by Soldati *et al.* [29]. Similarly, Pottner **et al.** developed a system with schedule construction for time-critical data delivery for periodic traffic [7]. As scheduling all flows in a network within a certain time-bound is a challenge, schedulability analysis under graph routing is studied in [6]. The common drawback of these solutions is their computational complexity.

Time slotted channel hopping (TSCH) is a descendant and improvement of WirelessHART that support time-bounded and reliable data delivery [30], [31]. Due to adherence to the layered architecture, it can be coupled with any routing and higher layer protocols. Like WirelessHART, TSCH also requires a scheduler to support multihop bounded-latency routing. Orchestra [32] proposes a distributed scheduler that is aimed for this purpose. Recently, the IETF Working Group 6TiSCH [33] is standardizing an IPv6 supported protocol suite that uses TSCH as MAC combined with RPL-based routing mechanism. However, a number of challenges are still open to be improved [34]–[36].

Clustering: The usage of clustering has been successfully proposed before [37], [38]. However, most of the existing research does not consider bounded-time data delivery. An exception is a work by Deng *et al.*, who propose a cluster-based data collection mechanism for delay-sensitive WSANs [39]. Their method, however, cannot quash the NP-hard scheduling problem. Moreover, it assumes that the topology of the network is known beforehand. Thus they cannot be extended to IoT scenarios involving actuator nodes. **Concurrent transmission for communication**: An alternative approach is to forgo routing altogether and operate the multi-hop network as a shared bus delivering all data to all nodes. The corner-stone of this approach is Glossy's fast flooding mechanism that exploits constructive interference [10] for successful communication. The LWB protocol [14] overlays a TDMA structure on top in which nodes take turns in initiating such an efficient flood. The reduction in complexity (doing away with routing) outweighs the overhead of flooding the whole network in each slot. LWB works remarkably well outperforming traditional tree-based routing by quite some margin. However, it does not scale well to large networks as the average latency grows

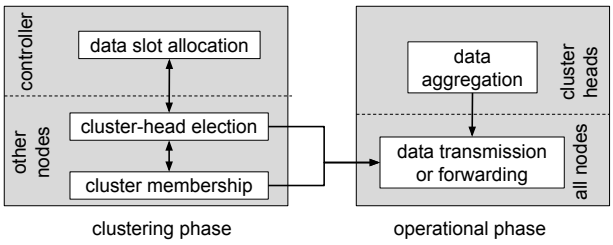


FIGURE 3. Components of FLEET grouped in two phases.

TABLE 1. Comparison of various system parameters between WirelessHART, TSCH, and FLEET.

Parameters	WirelessHART	TSCH	FLEET
superframe length	max. 15 s	flexible	flexible
sync duration	1.5 s	not defined, usually <90 s	20 ms
slot length	10 ms	10 ms (usually), 15 ms, 32 ms	10 ms and 20 ms
# of slots in a superframe	max. 1350	flexible	flexible

linearly with the number of nodes in the network. Though LWB is shown to be efficient compared to state-of-the-art data collection protocols, such as CTP [19], RPL [22], and BCP [40], the inherent all-to-all communication pattern of LWB introduces a significant amount of energy overhead on the nodes.

A number of other protocols are developed that exploits the concurrent transmission primitive of Glossy. For example, CXFS [41] and Sparkle [42] use a forwarder selection mechanism and transmission power control respectively on top of LWB that reduces the unnecessary participation of nodes in every Glossy flood while uses concurrent transmission in a minimalist way to ensure reliability. Similarly, LaneFlood [43] is another protocol that improves over LWB and shows protocols like TCP/UDP and CoAP can run efficiently by utilizing current transmission.

Zimmerling *et al.* proposed Blink [8], which is the first Glossy-based protocol that is tailored for real-time traffic. It proposes a real-time scheduler (earliest deadline first) on top of non-real-time LWB that ensures flows meet their deadline. However, we show later that it can guarantee deadline only for a few flows within a given stipulated time. Moreover, it assumes periodic traffic (though for different flows the period can be different). Thus, it causes significantly high energy wastage due to idle listening in case of event-driven traffic.

Fig. 2 shows how the various protocols fit in the design space for energy-efficient bounded-time communication protocols spanned by the two dimensions – latency and energy consumption. FLEET has been designed to capitalize on the best practices from the WSN and control domains. The combination of a slot-based approach and (selective) flooding by means of clustering yields a solution that results in both lower latency and lower energy consumption.

III. OVERVIEW

The overall objective of FLEET is to provide a bounded-time communication primitive that supports both periodic and event-driven traffic while ensuring high energy efficiency. This amounts to the following design goals.

Goal 1: support data collection from as many nodes (flows) as possible within the given latency constraints. This implies a slot-based approach since it provides predictability as opposed to asynchronous approaches.

Goal 2: reduce the average latency as much as possible. This hints at maximizing the amount of communication happening in parallel in the network by means of spatial reuse.

Goal 3: handle topology changes in a timely manner. This rules out running complex, and time-consuming flow scheduling algorithms at the central controller (or sink node). This advocates the use of advanced topology-agnostic flooding primitives, but without the associated scaling problems.

Goal 4: make the protocol dynamic such that it can effectively support event-based applications. This implies that the protocol should adapt at the local level instead of at the global controller.

Fig. 3 shows the key building blocks of FLEET grouped in two different phases: *clustering* and *operational*. During the clustering phase, FLEET solicits an election process in which a set of cluster heads is determined such that the remaining sensor nodes are one-hop away from a cluster-head. Once the clusters are formed FLEET enters the operational phase in which data is sent from the sensor nodes via the cluster heads to the global controller. Subsequently, the controller determines the necessary actions and sends out commands (if any) to the actuators in the network.

A. SLOT-BASED COMMUNICATION

FLEET adopts a TDMA approach where activities are mapped onto slots. To support bounded-time communication, slots are carefully allocated to minimize energy consumption and maximize parallel execution. The resulting transmission schedule is compact, yet comprises retransmissions and path diversity to account for the ever-changing wireless environment. FLEET mimics WirelessHART, whose operation is driven by a superframe consisting of multiple slots specifying what needs to happen and when. For reference, Table 1 shows how the frame structure of FLEET and WirelessHART compare. The main difference is that FLEET uses two kinds of slots: unicast slots for local (intra-cluster) communication, and (selective) flooding for global (multihop) communication.

B. FLEET SUPERFRAME

A FLEET superframe consists of radio-on and off duration, where any data transmission occurs during the radio-on period. The duration of a superframe matches the required sensing frequency of the application, with as small radio-on duration as possible (see Fig. 4, which is not drawn to scale). Even during the radio-on duration, a node turns its radio on

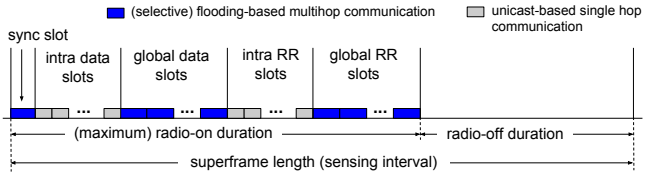


FIGURE 4. Structure of the superframe as the building block of slotted communication in FLEET.

in a particular slot only if it is either a source, forwarder, or destination. The aim is to make a node participate in the minimum number of slots to save as much energy as possible while ensuring end-to-end data delivery of all flows in the network. Moreover, even within a slot, a node turns-off its radio immediately after completing the communication. There are five different types of slots for different purposes, which are either a unicast or a flooding slot.

Synchronization slot: At the beginning of every superframe, a sync slot (re)synchronizes the whole network with the controller similar to Glossy. Thus, it is a flooding slot in which every node participates. Additionally, the sync packet contains the structural information of the superframe, i.e., the number of different slots within the superframe.

Intra-cluster data slot: These slots (using unicasting) are used to deliver sensed data from cluster members to their respective cluster heads. Though each member within a cluster has a separate slot assigned to it, nodes in different clusters can simultaneously communicate reusing these intra-cluster slots. Transmissions by the nodes at the border of a cluster may interfere with other clusters. Thanks to the capture effect, we see fewer packets being lost.

Global data slot: The global data slots are used to deliver aggregated sensed data from the cluster heads to the controller as well as the commands from the controller to the actuators using flooding. Each cluster-head is assigned a unique global data slot.

Request/reply (RR) slot: Request/reply (RR) slots are used to acquire data slots. Intra RR slots (through unicast) are used to acquire an intra data slot by the cluster members and global RR slots (through flooding) are used to acquire a global data slots by the cluster heads.

In the next section, we provide the details of how FLEET works utilizing this superframe and slot-based communication.

IV. PROTOCOL DESCRIPTION

As mentioned earlier, FLEET has two phases – clustering and operational. The first requirement for any node is to synchronize itself with the controller immediately after joining the network. Thus, after a node is powered on, it keeps its radio on until it receives a sync packet from the controller. As soon as it receives this packet, it learns the superframe structure and adjusts its radio-on time accordingly. Subsequently, the node starts acquiring a data slot (if it is a sensor node) in the clustering phase.

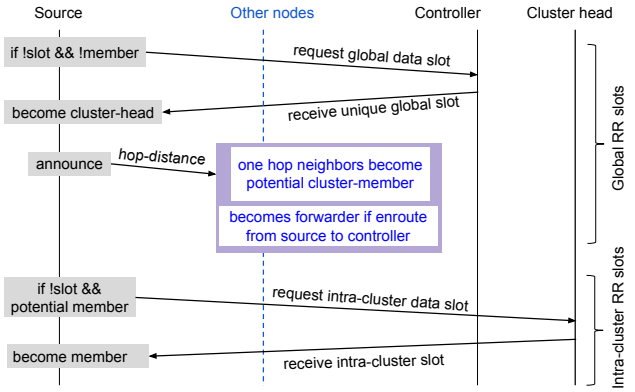


FIGURE 5. Message exchange among the nodes during cluster formation, which is intertwined with data slot assignment.

A. CLUSTERING PHASE

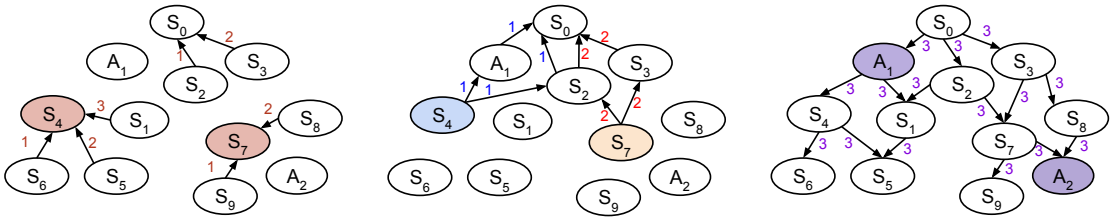
In this phase, every node performs three major tasks: (i) decide a role for itself, i.e., either become a cluster head or a member, (ii) acquire a data slot to deliver its data, and (iii) determine in which global data slots to participate to help in routing/flooding packets to/from the controller.

1) Election of cluster heads and assignment of global data slots

Fig. 5 provides an overview of the cluster-formation process. At the beginning of the clustering phase, a superframe contains only global Request/reply (RR) slots (apart from a sync slot), and nodes have neither cluster role nor a data slot. Thus, everyone requests for a global data slot from the controller in the first (1^{st}) available RR slot. Note that collisions are (usually) resolved due to the capture effect, ensuring that only one request reaches the controller. Note that in the beginning there are more collisions, but that will reduce sharply over a few slots when nodes start succeeding (and stop contending).

Upon receiving a request, the controller sends back a unique global data slot in the next (2^{nd}) RR slot. After receiving that reply, the requester node becomes a cluster head. It announces this fact in the next (3^{rd}) RR slot to its immediate neighbors, who become potential cluster members, and the "upstream" nodes who need to aid in (selectively) flooding the data from the cluster head to the controller. After this, the next round starts with all remaining nodes requesting a global data slot. The process terminates once all the nodes in the network have either been appointed as cluster head or adopted a potential member status.

It is clear that if there are only a few global RR slots in every superframe, it will take a large number of rounds before every node can acquire a data slot. To solve this issue, FLEET uses as many global RR slots as can possibly fit within a superframe. After a couple of unutilized global RR slots, the number of global RR slots is reduced to the minimum (i.e., *three* slots for request/reply/announcement for nodes joining late and/or upgrading to cluster-head status) to reduce energy waste. This switch is accompanied by a change in the



(a) intra-cluster data delivery (slot 1,2,3) (b) aggregated data delivery (slot 1,2) (c) actuation command delivery (slot 3)

FIGURE 6. Packet routing in the network during the operational phase.

superframe structure, which will now include intra-cluster RR slots.

2) Registering cluster membership and assignment of intra-cluster data slots

Potential members record a list of all neighboring cluster heads along with their received signal strengths (RSS). This allows them to become a member of the cluster they are closest to, providing better protection against inter-cluster interference (see below in Section IV-B1). In the first intra-cluster RR slot, potential members send a request to their cluster head to allocate them an intra-cluster data slot. As before, collisions are anticipated to be resolved by the capture effect, and the cluster head sends back a reply to the winning (strongest) member with its slot index. As only one slot can be handed out at a time, nodes have to repeatedly send in requests. If the maximum number of cluster members has been reached the head responds with a negative reply, and the potential member can then switch to another cluster head with the next-best RSS value. When a node has exhausted the list of neighboring cluster heads, it must become a cluster head itself and starts requesting the controller for its own global data slot.

3) Participation in global data slots

Though intra-cluster data transmission occurs over a single-hop using unicasting, the cluster heads send the aggregated data to the controller through multiple hops using selective flooding. Thus, a number of intermediate nodes need to forward the packet within the same global data slot. Unlike LWB, FLEET every node distributedly determine whether or not they should become a forwarder in a particular global data slot. We follow an approach similar to CX-LWB [41].

The idea is to widen the shortest path between a node and the controller by using all paths of equal length[2] connecting the two. A requesting node (n), upon receiving a global data slot from the controller, notes the hop count (h) from the relay-count field in the packet header (as per virtue of the original Glossy protocol). Since the global RR slots are flooding based, every other node also receives the reply message and notes its hop count from the controller (h_c). Next, when node n announces itself as a cluster head, it attaches its hop-distance h from the controller to the announcement message. All other nodes receive this message and record h as well as their hop distance (h_n) to node n. This allows them to check if they are "en route" to the controller. If $h_n + h_c \quad h$ then they add the global data slot of node n to their list of active slots.

[2]For extra reliability, paths with (an) additional hop(s) can also be included.

B. OPERATIONAL PHASE

From the above description, one can see that the clustering process is highly efficient and its length is customizable during the deployment, i.e., proportional to the number of nodes in the network. The operational phase starts after the clustering phase and is initiated by the controller adjusting the layout of the superframes. The intra-cluster RR slots are dropped, the global RR slots are set to the minimum, and the appropriate number of intra-cluster and global data slots are included.

1) Intra-cluster data collection

As mentioned earlier, cluster members use intra-cluster data slots to send their data to their respective cluster heads through unicasting (see Fig. 6(a)). Though the slots are unique among the members of the same cluster, slots are reused across clusters. Even if there are simultaneous transmissions in their vicinity, cluster heads can successfully receive the packets from their cluster members due to capture. When a packet is received correctly, the cluster head immediately sends an ACK. If no ACK is received, the cluster member retransmits the packet after a small timeout period within the same slot (up to 2X). If the retransmission proves in vain the packet is discarded.

2) Data collection and delivery of actuation commands

As mentioned earlier, global data slots are used by the cluster heads for data collection (Fig. 6(b)) and by the controller for delivery of actuation commands (Fig. 6(c)). To report the aggregated sensed data, only a subset of nodes participate in forwarding the packets using selective flooding. However, all nodes forward packets containing actuation commands such that multiple destinations (actuators) can be reached within the same slot. Fig. 7 shows the difference between unicasting and (selective) flooding in the intra and global slots, respectively.

When none of the cluster members detects any event in sensing round, the cluster head has no data to report. To

Slot-based communication – node activity during the operational phase for the network in Fig. 6.

save energy, the cluster head sends a dummy packet so that participating nodes can switch-off their radio immediately after forwarding the packet. Without a dummy packet nodes keep listening unnecessarily for the whole slot as the exact moment a ooding ripple passes by is source/destination/interference dependent. Dummy packets are also used when no actuation is required. Thus some more energy is saved because of this technique.

3) Deciding the number of global data slots

The number of global data slots is typically one slot higher than the number of cluster heads to accommodate actuation commands. Like sensed data, multiple actuation commands are aggregated in one packet and sent to multiple actuators in the same global data slot. If there are more actuation commands that can t within a single packet, multiple slots are used. Each actuation packet contains a ag signaling if it is the last in the pipeline or not.

C. ACHIEVING BOUNDED LATENCY

Due to the retransmissions within a cluster, and the path diversity between the cluster heads and controller, sensor data is delivered with high probability. FLEET also does it faster, as the name suggests. Exactly how fast FLEET handles this depends on the number of clusters. In step one (aggregation)$n-c$ members may send their data. In the next step,c cluster heads forward their data to the controller, who then sends the actuation commands back into the network in a single slot. That amounts to a total of$n-c+c+1$ slots. As the number of clusters varies from 1 (a clique network) to$n-2$, the length of the schedule is bounded by$O(1)$ and$O(\sqrt{n})$, respectively. As remarked earlier practical constraints on the maximum payload limits the number of members in a cluster, so the degenerated case of clique topology will not apply. Thus, the schedule length is typically in the order of$\sqrt{n}$ slots. Note that this compares favorably to ow scheduling on top of Glossy, which requires at least$n$ slots, one for each node to report its data.

FLEET is implemented using the Contiki operating system [11]. As mentioned earlier, it utilizes constructive interference for effective communication. The glossy protocol uses constructive interference based ooding for synchronizing the network. Thus, we used some functionalities of Glossy with some adaptation to obtain network-wide time synchronization, and as a basis for route-free multihop communication. Glossy uses a combined yet simple routing and MAC layer with a simple communication model. Every node has a common understanding of time as well as the starting time of a ood. The initiator node starts the ood by transmitting the packet, and every other node simply transmits the packet after receiving it. Glossy ensures a xed switching delay between receiving and transmitting packets in order to ensure constructive interference. On the other hand, the design of FLEET involves various other types of communication while maintaining the simple design principle.

Without loss of generality, we xed the message size for intra-cluster communication to 4 bytes of sensor-data payload and 4 bytes of headers (2-byte source and destination addresses), thus 8 bytes in total. For inter-cluster communication the message size depends on the number of cluster members$n$, which we varied over 2, 4, and 8, yielding packet sizes of$(n + 1)$ $(4 + 2) + 2 + 2$ bytes to accommodate the payload of the members and head (with source ID), the destination address, and a length eld (22, 34, and 58 bytes respectively). To ensure reliable data delivery without end-to-end control (to reduce latency), FLEET forms clusters such that the link quality between a cluster head and its member nodes are suf ciently high. We used an RSS value of -75 dBm as the threshold to lter out good links. This setting, in combination with up to two retransmissions within a cluster, proved to achieve good packet reception rates across a range of different topologies (as reported in the next section). Only in the case of very sparse networks the chosen threshold had a noticeable impact on the performance as many clusters were created with just a few, or even zero, members due to a lack of quality links, compromising aggregation ef ciency.

The code footprint of FLEET is very small. The implementation added about 900 lines of C code on top of the existing implementation of Glossy. The compiled rmware is only about 26.4 kB as compared to 22.6 kB and 24.8 kB for Glossy and LWB, respectively.

Though the current implementation of FLEET is targeted for the Tmote sky platform with CC2420 radio, it can easily be adapted to work on other types of devices. One of the biggest hurdles is to achieve constructive inference by ensuring multiple simultaneous transmissions, i.e., keeping the switching delay from receive to transmit mode constant across multiple devices. The developers of Glossy already described how it can be ported to other radio platforms, and indeed a number of CI-based systems have emerged running on other radios, e.g., CC430 [41] and CC2530 [44]. Porting FLEET to these platforms should be no more complicated.

We evaluated FLEET both in simulation, which provides repeatable experiments suitable for cross-protocol comparison

(a) Topology (b) HRP, completion in 90 ms (c) Blink, completion in 180 ms

Sample 9- ow network with schedules for optimal source routing (HRP) and Blink. The slot length for HRP and Blink is 10 ms and 20 ms, respectively.

Sample 9- ow network with 3 FLEET clusters and selective ooding, completing in just 70 ms. The length of the intra-cluster and global slots are 10 ms and 20 ms, respectively.

and on a real-world testbed to study resilience to externa interference and other practicalities. For the simulations, we used the Cooja software that comes with the Contiki operating system. The real-world experiments were carried out on two publicly-available testbeds, i.e., Indriya [12] and FlockLab [13] along with some testing on local nodes in our laboratory. Since the testbed measurements involved a lot of randomnesses (e.g., interference from 802.11 traf c during working hours) we ran different protocols back-to-back to ensure they endured similar conditions.

A. TEST SCENARIOS

The testing code on top of FLEET was set to mimic a control application in which each node periodically reads out a sensor and reports it to the central controller. We assume that all the ows have the same priority, though they can have a different deadline, which decides the sensing frequency of the source node. Please note, if there are a signi cant number of nodes in the network and the deadline is very small, there is no way that all the ows can be assigned a slot.

We experimented with four different setups– Default deadline (DD), Tight deadline (TD), Extreme deadline (ED) and Mixed deadline (MD). In DD, TD, and ED scenarios, all ows have the same deadline of 10 s, 1 s, and 250 ms, respectively. These lead to sensing frequency of $\frac{1}{10}$ Hz, 1 Hz, and 4 Hz, and the FLEET superframe length of 10 s, 1 s, and 250 ms, respectively. In the extreme setup, only a few ows can be scheduled within this stipulated time (250 ms). However, using FLEET, a higher number of ows are able to meet their deadline as compared to any other method (as shown later). In the MD scenario, different ows have different deadlines, which leads to a different sensing frequency at the sensor nodes. We used almost 50% of the nodes with 250 ms deadline (sensing frequency of 4 Hz), and the remaining with 1 s deadline (sensing frequency of 1 Hz). The FLEET superframe length is set to 1 s, where frequently sensing nodes would get 4 slots in every superframe.

B. COMPARISON WITH AN OPTIMAL ALGORITHM

The lack of an open implementation of any prevalent real-time routing protocol restricted our options for performing a one-on-one comparison with FLEET. Before providing a detailed study based on the actual implementation –to thoroughly evaluate FLEET vis-á-vis latency– we, therefore, compare FLEET with two protocols based on their design principals. The rst one is a Hypothetical Real-time Protocol (HRP) that packs the data packets tightly to avoid any wastage of slots. This allows us to nd the lower bound on the delay for a particular scenario. Let us assume that HRP packs the data packets tightly to avoid any wastage of slots, and it combines source routing with optimal ow scheduling as described by Pottner et al. [7]. For the sample 9- ow (10-node) network shown in Fig. 8(a), the optimal HRP scheduler nds the shortest possible schedule to complete all the ows as shown in Fig. 8(b). As messages are not aggregated at least 9 slots are needed. HRP manages to do this by scheduling the remaining transfers in parallel. Thus, the total time required is 90 ms considering a slot length of 10 ms according to the WirelessHART standard. In practice, completion will take longer as the ow scheduling problem is NP-hard, thus we need to resort to heuristics. Moreover, for resilience one may prefer graph routing over source routing, adding even more slots to the schedule.

To consider a realistic case for comparison, we turn to Blink [8], which is the closest state-of-the-art routing protocol in WSNs that offers data delivery with the lowest latency

Comparison between Blink and FLEET– bounded-time guarantee for number of ows under different deadlines.

Deadline	DD	TD	ED	MD
Blink	499	99	11	19
FLEET	2725	225	35	56

and high energy ef ciency. Blink also requires 9 slots (like HRP), but these slots are 20 ms long to ensure the data can be ooded across multiple hops. Thus, it takes Blink a total of 180 ms to complete all the ows (Fig. 8(c)), which is twice as long as for HRP.

FLEET combines the unicast slots from HRP and network-wide oods from Glossy (which is also used by Blink) through its clustering approach, see Fig. 9. The maximum number of cluster members was set to 3, leading to 3 clusters headed by nodes S_0, S_4, and S_9 with 3, 2 and 2 members respectively. Consequently, FLEET needs 3 intra-cluster slots, followed by 2 global data slots to forward the aggregated data from remote clusters S_4 and S_9 to the controller S_0. Thus, the total time required to complete the ows is 70 ms ($3 \quad 10 + 2 \quad 20$), where the length of an intra-cluster and a global slot is 10 ms and 20 ms, respectively.

The key observation is that FLEET outperforms both HRP and Blink. FLEET beats HRP because it aggregates data allowing it to use fewer slots while offering additional resilience through its use of ooding. FLEET beats Blink because it exploits parallelism at a local level (spatial reuse) allowing it to use shorter and fewer slots. Please note that for a network with a smaller diameter, length of a Glossy ood can be set to a smaller duration, e.g., 10 ms. In that case, both Blink as well as FLEET would be able to complete the ows in lesser time.

We also compared Blink and FLEET for different deadlines. Table 2 summarizes the maximum number of ows (in an ideal situation) that can meet the deadline using these two protocols. For, DD, TD, and ED scenarios, Blink uses 20 ms for the sync slot and remaining time for data delivery. Similarly for FLEET, 20 ms for the sync slot, 80 ms (8 intra-cluster slots) for intra-cluster communication, and the remaining time for data delivery. For the mixed deadline in Blink, 10 ows are scheduled 4 times in every second (250 ms deadline) consuming 800 ms. Remaining 180 ms (barring 20 ms for sync slot) can be used for ows with 1 s deadline. In the case of FLEET, every 250 ms 35 ows can be scheduled (like ED). Out of these 35 ows, 28 can be of high frequency (250 ms deadline) and the remaining 7 ows would be less frequent. These 28 ows will have a slot in every 250 ms, whereas the remaining 7 slots can be assigned to different ows, totaling 56($28 + 4 \quad 7$) ows in every second. Please note that if there are less number of ows some part of the superframe would not be utilized, which leads to the lower duty cycle. As FLEET packs the ows more tightly, it achieves higher energy-ef ciency than Blink.

Comparison of total time and number of slots required to complete 50 ows for different node degrees.

Packet reception ratio vs. node degree.

Node degree	LWB	CXFS	FLEET
2	100%	99%	98.4%
4	100%	99%	97.3%
8	100%	100%	99.6%

C. SIMULATION RESULTS

To study the performance of FLEET in more detail we evaluate three aspects: (i) delay bound, (ii) energy ef ciency, and (iii) reliability of data collection. In particular, we are interested in the associated metrics of the total time (latency) taken for scheduling all ows, the average duty cycle of the nodes, and the packet reception ratio (PRR). Since studying various parameters under various topologies is not possible in a static testbed, we rst performed a simulation-based study.

1) Performance under periodic traf c

We considered three different network topologies of 50 nodes, in which the average node degree was set to 2, 4 and 8. We did so by changing the total deployment area while keeping the total number of nodes (ows) xed. We compared the performance of FLEET with LWB and its successor Forwarder-Selection LWB (CXFS), which achieves higher energy ef ciency in data collection scenarios by limiting the set of participating nodes in a ood [41]. Note that FLEET also uses such forwarder selection when cluster heads report the aggregated data to the sink. As an open implementation of Blink is not available, we limit our comparison with LWB and CXFS only. However, we use the same deadline for every ow such that LWB becomes equivalent to Blink. Fig. 10 shows the total time (in blue) and the number of slots (in yellow) required to complete the 50 ows in the network. Note that for LWB and CXFS, the numbers remain the same irrespective of the network density as each ow uses one global slot (and the number of ows is constant). Hence, we only plot one bar. For FLEET, however, the increase in node degree raises the number of members per cluster, which in turn increases the scope for parallel communication. The net effect is that FLEET uses more intra-cluster slots (for members) and fewer global slots (for cluster heads) as the breakdown shows. This translates into a lower number of total slots per superframe (more parallelism) leading to lower latency. Note that this effect is ampli ed by intra-cluster slots taking only 10 ms vs. 20 ms for global slots. FLEET excels not only in terms of overall latency, but it

Average duty cycle in a 50 node network.

Duty cycle of individual nodes in a 50 node network (node degree of 4).

Duty cycle of FLEET nodes, sorted by hop-distance from the controller.

also reduces the energy consumption of the network. Fig. 11 shows a comparison of the average duty cycle of the nodes. For LWB and CXFS, unlike with latency, the average duty cycle of the nodes changes when the density of the network increases. Recall that the density was increased by shrinking the deployment area, which causes nodes to be located closer to the controller. For LWB this results in Glossy oods completing faster since fewer hops need to be traversed. This leads to (slightly) lower duty cycles. This effect also applies to CXFS, but at the same time, the increased density leads to more redundant paths between a node and the controller. The effect of involving more nodes per ood is stronger than the reduction in hops, causing CXFS's energy consumption to go up with increasing density. For FLEET, a third factor comes into play. Its shift from global (ooding) to local (unicast) communication counters the reduction in ef ciency of the selective forwarding optimization. The overall effect is that FLEET's ef ciency is almost insensitive to network density.

The nal performance metric to consider is the average PRR, which is listed in Table 3. LWB achieves a 100% PRR due to a high degree of redundancy (all nodes participate in all oods). CXFS does slightly worse, especially for low-density networks as the number of redundant paths is limited, causing about 1 % of the packets being lost. FLEET suffers from the same effect and loses up to 3 % of the packets in the worst case. Closer inspection of the results revealed that this extra packet loss is incurred during intra-cluster communication. We conjecture that this is due to the capture effect failing to resolve all interference from communication in neighboring clusters. Further research is needed to study if careful cluster formation or advanced retransmit policies can bring FLEET's reliability in line with LWB/CXFS. Alternatively, more intra-cluster slots can be used to reduce the amount of parallel communication.

To gain a deeper understanding of the ef ciency of the protocols, we have analyzed the duty cycle of individual nodes. Fig. 12 shows the duty cycle of the 50 nodes for the three protocols. According to the expectations, LWB shows hardly any difference between the nodes as they all participate in all slots. CXFS and FLEET, on the other hand, do show signi cant uctuations. The selective forwarding scheme only involves a subset of the nodes, with those close to the sink being active for all (global) slots and those at the edge only in case of sending their own data. Note that the maximum duty cycle for CXFS (by Node 15) is roughly the same as for LWB, so the network lifetimes, that is until the rst node dies, are the same. In that respect, FLEET does much better (i.e., doubles the lifetime) due to its cluster-based approach.

Fig. 13 provides additional insight by plotting the duty cycle of FLEET for members (blue) and cluster heads (purple) sorted by hop distance to the controller. Two important observations can be made. First, nodes close to the controller spend more energy than nodes 2 or 3 hops away, because of the selective forwarding optimization. Second, cluster heads consume only a little more than cluster members. That somewhat surprising result follows from the low number of members per cluster. Note that the maximum number of cluster members was set to 4, yet only 30 out of 50 nodes became cluster members (and 40 = 4=5 50) due to solitary nodes becoming cluster heads. Apparently, the requirement for having a strong link with the head of a cluster pruned away too many options for edge nodes, who are relatively poorly connected, to begin with. Having few members per cluster automatically reduces the overhead, putting the duty cycle of the cluster heads in line with that of the members.

2) Performance under event-driven traf c

To mimic the event-driven traf c, we set a node to transmit with a probability of 0.5 (this can be set differently to change the event pattern and we may use different distribution too). We tested only two scenarios for the event-driven traf c, i.e., DD and TD. As LWB/CXFS cannot guarantee energy-ef ciency, low latency, and guaranteed delivery simultaneously for bounded-time event-driven traf c, we followed two strategies – guaranteed delivery and best-effort delivery. Table. 4 and 5 shows performance of LWB with these two strategies compared to FLEET. In guaranteed delivery, every node has a dedicated slot irrespective whether they have data to transmit or not. Thus, it ensures guaranteed data

TABLE 4. Duty cycle comparison under event-driven traffic.

Deadline	DD		TD	
	Guarantee	Best-effort	Guarantee	Best-effort
LWB	4.6%	3.75%	46.2%	41.9%
FLEET	1.29%		13.1%	

TABLE 5. PRR comparison under event-driven traffic.

Deadline	DD		TD	
	Guarantee	Best-effort	Guarantee	Best-effort
LWB	100%	65%	100%	62%
FLEET	97.3%		97.2%	

TABLE 6. Performance comparison of various protocols on the Indriya testbed with 106 flows (96 sensors and 10 actuators).

	LWB	CXFS	FLEET	FLEET-X
Total latency (ms)	2120	2120	980	980
Avg. duty cycle (%)	8.11	4.81	2.92	5.23
PRR (%)	98.3	96.5	95.7	97.9

delivery whenever there is data, but energy is being wasted if no meaningful data is sent (if there is no valid data, a dummy packet is sent to reduce idle listening). Thus, the duty cycling is almost the same as the periodic traffic irrespective of sparsity in data traffic.

In the case of best-effort delivery, two nodes share the same slot. Energy-efficiency of this scheme is better since, during no traffic times, idle listening is reduced. Since two nodes share a common slot, there is a delivery failure whenever both the nodes have to send data at the same time. Naturally, the overall packet reception ratio gets reduced. Moreover, as nodes share slots, they cannot send a dummy packet if they do not have valid data.

On the other hand, FLEET always uses a (globally) dedicated slot for cluster heads, and dedicated intra-cluster slots for cluster members. Thus, whenever there is no valid data, a node can send a dummy packet to reduce the idle listening.

D. TESTBED RESULTS

To validate the findings from the Cooja simulations, we conducted real-world tests on the Indriya and FlockLab testbeds with 97 and 32 TelosB nodes, respectively. On Indriya, we used Node 1 as the sink and the other 96 nodes as sensors, of which 10 nodes served the dual role of sensor and actuator, totaling 106 flows in the network. Similarly, FlockLab has 35 flows in total (31 sensors and 4 actuators). For the testbed

TABLE 7. Performance comparison on the FlockLab testbed with 35 flows (31 sensors and 4 actuators).

	LWB	CXFS	FLEET	FLEET-X
Total latency (ms)	720	720	360	360
Avg. duty cycle (%)	3.91	1.95	1.02	1.88
PRR (%)	99.9	99.5	97.1	98.8

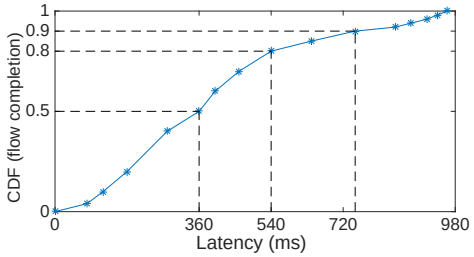


FIGURE 14. Cumulative distribution showing the number of flows that are completed over time (Indriya testbed).

experiments, we used a maximum of 8 intra-cluster slots. The results reported in Table 6 and Table 7 summarize the outcome of over 100 hours of experimentation. The performance numbers confirm that FLEET provides a multi-fold latency reduction compared to LWB and CXFS while ensuring significantly higher energy efficiency. Specifically, FLEET achieves a latency reduction by a factor of 2.2 and 2 over Indriya and FlockLab, respectively.

The average duty cycle of the nodes depends on the total number of flows in the network and the utilization of intra-cluster slots. On Indriya FLEET is 2.8 and 1.6 times more energy efficient compared to LWB and CXFS, respectively. On FlockLab the efficiency is even higher at 3.8 and 1.9 times, respectively. The reason is that the percentage of global slots in FlockLab is lower compared to Indriya.

The harsh conditions on the testbed reflect in the packet reception ratios, which are all lower than recorded in the simulation. Even LWB loses a fraction of packets, while in simulations it lost none. CXFS and FLEET do slightly worse and lose about 3-4 % of packets. Note however that these numbers are comparable to the 95 % average reliability of flow delivery reported by Lu *et al.* using graph routing [45]. To improve the reliability of FLEET, we have created a special version (named FLEET extra, or FLEET-X for short) in which we disable forwarder selection and resort to full Glossy floods (like LWB). The immediate benefit is an improvement in PRR. On Indriya, FLEET-X even surpasses the PRR of CXFS. The remaining packet loss, especially on FlockLab in comparison to LWB, is due to collisions in the intra-cluster slots. Of course, there is no free lunch. FLEET-X's improved PRR goes at the expense of its energy efficiency, which is similar to CXFS (but with a much shorter latency). Whether or not the improved PRR of FLEET-X outweighs the cost in terms of energy efficiency depends on the application at hand.

So far we have discussed the total time required to complete all flows. An important question is what will happen if the deadline is too small to accommodate all flows. The flow completion pattern over time is shown in Fig. 14. It is clear that the majority of the flows can be completed in about half of the time (80% complete in 540 ms). All the members belonging to the controller's cluster complete real quickly during the intra-cluster phase. Then, with each global slot,

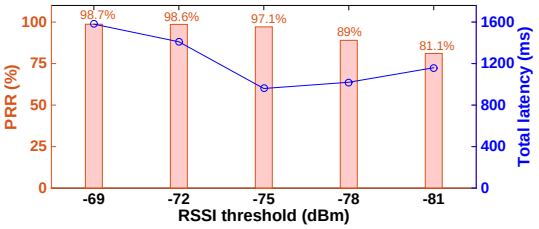


FIGURE 15. The impact of RSSI threshold on latency and reliability (Indriya testbed).

a set of flows complete together as the aggregated message contains the data of all members belonging to the cluster head that was allocated to that slot. The number of flows varies per global slot as the topology dictates the cluster formation. In particular, towards the end slots are taken by solitary cluster heads who failed to join any cluster and are forced to use a global slot just for themselves.

Recall that a node decides to join a cluster only if it has a good link to the respective cluster head. FLEET uses the RSSI to discriminate good from bad links by means of a threshold set to -75 dBm. Changing the threshold will impact the cluster formation process. If we tighten the threshold, the number of cluster members will be lower leading to more clusters, hence, a longer schedule leading to higher latency. In return, the reliability will increase as the links are of better quality. If we relax the threshold, the reverse will happen with schedules completing faster and PRR becoming worse. To study the exact trade-off, we have run a number of experiments with different thresholds on the Indriya testbed, see Fig. 15. Observe that, as expected, the PRR decreases when relaxing the threshold from -69 dBm to -81 dBm. The latency, however, does initially go down, but then raises again once the threshold is relaxed beyond -75 dBm. A detailed inspection of this surprising result revealed that this behavior is a consequence of the safety mechanism built into FLEET. When a cluster member fails to transmit in two successive superframes, i.e., it does not receive an ACK from its cluster head, it leaves the cluster and acquires a global slot. This does keep the PRR under control, but at the expense of additional latency (an extra global slot). Without this fail-safe mechanism the latency does decrease further, but the PRR drops to unacceptably lower values for typical IoT applications. Therefore, FLEET's default threshold of -75 dBm appears to be the sweet spot to achieve the lowest latency and marginal packet loss.

VII. DISCUSSION

In this section, we step back and discuss FLEET in a broader context. While developing the protocol, we did not consider all possible application scenarios. In the following, we discuss the usage of FLEET*vis-á-vis* different scenarios and also how FLEET can be adapted in such cases.

In the evaluation, we assumed a maximum time bound of 1 s and showed that more than 100 flows can be scheduled within such a deadline. The design of FLEET provides the flexibility to set the deadline as required by the application without any design or implementation changes. Of course, there is a limit on how many flows can be scheduled within a given time bound. From the bounded-time perspective, the benefits of FLEET are two-fold (i) it can schedule more flows within a given time bound, and (ii) it provides a guaranteed delay bound with high accuracy. When there are only a few flows in the network, it becomes easy to accommodate all of them within the required time bound. In such cases, not only FLEET, but any other protocol would meet the deadline for all the flows. However, FLEET will still be effective as it provides significantly higher energy efficiency than any other existing protocol.

FLEET assumes a fixed priority for all the flows in the network. Thus, it targets to complete as many flows as possible within a given time. However, in many applications, flows may have different priorities. In such cases, high priority tasks need to be scheduled earlier, and if time permits only then low priority tasks need to be scheduled. To accommodate more flows, the low priority flows can be assigned a slot in every, say `S`, superframes in a round-robin fashion. A suitable scheduler on top of FLEET can tackle such priority-based scheduling while the communication mechanism remains the same.

VIII. CONCLUSIONS

Latency, reliability, and energy-efficiency are important aspects of a communication protocol in the domain of the cyber-physical system. Though many routing protocols proposed that tackle these issues partially, they do not address all of them together. We presented FLEET (Flat Latency Energy Efficient Transmission), a highly energy-efficient communication protocol that provides a bounded-time guarantee. It uses a cross-layer approach – slot-based, synchronous communication at the link layer and flooding and clustering at the network layer. Using on-the-fly clustering, which is highly dynamic, we split the data collection into two levels. This allows us to achieve parallel operations leading to higher energy-efficiency and lower latency at the same time. We compared FLEET with source routing and Blink to explain how it behaves and compared its performance with LWB and CXFS on two public testbeds. We recorded a 2.8 and 3.8 times reduction in the average duty cycle at 2.2 and 2 times reduction in data delivery latency compared to LWB on the testbeds. These benefits are primarily achieved by the use of fast flooding eliminating explicit routing and hop-by-hop scheduling for each flow. We gained further by adopting a selective flooding mechanism to tackle the inefficiency of flooding the whole network. Moreover, FLEET is the first protocol that can address both periodic and event-driven traffic equally well.

REFERENCES

[1] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications. Acm, 2002, pp. 88–97.

[2] M. Mazo and P. Tabuada, "Decentralized event-triggered control over wireless sensor/actuator networks," IEEE Transactions on Automatic Control, vol. 56, no. 10, pp. 2456–2461, 2011.

[3] G. Schmidt and P. C. B. Unit, "Getting the most out of your WirelessHART system [online]," http://www.phoenixcontact.com/, 2010, accessed: 2016-04-30.

[4] R. C. Carrano, D. Passos, L. Magalhaes, and C. V. Albuquerque, "Survey and taxonomy of duty cycling mechanisms in wireless sensor networks," Communications Surveys & Tutorials, IEEE, vol. 16, no. 1, pp. 181–194, 2014.

[5] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-time scheduling for WirelessHART networks," in Real-Time Systems Symposium. IEEE, 2010, pp. 150–159.

[6] A. Saifullah, D. Gunatilaka, P. Tiwari, M. Sha, C. Lu, B. Li, C. Wu, and Y. Chen, "Schedulability analysis under graph routing in WirelessHART networks," in Real-Time Systems Symposium. IEEE, 2015.

[7] W.-B. Pöttner, H. Seidel, J. Brown, U. Roedig, and L. Wolf, "Constructing schedules for time-critical data delivery in wireless sensor networks," ACM Transactions on Sensor Networks, vol. 10, no. 3, p. 44, 2014.

[8] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele, "Adaptive real-time communication for wireless cyber-physical systems," ACM Transactions on Cyber-Physical Systems, vol. 1, no. 2, p. 8, 2017.

[9] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Capture effect based communication primitives: Closing the loop in wireless cyber-physical systems," in Embedded Network Sensor Systems. ACM, 2012, pp. 341–342.

[10] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in Information Processing in Sensor Networks. ACM/IEEE, 2011, pp. 73–84.

[11] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in 29th Annual Conf. on Local Computer Networks. IEEE, 2004, pp. 455–462.

[12] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3d wireless sensor network testbed," in Testbeds and Research Infrastructure. Development of Networks and Communities. Springer, 2011, pp. 302–316.

[13] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in Information Processing in Sensor Networks (IPSN), 2013 ACM/IEEE International Conference on. IEEE, 2013, pp. 153–165.

[14] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in Embedded Network Sensor Systems. ACM, 2012.

[15] C. Sarkar, "Virtualizing the internet of things," Ph.D. dissertation, 2016.

[16] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in Embedded Networked Sensor Systems. ACM, 2006, pp. 307–320.

[17] A. Dunkels, "The contikimac radio duty cycling protocol," 2011.

[18] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis, "Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless," in Embedded Networked Sensor Systems, 2010, pp. 1–14.

[19] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in Embedded Networked Sensor Systems. ACM, 2009.

[20] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson, "Low power, low delay: opportunistic routing meets duty cycling," in Information Processing in Sensor Networks. IEEE, 2012, pp. 185–196.

[21] D. Puccinelli, S. Giordano, M. Zuniga, and P. J. Marrón, "Broadcast-free collection protocol," in Embedded Network Sensor Systems. ACM, 2012, pp. 29–42.

[22] N. Tsiftes, J. Eriksson, and A. Dunkels, "Low-power wireless ipv6 routing with contikirpl," in Information Processing in Sensor Networks, 2010, pp. 406–407.

[23] S. Duquennoy, O. Landsiedel, and T. Voigt, "Let the tree bloom: scalable opportunistic routing with orpl," in Embedded Networked Sensor Systems. ACM, 2013, p. 2.

[24] T. Istomin, C. Kiraly, and G. P. Picco, "Is RPL ready for actuation? a comparative evaluation in a smart city scenario," in Wireless Sensor Networks, pp. 291–299.

[25] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "WirelessHART: Applying wireless technology in real-time industrial process control," in Real-Time and Embedded Technology and Applications Symposium. IEEE, 2008, pp. 377–386.

[26] W. Specification, "75: TDMA data-link layer," HART Communication Foundation Std., Rev, vol. 1, 2008.

[27] S. Han, X. Zhu, A. K. Mok, D. Chen, and M. Nixon, "Reliable and real-time communication in industrial wireless mesh networks," in Real-Time and Embedded Technology and Applications Symposium. IEEE, 2011, pp. 3–12.

[28] B. Li, Y. Ma, T. Westenbroek, C. Wu, H. Gonzalez, and C. Lu, "Wireless routing and control: a cyber-physical case study," in Cyber-Physical Systems. ACM/IEEE, 2016.

[29] P. Soldati, H. Zhang, and M. Johansson, "Deadline-constrained transmission scheduling and data evacuation in WirelessHART networks," in European Control Conference (ECC). IEEE, 2009, pp. 4320–4325.

[30] I. P. . e 2012, "Part 15.4: Low-rate wireless personal area networks (wpans), amendment 1: Mac sub-layer."

[31] T. Watteyne, M. Palattella, and L. Grieco, "Using ieee 802.15. 4e time-slotted channel hopping (tsch) in the internet of things (iot): Problem statement," Tech. Rep., 2015.

[32] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled tsch," in Proceedings of the 13th ACM conference on embedded networked sensor systems. ACM, 2015, pp. 337–350.

[33] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6tisch: deterministic ip-enabled industrial internet (of things)," IEEE Communications Magazine, vol. 52, no. 12, pp. 36–41, 2014.

[34] T. Watteyne, J. Weiss, L. Doherty, and J. Simon, "Industrial ieee802. 15.4 e networks: Performance and trade-offs," in Communications (ICC), 2015 IEEE International Conference on. IEEE, 2015, pp. 604–609.

[35] A. Mavromatis, G. Z. Papadopoulos, X. Fafoutis, A. Elsts, G. Oikonomou, and T. Tryfonas, "Impact of guard time length on ieee 802.15. 4e tsch energy consumption," in Sensing, Communication, and Networking (SECON), 2016 13th Annual IEEE International Conference on. IEEE, 2016, pp. 1–3.

[36] S. Duquennoy, A. Elsts, A. Nahas, and G. Oikonomou, "Tsch and 6tisch for contiki: challenges, design and evaluation," in Proceedings of the International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2015), 2017.

[37] D. Karaboga, S. Okdem, and C. Ozturk, "Cluster based wireless sensor network routing using artificial bee colony algorithm," Springer Wireless Networks, vol. 18, no. 7, pp. 847–860, 2012.

[38] S. Yoon and C. Shahabi, "The clustered aggregation (cag) technique leveraging spatial and temporal correlations in wireless sensor networks," ACM Transactions on Sensor Networks, vol. 3, no. 1, p. 3, 2007.

[39] X. Deng and Y. Yang, "Cluster communication synchronization in delay-sensitive wireless sensor networks," in Distributed Computing in Sensor Systems. IEEE, 2013, pp. 36–43.

[40] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: the backpressure collection protocol," in 9th ACM/IEEE Conf. on Information Processing in Sensor Networks, 2010, pp. 279–290.

[41] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali, "Forwarder selection in multi-transmitter networks," in Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on. IEEE, 2013, pp. 1–10.

[42] D. Yuan, M. Riecker, and M. Hollick, "Making 'glossy' networks sparkle: Exploiting concurrent transmissions for energy efficient, reliable, ultra-low latency communication in wireless control networks," in European Conference on Wireless Sensor Networks. Springer, 2014, pp. 133–149.

[43] M. Brachmann, O. Landsiedel, and S. Santini, "Concurrent transmissions for communication protocols in the internet of things," in Local Computer Networks (LCN), 2016 IEEE 41st Conference on. IEEE, 2016, pp. 406–414.

[44] V. S. Rao, M. Koppal, R. V. Prasad, T. Prabhakar, C. Sarkar, and I. Niemegeers, "Murphy loves ci: Unfolding and improving constructive interference in wsns," in IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. IEEE, 2016, pp. 1–9.

[45] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-time wireless sensor-actuator networks for industrial cyber-physical systems," Proceedings of the IEEE, vol. 104, no. 5, pp. 1013–1024, 2016.