SpeedCollect: Data Collection Using Synchronous Transmission for Low-Power Heterogeneous Wireless Sensor Network

Ebram Kamal William
National University of Singapore
ebramkw@comp.nus.edu.sg

Mun Choon Chan
National University of Singapore
chanmc@comp.nus.edu.sg

Abstract

In this paper, we present the design of SpeedCollect, an optimized many-to-one communication that uses capture effect based synchronous transmission. SpeedCollect is designed to be more reliable to network characteristics by requiring only simple network topology information that can be easily supported. In addition, SpeedCollect provides more flexibility in the application computation's timing to ease application and protocol development.

We have designed and implemented SpeedCollect on Contiki OS. SpeedCollect can run on three hardware platforms using the same source code without modification. The evaluation shows that SpeedCollect can reduce the latency of data collection from a base many-to-many communication protocol by an average of 20%. Compared to the state-of-the-art, SpeedCollect can increase data collection throughput by up to 3.4x.

1 Introduction

Wireless sensor network (WSN) plays an important role in smart cities and industry 4.0 [23, 31]. WSN needs flexible and reliable protocols that can adapt to its dynamic characteristics [5, 16, 27] and low-power constraints [1, 15]. Data collection is a common use case in WSN. With more sensors modality, a massive amount of data communication is required which motivates the research direction for high throughput and efficient data collection. Traditional protocols in WSN use MAC protocols to manage collision (e.g. CSMA) and routing protocols (e.g. Collection Tree Protocol (CTP) [14], RPL [34]) that require link estimation and end-to-end route discovery. These MAC and routing protocols incur substantial overhead that significantly impacts throughput, latency, and energy efficiency.

Use of synchronous transmission in WSN protocol designs have provided some unique advantages that overcome some of the limitations of traditional approaches [37]. Pioneered by Glossy [13] many synchronous transmission based protocols have shown to achieve high reliability and low latency. By using broadcast-based communications, these protocols avoid the need for link estimation and end-to-end route discovery, thus substantially reducing the overhead incurred.

Many existing protocols are based on synchronous transmission such as LWB [12], Blink [36], and Chaos [22]. Hence, even if a data collection application that requires many-to-one communication is to be supported, these protocols still have to perform many sequential one-to-all communication rounds. The performance of Glossy-based protocols are also very sensitive to small drift in timing and thus are harder to be ported to a more diverse set of devices.

Another approach to implementing synchronous transmission is to rely on capture effect. This is the approach taken by Codecast [29], Mixer [17], Crystal [19], and Weaver [32]. With capture effect, these protocols have less stringent timing requirements and have the potential to be more portable to new platforms. However, Codecast and Mixer are designed to support many-to-many communication and are not optimized for many-to-one communication while Crystal and Weaver operate by performing many concurrent one-to-all Glossy floods.

In this work, we aim to answer the following questions. (1) Since many-to-one communication is a common use case in WSN, rather than supporting general many-to-many communication, can we do better if the protocol is optimized to support many-to-one communication? (2) How can the protocol be designed such that it imposes minimum timing requirements on both processing and communication? Removing these timing constraints would make the protocol easier to modify and port to different platforms, important considerations for practical deployment.

Many-to-one communication offers an opportunity for improvement over many-to-many communication since data is needed to be delivered to a single node (sink). On the other hand, a key performance bottleneck is the congestion/contention around the sink whereby all the nodes are trying to deliver to the sink. One surprising observation we have is that sharing the state of the data items collected by the sink through acknowledgments does not significantly reduce the data collection latency by itself. Instead, a small amount of topology information, in the form of strong 1-hop

bidirectional neighbors of the sink, goes a long way.

Taking these observations into account, we present SpeedCollect, a data collection protocol that uses capture effect based synchronous transmission. Building on a base layer that uses local broadcast and network coding, we incorporate additional features that include (1) acknowledgments from sink and neighbor nodes, (2) topology information of one-hop neighbors of the sink, (3) schedule transmissions of the sink's 1-hop neighbors, and (4) a polling mode by the sink node.

In terms of implementation, SpeedCollect does not impose strict timing requirements. Similar to Mixer [17], SpeedCollect's design decouples the communication and computation layers to make porting and programming easier. The difference is that the processing time in SpeedCollect can be varied independently of the communication slot length and there is no need to ensure that packet processing has to be completed within a strict time limit.

The contributions of this paper are as follows:

- We present the design of SpeedCollect, an optimized data collection protocol that uses capture effect based synchronous transmission. SpeedCollect incorporates sink/local acknowledgments and collision management at the sink node (using TDM and polling) to improve data collection performance.
- SpeedCollect allows application computation to be more flexible, while allowing communication slot time to be reduced independently from the computation time. Specifically, it requires software timers that support resolutions in orders of tens of microseconds, a requirement that can be easily supported on existing platforms.
- We have implemented SpeedCollect on Contiki OS. SpeedCollect requires minimum support from the underlying hardware and has been ported to run on TelosB, CC2650 SensorTag, and Zolertia. The three platforms operate in the 2.4GHz spectrum. The same source code can be used on all three hardware platforms with only changes needed in the compilation flags for the different targets.

In the evaluation performed on the Indriya testbed [3], we show that SpeedCollect can reduce the latency of data collection from a base many-to-many implementation by an average of 20% when many-to-one enhancements are included. Compared to Crystal and Mixer, SpeedCollect increases data collection throughput by up to 3.4x when collecting 60 bytes of data from 33 nodes. In particular, SpeedCollect has a more robust performance compared to Crystal and Mixer when the sink node is varied and works well even in scenarios with a poorly connected sink.

The paper is organized as follows. Background and related work are presented in §2. The design of SpeedCollect is presented in §3, followed by the design of the underlying communication protocol in §4. The evaluation details are presented in §5. Finally, we conclude in §6.

2 Background & Related Work

The conventional approach in wireless protocols is to avoid collisions due to concurrent wireless transmissions because such collisions result in likely packet losses. CSMA and TDMA MAC protocols are examples of such approaches. On the other hand, the idea of synchronous transmission presents a different design trade-off. In synchronous transmission, collisions do not always lead to packet losses. Instead, under specific circumstances, it is possible for a receiver to successfully receive a packet even in the presence of superimposed signals from multiple transmitters.

There are generally two forms of synchronous transmissions that researchers have exploited. The first is the so-called "constructive interference" introduced by Ferrari et al. in [13] and the second is based on capture effect [25]. An overview of synchronous transmission and a survey of related protocols can be found in Zimmerling et al [37].

Glossy-Based Protocols Glossy is a protocol proposed by Ferrari et al. [13] that utilizes the observation that reception has a high likelihood to be successful if multiple senders transmit identical packets with small-time offsets among the transmitted signals. In Glossy's design, using the IEEE 802.15.4 radios, multiple senders have to initiate their transmissions with a tiny temporal difference that should be smaller or equal to 0.5μs. Based on performing *network-wide flooding of the same packet*, Glossy can achieve high-resolution time synchronization in multi-hop networks. For example, the evaluation shows that Glossy can achieve an average synchronization error of 0.4μs over 8 hops.

Many synchronous transmission protocols were subsequently developed based on Glossy, including low-power wireless bus (LWB [12]), data dissemination (Splash [8]), and point-to-point communication (P^3 [9]). As these protocols are based on Glossy, operations are performed using many rounds of one-to-all flooding of the same packets and strict time constraints need to be followed to ensure high reliability.

Capture Effect Based Protocols Capture effect [25, 4, 11, 24, 35] is a well-known phenomenon in wireless transmission. Successful packet reception can occur when the signal strength of one transmission dominates over all the other transmissions. Specifically, the receiver can decode a packet if the received signal from one node is at least 3 dB stronger than the sum of the received signals from all other nodes. However, the difference in signal strength alone is not sufficient to ensure packet reception. Timing matters as well. For capture effect to be successful, the competing transmissions have to be transmitted within the preambles of other transmissions. For the IEEE 802.15.4 ZigBee radio, this is about 128μs, which is more than 200 times larger than the timing requirement of Glossy. Thus, synchronous transmission protocols based on capture effect have the benefit that the timing requirement is much more relaxed, allowing these protocols to have the potential to be implemented on a more diverse set of platforms. For receptions to be successful, difference in signal strength is the determining factor. Thus, packet reception can be successful even if *multiple transmitters transmit different packets*. This is an important difference that provides significant flexibility to the protocol design over Glossy-based protocols.

Several capture effect based protocols have been proposed, including for many-to-many communications (Codecast [29], Mixer [17]), network agreement (Chaos [22] and

A^2 [2]), network flooding [26], collision resolution [33], and data collection (Crystal [19] and Weaver [32]).

Protocols like Chaos and A^2 are designed as network services and are not suitable for dissemination of large amount of data. On other hand, Codecast and Mixer are designed for many-to-many data dissemination. Since packet reception using capture effect is much less reliable than Glossy floods, network coding is used in Codecast and Mixer to improve reliability and to increase the rate of useful information flow in order to achieve higher throughput. Both protocols support general many-to-many communication and do not optimize for many-to-one communication pattern. Crystal and Weaver operate by performing many concurrent one-to-all Glossy floods. In Crystal and Weaver, a fixed initiator (sink) starts the communication with a Glossy synchronous flood. After the end of the Glossy round initiated by the sink node, all the source nodes start Glossy rounds competing to deliver their values to the sink which will initiate a Glossy round to acknowledge the value received and the corresponding sender will stop sending its value in the subsequent rounds.

Summary SpeedCollect differs from previous work in the following ways. First, SpeedCollect does not need many rounds of one-to-all flooding of the same packet as is the case in Crystal and Weaver. Such protocols are designed for less number of sources as the probability of collisions will be increased with more sources competing to deliver their messages to the sink. Second, while SpeedCollect also uses capture effect and network coding like Codecast and Mixer, it is specifically designed to provide better performance for many-to-one communication.

3 Data Collection Design

In many-to-one communication, a gateway node (or sink) collects data from sensor nodes which is a common usage scenario in WSN. In this section, we first present an overview of the data collection challenges and then techniques that address these challenges. By incorporating these techniques, the sink can perform data collection with high reliability and low latency even when the sink node has poor connectivity.

3.1 Overview

Even though many-to-one communication can be supported as a special case of many-to-many communication, if the objective is to specifically improve the performance of many-to-one communication, there are specific challenges that need to be addressed:

1. In many-to-one communication, it is required to deliver all the messages from the entire network to one node. Hence, besides needing to deal with unreliable wireless links in general, a poorly connected gateway poses even more severe challenges.
2. Directing the traffic toward the sink is a challenge as it requires some knowledge of network topology and location of the sink node. As wireless links are unreliable and asymmetric links are common, the overhead of approximating the topology can easily negate any gain from using synchronous transmission.
3. With a many-to-one communication pattern, nodes close to the sink are often required to relay data for all other nodes to the sink. This naturally leads to high transmission rates and likely collisions around the sink node.

To address the first challenge, we incorporate an important key design element with the main idea of replicating the messages from their originator to many other nodes in the network. This can help handle the first challenge as even with a poorly connected gateway, having the same messages replicated at many nodes in the network will help to increase the delivery probability to the sink.

To achieve message replication, we use local broadcast performed using capture effect based synchronous transmission as a basic layer. However, such local broadcast has low reliability. In SpeedCollect, we overlay the floods by letting nodes mix packets which achieves high efficiency with many-to-many communication [10, 21]. More details will be discussed in Section 3.2.

For the second challenge, when it comes to directing the traffic towards the sink, using topology information can help. This can be done by defining the node position relative to the sink node using the hop count for each node from the sink. However, such an approach has limitations due to the existence of asymmetric links whereby the link quality in one direction is much better than in the other direction. Link asymmetry makes the usage of hop count from the sink much more complex since for the distance to work well, we may need to determine the link quality between two nodes in both directions. For example, a node that can hear the sink may decide that it is one hop from the sink. However, if the link is asymmetric and the sink cannot receive from this node, then the node should not be considered a neighbor of the sink as it cannot deliver any message to the sink directly. Such link asymmetry issue applies to all nodes since nodes "closer" to the sink may not be able to deliver messages to nodes "further" from the sink. To address this challenge, we propose to use the notion of distance from the sink only for nodes that are bidirectional neighbors of the sink. While extra overhead will be incurred, the number of such bidirectional neighbors of the sink is small. If the link quality is sufficiently high, these links have also been observed to be very stable [7]. The use of these bidirectional neighbors is presented in Section 3.3.

Finally, with the last challenge, a simple coordination scheme among the sink node and its 1-hop bidirectional neighbors, using Time Division Multiplexing (TDM) and polling, can significantly mitigate congestion around the sink. Note that TDM is only for communication between the sink and its bidirectional neighbors. Synchronous transmission is used everywhere else.

3.2 Network Coding

Many network coding schemes have been presented in the literature (LT codes [6], RL codes [18], Raptor codes [30], Online codes [28], and Growth codes [20]). Network coding scheme addresses the issue of **what to transmit**. However, given that capture effect is used, constant transmissions will result in a very high collision and low reliability. So another important issue is to determine **when to transmit**.

In SpeedCollect, we use a combination of two techniques, namely, the network-assisted network coding (NANC) intro-

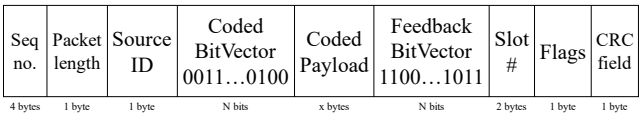


Figure 1: SpeedCollect's packet structure.

duced in Codecast [29] and Random Linear Network Coding (RLNC) [18]. Codecast uses neighbors' feedback to help in the decision of what and when to transmit. In NANC, nodes encode new vectors to send from what they have already decoded. Crafting the new packet from what is already decoded allows for choosing the degree (number of data items to be added) of the new coded bit-vector and its corresponding coded payload. This is useful as it gives a high level of control on what to send next and reduces uncertainty. However, depending only on this technique has a limitation as it does not allow nodes to create combinations from received packets that are not yet decoded. These limitations are addressed by using Random Linear Network Coding (RLNC) which allows nodes to send random linear combinations of received packets. In RLNC, sending nodes generate a new packet by adding random rows from its coding matrix and the received coded payloads.

Figure 1 shows the layout of SpeedCollect's packet. The structure has the following fields:

- Source node ID: which is used for the sink 1-hop bidirectional neighbors.
- Coded bit vector: meta-information for the message in the coded payload.
- Coded Payload: the payload of the packet.
- Feedback bit vector: which is used by nodes to feedback on their current status.
- Slot #: for slot number consistency between the sink and its neighbors.
- Flags: for signals initiated by the sink nodes.

Following, we will describe the network coding design choices that determine what and when to transmit?

What to transmit? Deciding on which messages to include in each transmitted packet is a decision that takes into account the node status, neighbors' status, and the stage of dissemination. We break the decision process into four parts: (1) at the beginning of a round, (2) rank increase, (3) based on what is decoded, and (4) based on what is received (not yet decoded).

- At the beginning of each round, each node with data starts by sending its data items which are considered new innovative information for its neighbors.
- When a node receives a new packet, it uses the received coded packet and the coding matrix to check if a new innovative packet is received. The packet is innovative if the rank of the coding matrix increases. Whenever a node increases the rank of its coding matrix, it will include the newly added row in the packet to be transmitted. This is to ensure it is sharing what it has recently learned from its neighbors. This is justified by the fact that if this information is new to a node, it will be useful to its neighbors with a high probability.
- Sending nodes to enter into a combined encoding process to build a new packet. The first is using what a node has already decoded and based on the feedback from its neighbors. The first step is more NANC-based. Each node chooses a degree based on what it has already decoded and on the least rank among its neighbors. The chosen degree determines how many decoded data items will be used to build up the new packet. Using this degree, a node will start by adding what it has and is missing from its recently connected neighbor. If the number of added data items is less than the prechosen degree, intersected data items between the node and its neighbors can be added. Adding more data items may help other neighbors that have not received these data items.
- If the first encoding process failed to create an innovative packet to the neighbor, another encoding mechanism is used which is based on RLNC. A node will loop through its coding matrix starting from a missing row in the recent feedback. If the missing row is the same as the last choice, the node will start the loop from a random position. Looping through the coded matrix, each row will be added to the packet until the resulting packet is innovative to the recent neighbor or until predefined threshold rows are added.

When to transmit? A node transmits whenever it learns something new (increases the rank of its coding matrix). When there is no new information, a node transmits with probability. In all the experiments, this probability is set to 0.1.

3.3 Bidirectional Neighbors

The sink's 1-hop bidirectional neighbors are those nodes that have links of high reliability in both directions. These bidirectional neighbors perform their tasks differently in two ways. First, these nodes control what all other nodes in the network transmit based on what they hear from the sink. Second, these nodes focus on helping only the sink, while other nodes in the network focus on disseminating new information to all their neighbors.

3.3.1 Acknowledgements

All nodes in the network use local ACK for feedback on what they have already received. This will help their neighbors to decide on the proper degree as well as what to include in the next packet. Using local ACK between nodes in the network is motivated by the idea of replicating the messages to many nodes in the network. The 1-hop bidirectional neighbors use different feedback compared to the rest of the network. Each of these nodes performs a union of what it has and what the sink node has already received to compute what we called the *union ACK*. Nodes receiving the union ACK incorporate this ACK information into their state. As a result, the rank of the completed matrix seen by other nodes

is the union of all the data received by the sink and all the bidirectional neighbors. In this way, nodes will prioritize these "unseen" data items reach the bidirectional neighbors faster and hence to the sink.

What to send? The 1-hop bidirectional neighbors will always use the feedback coming from the sink when encoding a new packet to send. For the first part of the combined encoding which is based on NANC, when these nodes build a new packet from what they have already decoded, they will only add what they have and missing at the sink. If the added rows are less than the chosen degree, they will not add from the intersection between them and the sink. This is to focus on always helping the sink node and directing the traffic towards it.

When to Send? A well-known challenge to address in capture effect based synchronous transmission is how to increase reception reliability while ensuring that some node is transmitting. If the decision is probabilistic and not coordinated, when the nodes are transmitting too aggressively, we end up losing a lot of packets due to collision. This is particularly problematic around the sink.

This motivates us to incorporate the information of the sink 1-hop bidirectional neighbors in the transmission decision. In SpeedCollect, the sink and its 1-hop bidirectional neighbors use a TDM scheme to decide on which slot to transmit. This is done using a slot number shared by the sink in each transmitted packet to update its neighbors. Note that such a scheme does not disable capture effect as other nodes in the network still perform probabilistic transmission decisions. Further, there can be inconsistency in the values of the slot number seen by the neighbors caused by packet losses and loss of synchronization. Nevertheless, adding the TDM scheme significantly reduces collisions around the sink while retaining high channel utilization.

Sink Node As the sink node does not have data to transmit, it also behaves differently than the rest of the network. First, it only needs to send/update feedback in the form of acknowledgments to the rest of the network. The sink sends based on the TDM allocation. In addition, it will only send when it has learned a sufficient amount of new information, determine by a sufficiently large change in the rank of its coding matrix.

3.4 Polling

In this section, we discuss another design mechanism which can mitigate congestion at the sink. This is motivated by the observation that if a sink's 1-hop bidirectional neighbor has a significant amount of new data, it can be more efficient for this neighbor and the sink to simply switch to another channel to communication rather than wait for the neighbor to transmit on the allocated TDM slot. In SpeedCollect, the sink can thus instruct a specific bidirectional neighbor to switch to another channel using a polling mode.

Polling uses a two-way handshake fashion. When the sink node decides to poll one of its neighbors, it adds the information inside the flag but it does not switch to the other channel until it receives an acknowledgment from the polled neighbor on the switch. Once the polled neighbor receives the switch flag from the sink, it acknowledges the message and then switches to the other channel.

One limitation of polling is that the switch command from the sink and the acknowledgment from the polled neighbor can be lost. The choice of a two-way handshake fashion is to make sure that if the switch is not completed properly by the two parties, the waste will be more on the polled neighbor rather than the sink as the polled neighbor will be the one switching first to the other channel. Another limitation of polling is that the sink can receive packets from other nodes between sending the switch command and receiving the acknowledgment making the switch redundant or less useful.

Due to these limitations, polling tends to work better in cases where the 1-hop bidirectional neighbors have more data to send initially. In such scenarios, the benefit of polling becomes more significant.

3.5 Smart Shutdown

The network shutdown is capped by the maximum number of slots in each communication round. However, we would like to terminate faster with a simple smart shutdown mechanism. The termination in data collection is determined by the sink node as the target is to deliver data to this specific node. The termination decision cannot be locally driven by individual nodes as even if a node delivers its data to the sink, it could be needed to relay other messages from the network. In SpeedCollect, the termination decision comes from the gateway node and is taken in two steps.

The first termination step is based on the fact that the sink node monitors the feedback from its bidirectional neighbors. Once the union of its bidirectional neighbors' feedback reaches full rank, the sink can send the rest of the network to sleep. This comes from the idea that these neighbors are reliable to deliver what they have to the sink. The second termination step comes when the sink reaches full rank. The sink node sends another sleep signal which puts the entire network into sleep.

Whenever a node receives a sleep signal, it will send the signal in the next transmitted packets for a predefined number of transmissions and then sleep if it's eligible for sleeping. A node in the group of 1-hop bidirectional neighbors is eligible to sleep on the second sleep signal while other nodes in the network can sleep after receiving the first sleep trigger.

4 Communication Layer

The design of SpeedCollect is driven by the goal to have a relaxed timing synchronization requirement. We implement our version of capture effect based synchronous transmission to decouple the communication and processing layers so that programming using SpeedCollect has a minimum timing constraints on the processing.

The typical implementation of communication in synchronous transmission communication protocols (e.g. [13], [12], [22], [29], [17], [19], [32]) is based on a fixed cycle with a bounded time given for packet reception/transmission and computation. The communication and computation cycles are tightly coupled. Given that the interval between communication slots needs to be small to maintain time synchronization, there is a limit on the time between consecutive

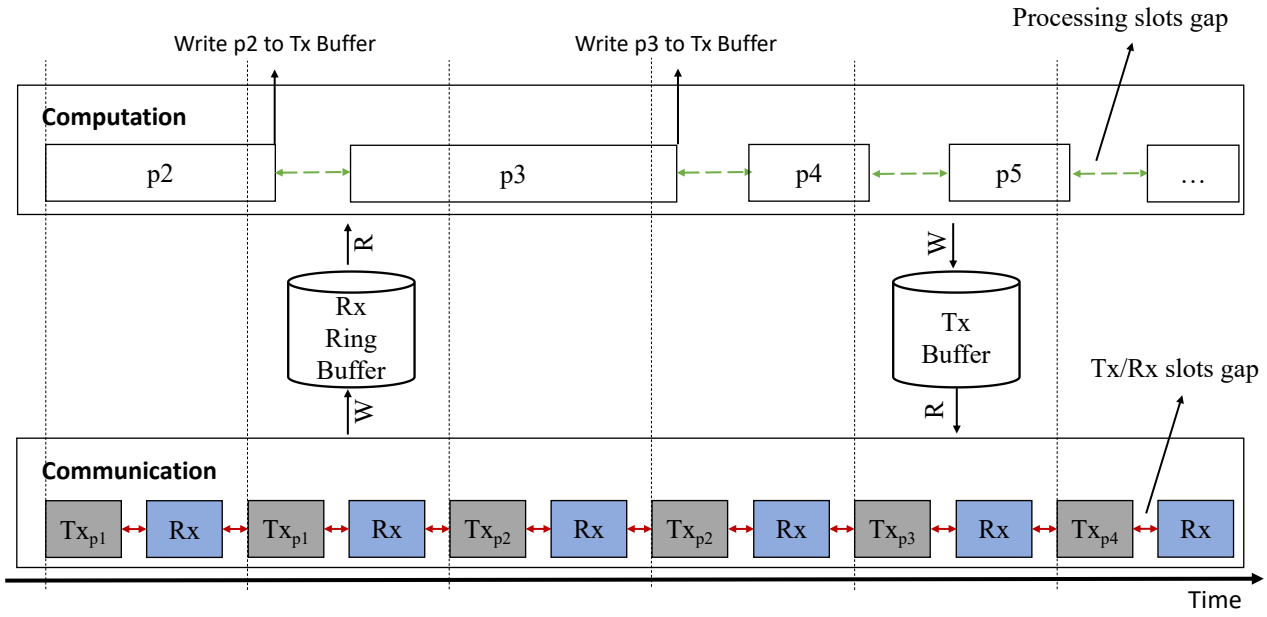


Figure 2: Overview of Communication and Computation Separation in SpeedCollect.

communications slots that can be used for computation. This coupling also results in a trade-off between communication slot time and computation slot time. If the time intervals between packet transmission and/or receptions are too small, there may be very little time left for computation. If the intervals are made larger, the throughput goes down, or latency increases since the amount of communication that can be done per unit time reduces.

SpeedCollect uses two separate processes for communication and computation to allow a complete decoupling of processing and transmissions/receptions. There is no dependency between the two processes during the round. They only communicate through shared buffers. The timing is decoupled in the sense that packet processing can take longer than a single communication slot. This separation gives the designer full control over how the communication and computation processes can work. The designer can choose to give more time for processing to support computational applications and yet choose short communication slots to increase communication and reduce latency.

Core Architecture: The communication layer is presented in Section 4.1 and the computation layer in Section 4.2. Figure 2 shows the relation between the two layers.

4.1 Communication and Tx/Rx Buffers

Once communication commences, the communication process is a time-grid slot of transmissions and receptions of fixed time intervals. Scheduling the next Tx/Rx slot is affected by the last Tx/Rx which implicitly uses the last Tx/Rx to be a reference time for the next Tx/Rx. The duration of each time grid consists of two components. The first component is the packet on-air transmission time. We assume that the packet sizes are the same throughout. Note that even though the packet sizes are the same, there can be still small variations in the packet transmission times. The second component is a timer-driven time gap between each TX/RX interval. This is called the **TX/RX slot gap** in Figure 2. Such gaps are used to provide buffers for variations in radio send/read times and clock drifts on different nodes. One can increase the gap length to tolerate a higher level of variations. The number of slots in a communication round determines the maximum time of the round. Nodes can terminate before the end of the round depending on the termination policy of the application.

We would like to emphasize that the time-grid slots are only needed during the communication round and not needed between rounds. Also, nodes are not synchronized to their neighbors or fixed initiator. Nodes lock onto their time grid after the reception of the first valid frame until communication for this round completes.

On the first successful reception, the new packet will also be inserted into the *Tx Buffer*. This is to ensure that there is always (exactly) one packet in the *Tx Buffer*. In transmission slots, a node transmits the packet in the *Tx Buffer*. By having only one packet in the *Tx Buffer*, the application can directly determine which packet will be transmitted next. SpeedCollect allows the application to specify the likelihood of a transmission with a value between 0 and 1. Hence, the application can ask the communication layer to re-transmit the same packet many times, transmit less often, or transmit the most recent data it has received by overwriting the

packet in the *Tx Buffer*. The decision on whether the node should transmit or not is thus determined by the computation process based on the application logic.

4.2 Computation

The computation process is triggered when a node decides to join a communication round (after receiving the first valid frame and writing it into the *Rx Ring Buffer*). Computation in SpeedCollect differs from existing synchronous transmission based protocols in fundamental ways. As illustrated in Figure 2, there is no need for computation to start and end within a communication slot. The duration of a computation can be much longer than a communication slot as shown in the left part of Figure 2 (e.g. $p2$ and $p3$). Thus, we do not control the actual processing time but instead, control the time gap between processing slots. At the end of a computation slot, a timer is set to trigger the next computation slot. This time gap is needed to release the processor resources for other processing tasks to be completed. This is called the **processing slot gap** in Figure 2. Processing times are thus completely decoupled from the transmissions/receptions times.

When the computation process is triggered, it checks on the *Rx Ring Buffer*. A new packet, if any, will be processed. When processing ends, a new packet, if any, is written to the *Tx Buffer*. After that, the process sets the timer for the next slot and then goes to sleep.

5 Evaluation

The performance of SpeedCollect has been evaluated on Indriya2 [3]. Indriya2 has two types of motes namely, TelosB and CC2650 SensorTag deployed over three floors. During the evaluations, Indriya2 has 34 TelosB and 17 CC2650 SensorTag available to run the experiments. Additional experiments were done in a lab environment on a desk for evaluation on additional hardware platforms not supported by the testbed. Each experiment runs for a maximum of 300 communication slots.

The following metrics are used:

1. Completion time: the duration between the time the sink wakes up to the time it collects all the data.
2. Network radio-on time: the average time for all the nodes from when they wake up until they sleep.
3. Reliability at the sink: the percentage of (unique) data from all sources received by the sink over all experiments.
4. Throughput at the sink: the size of (unique) data from all sources received by the sink divided by the time taken over all experiments.

We first evaluate the performance of different design mechanisms of SpeedCollect (Section 5.1). Next, we evaluate the performance of SpeedCollect using Crystal [1] and Mixer [2] as the baseline (Section 5.2). Next, we demonstrate the effect of varying the Tx/Rx slots gap on the performance of SpeedCollect by running SpeedCollect on a small network of devices and vary the Tx/Rx slots gap (Section 5.3). Finally, we show the performance of SpeedCollect running on a network of three different types of sensor devices in Section 5.4.

Unless otherwise stated, all settings in Crystal and Mixer are set to the default values. In Crystal, the default slot length is 10ms for the synchronization slot, 8ms for the transmission slot, and 8ms for the acknowledgment slot. In Mixer, the slot length is set to 4ms. For SpeedCollect, the Tx/Rx slots gap and processing slots gap are set to 3ms and 2ms, respectively. In all the experiments, the message payload size is set to 60 bytes. Note that since we set the Tx/Rx gap length to 3ms, the corresponding communication slot length for SpeedCollect (Tx/Rx gap + Tx/Rx time) is around 5.6ms.

5.1 Design Elements

In this section, we evaluate the effect of different design elements on performance. We focus on evaluating the data collection optimization mechanisms. We run four different versions, each adding a new feature to the previous version, starting with a base layer:

- **Basic** The base layer uses capture effect and network coding. The transmission probability of the sink is reduced since the sink mostly receives data.
- **ACK** In this version, bidirectional neighbors to the sink and sink acknowledgments are added to **Basic**.
- **TDM** In this version, TDM transmission among the sink's bidirectional neighbors is added to **ACK**.
- **Polling** In this version, a polling mode by the sink is added to **TDM**.

As the performance varies with the choice of the sink node, we show the result for a specific sink node to illustrate the overall trend. Results for different sink nodes will be presented later. In this evaluation, all nodes start with one data item with a payload of 60 bytes to be sent to the sink.

Figure 3 shows the average number of successful receptions at the sink and the median completion times at the sink. The error bars show the range between the maximum and minimum values.

Note that successful receptions do not always imply that the reception is useful (new information learned). The utility of the receptions is reflected by the completion time. The performance of Mixer is included as a baseline. There are at least 50 runs for each version.

Besides differing in the detailed implementations of network coding and transmission probabilities, Basic also differs from Mixer in the sink's transmission probability. The results show that **Basic** outperforms Mixer in terms of a slightly higher average number of successful receptions at the sink and shorter sink completion latency. However, when the sink's bidirectional neighbors and the sink's feedback are added, there is surprisingly little improvement in terms of latency even though there is an increase in terms of successful receptions at the sink. This can be explained as follow. While adding acknowledgments helps to reduce redundant transmission, there is a substantial collision at the sink. Without a coordinated transmission from the neighbors, the impact on the latency is minimum.

[1] https://github.com/d3s-trento/crystal

[2] https://gitlab.com/nes-lab/mixer

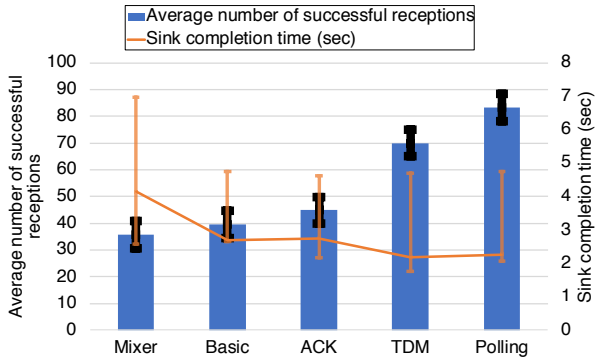


Figure 3: Average number of successful receptions and sink completion time at the sink under different versions of the proposed mechanism.

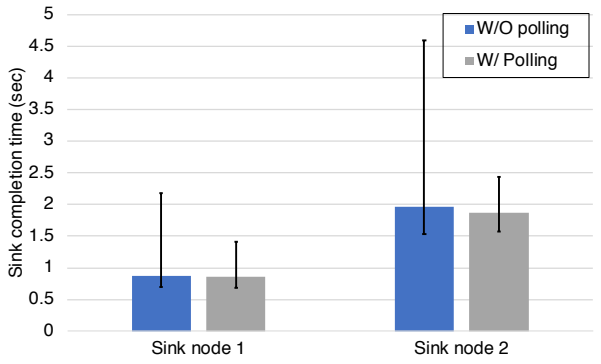


Figure 4: Sink Completion time using different message distribution using SpeedCollect without and with polling design mechanism.

The results improve significantly when TDM is added to manage the bottleneck around the sink node. Scheduled transmissions from bidirectional neighbors increase the average number of successful receptions at the sink and reduce the median completion time at the sink by 56% and 25%, respectively over ACK. Note that performance improvement builds on the use of acknowledgments.

The last feature to be added is polling, where the sink node can ask a certain neighbor to switch to another channel to exchange packets and back to the original channel after that. The result shows that while polling can increase the number of successful slots, it does not decrease median latency. This can be explained as follow. Polling incurs overhead for channel switching and in some cases, we have observed that it takes several message exchanges before channel switching can be completed successfully. Hence, without sufficient data on the bidirectional neighbors, polling may not provide much benefit.

Overall, in this evaluation, comparing Mixer to SpeedCollect (with all features included), we see a 133% increase in the average number of successful receptions and a 45% reduction in the median completion time at the sink.

When Polling is Useful: To evaluate the impact of polling, we redistribute the data items such that the sink's bidirectional neighbors start with 5 messages, while the other nodes have 1 data item each. Hence, the bidirectional neighbors have more data to transmit than other nodes.

Figure 4 shows the median completion time of 2 different sink nodes running SpeedCollect without and with polling. The error bars represent the maximum and minimum values. As shown in the figure, while the median completion times do not change significantly, the maximum latency is much smaller. This shows that in cases whereby the latencies are much longer, polling can be useful when the overhead of channel switching is small compared to the overall latency.

5.2 Data Collection

In this section, we evaluate the overall performance of SpeedCollect using Crystal [19] and Mixer [17] as the baseline protocols. Each experiment runs on a network of 34 TelosB nodes, with 1 sink and 33 sources. Each source shares 1 message of size 60 bytes payload. We show the results for 3 different sink nodes with different completion times on Indriya. We run each experiment for at least 110 iterations to compute the average values.

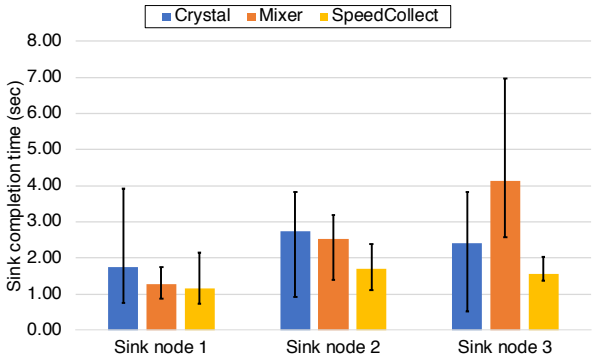
Figure 5 shows the completion time, network radio-on time, reliability at the sink, and throughput measured at the sink. We have chosen a node (sink 1) with good connectivity, a node with average connectivity (sink 2), and a corner node with bad connectivity (sink 3).

Figure 5a shows the median completion time. The error bars in the figure represent the minimum and maximum completion times. As shown in the figure, for all the cases with different sink nodes, the completion times for SpeedCollect are less than Mixer. For sink node 3, the median completion time of SpeedCollect is up to 62% lower than Mixer.

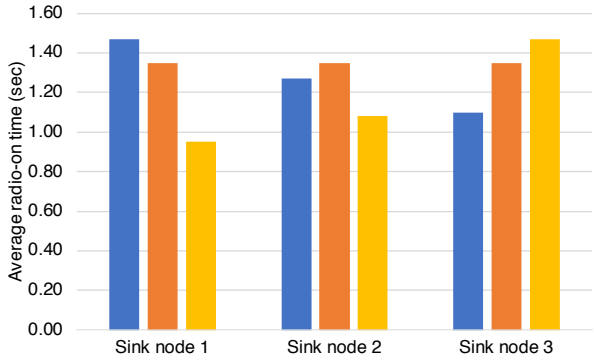
The case is different for Crystal where we can see a higher completion time compared to Mixer and SpeedCollect in the case of sink node 1. However, the completion time is lower for the poorly connected sink nodes 2 and 3. This can be explained by the fact that with the poorly connected sink, the sink node is not able to receive information for subsequent rounds which results in a time-out at the sink in some cases and the sink node goes to sleep prematurely. This is also indicated by the lower reliability shown in Figure 5c.

Figure 5b shows the network radio-on time. For Mixer, the network radio-on times remains more or less the same regardless of the node chosen as the sink. The radio-on time for each node is determined based on a local decision on their completion and their neighbors' completion. Hence, some nodes can go to sleep much early when the sink is still running. For Crystal, nodes locally decide to sleep if they are not able to receive for multiple rounds which results in lower radio-on time for sink node 3 compared to sink node 1. However, this maps to very low reliability. For SpeedCollect, the network goes to sleep on a sleep signal generated by the sink node. When the sink node has very bad connectivity, it can take more time for the sink to collect data and thus delay the trigger of the sleep signal.

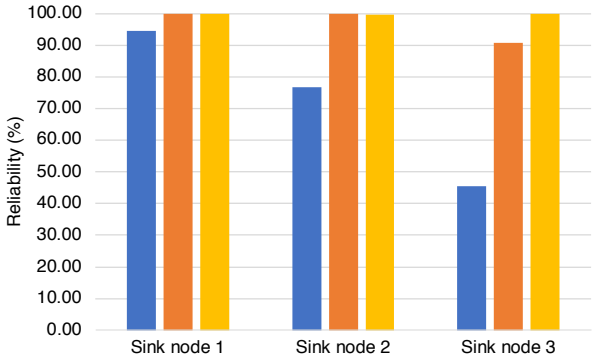
Figure 5c shows the reliability at the sink node. As shown in the figure, SpeedCollect can be more robust than Crystal and Mixer.



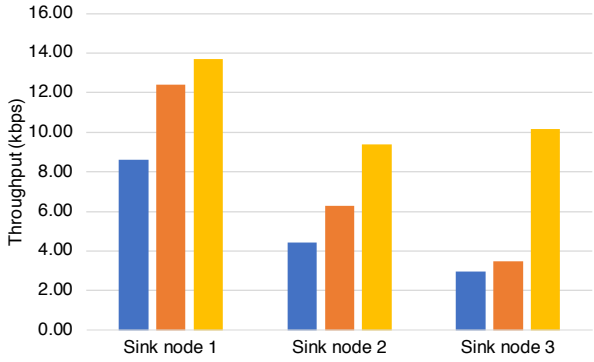
(a) Sink completion time (sec).



(b) Network average radio-on time (sec).



(c) Sink reliability (%)



(d) Sink throughput (kbps).

Figure 5: Performance of SpeedCollect vs. Mixer on Indriya testbed using different nodes as the sink.

Figure 5d shows the throughput at the sink node. The throughput shown is calculated over all experiments, including those that did not successfully receive all data items, by keeping track of the number of unique data items received by the sink within the given time.

SpeedCollect can achieve higher throughput in all cases compared to Crystal and Mixer and the improvement is up to 2.3x for a sink node with bad connectivity (sink node 3). It is expected for Mixer's completion time to increase and reliability to decrease with bad connectivity at the sink node as it's harder for nodes to deliver their data to the sink. It's even more challenging for Crystal as it is designed for a small number of concurrent sources and it does not work as well with a poorly connected sink.

Figure 6 shows the CDF of sink completion times of SpeedCollect and Mixer using 5 different nodes. For each sink node, we have up to 100 different runs. As shown in the figure, while the completion times of SpeedCollect are concentrated within a smaller range between 0.67s to 4.3s, the completion times of Mixer vary over a much large range with about 10% of the completion times larger than 4s.

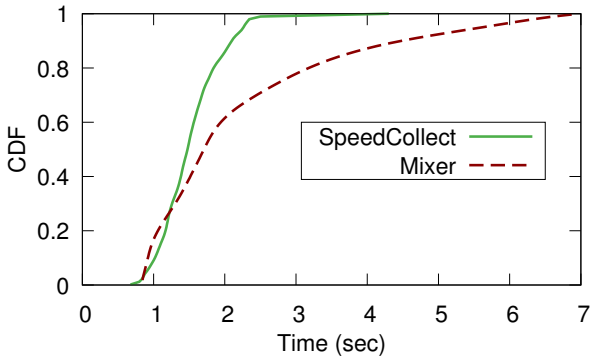


Figure 6: Sink completion time CDF on Indriya testbed of running experiments using a different sink node in each run with a total of five different nodes to be the sink.

5.3 Varying the Tx/Rx Slots Gap

We experimented to show how the completion times vary when the Tx/Rx slots gap is varied. Note that while a smaller gap time can improve performance by allowing more Tx/Rx in a fixed duration, the intervals between Tx/Rx should also be large enough so that sufficient processing can be completed to generate new information.

In the experiments, 6 nodes served as sources sending data to the sink. All source nodes are 1-hop neighbors of the sink. Each of the source nodes sends 8 60-byte data packets to the sink. For SensorTag, the Tx/Rx slots gap time is varied from 0.1ms to 3ms. For TelosB, the Tx/Rx slots gap time is varied from 3ms to 10ms. We run each set for at least 10 rounds.

Figures 7 and 8 show the sink median completion time in

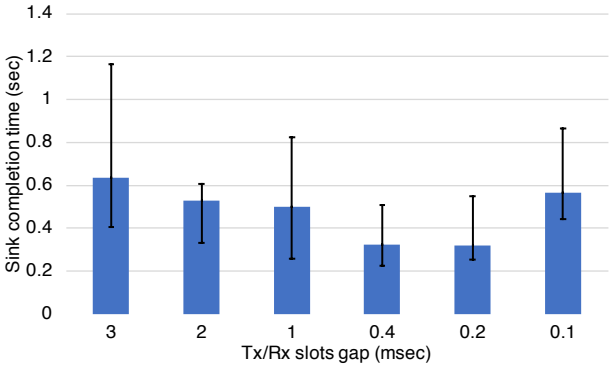


Figure 7: SpeedCollect running on 7 CC2650 SensorTag nodes on desk with a message size of 60 bytes while varying the Tx/Rx slots gap length.

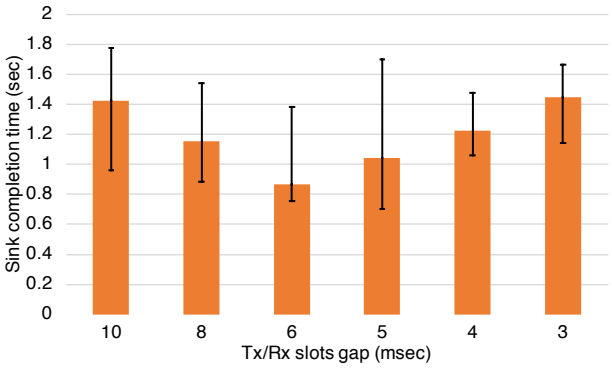


Figure 8: SpeedCollect running on 7 TelosB nodes on desk with a message size of 60 bytes while varying the Tx/Rx slots gap length.

seconds with the error bars representing the maximum and minimum values. For SensorTag, it can be observed that as the time gap reduces from 3ms to 0.2ms, there is a corresponding around 50% reduction in median completion latency. However, reducing the Tx/Rx slots gap length further to 0.1ms increases the completion latency. This shows the need to have sufficient time for processing between Tx/Rx slots. Similar behavior can be observed for the TelosB. However, since the processor of TelosB is much slower than SensorTag's processor, the reduction is from 10ms to 6ms, and completion times increase after that.

Note that the processing load in this section (48 messages) is higher than the load in the previous section (33 messages). Hence, a smaller time gap of 3ms is used in the previous section.

5.4 Heterogeneous Devices

In this section, we validate the portability of SpeedCollect and its ability to run on heterogeneous devices with different clock rates and processor types working together. The same source code is compiled to run on three platforms (Figure 9) namely: TelosB, CC2650 SensorTag, and Zoleria. In the experiment, we have a sink node (TelosB) collecting data from 6 other nodes, 2 TelosB, 2 CC2650 SensorTag, and 2 Zoleria. Each of the sources has 6 messages of size 60 bytes to be transmitted to the sink.

Figure 10 shows the CDF of the completion times at the sink. As shown in the figure, SpeedCollect works on the three different platforms and can finish as fast as 0.7sec with a median completion time of 1.19s which gives a throughput of about 15kbps with 100% reliability.

6 Conclusions

We have designed a portable and flexible many-to-one communication protocol, SpeedCollect, which is based on synchronous transmission and capture effect. SpeedCollect is hardware-independent which allows us to evaluate it on networks of heterogeneous devices. We have shown that SpeedCollect is effective in improving the performance of many-to-one communications through the combinations of acknowledgment, 1-hop bidirectional neighbors, TDM and polling compared to the baseline. With the decoupling of communication and computation, SpeedCollect is also able to support protocols that required more processing. We believe that SpeedCollect opens up the potential to develop more diverse classes of applications that can simultaneously run on multiple platforms.

7 Acknowledgments

This research was supported by the Singapore Ministry of Education Academic Research Fund Tier 2 (MOE2019-T2-2-134).

8 References

[1] J. Åkerberg, M. Gidlund, and M. Björkman. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In *2011 9th IEEE International Conference on Industrial Informatics*, pages 410–415. IEEE, 2011.

[2] B. Al Nahas, S. Duquennoy, and O. Landsiedel. Network-wide consensus utilizing the capture effect in low-power wireless networks. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–14, 2017.

[3] P. Appavoo, E. K. William, M. C. Chan, and M. Mohammad. Indriya2: A heterogeneous wireless sensor network (wsn) testbed. In *International Conference on Testbeds and Research Infrastructures*, pages 3–19. Springer, 2018.

[4] J. Arnbak and W. Van Blitterswijk. Capacity of slotted aloha in rayleigh-fading channels. *IEEE Journal on Selected Areas in Communications*, 5(2):261–269, 1987.

[5] I. Bekmezci, O. K. Sahingoz, and Ş. Temel. Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254–1270, 2013.

[6] J. W. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE journal on selected areas in communications*, 20(8):1528–1540, 2002.

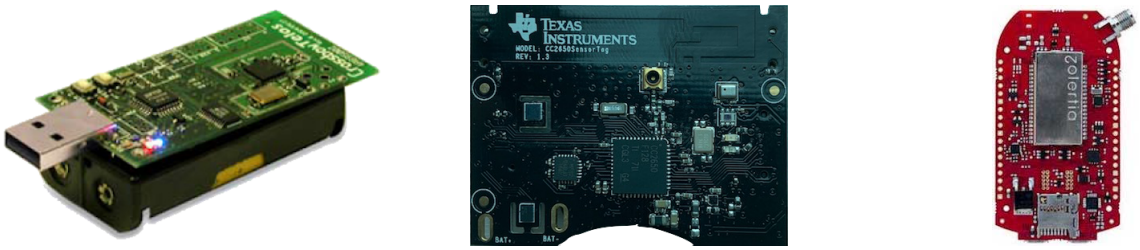
[7] M. Doddavenkatappa, M. C. Chan, and B. Leong. Improving link quality by exploiting channel diversity in wireless sensor networks. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 159–169. IEEE, 2011.

[8] M. Doddavenkatappa, M. C. Chan, B. Leong, et al. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *NSDI*, pages 269–282, 2013.

[9] M. Doddavenkatappa and M. Choon. P 3: a practical packet pipeline using synchronous transmissions for wireless sensor networks. In *IPSN*, pages 203–214. IEEE, 2014.

[10] W. Du, J. C. Liando, H. Zhang, and M. Li. When pipelines meet fountain: Fast data dissemination in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 365–378. ACM, 2015.

[11] 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



(a) TelosB (b) CC2650 SensorTag (c) Zolertia - RE-Mote

Figure 9: Platforms that SpeedCollect is running on and working together.

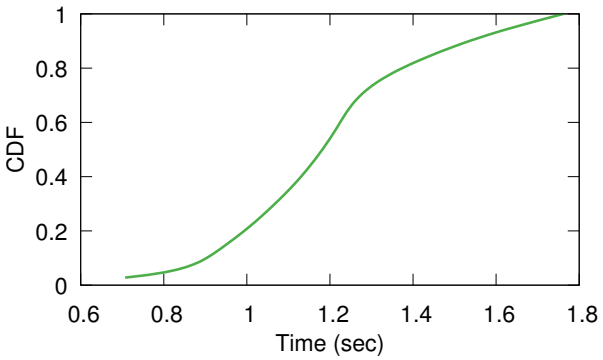


Figure 10: SpeedCollect running on a sink node and 6 devices of 3 different platforms (TelosB, CC2650 SensorTag, and Zolertia) each sharing 6 messages with 60 bytes of payload.

layer for low-power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14, 2010.

[12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *SenSys*. ACM, 2012.

[13] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *IPSN*. IEEE, 2011.

[14] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys*. ACM, 2009.

[15] S. Hayat, E. Yanmaz, and R. Muzaffar. Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Communications Surveys & Tutorials*, 18(4):2624–2661, 2016.

[16] H. Hellwagner and C. Bettstetter. Networking research challenges in multi-uav systems, 2016.

[17] C. Herrmann, F. Mager, and M. Zimmerling. Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks. In *SenSys*. ACM, 2018.

[18] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[19] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza. Data prediction+ synchronous transmissions= ultra-low power wireless sensor networks. In *SenSys*. ACM, 2016.

[20] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth codes: Maximizing sensor network data persistence. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 255–266, 2006.

[21] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: Practical wireless network coding. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 243–254, 2006.

[22] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *SenSys*. ACM, 2013.

[23] E. A. Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, pages 363–369. IEEE, 2008.

[24] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi. An experimental study on the capture effect in 802.11 a networks. In *Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 19–26, 2007.

[25] K. Leentvaar and J. Flint. The capture effect in fm receivers. *IEEE Transactions on Communications*, 24(5):531–539, 1976.

[26] J. Lu and K. Whitehouse. *Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks*. IEEE, 2009.

[27] M. Luvisotto, Z. Pang, and D. Dzung. Ultra high performance wireless control for critical applications: Challenges and directions. *IEEE Transactions on Industrial Informatics*, 13(3):1448–1459, 2016.

[28] P. Maymounkov. Online codes. Technical report, Technical report, New York University, 2002.

[29] M. Mohammad and M. C. Chan. Codecast: Supporting data driven in-network processing for low-power wireless sensor networks. In *IPSN*. ACM, 2018.

[30] A. Shokrollahi. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.

[31] J. A. Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3–9, 2014.

[32] M. Trobinger, D. Vecchia, D. Lobba, T. Istomin, and G. P. Picco. One flood to route them all: ultra-fast convergecast of concurrent flows over uwb. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 179–191, 2020.

[33] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II.*, pages 45–52. IEEE, 2005.

[34] T. Winter, P. Thubert, A. Brandt, J. W. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.-P. Vasseur, R. K. Alexander, et al. Rpl: Ipv6 routing protocol for low-power and lossy networks. *rfc*, 6550:1–157, 2012.

[35] D. Yuan and M. Hollick. Let's talk together: Understanding concurrent transmission in wireless sensor networks. In *38th Annual IEEE Conference on Local Computer Networks*, pages 219–227. IEEE, 2013.

[36] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele. Adaptive real-time communication for wireless cyber-physical systems. *ACM Transactions on Cyber-Physical Systems*, 1(2):1–29, 2017.

[37] M. Zimmerling, L. Mottola, and S. Santini. Synchronous transmissions in low-power wireless: A survey of communication protocols and network services. *ACM Computing Surveys (CSUR)*, 53(6):1–39, 2020.