CodeDrip: Data Dissemination Protocol with Network Coding for Wireless Sensor Networks

Nildo dos Santos Ribeiro Júnior[1], Marcos A.M. Vieira[1], Luiz F.M. Vieira[1], and Omprakash Gnawali[2]

[1] Universidade Federal de Minas Gerais, Brazil
{nildo,mmvieira,lfvieira}@dcc.ufmg.br
[2] University of Houston, USA
gnawali@cs.uh.edu

Abstract. In this paper, we present CodeDrip, a data dissemination protocol for Wireless Sensor Networks. 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. Network coding allows recovery of lost packets by combining the received packets thereby making dissemination robust to packet losses. 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. Results show that CodeDrip is faster, smaller and sends fewer messages than Drip, DHV and DIP protocols.

1 Introduction

Wireless Sensor Networks (WSNs) consist of a large number of nodes with sensing, computation, and wireless communication capability. These sensor networks are typically deployed to collect data from the environment or other physical spaces. Many sensor networks have been deployed in outdoor environment such as forests and streets and in indoor setting such as buildings and factories. Wireless communication and energy efficiency are key requirements of these networks, especially in applications where we retrofit existing infrastructure with smart sensing and actuation capabilities.

Many WSN applications require the capability to send messages from a central server or controller node to all the nodes in the network. This type of communication pattern is called *dissemination*. Dissemination is typically used to query, send commands, 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 communication protocols. Finally,

B. Krishnamachari, A.L. Murphy, and N. Trigoni (Eds.): EWSN 2014, LNCS 8354, pp. 34–49, 2014.

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, 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 dissemination 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 [11], can not be applied to WSNs because these algorithms require large memory overhead. Network coding schemes such as Rateless Deluge [8] and AdapCode [9] 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 and require $O(n^3)$ instructions for decoding with the Gaussian elimination. 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 [20]. Third, we compare CodeDrip to the data dissemination of small values with Drip, DIP [18] 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 work related to CodeDrip. In Section 3, we give an overview of network coding. CodeDrip is explained in Section 4. We present the simulation results in Section 5 and the testbed experiments in Section 6. We conclude in Section 7.

2 Related Work

There is a large body of work in dissemination protocols for WSN. Figure 1 summarizes the major classes of dissemination protocols. In this section, we describe

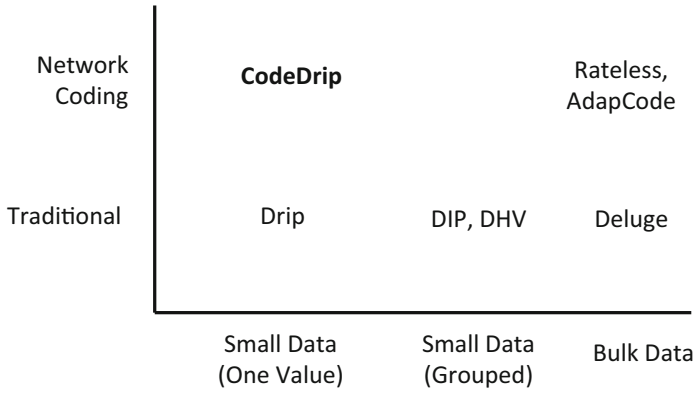

Fig. 1. Selected classes of dissemination protocols in sensor network. CodeDrip uses network coding to make dissemination of small data efficient and fast.

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 [16] 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.

Drip [21] 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. Figure 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) [18] 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

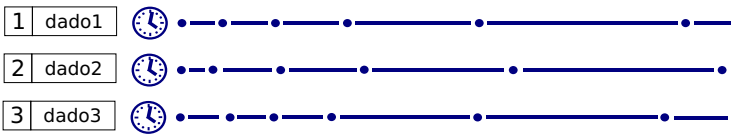

Fig. 2. Drip example. There are three values to be disseminated. Each value is independently associated with a Trickle timer. Each packet has a value (rectangles on the left side). Each dot represents when the timer fires and a message is sent.

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 [7].

2.2 Bulk and Middle-sized Data Dissemination

A different set of dissemination protocols have been proposed for middle or large-sized objects. Maté [14] and Tenet [19] 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 [10] 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, in-network 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 [17]. 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 [11] and CodeTorrent [13]. COPE [11] demonstrated that the use of network coding can improve the overall wireless network throughput. CodeTorrent [13] 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. [12] 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 [8] is an implementation of Deluge with Network coding. AdapCode [9] 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 [23], 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.

3 Network Coding

Network Coding [3] is a technique that combines packets in the network thereby increasing the throughput, decreasing energy consumption, and reducing the number of messages that are transmitted [22].

In wireless networks, 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. We explain how Network Coding works with a simple example.

Consider the topology in Figure 3. The sink node, at the center of the figure, wishes to disseminate two packets, denoted P_1 and P_2. Sensor nodes 1 and 2 are in the communication range of the sink node and might receive the packets. The sink node transmits packet P_1 but only sensor node 1 receives it. Later, the sink node transmits packet P_2, but at this time, only sensor node 2 properly receives packet P_2. Thus, each sensor node receives a different packet.

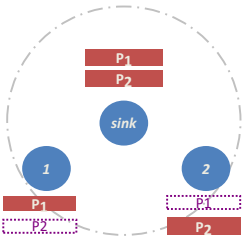

Fig. 3. Sink node needs to disseminate two packets to the nodes 1 and 2

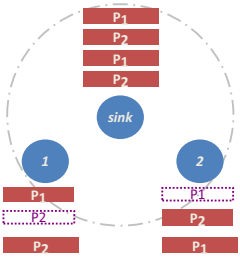

Fig. 4. With traditional retransmission, we need a total of 4 transmissions

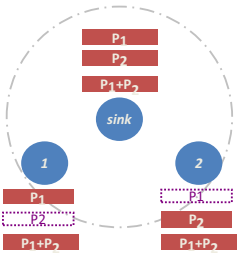

Fig. 5. With retransmission with Network Coding, we need only 3 transmissions

With the traditional approach, lost packets are recovered using retransmissions. In our example, the sink node retransmits packets P_1 and P_2. If these retransmissions are successful, sensor node 1 will receive the missing packet P_2 and sensor node 2 will receive the missing packet P_1. Thus, even if these first retransmissions are successful, we need a total of 4 transmissions for both the nodes to successfully receive the message. We show this scenario in Figure 4.

Network Coding allows packets to be combined using a logic operator. We can combine packets using, for instance, the XOR (exclusive or $\oplus$) operator. A new packet is created by performing a bit-wise XOR of each each bit in the packets P_1 and P_2. The new packet is of the same size as the packets P_1 and P_2.

In a network coding system, the sink node, instead of retransmitting packets P_1 and P_2, retransmits a new packet which is $P_1 \oplus P_2$. Sensor node 1, after receiving this packet, is capable of decoding packet P_2 by applying the XOR operator between the packet P_1, which it has already received, over the new packet. Thus, sensor node 1 decodes P_2 since $P_2 = P_1 \oplus (P_1 \oplus P_2)$. In the same way, sensor node 2 is capable of decoding P_1 when it receives the new packet and applies the XOR operator with packet P_2, which it has already received. Thus, sensor node 2 decodes P_1 since $P_2 = P_1 \oplus (P_1 \oplus P_2)$. Thus, we are able to recover both the packets with only three total transmissions, as shown in Figure 5.

This example illustrates the benefit of using Network Coding in the single hop topology example. The number of messages was reduced from 4 to 3. For a larger topology with many more hops, the gains are much larger.

4 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.

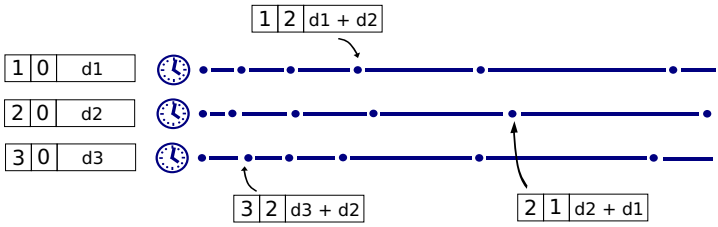

Fig. 6. CodeDrip example. There are three values to be disseminated. Each value has an associated Trickle timer. Each packet transmission might combine the packets.

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 $\oplus$ operator, which is a Galois field of 2, instead of a more complex finite field. This choice allows Drip 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 consist of the result of applying the XOR operator among the data that are identified by the list of identifiers.

Figure 6 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. Figure 6 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 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 6.

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.

5 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.

5.1 Evaluation Methodology

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

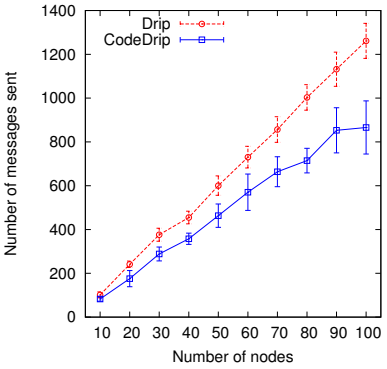

Fig. 7. Number of messages sent for Drip and CodeDrip for different network sizes

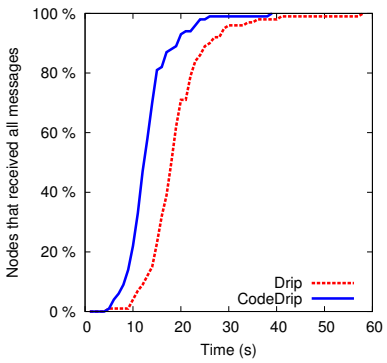

Fig. 8. Number of sensor nodes (in percentage) that already received all messages over time. Topology has 100 sensor nodes and 50% packet loss rate.

the metrics. In each simulation, the network disseminated three different data items. We use the simulator called TOSSIM [15] 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.

5.2 Performance Analysis

In the first set of simulations, we compare the number of transmitted messages for CodeDrip and Drip. Figure 7 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 Figure 7. 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. Figure 8 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

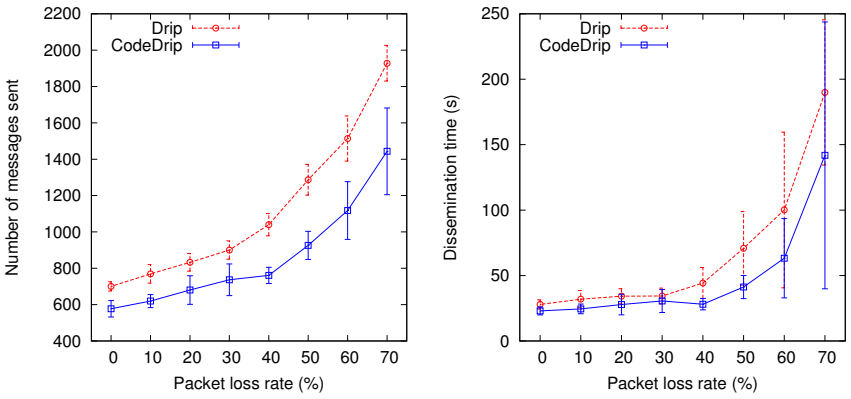

Fig. 9. Number of messages sent and Dissemination time for Drip and CodeDrip for differnet packet loss rates in topologies with 100 sensor nodes

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 latency.

In the third set of simulations, we study how the link quality affects CodeDrip and Drip protocols. Figure 9 illustrates 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 succssfully received.

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.

6 Testbed Experiments

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

6.1 Evaluation Methodology

Testbed experiments involve implementing and running the protocol code on physical motes and collecting instrumentation data to understand the performance.

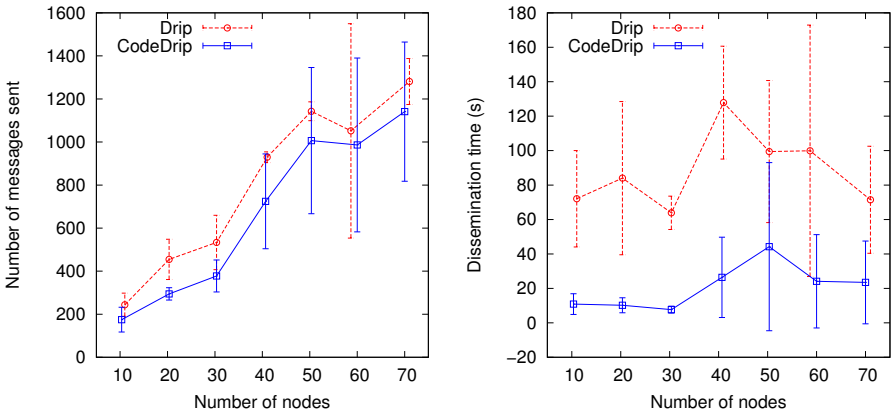

Fig. 10. Number of messages sent and dissemination time for CodeDrip and Drip as we increase the number of physical sensor nodes used in the testbed experiments

We used the public KanseiGenie Wireless Sensor Network Testbed [20] to run our experiments with real sensor motes. This testbed provides access to TelosBs. The motes on the testbed have a 3dB 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.

6.2 Performance Analysis

In the first set of testbed experiments, our goal is to compare CodeDrip performance with Drip for different network sizes. Figure 10 shows the number of messages sent and the dissemination time for Drip and CodeDrip. 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. Figure 10 shows that CodeDrip sends fewer messages than Drip and CodeDrip is 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

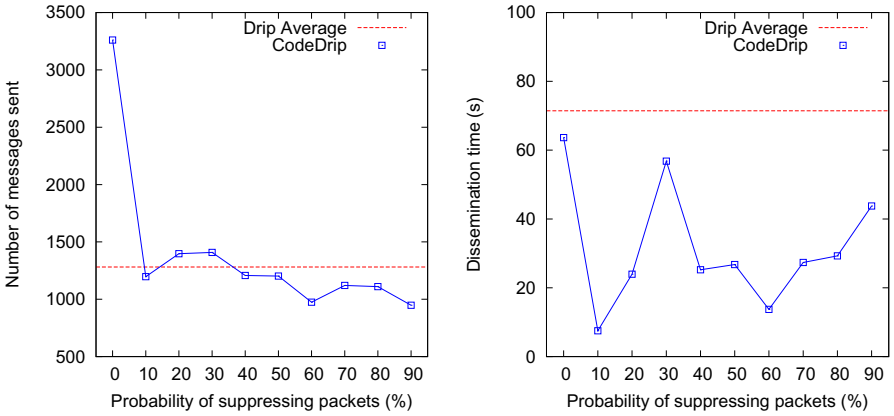

Fig. 11. Number of messages sent and dissemination time for CodeDrip and Drip for different probability of suppression in the testbed experiments

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.

Figure 11 depicts the number of sent messages and the dissemination time 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 small probability of suppression, CodeDrip sends more messages than Drip. However, the average dissemination time for CodeDrip is always smaller than Drip. For better performance, the probability of suppression should be set between 40% and 80%. In general, increasing the probability of suppression decreases the number of messages but, on the other hand, increases the dissemination time.

Figure 12 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%. Again, the average dissemination time for CodeDrip is always smaller than Drip. For better performance, the probability of combination should be set between 10% and 30%.

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%.

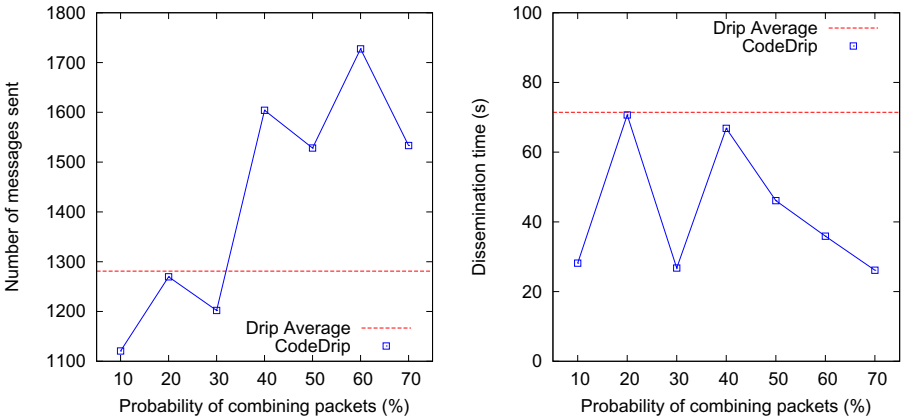

Fig. 12. Number of messages sent and dissemination time for CodeDrip and Drip for different probability of combining packets in the testbed experiments

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. Figure 13 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 seconds, while Drip needs 15 seconds, DIP spends 43 seconds and DHV is not able to get 90% in over 100 seconds.

Figure 14 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 Tricke timer for each value that is being disseminated in the network. DIP and DHV use only one Tricke timer for all values. Although DIP sends fewer messages, not all nodes receive the dissemination content as shown in Figure 13. Thus, CodeDrip transmits fewer packets than DHV and Drip. Although CodeDrip transmits more packets than DIP, CodeDrip does the job of completely disseminating the values, which was not the case for DIP.

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.

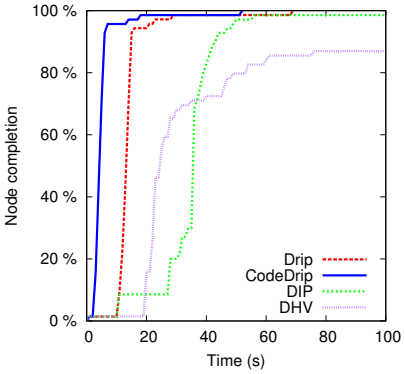

Fig. 13. Percentage of nodes that received all the 10 disseminated values over time for CodeDrip, Drip, DIP, and DHV protocols in the testbed experiments with 70 motes

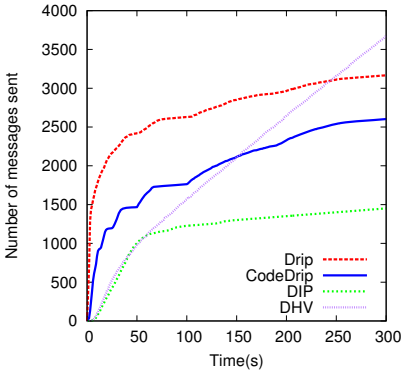

Fig. 14. Number of messages sent over time for CodeDrip, Drip, DIP, and DHV protocols in the testbed experiments with 70 motes

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)	17980	21274	22130	21478
Code size (bytes)	42655	50123	49493	49003

7 Conclusions

In this work, we presented CodeDrip, a data dissemination protocol for Wireless Sensor Network. The main idea behind this protocol is to apply Network Coding in the dissemination process, decreasing the number of transmitted messages and consequently saving 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. 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.

For future work, we plan to analyze the impact of different topology types and link qualities on the performance of CodeDrip. Another interesting work is to develop new policies to combine messages using more complex operators.

References

1. Aguayo, D., Bicket, J., Biswas, S., Judd, G., Morris, R.: Link-level measurements from an 802.11b mesh network. In: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 121–132. ACM, New York (2004)
2. Ahlswede, R., Cai, N., Li, S.-Y.R., Yeung, R.W.: Network information flow. IEEE Transactions on Information Theory 46(4), 1204–1216 (2000)
3. Ahlswede, R., Cai, N., Li, S.Y.R., Yeung, R.W.: Network information flow. IEEE Transactions on Information Theory 46(4), 1204–1216 (2000)
4. Chandanala, R., Stoleru, R.: Network coding in duty-cycled sensor networks. In: 2010 Seventh International Conference on Networked Sensing Systems (INSS), pp. 203–210 (2010)
5. Dang, T., Bulusu, N., Feng, W.-c., Park, S.: DHV: A code consistency maintenance protocol for multi-hop wireless sensor networks. In: Roedig, U., Sreenan, C.J. (eds.) EWSN 2009. LNCS, vol. 5432, pp. 327–342. Springer, Heidelberg (2009)
6. Deb, S.: Network coding for wireless applications: a brief tutorial. In: International Workshop on Wireless Ad-hoc Networks (IWWAN) (May 2005)
7. Gnawali, O., Guibas, L., Levis, P.: A case for evaluating sensor network protocols concurrently. In: Proceedings of the Fifth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterizationn, WiNTECH 2010, pp. 47–54. ACM, New York (2010)
8. Hagedorn, A., Starobinski, D., Trachtenberg, A.: 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, IPSN 2008, pp. 457–466. IEEE Computer Society, Washington, DC (2008)
9. Hou, I.-H., Tsai, Y.-E., Abdelzaher, T.F., Gupta, I.: Adapcode: Adaptive network coding for code updates in wireless sensor networks. In: The 27th Conference on Computer Communications. INFOCOM 2008., pp. 1517–1525. IEEE (2008)
10. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd International, pp. 81–94. ACM Press (2004)
11. Katti, S., Rahul, H., Hu, W., Katabi, D., Medard, M., Crowcroft, J.: XORs in the Air: Practical Wireless Network Coding. IEEE/ACM Transactions on Networking 16(3), 497–510 (2008)
12. Keller, L., Karaagac, A., Fragouli, C., Argyraki, K.: Evaluation of network coding techniques for a sniper detection application. In: 2011 International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), pp. 327–333 (2011)
13. Lee, U., Park, J.-S., Yeh, J., Pau, G., Gerla, M.: Code torrent: content distribution using network coding in vanet. In: MobiShare 2006: Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking, pp. 1–5. ACM, New York (2006)
14. Levis, P., Gay, D., Culler, D.: Active sensor networks. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, NSDI 2005, vol. 2, pp. 343–356. USENIX Association, Berkeley (2005)
15. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos applications. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, pp. 126–137. ACM, New York (2003)

16. Levis, P., Patel, N., Culler, D., Shenker, S.: Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), pp. 15–28 (2004)
17. Li, S.-Y.R., Yeung, R.W., Cai, N.: Linear network coding. IEEE Transactions on Information Theory 49(2), 371–381 (2003)
18. Lin, K., Levis, P.: Data discovery and dissemination with dip. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN 2008, pp. 433–444. IEEE Computer Society, Washington, DC (2008)
19. Paek, J., Greenstein, B., Gnawali, O., Jang, K.-Y., Joki, A., Vieira, M., Hicks, J., Estrin, D., Govindan, R., Kohler, E.: The tenet architecture for tiered sensor networks. ACM Trans. Sen. Netw. 6(4), 34:1–34:44 (2010)
20. Sridharan, M., Zeng, W., Leal, W., Ju, X., Ramnath, R., Zhang, H., Arora, A.: From kansei to kanseiGenie: Architecture of federated, programmable wireless sensor fabrics. In: Magedanz, T., Gavras, A., Thanh, N.H., Chase, J.S. (eds.) TridentCom 2010. LNICST, vol. 46, pp. 155–165. Springer, Heidelberg (2011)
21. Tolle, G., Culler, D.: 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, pp. 121–132 (2005)
22. Vieira, L.F.M., Misra, A., Gerla, M.: Performance of network-coding in multi-rate wireless environments for multicast applications. In: IEEE Military Communications Conference, MILCOM 2007, pp. 1–6. IEEE (2007)
23. Wang, X., Wang, J., Xu, Y.: Data dissemination in wireless sensor networks with network coding. EURASIP Journal on Wireless Communications and Networking 2010(1), 465915 (2010)