CodeDrip: Improving data dissemination for wireless sensor networks with network coding

Nildo dos Santos Ribeiro Júnior[a], Rodrigo C. Tavares[a], Marcos A.M. Vieira[a,*],
Luiz F.M. Vieira[a], Omprakash Gnawali[b]

[a] *Universidade Federal de Minas Gerais Belo Horizonte, Brazil*
[b] *Department of Computer Science, University of Houston, Houston, TX, USA*

ABSTRACT

Network Coding is a technique that combines packets in the network with the potential to mitigate packet lost. In wireless communication systems, traditionally, dropped packets are recovered using retransmissions. By combining packets using network coding, it is possible to recover the transmitted information without needing to retransmit all the lost packets to all the nodes. In this paper, we present a show-case that applying Network Coding to data dissemination for Wireless Sensor Networks provides benefits even for small values. CodeDrip is a data dissemination protocol with Network Coding capability. Dissemination is typically used to query nodes, send commands, and reconfigure the network. CodeDrip utilizes Network Coding to improve energy efficiency, reliability, and speed of dissemination. While previous work in combining network coding and dissemination focused on bulk data dissemination, we optimize the design of CodeDrip for dissemination of small values. We perform extensive evaluation of CodeDrip on simulations and a large-scale testbed and compare against the implementations of Drip, DIP and DHV protocols. We evaluate with more than 350 physical sensor nodes. Results show that CodeDrip is faster, smaller and sends fewer messages than Drip, DHV and DIP protocols.

1. Introduction

Network Coding [3] is a technique that combines packets in the network with the potential to increase throughput, decrease energy consumption, and reduce the number of messages in Wireless Communication Systems.

Wireless Sensor Network (WSN) consists of a large number of nodes with sensing, computation, and wireless communication capability. This sensor network is typically deployed to collect data from the environment or other physical spaces. Wireless communication and energy efficiency are key requirements for WSNs, especially in applications where we retrofit existing infrastructure.

Many WSN applications require the capability to send messages from a base station or controller node to all the nodes in the network. This type of communication pattern is called *dissemination*, or one-to-many. Dissemination is typically used to send commands, query, reconfigure and reprogram the network. A data dissemination protocol for sensor networks needs to overcome several

challenges. First, the energy in each sensor node is limited by the battery or energy harvesting capacity, thus it is important to save energy to increase the sensor node's operational lifetime. Second, sensor nodes typically do not have powerful CPUs, so they might not be capable of executing complex matrix operations or complex communication protocols. Finally, wireless communication is susceptible to transmission errors and packet loss. A dissemination protocol should not only be reliable and energy efficient, but also fast.

In this paper, we present CodeDrip,[1] a data dissemination protocol for Wireless Sensor Networks. CodeDrip uses Network Coding to improve energy efficiency, reliability, and speed of dissemination. Instead of simply retransmitting received data packets, sensor nodes combine various packets into one, and retransmit the combined packet to its neighbors. Therefore, packet loss is mitigated since lost packets might be recovered through the decoding of others combined packets. By avoiding retransmission, the dissemination process might finish faster.

Existing data dissemination protocols for Wireless Sensor Networks present a tradeoff: save energy at the expense of dissemina-

---

* Corresponding author.
  E-mail addresses: nildo@dcc.ufmg.br (N.d.S. Ribeiro Júnior),
rodrigoct@dcc.ufmg.br (R.C. Tavares), mmvieira@dcc.ufmg.br (M.A.M. Vieira),
lfvieira@dcc.ufmg.br (L.F.M. Vieira), gnawali@cs.uh.edu (O. Gnawali).

---

[1] This work is based on preliminary conference version [8].

Network Coding — CodeDrip — Rateless, AdapCode

Traditional — Drip — DIP, DHV — Deluge

Small Data (One Value) — Small Data (Grouped) — Bulk Data
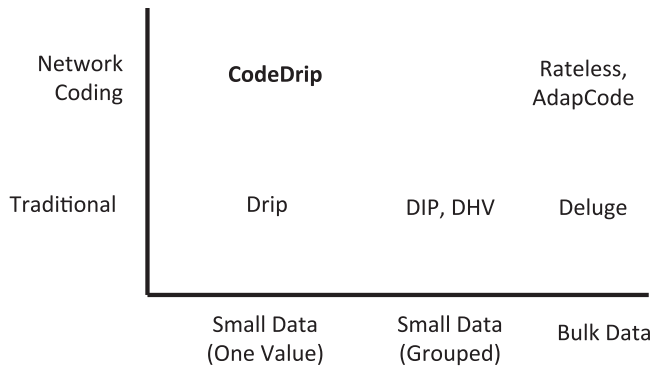
Fig. 1. Selected classes of dissemination protocols in sensor network. CodeDrip uses network coding to make dissemination of small data efficient and fast.

Fig. 2. Drip example. There are three values to be disseminated. Each value is independently associated with a Trickle timer. Each packet has a value (rectangles on the left side). Each dot represents when the timer fires and a message is sent.

tion speed. These protocols use transmission of summaries or version numbers and attempt to selectively transmit the missing data to avoid redundant transmission. While this strategy saves energy, it could incur large delays. Through extensive experiments, we find that CodeDrip provides faster data dissemination while transmitting fewer messages than most previous approaches.

Network Coding is not a new idea in wireless communication. However, previous work such as COPE [16], cannot be applied to WSNs because these algorithms require large memory overhead. Network coding schemes such as Rateless Deluge [13] and Adap-Code [14] have been previously proposed in wireless sensor networks and are shown to have low memory and computational overhead. However, these WSN dissemination protocols are designed for bulk data dissemination. There has been no previous study of effectiveness of network coding in dissemination of *small values*. Previously, it was thought that there would be limited opportunity to combine packets in dissemination of small values, hence the focus on bulk data dissemination. We identify the opportunity to make dissemination of small values efficient with network coding and fill this gap in sensor network protocol design space.

Our main contributions are as follows. First, we present the design and implementation of CodeDrip. Second, we study the performance of CodeDrip through extensive simulation and testbed experiments on the KanseiGenie testbed [26]. We evaluate CodeDrip with more than 350 physical sensor nodes. Our work is a realistic show-case that Network Coding benefits real-world applications. Third, we compare CodeDrip to the data dissemination of small values with Drip, DIP [23] and DHV [5] and quantify the Network Coding gain and show that Network Coding is useful even for dissemination of small values.

Our work is organized as follows. In the next section, we present the related work. CodeDrip is explained in Section 3. We present the simulation results in Section 4 and the testbed experiments in Section 5. We conclude in Section 6.

2. Related work

There is a large body of work in dissemination protocols for WSN. Fig. 1 summarizes the major classes of dissemination protocols. In this section, we describe how some of the dissemination protocols fit in the design space of dissemination protocols and application of network coding in dissemination.

2.1. Dissemination of small values

Trickle [21] is used as a building block for a number of dissemination protocols that propagate code or small values in WSN. Trickle has two key features that allows it to be efficient: the timer

control algorithm and duplicate suppression. Trickle timer doubles its interval every time it fires. Thus, over the long run, the interval converges to a very large maximum value. The timer can be reset to a small value when a new message needs to be sent. Trickle uses version numbers to detect and suppress duplicate transmissions. A node periodically broadcasts its version but stays silent and increases the interval if it hears several messages from its neighbors containing the same version number. When a node receives a new version number, the node resets the timer and transmits the message. CodeDrip uses Trickle timer in its design.

Some works [9,27] study the Trickle performance. In [27], the authors present an analytical study of the performance of Trickle algorithm for data dissemination. In [9], the authors analyzed Trickle parameters and proposed a modification when a new node joins the network. Moreover, the CTP work[11] evaluates the Trickle adaptive beaconing and presents some discussion about Trickle timers and different parameters.

Drip [28] is the simplest data dissemination protocol that uses Trickle timer. Each time an application transmits a message with Drip, a new version number is used. The new version number causes the protocol to reset the Trickle timer and thereby transmissions in the network to disseminate the new value. Redundant transmissions are detected using version numbers and suppressed. When the application does not inject new messages, the timer interval increases which causes the control overhead to level off. When a new message is injected, the new version number causes the timer to reset and the nodes disseminate the message. Fig. 2 shows how Drip works when it is used to disseminate three values. Dissemination of each message is paced by its own Trickle timer.

DIssemination Protocol (DIP) [23] is a data discovery and dissemination protocol. DIP continuously measures network conditions and estimates whether each data item requires an update. In DIP, a node periodically broadcasts a summary message, containing hashes of its keys and versions. A hash-tree based algorithm detects if there is an update. DIP scales logarithmically with the total number of items. DHV [5] is a code consistency maintenance protocol. DHV's key contribution is its technique to efficiently determine when to perform code updates. DHV detects and identifies which code item need updates at the bit level. DHV uses the Trickle timer to control transmission rate and duplicate suppression.

DIP and DHV are examples of dissemination protocols that operate at the level of a group of messages (for example, to compute summaries). On the other hand, Drip operates at the granularity of a single message. In DIP and DHV, all nodes must agree on a fixed set of data item identifiers before dissemination. DIP and DHV can scale to a large number of data item updates, however perform worse than Drip on small number of data items or updates [12].

2.2. Bulk and middle-sized data dissemination

A different set of dissemination protocols have been proposed for middle or large-sized objects. Maté [19] and Tenet [24] optimize the design of their dissemination protocols for middle-sized objects. Maté virtual machine disseminates code capsules to install

small virtual programs. Tenet disseminates tasks, enabling users to decide what to run in the sensor network during run-time.

Deluge [15] is a data dissemination protocol for code updates. It focuses on disseminating bulk data. Several optimizations in storage, buffering, and transmission enable it to efficiently disseminate objects (such as sensor node code) that do not fit in the limited RAM of sensor nodes. Deluge also uses Trickle timer to time its control packets.

2.3. Network coding and dissemination

Early work in network coding showed that, in general, innetwork encoding of packets could achieve an optimal capacity that cannot be realized via any feasible routing-only scheme [2]. It was shown that network coding can achieve multicast capacity while routing-only scheme may not. Follow on work showed that it is sufficient for the encoding function to be linear [22]. In wireless environments, network coding has been demonstrated to offer several benefits, such as improved energy efficiency [6](by reducing the number of distinct transmissions), higher throughput and robustness (by allowing nodes to receive potentially multiple copies of a single packet).

A number of network-coding protocols have been proposed in wireless communication, for example COPE [16] and CodeTorrent [18]. COPE [16] demonstrated that the use of network coding can improve the overall wireless network throughput. CodeTorrent [18] performs content distribution in VANETs using network coding. Unfortunately, a direct application of these protocols from mesh networks to WSN is challenging primarily due to memory constraints on sensor nodes that limit the cache size for overheard packets.

Researchers have proposed network coding protocols with low memory and computational overhead. Keller et al. [17] experimentally investigate the delay of flooding based multicast protocols for a sniper detection application using network coding. DutyCode [4] combines the idea of Network Coding and duty-cycle in the MAC layer.

Rateless Deluge [13] is an implementation of Deluge with Network coding. AdapCode [14] is another reliable data dissemination protocol for code update with Network Coding. Both protocols take advantage of Network Coding to improve reliability and send fewer messages than Deluge. These protocols are optimized for bulk data dissemination and have high memory overhead and running time of $O(n^3)$ due to their use of Gaussian elimination. CodeDrip is optimized for dissemination of small values and uses XOR operator for encoding and hence has low computational overhead.

A data dissemination protocol in WSNs with network coding is also presented in [29], but their focus is different from ours. They assume a TDMA-like MAC and focus on a packet-scheduling, determining which packets to combine and transmit given the radio on-off schedule. They prove that the problem is NP-hard and provide a LP formulation. In contrast to their system, CodeDrip runs on CSMA MAC. We also provide a rigorous testbed-based evaluation of dissemination of small values using network coding.

Splash [7] is a dissemination protocol for large objects in WSNs. The approach of this protocol is to combine the advances in constructive interference broadcast and multiple-channel pipelining to eliminate contention overhead among the nodes in the network. Its constructive interference established that concurrent transmissions by notes synchronized can improve reception rates by neighbors. Splash uses a mechanism on its final round to recover missing packets at different nodes, where each node broadcasts the packets it has XOR'ed with 19 other packets. This ensures that every single packet is retransmitted at least once, and the probability of the node being able to decode the missing packets is high. Splash only uses XOR coding to recover lost packets at the end of the dissemination process.

MT-Deluge [10] is a multi-threaded design for a coding-based data dissemination protocol. The main idea is to separate the coding operation and the transmission operation into two threads. An adaptive packet-level synchronization provides precise synchronization between multiple threads and an incremental decoding algorithm is proposed where the receiver starts to decode after obtaining a small number of packets. This algorithm can make use of the idle-durations in receiving, mitigating the impact of decoding to dissemination delay.

Synapse [25] is a protocol which was designed to improve the efficiency of the error recovery phase using network coding to encode a packet before transmission. Synapse implements a hybrid ARQ (HARQ) where data are encoded prior to transmission and incremental redundancy is used to recover from losses. For the coding, digital Fountain Codes were selected as they are rateless and allow for lightweight implementations. After receiving an expected number of encoded packets, the receiving node uses an alternative decoding algorithm by Gaussian elimination with a high probability of packets with a high degree.

3. Algorithm

CodeDrip uses Network Coding to improve the efficiency of dissemination in Wireless Sensor Networks. In a sensor network deployment, we often need to disseminate different information (e.g., configuration, commands) to the network. With network coding, we can combine different packets and make dissemination more resilient to failure. We can recover lost packets if the sensor node had received a combined message and an original message.

Like Drip, CodeDrip uses the Trickle timer to time the message transmissions with the goal that the data will eventually arrive at all the nodes in the network. Unlike Drip, CodeDrip sometimes combines messages and transmits the combined messages. CodeDrip does not use topology information to inform its decision about which messages to combine, thus has no control overhead for topology discovery. Later we will describe when CodeDrip decides to transmit combined messages.

To combine messages, different network coding protocols use different operators. CodeDrip uses the ⊕ operator, which is a Galois field of 2, instead of a more complex finite field. This choice allows CodeDrip to run efficiently on resource constraint nodes. On most CPUs, the XOR operator is just one hardware instruction.

We modify the packet format for Drip to accommodate the control fields required by network coding. The decoding process needs to know which messages were combined to create the given payload. We add to the message header a field indicating what messages where combined. Each data to be disseminated has a 1 byte identifier. For more than 256 items, we could extend this identifier. Each message, besides its payload, has a set of these identifiers. This set of identifiers is the CodeDrip overhead necessary for the coding and decoding processes. The unmodified Drip message has only one identifier and its payload is the data to be disseminated. A combined message has two or more identifiers corresponding to the packets that were combined. The payload consists of the result of applying the XOR operator among the data that are identified by the list of identifiers.

Fig. 3 shows how CodeDrip works. There are three values to be disseminated. Each value has an associated Trickle timer. When the Trickle timer fires (represented by a dot), a message is sent. Fig. 3 also illustrates the message content, where packets are combined before transmission.

Each sensor node has two buffers, one for the original messages and one for the combined messages, which are initially empty. After receiving a message, the sensor node verifies if the message is
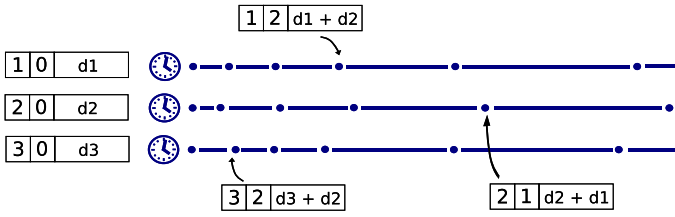
Fig. 3. CodeDrip example. There are three values to be disseminated. Each value has an associated Trickle timer. Each packet transmission might combine the packets.

original or combined. If the message is original, the sensor node stores it in the original message buffer and will transmit this message when the Trickle timer fires. Drip also requires an original message buffer to store the data item and its version number.

If the sensor node receives a combined message, the sensor node checks if it can decode the new message from the original messages which it had already received. If it is not possible, the message will be stored in the combined buffer until new messages make it possible to decode the message. In practice, for example, consider the case where only two messages can be combined, the node will only store a combined message in the buffer if it does not have any of its combined packets.

There are two probabilistic decision policies in CodeDrip. The first policy decides what to do when a sensor node receives a message it has already received. Receiving the same message many times indicates that its neighbors already have the message. Thus, it is reasonable that the sensor nodes does not need to send the message right away, since this message is not missing in the neighborhood. A sensor node might decide to suppress the message, in other words, delay sending the message. This process is called suppression and is also present in Drip. The suppression probability determines if the protocol should suppress the message.

The second policy decides if CodeDrip should send the original message or a combined message. This decision is made before sending the message, in other words, when the Trickle timer fires. Each original message is associated with a timer. When this timer fires, the sensor node decides to either send the original message, or to combine with other messages and to send the combination. The other messages to be combined are selected randomly. The combination can happen independently at every node and not just the node that initially generated the packet. The combination probability affects the protocol performance and is evaluated in Section 5.

Since CodeDrip has the potential to decode more than one message with just one transmission, CodeDrip suppresses sending unnecessary messages faster than Drip. For example, CodeDrip can receive the combination of two redundant messages and delay the sender timer of these two messages in a single step while Drip would need to receive both messages.

4. Simulation experiments

We first perform a set of simulation experiments to study the performance of CodeDrip and how its performance compares to other dissemination protocols. Although simulation experiments use modeled wireless propagation characteristics, which might be very different from what we find in realistic wireless networks, the results are nevertheless helpful in understanding the basic high-level properties of the protocols.

4.1. Evaluation methodology

We generated WSN topologies by placing the sensor nodes in random locations in the network and constructing a communica-
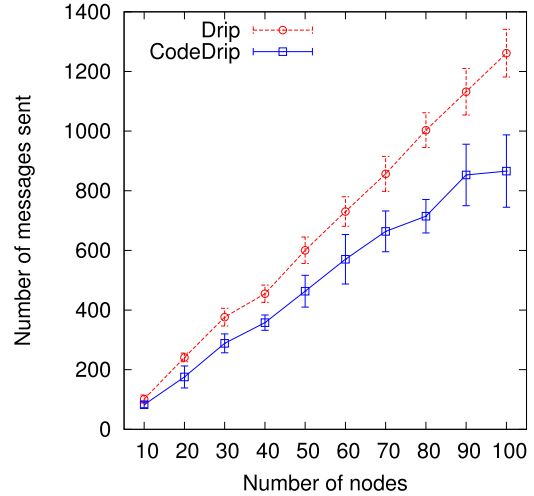

Fig. 4. Number of messages sent for Drip and CodeDrip for different network sizes.

tion graph where the edge weights represent packet loss rate on that link. We made sure that all the topologies are connected. We simulate each scenario ten times and we report the median of the metrics. In each simulation, the network disseminated three different data items. We use the simulator called TOSSIM [20] for our experiments. TOSSIM is designed to simulate TinyOS applications. We use these metrics in our study:

- Efficiency: We use the number of messages as a measure of efficiency of the protocol. With energy efficient and duty-cycling MAC, fewer number of transmissions typically leads to less energy expenditure.
- Reliability: We use the fraction of nodes that receive the disseminated message as a measure of reliability.
- Speed: We use the time it takes for all the nodes to receive the disseminated message as a measure of speed. We call it dissemination time or latency.

4.2. Performance analysis

In the first set of simulations, we compare the number of transmitted messages for CodeDrip and Drip. Fig. 4 shows the performance of Drip and CodeDrip for different number of sensor nodes. Roofnet project [1] collected link-level measurements from a realistic mesh network and shown that most communication pairs (sender/receiver) have intermediate (50%) delivery probabilities. Thus, we fix the packet loss rate to 50%. This setting allows us to study CodeDrip's performance in a realistic network. We found that, in larger networks, the overhead for CodeDrip relative to Drip improves as shown in Fig. 4. In all our simulation experiments, CodeDrip sends fewer total messages than Drip.

In the second set of simulations, we compare the dissemination latency of CodeDrip and Drip. Fig. 5 shows the cumulative fraction of sensor nodes that already received all the disseminated messages according to the dissemination process time in topologies with 100 sensor nodes. Observe that Drip suffers from packet loss in the initial stage since there are few nodes transmitting and many packets are lost. In CodeDrip, even when there are few nodes transmitting in the beginning, the data propagates through larger part of the network. We can also observe that dissemination latency is shorter for CodeDrip. While some nodes do not receive the last message with Drip, increasing the dissemination time, with CodeDrip the combined messages increases the probability of receiving the last message, decreasing the dissemination
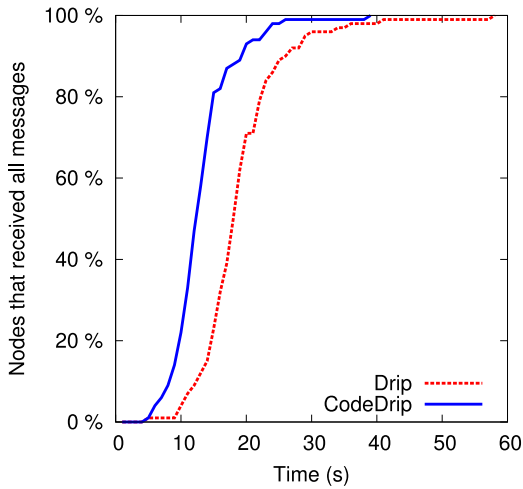
Fig. 5. Number of sensor nodes (in percentage) that already received all messages over time. Topology has 100 sensor nodes and 50% packet loss rate.
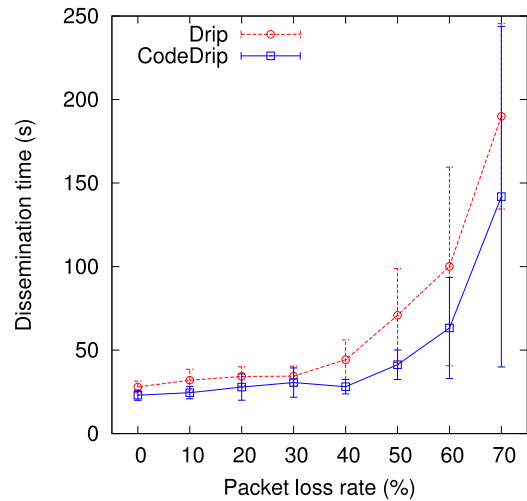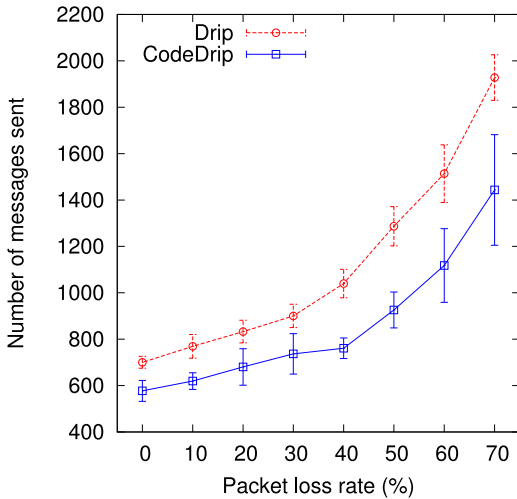

Fig. 6. Number of messages sent for Drip and CodeDrip for different packet loss rates in topologies with 100 sensor nodes.
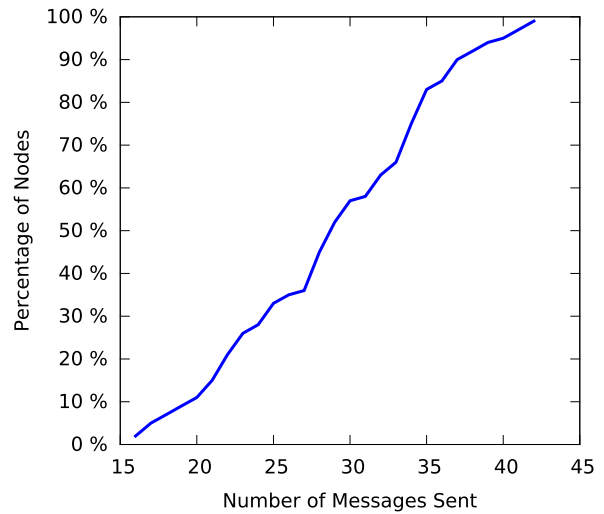

Fig. 7. Dissemination time for Drip and CodeDrip for different packet loss rates in topologies with 100 sensor nodes.

Fig. 8. The distribution of packet sent per sensor node with 10 messages being disseminated.

latency. **Summary:** Figs. 4 and 5 show that CodeDrip is faster and more efficient than Drip.

In the third set of simulations, we study how the link quality affects CodeDrip and Drip protocols. Figs. 6 and 7 illustrate the performance of Drip and CodeDrip as we vary the link quality in the network with a 100 sensor node topology. We observe that CodeDrip is less affected by the packet loss rate than Drip since CodeDrip needs fewer messages to finish the process. CodeDrip also has smaller dissemination latency compared to Drip. When the packet loss rate is higher than 50%, the measurements have higher variation, which is expected because different decisions are made in dissemination depending on whether the packets are successfully received. **Summary:**Figs. 6 and 7 indicate that CodeDrip gain increases in lossy topologies.

These results indicate that the dissemination process sends fewer messages when coupled with Network Coding technique. The gains are improved when the packet loss rate are higher.

Next, we measured the number of packets sent by each node during dissemination process. We disseminated 10 messages in random topologies with 100 sensor nodes. Fig. 8 shows the cumulative distribution probability (cdf) of number of packet sent. Y-axis represents the fraction of sensor nodes. X-axis indicates the number of messages sent. We can observe that 50% of nodes sent

30 or fewer packets. This metric is important to evaluate the load balance behavior in wireless sensor network.

4.3. Buffer size analysis

Here, we evaluate how many positions in the buffer are actually used during the dissemination process. We modified the source code of CodeDrip to report the number of messages stored in the buffer in each node at each time. Then we ran several simulations with this implementation using TOSSIM.

Our simulated network has 100 nodes. It is generated randomly making sure that every node inserted is connected to the network.

Fig. 9 presents a histogram using blue bars for a single simulation disseminating 50 messages. In the x axis, we have the maximum buffer size used by a node. The y axis represents how many nodes that fall in each category. We see that most nodes used at most 3 buffer positions during the dissemination process. The node that used more buffer positions need 12 positions.

Fig. 9 also presents cumulative distribution with a line. It is easy to see through this chart that 80% of the nodes of the network used a maximum of 4 positions in the buffer. If we set the maximum buffer size to 5 in this case, approximately 8% would loose
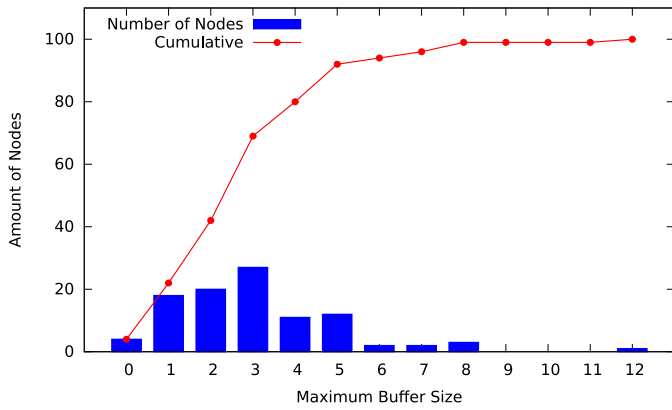
Fig. 9. The distribution of the maximum utilization of the buffer in the experiment with 50 messages being disseminated. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)
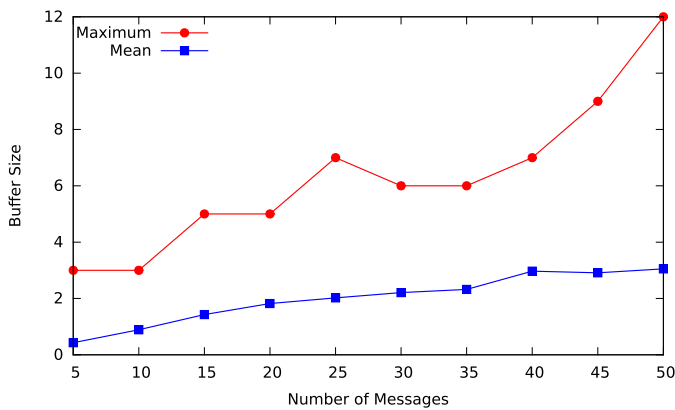

Fig. 10. Chart showing the maximum and mean buffer sizes in each simulation.

Fig. 11. Number of messages sent for CodeDrip and Drip as we increase the number of physical sensor nodes used in the testbed experiments.

some combined message. Since the number of possible combinations of messages grows exponentially with the number of messages, a combined message usually does not take too long to be decoded. Remember that a combined messages only stays in the buffer until it is able to be decoded.

Next, we evaluate how the number of messages being disseminated affects the buffer utilization. We varied the number of messages from 5 to 50, increasing it by 5 each time. In Fig. 10, we show the maximum and the mean buffer size for of all 100 motes in the network for each experiment. We can observe that the maximum utilization of the buffer grows quickly, but as we previously observed, very few nodes actually use the maximum buffer size in the network. The mean of the buffer size does not grow as the maximum buffer size. Even if we increase the number of the messages being disseminated in the network, the overhead used with the buffer in practice do not increase as the number of possible combination of messages. We can use relatively small buffers without compromising too much of the performance.

5. Testbed experiments

In this section, we describe testbed experiments used to validate the performance results for CodeDrip in a more realistic wireless environment.

5.1. Evaluation methodology

Testbed experiments involve implementing and running the protocol code on physical motes and collecting instrumentation data to understand the performance.
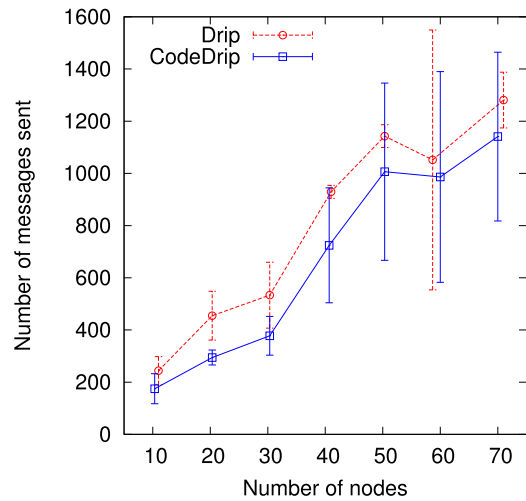
We used the public KanseiGenie Wireless Sensor Network Testbed [26] to run our experiments with real sensor motes. This testbed provides access to TelosBs. The motes on the testbed have a 3 dB attenuator attached to their antennas. At the lowest power level of transmission, the reliable range is 3 feet. Thus, this testbed allows us to evaluate protocols in a realistic wireless environment with a mix of reliable and unreliable links. We run the protocol code for CodeDrip, Drip, DIP, and DHV on this testbed and collect performance information.

To collect data related to the performance of the protocol, we instrumented the code of the protocols to send a message to the serial port of the TelosB motes. The messages contain the id of the mote, the number of packets the mote sent since the beginning of the dissemination and the number of messages the mote have already received. The system adds a timestamp when a message is sent. These messages are then sent and stored in a central server from which we can download them for performance analysis.

5.2. Performance analysis

In the first set of testbed experiments, our goal is to compare CodeDrip performance with Drip for different network sizes. Figs. 11 and 12 show the number of messages sent and the dissemination time for Drip and CodeDrip, respectively. We disseminated 10 values from the sink node. We tested topologies from 10 to 70 sensor nodes. We changed the size of the network by programming the protocol code on select nodes and programming "Blink" program on rest of the nodes. Each point on the graph represents an average of results from 3 experiments and the error bar is the standard deviation. **Summary:** Figs. 11 and 12 show that CodeDrip is more efficient and faster than Drip.

In the second set of experiments, we study the impact of suppression and combination probability on different performance metrics. The probability of suppressing a received message or combining multiple packets into a single packet before transmission affects the performance of CodeDrip protocol. These results can give us guidelines on how to configure these two parameters in a real-world deployment.

Fig. 13 depicts the number of sent messages and by the probability of suppression. The experiments contain topologies with 70 physical sensor nodes. For comparison against Drip, we add a horizontal line that indicates the Drip's average of sent messages and also Drip's average dissemination time. We can observe that, for better performance, the probability of suppression for CodeDrip
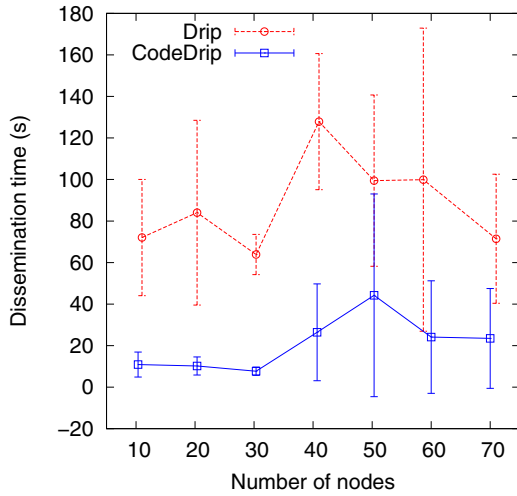
Fig. 12. Dissemination time for CodeDrip and Drip as we increase the number of physical sensor nodes used in the testbed experiments.
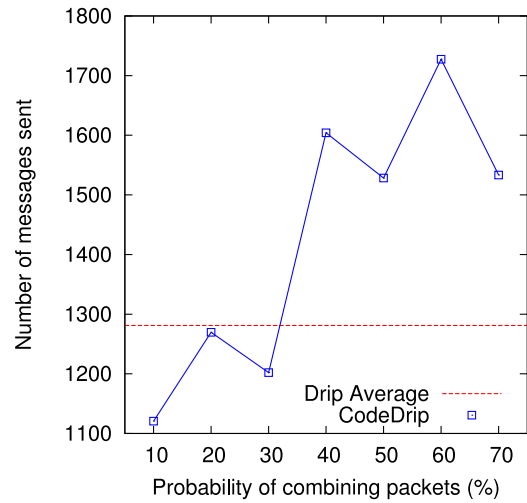

Fig. 14. Number of messages sent for CodeDrip and Drip for different probability of combining packets in the testbed experiments.
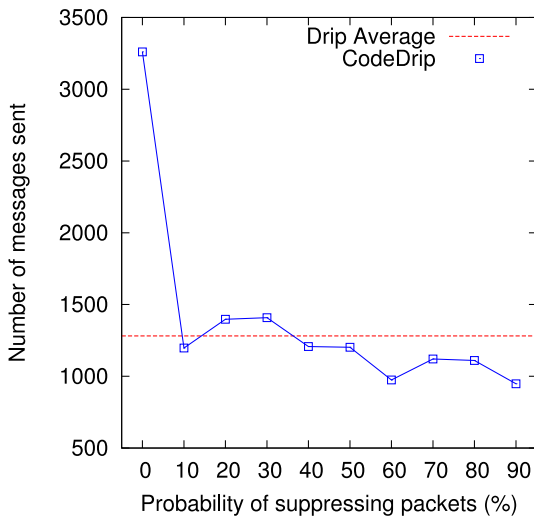

Fig. 13. Number of messages sent for CodeDrip and Drip for different probability of suppression in the testbed experiments.

Fig. 15. Percentage of nodes that received all the 10 disseminated values over time for CodeDrip, Drip, DIP, and DHV protocols in the testbed experiments with 70 motes.

should be set between 40% and 90%. In general, increasing the probability of suppression decreases the number of messages but, on the other hand, increases the dissemination time. **Summary:** Suppression Probability between 0.4 and 0.9 maybe appropriate.

Fig. 14 shows the results from experiments that analyze the impact of the probability of combination parameter on dissemination performance. The probability of combination determines if the node should combine the messages when the node transmits a message. The results suggest a trend where the number of sent messages increases with the larger probability of combination. We presume that by combining many times, we do not get an original message that could be useful to decode all previous messages. CodeDrip is better than Drip for values below 30%. **Summary:** Larger combining probability reduces decoding probability.

Thus, we investigated how the probability of suppression and combination affects CodeDrip performance. For the next set of experiments, we set the probability of combination to 30% and probability of suppression to 50%.

DIP and DHV are newer dissemination protocols that improve upon Drip for certain types of dissemination. Drip treats every data item as a separate entity for dissemination and DIP and DHV treat
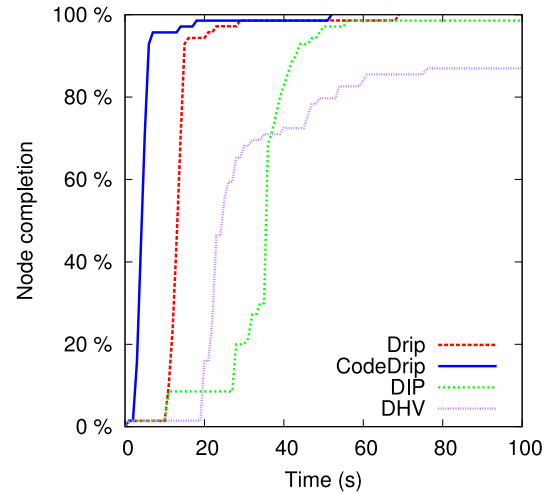
them as a group. We now evaluate CodeDrip against both DHV and DIP.

One of the benefits of applying Network Coding is that it reduces the dissemination time. Fig. 15 shows the evolution of dissemination with CodeDrip, Drip, DIP, and DHV protocols over time in a topology with 70 physical sensor nodes. The x-axis represents time since the dissemination process started. The y-axis indicates the percentage of nodes that received all 10 disseminated messages. For at least 90% of nodes to receive all messages, CodeDrip takes only 6 s, while Drip needs 15 s, DIP spends 43 s and DHV is not able to get 90% in over 100 s.

Fig. 16 shows the number of messages sent over time for CodeDrip, Drip, DIP, and DHV protocols in a topology with 70 physical sensor nodes. We observe that the CodeDrip and Drip show a faster growth in the number of sent messages at the beginning of the dissemination process, while DIP and DHV grow more slowly. This is a consequence of how the protocols are implemented. CodeDrip and Drip use different Trickle timer for each value that is being disseminated in the network. DIP and DHV use only one Trickle timer for all values. Although DIP sends fewer messages, not all nodes receive the dissemination content as shown in Fig. 15. Thus, CodeDrip transmits fewer packets than DHV and Drip. Although
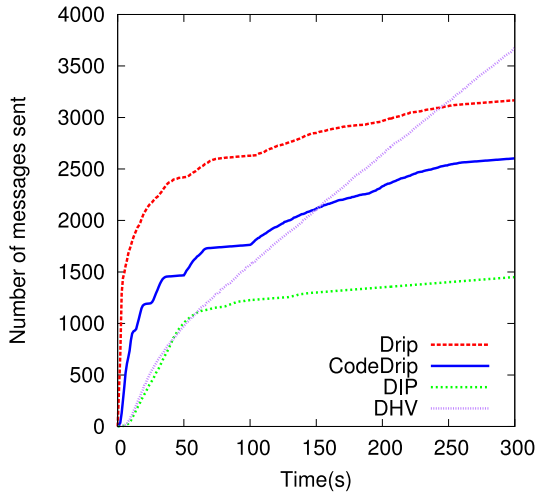
Fig. 16. Number of messages sent over time for CodeDrip, Drip, DIP, and DHV protocols in the testbed experiments with 70 motes.
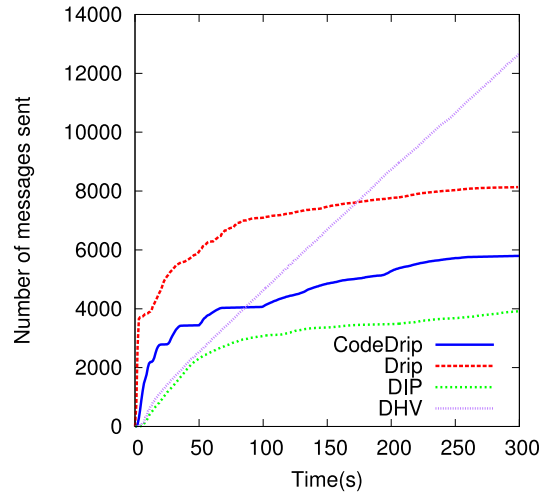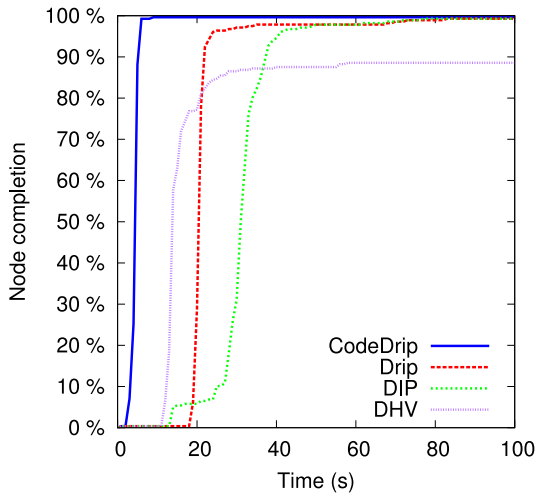

Fig. 17. Percentage of nodes that received all the 10 disseminated values over time for CodeDrip, Drip, DIP, and DHV protocols in a topology with 368 physical sensor nodes.

Fig. 18. Number of messages sent over time for CodeDrip, Drip, DIP, and DHV protocols in a topology with 368 physical sensor nodes.

Table 1
RAM, ROM and Code size usage from CodeDrip, Drip, DIP, and DHV.

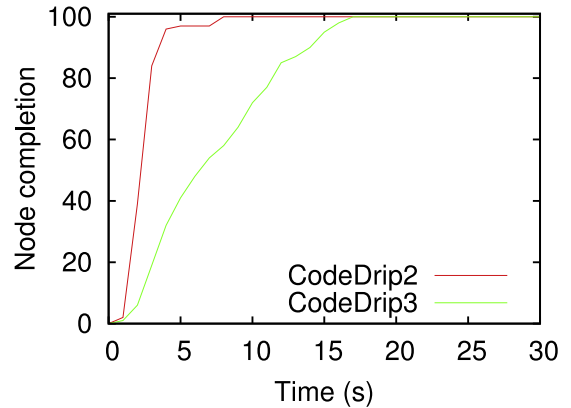| Metric	CodeDrip	Drip	DIP	DHV
RAM (bytes)	900	845	959	928
ROM (bytes)	17,980	21,274	22,130	21,478
Code size (bytes)	42,655	50,123	49,493	49,003

Fig. 19. Percentage of nodes that received all the 10 disseminated values over time for CodeDrip with 2 and 3 combined messages.

CodeDrip transmits more packets than DIP, CodeDrip does the job of completely disseminating the values, which was not the case for DIP.

Someone might wonder if these results hold for even larger networks. Thus, we conducted experiments with 368 physical sensor nodes, which was the maximum number of sensor nodes available for Kansei testbed. Fig. 17 shows the evolution of dissemination with CodeDrip, Drip, DIP, and DHV protocols over time in a topology with 368 physical sensor nodes. The x-axis represents time since the dissemination process started. The y-axis indicates the percentage of nodes that received all 10 disseminated messages. CodeDrip is the only dissemination protocol to finish before 20 s. Again, DHV is not able to get 90% in over 100 s.

Fig. 18 depicts the number of messages sent over time for CodeDrip, Drip, DIP, and DHV protocols with 368 physical sensor nodes. The behavior is the same as the one illustrated in Fig. 16, except for the number of messages sent on the x-axis. **Summary:** CodeDrip completes faster and is generally more efficient.

Finally, we compare memory overhead of the four dissemination protocols. Table 1 compares the RAM, ROM, and code size usage (in bytes) from CodeDrip, Drip, DIP and DHV. CodeDrip uses more RAM than Drip because of buffers. But, this overhead is marginal since it still consumes less memory than DIP and DHV. CodeDrip consumes fewer ROM memory than Drip, DHV and DIP. We conjecture that CodeDrip has smaller ROM size than Drip because all data was encoded in one interface while Drip provides one interface for each datum.

5.3. Extension

The next research question we address is: what if CodeDrip combined more messages? Thus, we extended CodeDrip to combine even 3 messages. We did experiments with 90 sensor nodes and 10 messages being disseminated. The suppression probability is 50% and the combination probability is 30%. We call CodeDrip2 (represented with red (darker) line) the CodeDrip version with 2 combined packets and CodeDrip3 (represented with green (lighter) line) the CodeDrip version that combines 2 and 3 packets. Fig. 19 depicts the percentage of nodes that received all the 10 disseminated values over time for CodeDrip with 2 and 3 combined mes-
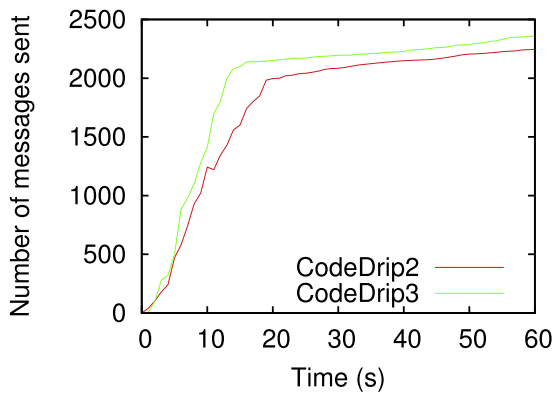
Fig. 20. Number of messages sent over time for CodeDrip with 2 and 3 combined messages.

sages. Both cases, all nodes received all messages before 20 s. CodeDrip 2 is faster than CodeDrip3. Fig. 20 illustrates the number of messages sent over time for CodeDrip with 2 and 3 combined messages. CodeDrip2 sends fewer messages than CodeDrip3. We conjuncture this is the same issue with the combination probability: by combining many times, we do not get an original message that could be useful to decode all previous messages.

6. Conclusions

Network Coding might mitigate packet lost in Wireless Communication System. Here, we presented a show-case of applying Network Coding to Wireless Sensor Networks. Unlike previous results, we show that Network Coding can benefits the data dissemination process of even small values. We developed CodeDrip, a data dissemination protocol for Wireless Sensor Networks that applies Network Coding in the dissemination process. We showed that this decreases the number of transmitted messages and consequently saves energy consumption. CodeDrip requires additional space in the packet to store message ids and buffers to store combined messages. These overheads can be controlled by specifying the maximum number of messages that can be decoded and the maximum buffer size. We evaluated the performance of CodeDrip with simulation and testbed experiments. We conducted experiments with more than 350 physical sensor nodes. Our results showed that CodeDrip is faster than Drip, DIP and DHV protocols to disseminate information. CodeDrip also requires less ROM memory than Drip, DHV and DIP. Furthermore, CodeDrip transmits fewer packets than DHV and Drip. Although CodeDrip transmits more packets than DIP, CodeDrip's dissemination reliability is higher than DIP's. Thus, CodeDrip is faster, smaller and sends fewer messages than Drip, DHV and DIP protocols.

Acknowledgement

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, R. Morris, Link-level measurements from an 802.11b mesh network, in: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and protocols for Computer Communications, in: SIGCOMM '04, ACM, New York, NY, USA, 2004, pp. 121–132.

[2] R. Ahlswede, N. Cai, S.-Y. Li, R. Yeung, Network information flow, Inf. Theory IEEE Trans. 46 (4) (2000) 1204–1216, doi:10.1109/18.850663.

[3] R. Ahlswede, N. Cai, S. yen Robert Li, R.W. Yeung, Network information flow, IEEE TRANS. INF. THEORY 46 (4) (2000) 1204–1216.

[4] R. Chandanala, R. Stoleru, Network coding in duty-cycled sensor networks, in: Networked Sensing Systems (INSS), 2010 Seventh International Conference on, 2010, pp. 203–210, doi:10.1109/INSS.2010.5572223.

[5] T. Dang, N. Bulusu, W.-c. Feng, S. Park, Dhv: a code consistency maintenance protocol for multi-hop wireless sensor networks, in: U. Roedig, C. Sreenan (Eds.), Wireless Sensor Networks, Lecture Notes in Computer Science, 5432, Springer Berlin Heidelberg, 2009, pp. 327–342, doi:10.1007/978-3-642-00224-3_21.

[6] S. Deb, Network coding for wireless applications: a brief tutorial, International Workshop on Wireless Ad-hoc Networks (IWWAN), 2005.

[7] M. Doddavenkatappa, M.C. Chan, B. Leong, Splash: fast data dissemination with constructive interference in wireless sensor networks, in: Presented as Part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013, pp. 269–282.

[8] N. dos Santos Ribeiro Júnior, M.A.M. Vieira, L.F.M. Vieira, O. Gnawali, Codedrip: data dissemination protocol with network coding for wireless sensor networks, in: EWSN, 2014, pp. 34–49.

[9] J. Eriksson, O. Gnawali, Poster Abstract: Synchronizing Trickle Intervals, in: Proceedings of the 11th European conference on Wireless sensor networks (EWSN 2014), 2014.

[10] Y. Gao, J. Bu, W. Dong, C. Chen, L. Rao, X. Liu, Exploiting concurrency for efficient dissemination in wireless sensor networks, IEEE Trans. Parallel Distrib. Syst. 24 (4) (2013) 691–700.

[11] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, P. Levis, CTP: an efficient, robust, and reliable collection tree protocol for wireless sensor networks, ACM Trans. Sens. Netw. (TOSN) 10 (3) (2013).

[12] O. Gnawali, L. Guibas, P. Levis, A case for evaluating sensor network protocols concurrently, in: Proceedings of the fifth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, in: WiNTECH '10, ACM, New York, NY, USA, 2010, pp. 47–54.

[13] A. Hagedorn, D. Starobinski, A. Trachtenberg, 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, in: IPSN '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 457–466.

[14] I.-H. Hou, Y.-E. Tsai, T. Abdelzaher, I. Gupta, Adapcode: adaptive network coding for code updates in wireless sensor networks, in: INFOCOM 2008. The 27th Conference on Computer Communications. IEEE, 2008, pp. 1517–1525, doi:10.1109/INFOCOM.2008.211.

[15] J.W. Hui, D. Culler, The dynamic behavior of a data dissemination protocol for network programming at scale, in: In Proceedings of the 2nd International, ACM Press, 2004, pp. 81–94.

[16] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, J. Crowcroft, XORs in the air: practical wireless network coding, IEEE/ACM Trans. Netw. 16 (3) (2008) 497–510.

[17] L. Keller, A. Karaagac, C. Fragouli, K. Argyraki, Evaluation of network coding techniques for a sniper detection application, in: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2011 International Symposium on, 2011, pp. 327–333, doi:10.1109/WIOPT.2011.5930035.

[18] U. Lee, J.-S. Park, J. Yeh, G. Pau, M. Gerla, Code torrent: content distribution using network coding in vanet, in: MobiShare '06: Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking, ACM, New York, NY, USA, 2006, pp. 1–5. http://doi.acm.org/10.1145/1161252.1161254.

[19] P. Levis, D. Gay, D. Culler, Active sensor networks, in: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2, in: NSDI'05, USENIX Association, Berkeley, CA, USA, 2005, pp. 343–356.

[20] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: accurate and scalable simulation of entire tinyos applications, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, in: SenSys '03, ACM, New York, NY, USA, 2003, pp. 126–137, doi:10.1145/958491.958506.

[21] P. Levis, N. Patel, D. Culler, S. Shenker, Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks, in: In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2004, pp. 15–28.

[22] S.-Y. Li, R. Yeung, N. Cai, Linear network coding, Inf. Theory IEEE Trans. 49 (2) (2003) 371–381, doi:10.1109/TIT.2002.807285.

[23] K. Lin, P. Levis, Data discovery and dissemination with dip, in: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, in: IPSN '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 433–444.

[24] J. Paek, B. Greenstein, O. Gnawali, K.-Y. Jang, A. Joki, M. Vieira, J. Hicks, D. Estrin, R. Govindan, E. Kohler, The tenet architecture for tiered sensor networks, ACM Trans. Sen. Netw. 6 (4) (2010) 34:1–34:44.

[25] M. Rossi, G. Zanca, L. Stabellini, R. Crepaldi, A.F. Harris III, M. Zorzi, Synapse: a network reprogramming protocol for wireless sensor networks using fountain codes, in: 2008 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, IEEE, 2008, pp. 188–196.

[26] M. Sridharan, W. Zeng, W. Leal, X. Ju, R. Ramnath, H. Zhang, A. Arora, From kansei to kanseigenie: architecture of federated, programmable wireless sensor fabrics, in: T. Magedanz, A. Gavras, N. Thanh, J. Chase (Eds.), Testbeds and Research Infrastructures. Development of Networks and Communities, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 46, Springer Berlin Heidelberg, 2011, pp. 155–165.

[27] M. Stolikj, T.M. Meyfroyt, P.J. Cuijpers, J.J. Lukkien, Improving the performance of trickle-based data dissemination in low-power networks, in: European Conference on Wireless Sensor Networks, Springer, 2015, pp. 186–201.

[28] G. Tolle, D. Culler, Design of an application-cooperative management system for wireless sensor networks, in: Second European Workshop on Wireless Sensor Networks (EWSN) Istanbul, Turkey, January 31, - February 2, 2005., 2005, pp. 121–132.

[29] X. Wang, J. Wang, Y. Xu, Data dissemination in wireless sensor networks with network coding, EURASIP J. Wireless Commun. Netw. 2010 (1) (2010) 465915, doi:10.1155/2010/465915.

Nildo dos Santos Ribeiro Júnior is a Master student at the Universidade Federal de Minas Gerais (UFMG). His research interests are in Computer Networking and Network Coding.

Rodrigo Costa Tavares is a junior undergrad at the Universidade Federal de Minas Gerais (UFMG). His research interests are in Computer Networking and Network Coding.

Marcos A. M. Vieira is a Professor of Computer Science at the Universidade Federal de Minas Gerais (UFMG). He received his undergraduate and M.S. at the Universidade Federal de Minas Gerais in Belo Horizonte, and M.S. and Ph.D. degrees in Computer Science from the University of Southern California (USC). His research interest is in Computer Networking.

Luiz F. M. Vieira is a Professor of Computer Science at the Universidade Federal de Minas Gerais (UFMG). He received his undergraduate and M.S. at the Universidade Federal de Minas Gerais in Belo Horizonte, and M.S. and Ph.D. degrees in Computer Science from the University of California Los Angeles (UCLA). His research interest is in Computer Networking.

Omprakash Gnawali is an Assistant Professor at the University of Houston, USA. He was a Postdoctoral Scholar at Stanford University, got his Ph.D. from the University of Southern California, and received his Masters and Bachelors degrees from the Massachusetts Institute of Technology. His research lies at the intersection of low power wireless networks and embedded sensing systems.