Codecast: Supporting Data Driven In-Network Processing for Low-Power Wireless Sensor Networks

Mobashir Mohammad
National University of Singapore
mobashir@comp.nus.edu.sg

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

ABSTRACT

This paper presents *Codecast*, a many-to-many communication protocol for low-power sensor networks that provide high throughput and reliable data sharing from multiple sources to multiple destinations of a network.

Codecast uses physical layer capture on concurrent transmissions for high spatial reuse and a network-assisted network coding for high throughput as the core techniques. Our extensive evaluation in two large-scale testbed deployments (Indriya and Flocklab) shows that Codecast provides up to 4x the throughput of Chaos and 1.8x the throughput of LWB for many-to-many data communication.

Finally, we demonstrate the utility of Codecast through a distributed channel selection mechanism and a link state based routing protocol.

CCS CONCEPTS

• **Computer systems organization → Sensor networks**;

KEYWORDS

Wireless sensor networks, synchronous transmission, capture effect, network coding

ACM Reference Format:
Mobashir Mohammad and Mun Choon Chan. 2018. Codecast: Supporting Data Driven In-Network Processing for Low-Power Wireless Sensor Networks. In *Proceedings of ACM conference (IPSN'18).* ACM, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

With over a decade of effort put towards improving communication protocols for the low-power sensor network, the research has reached a point where we can find very efficient approaches of designing typical sensor network protocols. However, quite often each protocol is optimized for a specific communication pattern. For instance, while a data collection protocol like CTP [18] is designed to handle many-to-one traffic pattern, data dissemination protocols like Splash [8] is designed for one-to-many. If an application demands a high throughput point-to-point bulk data transfer, P^3 [9] instead of either CTP or Splash would be a better choice.

IPSN'18, April 2018, Porto, Portugal

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

While existing protocols allow efficient data transfer for the interaction patterns that they have been designed for, they are not flexible enough to cater to the needs of applications that demand more complex interactions. [28, 30, 33] Moreover, there is an increasing trend in the machine learning and other related research communities to support in-network processing that involves substantial data exchanges among many nodes. Such interaction can enable different nodes to take coordinated actions based on common information. For instance, there exists an entire class of protocols that rely on multi-channel communication for improved reliability [1, 7, 12, 35, 36, 40] and/or improved throughput [8–10]. Such protocols typically either hard-code or rely on a centralized scheduler to decide the set of channels for the entire network. Since different parts of the network experiences different interference and would have different sets of usable channels, a distributed channel selection protocol requiring many-to-many channel quality data exchange and substantial in-network processing is required.

However, to the best of our knowledge, there is no protocol that is designed to disseminate significant amount of data from many sources to many destinations reliably and efficiently. Existing many-to-many communications approaches are based on workarounds that utilize data dissemination (LWB) or data collection (CTP) protocols. LWB is centralized and CTP is extremely slow due to the underlying contention-based CSMA/CA layer. Protocols that are designed to support agreement protocols (Chaos [26], A^2[3]), such as leader election or two-phase commit, have also been proposed. However, these protocols are not designed for sharing/disseminating a large amount of data and to support more complex interactions.

To address the above limitations, we present *Codecast*, a high throughput many-to-many communication protocol that can reliably disseminate data from many source nodes to many destination nodes. In order to achieve high throughput through allowing multiple nodes to transmit different data packets at the same time, Codecast leverages synchronous transmissions and capture effect to increase the likelihood of multiple successful packet receptions of different data packets by different nodes.

Codecast incorporates two key components to improve performance. First, it uses a network coding scheme that utilizes feedback to craft specialized codewords to improve the rate of data distribution. Second, it incorporates a transmission decision logic to achieve higher spatial reuse during uncoordinated synchronous transmissions from many senders.

To summarize, we make the following contributions:

- As standard network coding with known distributions like Robust Soliton [31] does not work well, we design an improved network coding degree distribution that leverages local feedback to create "better" codewords.

- We incorporate feedback to improve spatial reuse and allows a faster spread of "new" information through the network from many sources to many destinations.
- We have implemented Codecast on Contiki and show that it is reliable and provides much higher throughput over existing solutions for many-to-many data communication.
- We demonstrate the utility of Codecast through two applications (1) distributed channel selection, and (2) link state routing. Both applications require sharing of information among a subset or all the nodes.

Our evaluation shows that Codecast significantly outperforms Chaos and LWB for many-to-many data communication in terms of achievable throughput on two wireless sensor network testbeds. In particular, Codecast offers up to 4 times and up to 1.8 times the throughput of Chaos and LWB respectively. The main benefit of Codecast lies in its ability to be a generic wireless communication protocol that can handle multiple types of traffic patterns.

The rest of the paper is organized as follow. Section 2 introduces some of the related works. The motivation and detailed design of Codecast is presented in Section 3. Section 4 evaluates Codecast for many-to-many communication followed by some of the applications presented in section 5. Finally, we conclude in Section 6.

2 RELATED WORK

During the last decade, we have seen dramatic improvements in the performance of low-power sensor network protocols. We can now build numerous applications having diverse requirements like reliable and high throughput over-the-air programming of remote deployments [8, 17, 19, 21, 32], robust and energy-efficient data collection [1, 7, 12, 13, 18, 27, 36, 45], etc. However, the ability to share information among many nodes using a generic many-to-many communication in low-power sensor networks has been long overlooked. There are two possible approaches to achieve many-to-many communication.

(1) **Data dissemination driven**: This approach is taken by LWB [15]. A centralized controller distributes transmission schedules to the nodes of the network. Each node then performs Glossy-based data dissemination to all the nodes during its own dedicated transmission slot.

(2) **Data collection driven**: This approach is the reverse of LWB. A centralized scheduler like previously distributes the transmission schedule to every network node. Each node then acts as Sink and initiate data collection using CTP [18], RPL [45], etc.

These naïve approaches are limited by the reliability of the performance of the underlying data collection and dissemination protocols. Also, because of the dependency on a central scheduler, these schemes suffer from a single point of failure. Failure to receive the transmission schedule causes complete data loss. Moreover, while the data dissemination driven approach have reliability issues due to the well known scalability problem of synchronous transmissions [35], data collection driven scheme offers extremely low throughput because of the use of underlying CSMA/CA and the overhead of maintaining uplink routes for every node acting as a sink.

Chaos [26] and A^2[3] are more distributed in nature but are designed to support agreement protocols (such as leader election or two-phase commit) and are not designed for sharing/disseminating large amount of data and more complex interactions. Chaos performs data dissemination in parallel by integrating an aggregate function (MAX, MIN, COUNT, etc.) into synchronous transmission schedules. Because of the limit on the number of information that they can fit into a maximum sized 802.14.5 packet to exchange, they are severely limited by the maximum throughput that they can offer.

Codecast, on the contrary, provides a more general approach to support many-to-many communication. It does so by introducing a new feedback driven network coding (NANC) into synchronous transmission schedules. The scheme enables extremely reliable and high throughput many-to-many data sharing. We demonstrate the utility of Codecast through two applications, a distributed channel selection algorithm and a link-state based routing scheme.

3 DESIGN

This section motivates the need for Codecast using a toy example and then describes in detail the key components of its design.

3.1 Codecast - Motivation

We first illustrate the advantages of Codecast through a simple example. Consider the 6 node grid network topology shown in Figure 1. The application needs to perform channel selection that demands many-to-many data sharing to achieve network-wide consensus. Nodes are supposed to use the two channels based on the channel quality estimates of nodes A and F. Hence, nodes A and F have to share their preferred channels D_1 and D_2 with the entire network.

Simple Multicast: Using multicast and allowing two simultaneous transmissions by nodes A and F in the first transmission round, it takes a total of 5 rounds of transmissions to distribute the information to all the 6 nodes as shown in the top row of Figure 1.

Network Coding: The middle row of Figure 1 shows the transmissions needed to complete the required task by leveraging network coding. It reduces the rounds needed by 1 as only 4 rounds of transmissions are now enough.

Codecast: The bottom row of Figure 1 shows the transmissions needed to complete the required task using Codecast. Just 3 rounds of transmissions are needed, that is 1 less than the use of network coding and 2 less than the use of simple multicast.

The key difference between the use of network coding and Codecast is that the later allows both nodes B and E to transmit at the same time even though both nodes are within the transmission range of each other. Such simultaneous transmissions improve spatial reuse and thus speeds up data exchange significantly.

3.2 Codecast in a Nutshell

Codecast runs as a service that initiates data communication periodically or whenever an application needs to by triggering a wake-up event. Deciding which set of nodes N should participate in the dissemination/sharing in a Codecast round is application specific. Some applications may require participation from all the nodes to make a distributed decision while other applications may require

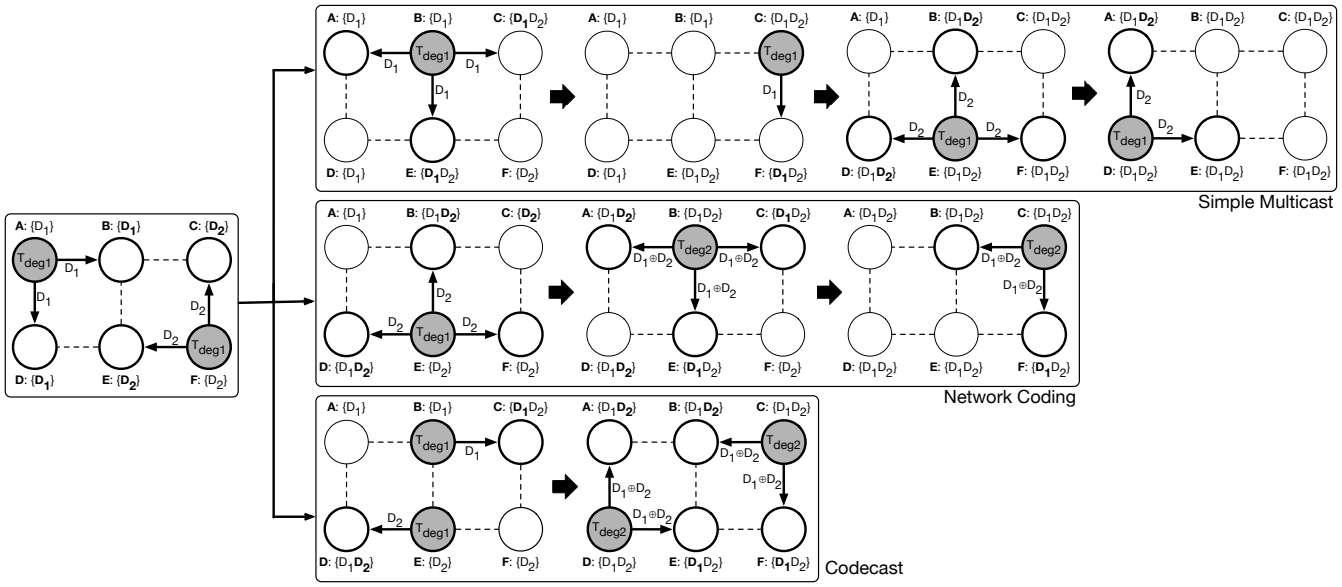


Figure 1: Comparison of different dissemination strategies

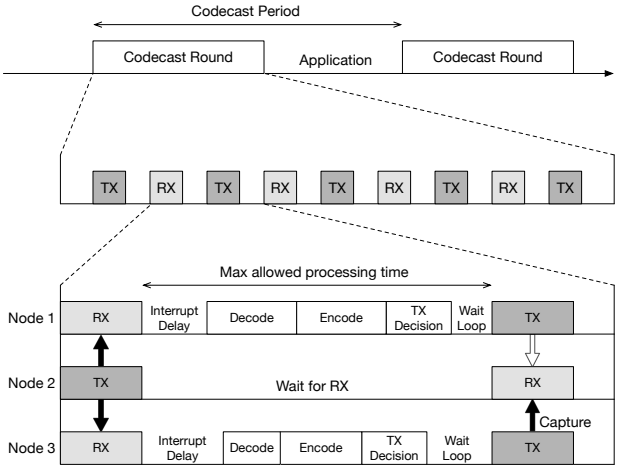


Figure 2: Overview of Codecast in action

participation from a subset of the nodes to make a decision. To disseminate the identities of the nodes in N, one approach is to have a coordinator uses a single Glossy [16] round for reliable sharing. Once the network nodes have received this information, they can start the Codecast dissemination process.

Operations during a Codecast round are tightly synchronized and organized into slots. Implementation of the synchronous operations are similar to Chaos [26] and is based on Glossy [16]. The protocol begins when a set of nodes want to share its data with others - periodic channel assessment, periodic routing update, battery status notification, sensor reading exchange, etc. A dedicated node starts the data sharing process by broadcasting the information that it has.

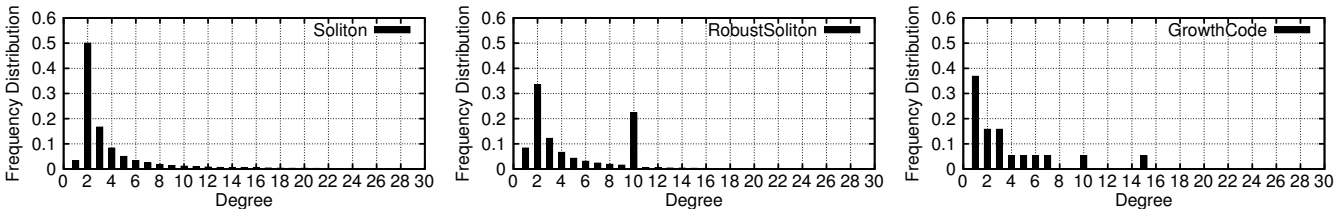
In each slot, a node is either in reception or transmission mode. With every packet reception, each node checks whether some new information has been learned. If new information is learned, the node chooses to transmit with some probability. Instead of contending with the neighboring nodes to acquire the channel and then transmitting, nodes in Codecast choose to transmit simultaneously in a tightly synchronized manner. Despite the fact that each node is transmitting different data over the same broadcast medium and that the packets are likely to collide and get corrupted, capture effect allows for successful packet receptions as long as some transmissions dominate over other relatively weak transmissions at the receivers [2, 14, 42].

Over time, nodes learn more and more data over multi-hops. However, note that when nodes learn *enough* data, the *Coupon Collector's Problem* sets in, slowing the rate at which nodes learn new data for every newly received packet.

The challenge is thus to design a protocol that answers the following key questions:

- What kind of information should be included in the packet to be transmitted?
- When should a node transmit or receive?

We discuss in detail the various components of Codecast and its implementation in the rest of this section. As the decision on whether to transmit, receive or terminate depends on the contents of information received in any round, we will first present the details of the information encoding process (Section 3.3). This is followed by Section 3.4 that decides when does a node transmit or receive.



(a) Soliton **(b) Robust Soliton** **(c) Growth Code**

Figure 3: Comparison of various distribution for $n = 30$ with $c = 0.1, \delta = 0.1$ for Soliton, Robust Soliton and Growth Code.

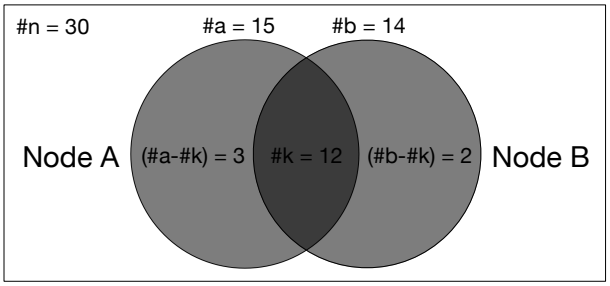


Figure 4: Decoding progress at a stage when node A and B have decoded a subset of the total data

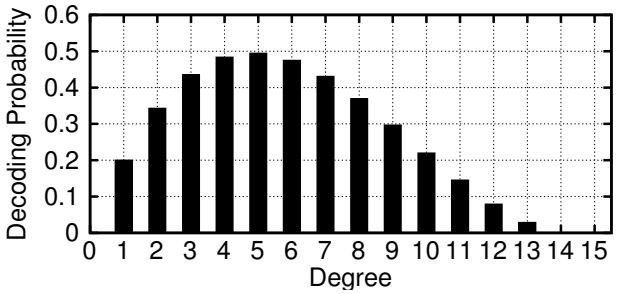


Figure 5: Decoding probability on receiving different degree codewords

3.3 Network Coded Data Distribution

In this section, we begin with a brief introduction to the basic of fountain codes and some existing degree distributions before presenting our variant of network coding that is optimized to improve throughput in a completely distributed many-to-many data sharing setting.

3.3.1 Fountain Codes Basics. Network Coding has been adopted in wireless sensor network for improving network's throughput and efficiency [10, 25]. Due to the resource constraints on the nodes, light-weight fountain codes (rateless codes) like Luby Transform codes (LT codes) [31], Random Linear codes (RL codes) [20], Raptor codes [41] and Online codes [34] are preferred because of their efficient encoding and decoding algorithms. The main idea behind network coding is to divide a data D into n symbols $\{D_1, D_2, ..., D_n\}$ (each from N nodes). These are then used to generate an infinite stream of encoded codewords $\{Y_1, Y_2, ...\}$. The encoding algorithm used to generate each Y_i codeword involves an application of XOR operation over d randomly selected symbols ($1 \leq d \leq n$) based on a chosen *degree distribution* (Section 3.3.2). It is referred to as the degree of the encoded codeword. A transmitting node distributes a stream of Y_i codewords. At the receiver's end, the node is able to recover the original data D upon receiving m ($m \geq n$) encoded codewords and performing simple decoding using Gaussian Elimination or Belief Propagation algorithms. Due to the robust and simplistic encoding and decoding algorithms of LT codes [32], we chose it in our implementation of Codecast. We implement an optimized "On the Fly Gaussian Elimination for LT Codes" [4] that takes less than 10ms on TelosB motes for up to $N = 30$.

3.3.2 Degree Distribution. Selecting a proper degree distribution is critical to achieve high throughput. The desired property of a degree distribution is to be able to retrieve the original data D with high probability P using a minimal number of encoded codewords m. Ideally, n codewords should be enough to recover all the n symbols. However, in practice, a little more than n codewords are required. *Soliton* distribution and its improved variant called the *Robust Soliton* distribution are the default and the most commonly used distributions. However, these distributions work better when all the data is available at a single source [24] and is thus more useful for application involving data dissemination [10]. In applications where the data is distributed throughout the network, the default *Robust Soliton* distribution does not work well. For instance, by following the *Robust Soliton* distribution, a node might decide to encode a codeword of a certain degree $d \geq 1$ when it has not decoded even a single symbol yet.

Growth Code [24] proposes an alternative degree distribution that requires a lower number of codewords to recover all the n symbols. It also speeds up the rate at which the nodes learn new symbols. The basic idea of Growth Code is that the transmitter should generate codewords with consistently increasing degree. Initially, when a node has recovered only a few symbols, it should encode low degree codewords to transmit. As time progresses, higher degree codewords are generated to improve the chances of the receivers to decode more and more symbols. Figure 3 compares the degree distributions for the three different distributions.

One key drawback of these distributions is that they are optimized for scenarios where all the data is available on a single

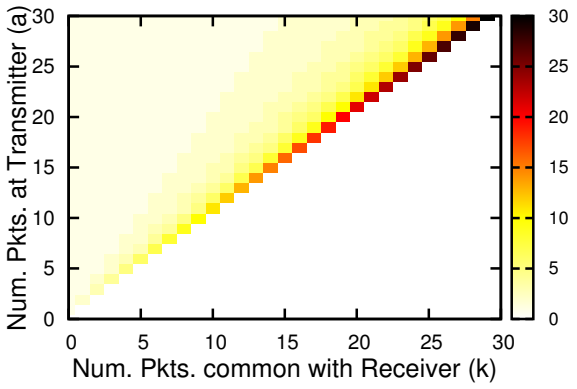


Figure 6: Network-Assisted Network Coding degree selection for a given $n = 30$, a, and k

source. However, in applications where the data is initially distributed across many nodes, degree selection based on the above distributions do not perform well.

Let us understand the problem through a two node example illustrated in Figure 4. Assume that a node A has decoded $a = 15$ symbols while node B has recovered $b = 14$ symbols out of the $n = 30$ symbols so far. Also, since both the nodes are neighbors, it is quite likely that many of the recovered symbols are common. Let us assume that $k = 12$ symbols are common between the two. Also, node A is trying to help node B to recover the missing symbols by sharing a degree d codeword. Node B can recover the missing symbol if and only if out of the d degree codeword encoded by A, exactly 1 symbol is missing at node B. Figure 5 plots the probability with which node B can decode an additional symbol when it receives codewords of different degrees from node A. Looking at the degree distributions, one can see that by following growth code, node A would encode a degree 1 codeword, while *Soliton* and *Robust Soliton* distribution might encode a degree 1 to 10 codeword, with 2 and 10 being the likely options respectively. From Figure 5 it is evident that the most optimal codeword is of degree 5. This enables node B to decode a new information with a decoding probability of 0.4945. Growth code, Soliton, and Robust Soliton distributions, on the contrary, yield low decoding probabilities of 0.2, 0.34, and 0.22 respectively.

3.3.3 Network-Assisted Network Coding (NANC). We design a data encoding and distribution technique called Network-Assisted Network Coding (NANC). Instead of following a degree distribution based on only local information, which leads to the generation of sub-optimal codewords, NANC incorporates feedback from the neighbors to generate "better" codewords.

In Codecast, nodes alternate between transmission and reception as shown in Figure 2. If a node has just transmitted a packet and its neighbor has successfully captured it, then the receiver is certain that the node that transmitted previously would currently be in reception mode. Further, the transmitting node also includes information on its decoding progress (number of symbols that it has decoded and therefore the symbols that it needs) in the packet. Specifically, in the transmitted packet, a node indicates in a N-bit vector (N is the number of nodes disseminating data) the data it has successfully decoded.

Selecting a "better" codeword degree. When a node decides to transmit, it has a record of the successfully decoded packets of each of its neighbors as it has heard from them earlier. However, note that except for the most recent data that it has received, all other information received from other nodes may be out-of-date because those nodes may have decoded more information since the time the feedback information was sent. Nevertheless, the decision will have to be made based only on the available feedback at that moment.

Let us revisit Figure 4. Given that a transmitting node A knows a (the number of symbols it has decoded) and k_i (the number of symbols it has common with its neighbor N_i), then $\rho_{a,k_i,d}$, the probability of decoding a new symbol by neighbor N_i on receiving a codeword of degree d from A can be defined as:

$$\rho_{a,k_i,d} = \frac{\binom{k_i}{d-1}\binom{a-k_i}{1}}{\binom{a}{d}} \tag{1}$$

Thus, node A selects the degree d_{opt_i} that maximizes the decoding probability for a particular neighboring node N_i as below:

$$d_{opt_i} = \underset{d}{\operatorname{argmax}}\, \rho_{a,k_i,d} \tag{2}$$

Figure 6 illustrates the degree (heatmap color) a node can select for $n = 30$, when a and k_i's are known.

Note that a node can have many neighbors and the optimal degree is likely to be different for each neighbor. The heuristic is to select the d_{opt_i} that maximizes the decoding probability for the slowest neighbor (the node that has solved the least number of symbols so far).

Crafting a specialized codeword. Once the degree is determined, the next step is to decide the set of data chosen to craft the codeword. Typically, the selection is done randomly [24, 31]. However, for Codecast, we use a hybrid scheme to utilize the available feedback. First, we select a single data that the most recent transmitter (B) has not decoded if such a data is locally available. This maximizes the likelihood this codeword to be useful since (1) the node that has just transmitted should be in the receiving mode and (2) its state has not changed and still needs this data. For the remaining $d_{opt} - 1$ symbols, we select up to $d_{opt} - 1$ data that both the sender and B have decoded already so that the first selected data can always be decoded by node B with certainty. Recall that the degree d_{opt} is already chosen to optimize the decoding probability of the slowest neighboring nodes. This hybrid scheme ensures the intended recipient can successfully decode a new data (if such a data is available locally) every time it receives a codeword and also allows the other neighbors to decode a symbol with high probability.

Codecast parameters. Figure 7 shows the Codecast packet structure. It comprises of 4 components: (1) *Codeword* of a certain degree decided as explained earlier, (2) *Codeword BitVector* to indicate

Table 1: Decision logic after every packet reception

I'm Done	Others Done	Decoded New	Transmit	Remark
0	0	0	P	Probabilistic transmission to avoid collisions.
0	0	1	1	Learned a new symbol, share with neighbors.
0	1	0	1	Only I need a symbol, inform neighbors.
0	1	1	1	Only I need a symbol, inform neighbors.
1	0	0	P	Probabilistic transmission to avoid collisions.
1	0	1	0	Not possible. Cannot learn a new symbol after I'm done.
1	1	0	0	Everyone has terminated. Stop transmission.
1	1	1	0	Not possible. Cannot learn a new symbol after all done.

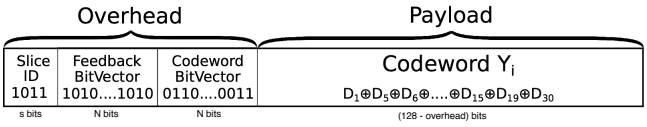


Figure 7: CodeCast packet structure

which data have been use to create a codeword, (3) *Feedback BitVector* to indicate which all data has been successfully decoded by the packet, and (4) *Slice ID* which is used to make Codecast scale to large network and would be explained later in section 4.

In the current implementation, Codecast slot length of 16ms is fixed to allow up to 10ms of encoding/decoding time and transmission time for a maximum sized 802.15.4 packet. Our optimized "On the Fly Gaussian Elimination for LT Codes" implementation [4] takes less than 10ms on TelosB motes for up to $N = 30$ nodes.

3.4 Transmit/Receive Decision

After a packet reception, a node decides if it wants to transmit. Since Codecast allows synchronous transmission to improve spatial reuse exploiting capture effect, it is important that not too many nodes transmit at the same time as that will reduce the likelihood of successful capture. A node makes use of a 3-tuple $< I'mDone, OthersDone, DecodedNew >$ to decide whether it should transmit or not.

The states are recorded at the end of a reception slot upon successful reception of a packet and are interpreted as follow:

- **"I'm Done":** 1 if the local node has received data from all sources, 0 otherwise.
- **"Others Done":** 1 if all other nodes (excluding the local node) have received data from all sources, 0 otherwise.
- **"Decoded New":** 1 if the local node has successfully decode a new data, 0 otherwise.

Table 1 illustrates the decision logic. The node spends most of its time in the states $< 000 >$ and $< 001 >$. The node always transmits when:

(1) It decodes a new data $< 0 * 1 >$, and
(2) It is the last incomplete node $< 01* >$

For cases when other nodes are not done and the local node has not decoded any new data in the last slot ($< *00 >$), a node chooses to transmit with probability P. The value of P depends on the network density. In the current evaluation, we select a small value of $P = 0.1$.

Finally, to confirm completion, Codecast uses a bit vector of size N bits where N denotes the number of source nodes sharing data. When a node has received from all sources, it broadcasts its data 5 times in quick succession similar to the *Aggressive Sharing upon Completion* in Chaos before turning off its radio.

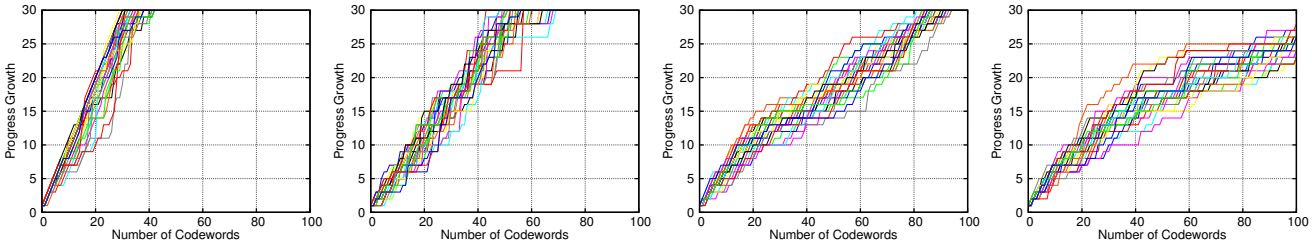
3.5 Scaling to Larger Networks

The Gaussian Elimination algorithm used to decode the Codecast codewords runs in $O(n^3)$ time. Thus, the time required to decode a codeword along with crafting the next specialized codeword increases rapidly with N, the number of nodes participating in the data exchange. The processing takes up to 10ms for $N = 30$ and much longer for $N \geqslant 30$ running on the telsoB motes. Use of larger slot lengths may result in loss of synchronization [35]. Moreover, given the size limitation of a maximum 802.15.4 packet, overhead would start to outweigh the payload when N becomes too large.

To enable scalable data exchange in larger networks, we utilize a hybrid approach. We first divide the network into multiple slices with $N \leqslant 30$ nodes in each slice. To indicate which slice a node belongs to, we add a slice ID in each Codecast packet as shown in Figure 7. Whenever a node receives a codeword, it checks which slice the codeword belongs to and transmits a newly encoded codeword from the same slice. This ensures that the explicit feedback in the received packet can be best utilized by the neighbors. Since every codeword now has a degree of up to the slice length ($N \leqslant 30$), we limit the decoding time to less than $10ms$. If a node has not decoded any information from the received codeword's slice yet, it will not be able to craft a new codeword belonging to the same slice. Instead, it transmits a codeword belonging to the next smallest slice ID that (1) the node has decoded information from; and (2) its neighbors have not completely decoded all data from the slice. With slicing, the number of nodes that participate in Codecast can be increased significantly.

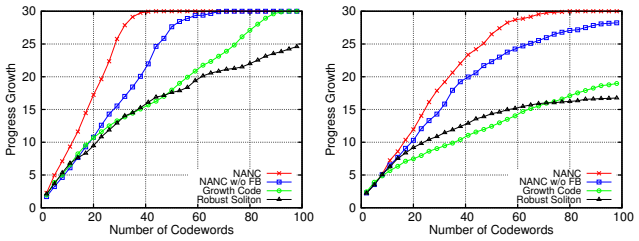
4 EVALUATION

In this section, we first evaluate Network-Assisted Network Coding (NANC) against the standard network coding (Robust Soliton and Growth Code) and highlight the importance of feedback in achieving faster data retrieval. Later, we incorporate NANC into Codecast to evaluate how it performs on real testbed evaluations.



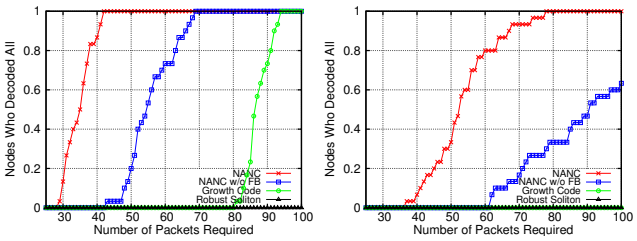
(a) NANC with Feedback (b) NANC (c) Growth Code (d) Robust Soliton

Figure 8: Decoding progress for each of the network node using different encoding schemes in Cooja



(a) Cooja (b) Indriya

Figure 9: Decoding progress when averaged across nodes



(a) Cooja (b) Indriya

Figure 10: CDF for the codewords required to decode

4.1 Network Code Performance

It is challenging to have a fair comparison of different network coding degree distributions on a real testbed. Unpredictable interference from multiple cross-technology interference (CTI) sources can change the results for different experiment runs. For consistency, we first evaluate the performance of different network coding schemes on Cooja [37], a simulator for sensor network using Multi-path Ray-tracer Medium (MRM) radio propagation model. We selected a 30 node network deployment over a 500mx500m simulated area for each run of the experiment. Nodes are allowed to transmit concurrently and packet receptions happen due to successful capture effect. The parameters for Robust Soliton and Growth Code are selected based on the recommended values [24, 31]. Figure 8 shows the number of packets required by individual nodes to decode the remaining 29 data of the other nodes for different schemes. Figure 9 highlights the decoding progress when averaged over all the network nodes to understand the general decoding trend.

As seen in Figure 9a, Growth Code and Robust Soliton distribution help to recover the first 15 symbols almost together. The progress using Robust Soliton slows down after that as the degree used to form codewords are too low. Growth Code, on the other hand, continues with a faster progress due to transmission of codewords with increasing degrees. In comparison, NANC is able to complete decoding with significantly smaller number of codewords - 55 codewords on average and 69 codewords in the worst case. The performance of NANC get further enhanced with explicit feedback. The number of codewords required to successfully decode all the information reduces to 35 on average and 42 in the worst case. This is close to the theoretical bound of 29 codewords. On the average, network coding used in Codecast requires 60% lesser codewords that the Growth code to recover the same amount of information as shown in Figure 10a.

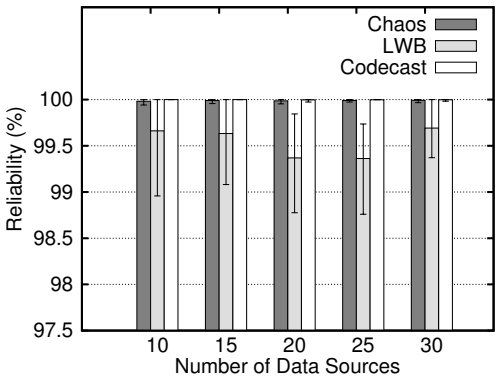
Similar observations can be made when we run the same network coding schemes on Indriya as shown in Figures 9b and 10b, however with slightly longer completion time. This is mainly due to the increase in packet collisions and interference present in a real testbed.

4.2 Many-to-Many Data Sharing

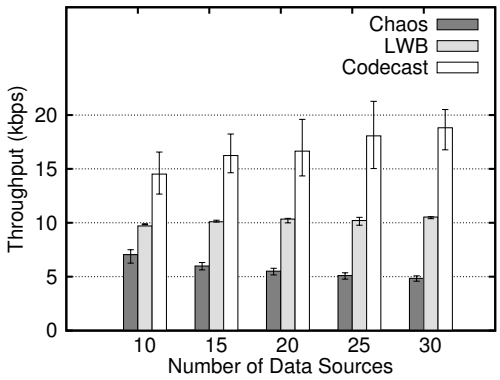
Codecast enables high throughput data sharing between many distributed nodes of a low-power wireless sensor network. In this section we compare Codecast against two popular protocols that also support many-to-many data communication.

4.2.1 Experiment Settings. We have implemented Codecast on Contiki OS [11] running on TelosB devices[38]. For each experiment, unless specifically mentioned, we used channel 26 of the IEEE 802.15.4 radio. Also, the maximum transmission power is used. To enable evaluation of the protocol multiple times, instead of allowing Codecast to run indefinitely, we timeout each codecast round after 4.5 seconds and restart the next. This gives codecast a maximum of 4.5 seconds to recover all the information.

4.2.2 Testbeds Considered. For extensive evaluation, we used Indriya [6] and Flocklab [29] testbeds. Indriya is a large and dense deployment, housing 90 TelosB nodes deployed over three floors of an academic institution. Flocklab on the other hand is a relatively smaller and less dense testbed comprising of 30 TelosB nodes deployed inside offices, hallways, and rooftop of an adjacent building. Each testbed roughly provides a diameter of around 5-6 hops when the highest transmission power of $0dBm$ is used.

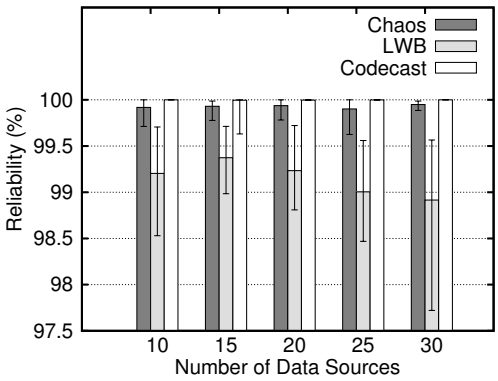


(a1) Reliability

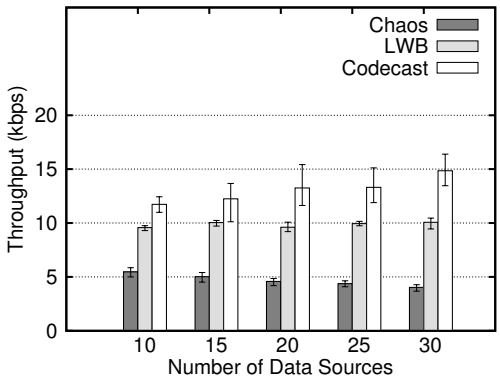


(a2) Throughput

(a) Flocklab



(b1) Reliability



(b2) Throughput

(b) Indriya

Figure 11: Performance comparison for many-to-many communication for different number of sources

Table 2: Testbed Characteristics

Testbed	Indriya	Flocklab
Number of Nodes	90	30
Tx Power (dBm)	0	0
Network Diameter (hops)	6	5
Ave. Neighbor per Node	15	8

Table 3: LWB Parameters

Parameters	Values
T_s	100ms
T_d	25 (32 bytes) - 100ms (128 bytes)
N_{tx}	Up to 7
Packet Size (Bytes)	32-128

Detailed testbed characteristics are enlisted in Table 2 for reference.

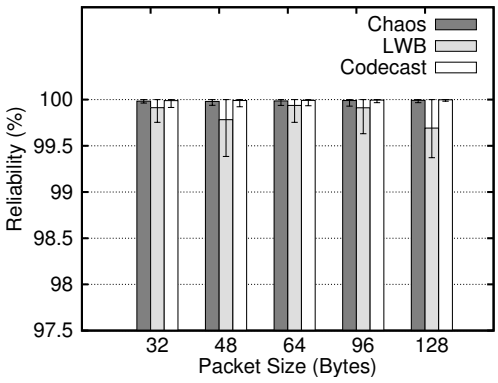
4.2.3 Baseline Protocols. We consider the following protocols to evaluate Codecast against.

Chaos: A synchronous transmission-based protocol that exploits capture effect and in-network processing to support many-to-many data sharing for applications like "Leader-Election". We used the Contiki implementation of Chaos for sky motes [26].

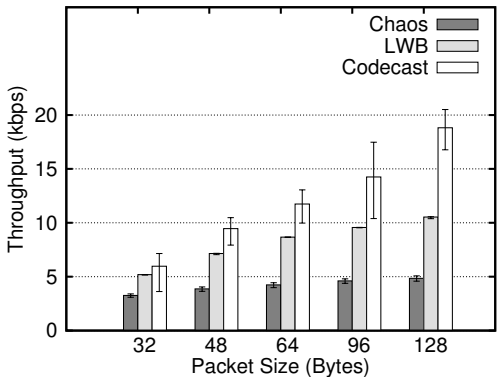
LWB: A synchronous transmission-based protocol that relies on a centralized scheduler to disseminate transmission schedules to network nodes. All the participating nodes then initiates a Glossy flood [16] for data dissemination in their own dedicated time slot. We used the Contiki implementation of LWB for sky motes [39].

The LWB settings chosen is shown in Table 3. We do not use the default LWB parameters as they work only for extremely small packets (15 bytes). Reliability for LWB depends on both the packet size and the length of data slot T_d. We choose the smallest data slot length that provided similar reliability of over 99% as reported in the paper [15] and showed high throughput to be fair as shown in Figure 13.

4.2.4 Metrics. We use the following key performance metrics for testbed evaluations:

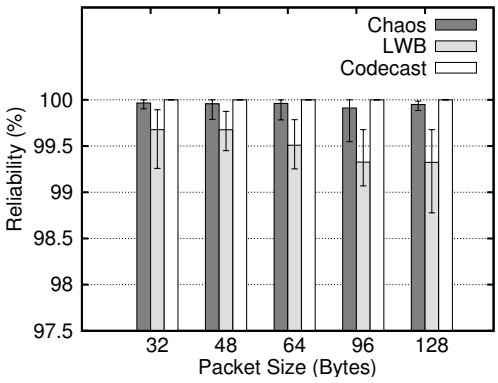


(a1) Reliability

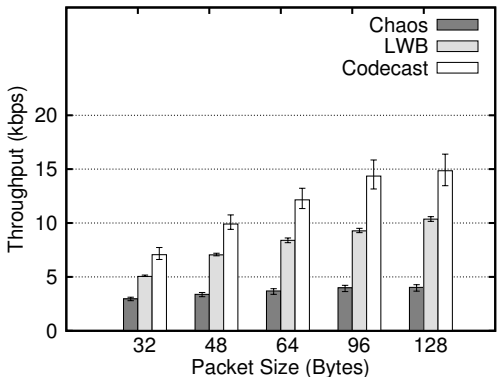


(a2) Throughput

(a) Flocklab



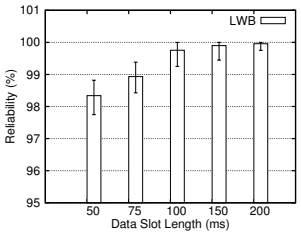
(b1) Reliability



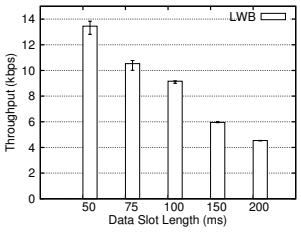
(b2) Throughput

(b) Indriya

Figure 12: Performance comparison for many-to-many communication for different packet sizes



(a) Reliability



(b) Throughput

Figure 13: Dependence of LWB reliability and throughput on data slot length T_d

Throughput: Calculated as the amount of data (exclude overhead) received by a node per unit time and represented in kilobits per second (kbps).

Reliability: Calculated as the total data received out of the total data generated by all the data sources. It is averaged over multiple rounds and represented as a percentage.

4.3 Impact of Varying Data Sources

In this section we test the performance of different protocols with varying number of data sources.

Setup: We select N randomly selected nodes from Flocklab and Indriya as data sources. N is varied from 5 to 30. The remaining network nodes not only act as relays to allow data exchange but are also data recipients. Maximum sized IEEE 802.14.5 packets are used for all the runs. Each experiment last for 30 minutes which involves multiple rounds of Codecast, Chaos and LWB. We plot the observed reliability and throughput averaged over all the rounds and across all the nodes.The error bars in the plots indicate the 5^{th} and the 95^{th} percentiles.

Result: As illustrated in Figures 11a1 and 11b1, both Chaos and Codecast are extremely reliable on both the testbeds. LWB, because of the lack of explicit feedbacks or acknowledgements, is not as reliable as the others but is still over 99% in reliability (achieved by choosing larger than default data slot length). Note that smaller data slot length will increase LWB's throughput but lower the reliability to below 99% as shown in Figure 13.

A significant difference is observed in the throughput comparison as shown in Figures 11a2 and 11b2. Codecast exhibits high throughput of up to 20kbps when 30 nodes act as data source. This is an improvement of up to 1.8 times in comparison to LWB and and 4.0 times in comparison to Chaos for the same number of data sources.

4.4 Impact of Varying Packet Sizes

Synchronous transmissions are known to be unreliable for large packet sizes [8]. Packet reception reliability drops with the increase in the packet size as well as with the increase in the number of synchronous transmitters [5, 23, 35, 43]. In this section we check the impact of different packet sizes on the performance of Codecast, LWB and Chaos.

Setup: We repeat the above experiment with 30 randomly selected nodes as data sources. However, this time we vary the size of the transmitted packets from 32 bytes to 128 bytes. Each time we record the reliability and throughput along with the 5^{th} and the 95^{th} percentile from both the testbeds.

Result: Figure 12 summarizes the results of the above experiment. The impact of packet sizes on reliability is not noticeable in Codecast and Chaos. As expected, LWB exhibits slight improvement in reliability as packet size decreases from 128 to 32 bytes. Codecast like previous section achieves extremely high throughput of up to 20kbps in comparison to a maximum throughput of 5.09kbps and 10.59kbps for Chaos and LWB respectively.

Table 4: Scalable many-to-many data sharing with network slicing

Nodes	# Slices	Throughput	Reliability
30-to-90	1	22.5 kbps	100.00%
60-to-90	2	19.4 kbps	99.95%
90-to-90	3	18.0 kbps	99.98%

4.5 Impact of Network Slicing

Slicing of network allows many-to-many data sharing to scale to any number of nodes to support large networks. Table 4 shows the average throughput and reliability when 30, 60 and 90 nodes share the data with the rest of the 90 nodes of Indriya. With 30 nodes in a slice, 60-to-all involve 2 slices being shared while all-to-all involves 3 slices being shared in parallel. Note that the throughput measured are different (higher) than those shown in Figure 11 since the experiments were performed on different days and experienced substantially different interference patterns.

4.6 Energy Consumption

To achieve high throughput data exchange between network nodes, Codecast rely on intensive in-network processing to perform network encoding and decoding on every packet that every node receives. Energy is a crucial resource for such resource constraint devices. Figure 14 compares the average energy consumption of Codecast (synchronous transmissions with in-network processing) and Glossy (only synchronous transmission that LWB is based on). Since radio accounts for the major portion of the consumed energy,

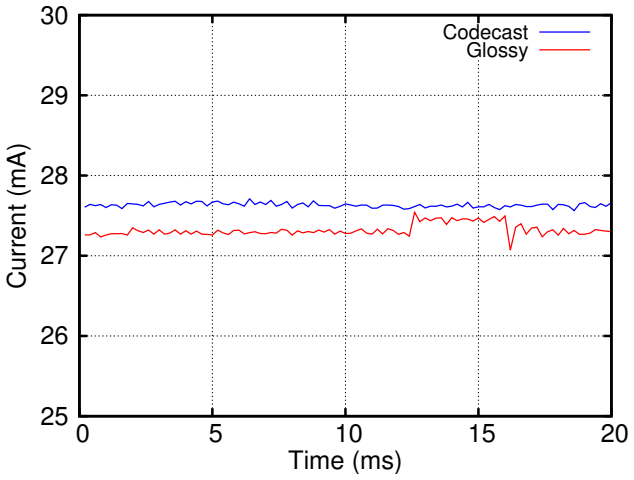


Figure 14: Energy Consumption with and without computation.

Codecast's additional intensive processing does not significantly add to it. Codecast's average current consumption during the active phase is 27.53μA. It is only 0.25μA increase when compared to 27.28μA current consumption of Glossy/LWB. There is negligible difference in current drawn when compared to Chaos since it also involves some simpler in-network processing. Combined with the fact that Codecast takes requires 1.8 to 4 times smaller radio-on time to share the same about of data, it provides an extremely efficient way of enabling many-to-many data sharing.

5 APPLICATIONS

In this section, we demonstrate the utility of Codecast through two applications.

5.1 Distributed Channel Selection

Cross-technology interference is becoming increasingly rampant [36]. Many techniques have been designed to mitigate its impact on packet losses. The most common method is to exploit channel diversity using channel hopping mechanisms [7, 8, 22, 36, 44]. However, none of these protocols specify a way to agree on a set of common channels. Quite often, these protocols hard-code a set of channels upon the initial setup. However, due to the dynamic nature of interference, channel selection should be performed in a more dynamic manner rather than statically determined in advance.

We implement a distributed channel selection scheme using Codecast. All 90 nodes on Indriya periodically assess the environment and individually pick one channel as the preferred channel out of 4 channels. These nodes then share this information with the rest of the network using Codecast. Early termination (before all information are received by all nodes) can happen if a node decides it has determined the channel that is preferred by the largest number of nodes or the *preferred channel*. Note that the preferred channel does not have to be the choice of the majority, it just needs to have the most votes.

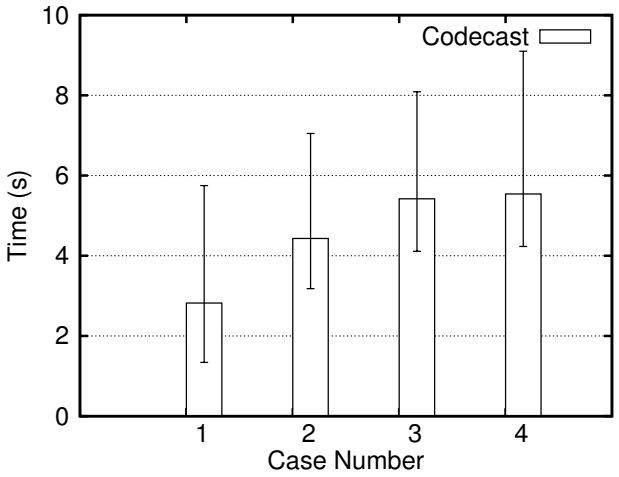


Figure 15: Completion Time of Distributed Channel Selection using Codecast with different Settings

Table 5: Channel selection distribution

	Ch 1	Ch 2	Ch 3	Ch 4
Case 1	100%	0%	0%	0%
Case 2	48%	24%	16%	12%
Case 3	40%	30%	20%	10%
Case 4	25%	25%	25%	25%

To be precise, let there be a total of N nodes, and the number of votes received for channels x_1, x_2, x_3 and x_4 so far be v_1, v_2, v_3 and v_4 respectively. Assume that $v_1 \geq v_2 \geq v_3 \geq v_4$. Selection can terminate if (1) $v_1 > \frac{N}{2}$ or (2) $v_1 > v_2 + (N - \sum_i v_i)$. In the worst case, there is no early termination and Codecast distributes all data to all nodes. In case of a tie, the channel with lower channel number is selected.

Table 5 shows the different distributed channel selection cases and Figure 15 shows the average completion time taken of distributed channel selection using Codecast. As expected, in the extreme case when all nodes chooses the same channel, Codecast terminates early in 2.97 sec once a node hears from a majority of the nodes. In the extreme case whereby all the channels are selected with almost equal likelihood, channel selection runs to completion in 5.54 sec. With other combinations of channel preferences, early terminations are triggered ranging from 4.43 sec to 5.42 sec.

This example illustrates the potential of Codecast to support more complex interaction. When more information is available, a decision can be made in a shorter time depending the decision logic and data characteristic.

5.2 Routing for Sensor Networks

Typically, routing in low-power sensor networks is distance vector based and a routing tree [18, 45] or a DODAG [13, 36] is constructed. Nodes share their distances from the sink (using metrics like hop-count, ETX, EDC, etc.) with their neighbors and the information is propagated until a stable network topology is constructed. Even though the process is simple, this however has certain disadvantages:

Routing loops: The nodes generally know only its parent or a set of forwarders along the direction of the sink. This limited information can cause routing loops and leads to packet losses when link quality changes.

Incomplete topology: The nodes do not have a complete and consistent view of the network topology. Communication from any source to any destination has to go through the sink, thus, introducing significant delays, even though the source and destination could be reachable directly.

With Codecast's ability to share larger amount of data efficiently, the question becomes whether it is possible to use link state based routing instead. In this evaluation, we use Codecast to distribute link state information from (1) 30 randomly selected nodes and (2) all nodes in Indriya. We refer to the former as Codecast30 and the later as CodecastAll. A standard shortest path algorithm is used to compute the shortest paths between two nodes.

Table 6: Mean and Max hops to reach from any source to any destination on Indriya for different schemes

Protocol	Ave. Time	Mean Hop	Maximum Hop
RPL	-	6.00	13
CTP	-	5.96	14
CodecastAll	5.6s	2.17	4
Codecast30	1.4s	2.58	5

Table 6 summarizes the results of the evaluation in terms of the distance in hops for any node to communicate with any other node on Indriya using routing schemes of RPL, CTP and Codecast. On the average, any two nodes on Indriya are within a communication range of 2.58 hops using Codecast30. This is almost a 60% reduction in hop count compare to existing tree based routing protocols. In addition, even though CodecastAll reduces the average path length to 2.17 hops, it does take more than 3 times longer to complete since there are much more data (data from 90 nodes vs 30 nodes) to transfer.

6 CONCLUSION

We present Codecast, a generic many-to-many communication protocol that enables reliable and high throughput data sharing among distributed network nodes. The protocol exploits synchronous transmission and capture effect to achieve high spatial reuse by allowing multiple nodes to transmit at the same time. Codecast incorporates a new network-assisted network coding scheme by utilizing feedback from the neighbors. Codecast not only generates optimal degree codewords but also crafts specialized codewords that maximize the probability of decoding a new information for the neighbors for every packet reception. Our data collection and data dissemination application shows significant improvements in comparison to other state-to-art protocols that supports many-to-many communications.

7 ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers and our shepherd, Shan Lin, for their valuable comments and suggestions to refine the paper.

This work is partially supported by the Singapore Ministry of Education Academic Research Fund Tier 1 (R-252-000-621-114).

REFERENCES

[1] Beshr Al Nahas, Simon Duquennoy, Venkatraman Iyer, and Thiemo Voigt. 2014. Low-power listening goes multi-channel. In *Proceedings of the 10th International Conference on Distributed Computing in Sensor Systems*. IEEE.

[2] JENSC Arnbak and Wim Van Blitterswijk. 1987. Capacity of slotted ALOHA in Rayleigh-fading channels. *IEEE Journal on Selected Areas in Communications* (1987).

[3] Simon Duquennoy Beshr Al Nahas and Olaf Landsiedel. 2017. Network-wide Consensus Utilizing the Capture Effect in Low-power Wireless Networks. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems*. ACM.

[4] Valerio Bioglio, Marco Grangetto, Rossano Gaeta, and Matteo Sereno. 2009. On the fly gaussian elimination for LT codes. *IEEE Communications Letters* (2009).

[5] Doug Carlson, Marcus Chang, Andreas Terzis, Yin Chen, and Omprakash Gnawali. 2013. Forwarder selection in multi-transmitter networks. In *Proceedings of the 9th International Conference on Distributed Computing in Sensor Systems*. IEEE.

[6] Manjunath Doddavenkatappa, Mun Choon Chan, and Akkihebbal L Ananda. 2011. Indriya: A low-cost, 3D wireless sensor network testbed. In *Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*. Springer.

[7] Manjunath Doddavenkatappa, Mun Choon Chan, and Ben Leong. 2011. Improving link quality by exploiting channel diversity in wireless sensor networks. In *Proceedings of the 32nd Real-Time Systems Symposium*. IEEE.

[8] Manjunath Doddavenkatappa, Mun Choon Chan, and Ben Leong. 2013. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*. USENIX.

[9] Manjunath Doddavenkatappa and Mun Choon. 2014. P 3: a practical packet pipeline using synchronous transmissions for wireless sensor networks. In *Proceedings of the 13th International Conference on Information Processing in Sensor Networks*. IEEE.

[10] Wan Du, Jansen Christian Liando, Huanle Zhang, and Mo Li. 2015. When pipelines meet fountain: Fast data dissemination in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM.

[11] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. 2004. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th International Conference on Local Computer Networks*. IEEE.

[12] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. 2015. Orchestra: Robust mesh networks through autonomously scheduled TSCH. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM.

[13] Simon Duquennoy, Olaf Landsiedel, and Thiemo Voigt. 2013. Let the tree Bloom: scalable opportunistic routing with ORPL. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM.

[14] Prabal Dutta, Stephen Dawson-Haggerty, Yin Chen, Chieh-Jan Mike Liang, and Andreas Terzis. 2010. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM.

[15] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. 2012. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*. ACM.

[16] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. 2011. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks*. IEEE.

[17] Yi Gao, Jiajun Bu, Wei Dong, Chun Chen, Lei Rao, and Xue Liu. 2013. Exploiting concurrency for efficient dissemination in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems* (2013).

[18] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. 2009. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM.

[19] Andrew Hagedorn, David Starobinski, and Ari Trachtenberg. 2008. Rateless deluge: Over-the-air programming of wireless sensor networks using random linear codes. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*. IEEE Computer Society.

[20] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. 2006. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory* (2006).

[21] Jonathan W Hui and David Culler. 2004. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems*. ACM.

[22] Venkatraman Iyer, Matthias Woehrle, and Koen Langendoen. 2011. ChryssoâĂŤA multi-channel approach to mitigate external interference. In *Proceedings of the 8th Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE.

[23] Xiaoyu Ji, Yuan He, Jiliang Wang, Kaishun Wu, Ke Yi, and Yunhao Liu. 2013. Voice over the dins: improving wireless channel utilization with collision tolerance. In *Proceedings of the 21st International Conference on Network Protocols*. IEEE.

[24] Abhinav Kamra, Vishal Misra, Jon Feldman, and Dan Rubenstein. 2006. Growth codes: Maximizing sensor network data persistence. In *ACM SIGCOMM Computer Communication Review*. ACM.

[25] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. 2006. XORs in the air: practical wireless network coding. In *ACM SIGCOMM Computer Communication Review*. ACM.

[26] Olaf Landsiedel, Federico Ferrari, and Marco Zimmerling. 2013. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM.

[27] Olaf Landsiedel, Euhanna Ghadimi, Simon Duquennoy, and Mikael Johansson. 2012. Low power, low delay: opportunistic routing meets duty cycling. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*. IEEE.

[28] Yingyu Liang, Maria-Florina Balcan, and Vandana Kanchanapally. 2013. Distributed PCA and k-means clustering. In *The Big Learning Workshop at NIPS*.

[29] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. 2013. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*. IEEE.

[30] Mohammad Ahamdi Livani and Mahdi Abadi. 2010. Distributed PCA-based anomaly detection in wireless sensor networks. In *Proceedings of the 5th International Conference for Internet Technology and Secured Transactions*. IEEE.

[31] Michael Luby. 2002. Digital Fountain, Inc. Email: luby@digitalfountain.com. (2002).

[32] David JC MacKay. 2005. Fountain codes. *IEE Proceedings - Communications* (2005).

[33] Sergio Valcarcel Macua, Pavle Belanovic, and Santiago Zazo. 2010. Consensus-based distributed principal component analysis in wireless sensor networks. In *Proceedings of the 11th International Workshop on Signal Processing Advances in Wireless Communications*. IEEE.

[34] Petar Maymounkov. 2002. *Online codes*. Technical Report. Technical report, New York University.

[35] Mobashir Mohammad, Manjunath Doddavenkatappa, and Mun Choon Chan. 2017. Improving Performance of Synchronous Transmission-Based Protocols Using Capture Effect over Multichannels. *ACM Transactions on Sensor Networks* (2017).

[36] Mobashir Mohammad, XiangFa Guo, and Mun Choon Chan. 2016. Oppcast: Exploiting spatial and channel diversity for robust data collection in urban environments. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE.

[37] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. 2006. Cross-level sensor network simulation with cooja. In *Proceedings of the 31st International Conference on Local Computer Networks*. IEEE.

[38] Joseph Polastre, Robert Szewczyk, and David Culler. 2005. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Conference on Information Processing in Sensor Networks*. IEEE Press.

[39] Chayan Sarkar. 2016. LWB and FS-LWB implementation for Sky nodes using Contiki. arXiv preprint - https://arxiv.org/pdf/1607.06622.pdf. (2016).

[40] Mo Sha, Gregory Hackmann, and Chenyang Lu. 2011. ARCH: Practical channel hopping for reliable home-area sensor networks. In *Proceedings of the 17th Real-Time and Embedded Technology and Applications Symposium*. IEEE.

[41] Amin Shokrollahi. 2006. Raptor codes. *IEEE Transactions on Information Theory* (2006).

[42] Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. 2006. Experimental study of concurrent transmission in wireless sensor networks. In *Proceedings of the 4th ACM Conference on Embedded Network Sensor Systems*. ACM.

[43] Yin Wang, Yuan He, Xufei Mao, Yunhao Liu, and Xiang-yang Li. 2013. Exploiting constructive interference for scalable flooding in wireless networks. *IEEE/ACM Transactions on Networking* (2013).

[44] Thomas Watteyne, Ankur Mehta, and Kris Pister. 2009. Reliability through frequency diversity: why channel hopping makes sense. In *Proceedings of the 6th ACM Symposium on Performance Evaluation of Wireless Ad-Hoc, Sensor, and Ubiquitous Networks*. ACM.

[45] Tim Winter. 2012. RPL: IPv6 routing protocol for low-power and lossy networks. (2012).