Exploiting Concurrency for Efficient Dissemination in Wireless Sensor Networks

Yi Gao*, Jiajun Bu*, Wei Dong*, Chun Chen*, Lei Rao†, Xue Liu†
*Zhejiang Provincial Key Laboratory of Service Robot, College of Computer Science, Zhejiang University, China
†School of Computer Science, McGill University, Canada
{gaoyi, bjj, dongw, chenc}@zju.edu.cn, {leirao, xueliu}@cs.mcgill.ca

***Abstract*—Wireless Sensor Networks (WSNs) can be successfully applied in a wide range of applications. Efficient data dissemination is a fundamental service which enables many useful high-level functions such as parameter reconfiguration, network reprogramming, and etc. Many current data dissemination protocols employ network coding techniques to deal with packet losses. The coding overhead, however, becomes a bottleneck in terms of dissemination delay. We exploit the concurrency ability of sensor nodes and propose MT-Deluge, a multi-threaded design of a coding-based data dissemination protocol. By separating the coding and radio operations into two threads and carefully scheduling their executions, MT-Deluge shortens the dissemination delay effectively. An incremental decoding algorithm is employed to further improve MT-Deluge's performance. Experiments with 20 TelosB motes on three representative topologies show that MT-Deluge shortens the dissemination delay by 37.6%∼48.6% compared to a typical data dissemination protocol while keeping the merits of loss resilience.**

I. Introduction

Wireless Sensor Networks (WSNs) can be successfully applied in a wide range of applications such as military surveillance [1], environment protection [2], habitat monitoring, etc. Data dissemination is a basic building block of WSN applications. Network operators need to disseminate data to a network of sensors, enabling parameter reconfiguration, sensor network reprogramming, and etc.

There are many bulk data dissemination protocols [3]–[12] in the literature. Existing approaches can be divided into two categories: non-coding-based approaches and coding-based approaches. The performance of non-coding-based approaches [3], [4], [10], [11] degrades in lossy networks with unreliable wireless links. Coding-based approaches [5]–[7] employ network coding techniques [13] to provide better loss resilience. For example, Rateless Deluge [6] employs rateless code to encode native packets before transmission. Receivers can recover the native packets when a sufficient number of encoded packets have been received. Rateless Deluge exploits spatial diversity so that a single encoded packet can be recovered at multiple receivers, reducing the total number of transmissions. The main workflow of Rateless Deluge is described as follows [6].

At the sender side, a long page X is divided into k native packets $\{X_1, X_2 \dots X_k\}$. These k native packets are further encoded into $m > k$ encoded packets $Y_1, Y_2 \dots Y_m$ which are linear combinations of the native packets, i.e., $Y_i = \Sigma_{j=1}^{k} \beta_{i,j} X_j$, where $\beta_{i,j}$ are randomly chosen numbers.

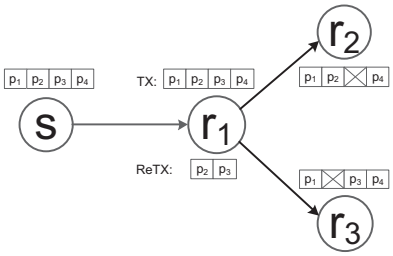


Fig. 1: A network consisting of four nodes.

At the receiver side, the encoded packets can be decoded by solving the set of linear equations by Gaussian elimination with back substitution after k linearly independent packets have been received. The benefit of network coding is shown in the following example. Suppose three receivers only get $k-1$ packets (the lost packets are distinct). In non-coding-based approaches, the sender needs to retransmit all three packets. In Rateless Deluge, the sender only needs one additional encoded packet as long as it is linearly independent with the previous $k-1$ packets. After decoding, the receivers obtain the native packets and become new senders for other nodes.

Network coding reduces the number of retransmitted packets, but it also introduces coding overheads. The time complexity of Gaussion elimination involved in the decoding procedure is $O(k^3)$. As reported in [6], on Tmote-sky motes, decoding a page consisting of 48 packets costs 6.96 seconds. Long decoding delay is a bottleneck in terms of the overall dissemination delay. Therefore, it is important to reduce the coding overhead.

One could argue that the receivers can simply forward the received encoded packets before decoding them. Then the decoding overhead does not contribute to the propagation delay. As shown in Figure 1, node s transmits four packets (i.e., p_1 to p_4) to nodes r_1, r_2 and r_3 using random linear codes. When node r_1 receives four encoded packets from node s, it immediately forwards these four encoded packets to nodes r_2 and r_3 without decoding them. However, this solution can hardly obtain the performance gains of network coding in practice. For example, when node r_2 and node r_3 lose one distinct packet (e.g., node r_2 loses p_3 while node r_3 loses p_2), node r_1 has several choices here.

1) Retransmit two packets, p_3 and p_2. In this case there are no benefits of network coding.
2) Decode the four packets before generating a newly encoded packet. In this case, the decoding delay still contributes to the dissemination delay.
3) Encode the encoded packets. In this case the coefficients of the encoded packet introduces additional transmission overhead since they cannot be efficiently generated using the same random number generator and seed across the network.

In general, simple patching to the original mechanism does not fundamentally address the irreconcilability of coding complexity and dissemination efficiency.

We propose MT-Deluge, a novel design in this paper to address the above problem. MT-Deluge exploits the concurrency ability of sensor nodes by separating the coding operation and radio operation into two threads. In Rateless Deluge [6], a node executes the following operations sequentially: receive, decode, encode, and send. With a multi-threaded design, we are able to amortize the overheads of encoding and decoding into the idle-wait durations during the radio operations, so as to reduce the dissemination delay of coding-based approaches.

Integrating multi-threading with current coding-based approaches has several practical challenges. First, when a sender receives a sufficient number of packets, how to recover the native packets and start dissemination as soon as possible? Second, when multi-threading is utilized, how to precisely synchronize multiple threads? To address the first challenge, an incremental decoding algorithm is proposed. Instead of starting the decoding procedure after receiving all packets, the receiver starts to decode after obtaining a small number of packets. The incremental decoding algorithm can make use of the idle-durations in receiving, mitigating the impact of decoding to dissemination delay. To address the second challenge, we introduce an adaptive packet-level synchronization to provide precise synchronization between multiple threads. When the receiver is receiving packets from the sender, the start time of the decoding procedure is adaptively chosen according to the packet receiving speed.

We implement MT-Deluge on TinyOS and evaluate it in a test-bed consisting of 20 Telosb [14] motes. Experiments on three different topologies show that MT-Deluge shortens the dissemination delay by 37.6%$\sim$48.6% compared to a typical data dissemination protocol while keeping the merits of loss resilience.

The rest of this paper is organized as follows. Section II introduces the related work and illustrates a motivating example. Section III gives an overview of MT-Deluge. Section IV describes the detailed design and implementation. Section V gives the real test-bed evaluation results and demonstrates that MT-Deluge shortens the dissemination delay by 37.6%$\sim$48.6%. Finally Section VI concludes the paper and presents the future work.

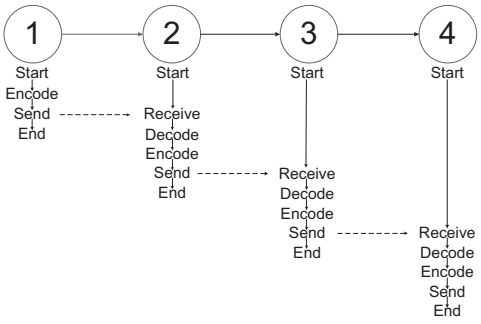


Fig. 2: Sequential workflow of Rateless Deluge.

II. Related Work

Data dissemination is a fundamental building block in WSNs. Many approaches have been proposed. Generally, existing approaches can be divided into two categories, i.e., non-coding-based and coding-based approaches. Deluge [3] is a standard non-coding-based dissemination protocol. In Deluge, a sender broadcasts a series of packets to many receivers and uses an NACK-based mechanism to provide reliability. MNP [4] is another non-coding-based approach which employs distributed sender selection to improve the dissemination efficiency. In addition, MNP employs sleep scheduling to reduce the energy cost. In order to speed up the dissemination procedure, Stream [11] reduces the code size by pre-installing the dissemination protocol on the external flash beforehand. Elon [15] introduces replaceable component to further reduce the transferred code size.

Because packets can be lost in wireless networks with unreliable links, retransmissions are necessary to provide high reliability. The performance of non-coding-based approaches degrades rapidly in lossy networks. In order to reduce the number of retransmitted packets, network coding techniques have been widely used in WSNs. Rateless Deluge [6] is a typical coding-based data dissemination protocol. In Rateless Deluge, random linear coding is used to reduce the retransmission overhead and two additional optimizations (i.e., precoding and ACKless transmission) further improve the performance. Adapcode [8] also employs random linear codes. Unlike Rateless Deluge, the receivers in Adapcode collect encoded packets from multiple senders instead of a single sender. Synapse [5] is another coding-based approach which uses Fountain Codes. All these approaches need to perform Gaussian elimination in the decoding phase which is time consuming on current resource-constrained sensor nodes. Our previous work ReXOR [7] uses XOR-based coding to improve the decoding efficiency.

Network coding is widely used in WSNs to improve the data dissemination performance, however encoding and decoding introduce computation overhead. Without loss of generality, we take Rateless Deluge as an example to describe how the computation overhead affects the dissemination delay. We set up a small network consisting of four sensor nodes

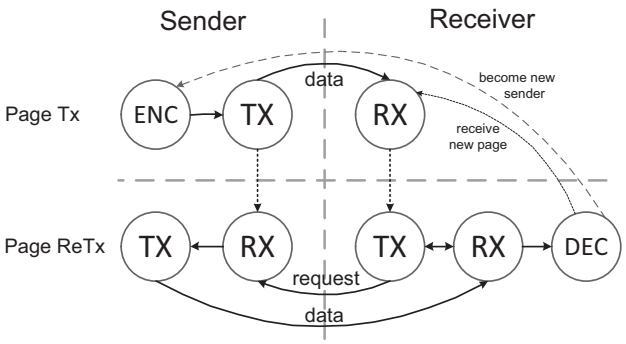


Fig. 3: State transition diagram of a coding-based protocol.

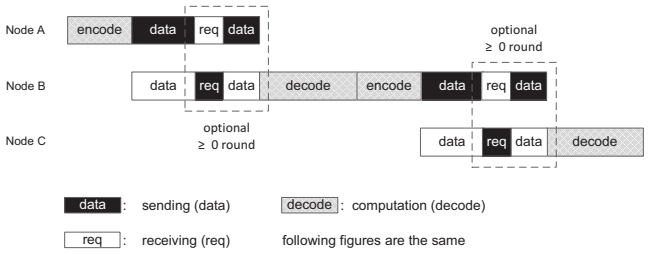


Fig. 4: Workflow of single-threaded design.

and these nodes sequentially obtain a series of packets using random linear codes. Figure 2 shows the behaviors of the sensor nodes during the experiment. We can see that every encode/decode operation prolongs the dissemination delay. We also measure the average CPU utilization of these four sensor nodes and find that it is below 14% when nodes are performing radio operations. If we could make good use of the computation ability during the radio operations for coding purpose, we can mitigate the impact of coding overhead to the dissemination delay. However, the current event driven model cannot provide enough concurrency. Therefore, we propose a multi-threading-based dissemination protocol in this paper, which is able to amortize the computation overhead into the packet sending/receiving procedure and hence shortens the dissemination delay significantly.

III. Overview

MT-Deluge exploits the concurrency ability of the sensor nodes and speeds up the dissemination of coding-based protocols. In this section, we give an overview of MT-Deluge. In MT-Deluge, when a code image needs to be disseminated, the image is divided into many pages. Each page is further split into a series of packets.

A. Background of Single-threaded Design

Before describing MT-Deluge, we introduce the workflow of a coding-based protocol of a single thread. Figure 3 depicts the main state diagram transition of a coding-based protocol. Every sensor node advertises about its local images periodically. When a node learns that one of its neighbor nodes has more available pages than itself, the node will start sending request to its neighbor. This advertise-request mechanism allows every node in the sensor network to know the current information about its neighbors. As shown in Figure 3, the workflow of a typical coding-based data dissemination protocol is divided into two phases: page transmission and page retransmission.

Page transmission: Figure 3 depicts three different states of the sender and receiver. There is one more state, $MAINTAIN$ which is not shown. The sender and receiver finish the advertise-request operations in $MAINTAIN$ state and we do not include this $MAINTAIN$ state to avoid complexity. After the advertise-request procedure, the sender enters the first state, Encoding (ENC). The k native packets belonging to one page are encoded using different algorithms in different protocols. In Rateless Deluge, the k native packets in the same page are encoded by random linear codes. Then the sender enters into Transmission (TX) state and transmits the encoded packets to the receiver. At the same time, the receiver is in Receiving (RX) state and receives the encoded packets.

Page retransmission: Due to unreliability of wireless links, the receiver may lose several encoded packets. Therefore, the page retransmission phase is needed for error recovery. Unlike the first phase, this phase is initiated by the receiver. The receiver enters into the TX state and transmits a request including the information of the lost packets. After the sender receives the request in the RX state, the sender enters into the TX state again and transmits the requested packets to the receiver. At the same time, the receiver switches to the RX state to receive the requested packets. If the receiver has obtained enough packets for decoding, it enters into the Decoding (DEC) state and starts to decode. Otherwise the receiver enters into the TX state and starts to send the request again.

These two phases constitute the workflow of one page transmission. A large code image usually has many pages. To disseminate an entire image, all the pages need to be disseminated. After the decoding procedure is finished, there are two possibilities. (1) the sender starts to transmit the next page (2) the receiver becomes a new sender and start to transmit pages to other sensor nodes. In this paper, we focus on one page transmission for simplicity of descriptions. However, the techniques can be simply extended to the case of multiple page transmissions.

B. Multi-threaded Design Overview

We propose MT-Deluge to improve the efficiency of coding-based dissemination protocols. The intuition of multi-threaded design is to allow the coding operations and the radio operations to work concurrently. Note that in the multi-threaded design, a sensor node can decode and send packets simultaneously, so there is no separate TX state or DEC state. Therefore, we use the time line figure to describe the workflow. Figure 4 and Figure 5 use time lines to describe the one page transmission of both single-threaded protocol and multi-threaded protocol. In Figure 4 and Figure 5, the black box

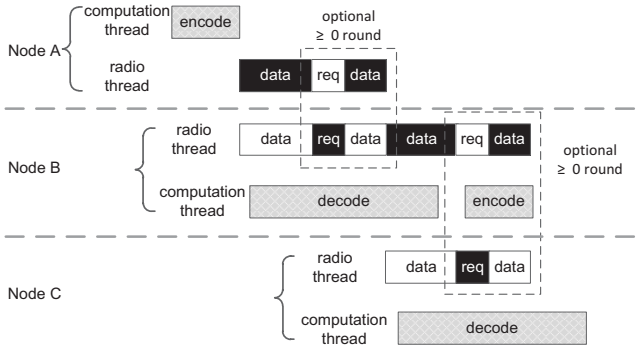


Fig. 5: Workflow of multi-threaded design.

represents the sending operation, the white box represents the receiving operation and the grey box represents coding operations (i.e., encoding and decoding). There are three nodes. Node A sends one page to node B and node B sends the same page to node C. In the single-threaded design as shown in Figure 4, node B only starts to send the page to node C after decoding and encoding. In the multi-threaded design as shown in Figure 5, a radio thread and a computation thread are executed concurrently. Node B can forward the page to node C once it has collected a small number of encoded packets sent by node A. The computation thread can start to decode even earlier than the sending thread. Once node B has obtained several packets, the computation thread can start to decode. Once node B has obtained enough encoded packets, the decoding procedure ends after a short period of time. Therefore, the page dissemination time can be shortened.

It is worth noting that Figure 5 only illustrates an ideal situation. In MT-Deluge, we need to consider several practical design issues. For example, when node C loses some packets and sends a request to node B, node B may have not finished the decoding and encoding procedures. Then node C must wait for node B. We will elaborate on this issue in more details in Section IV.

IV. Multi-threaded Design and Implementation

The implementation of multi-threaded design has two main challenges. One is to shorten the total dissemination delay by minimizing the waiting time throughout the dissemination procedure. The other is to synchronize different threads running on each sensor node. To address the first challenge, we propose an incremental decoding algorithm to start the decoding procedure earlier. To synchronize multiple threads precisely, a packet-level synchronization mechanism is employed. In addition, an adaptive page size adjustment scheme is utilized to optimize the page size given the link quality. We implement MT-Deluge based on Rateless Deluge [6], a typical coding-based data dissemination protocol which has both encoding and decoding procedures. MT-Deluge changes the page transfer mechanism from single-threaded design to multi-threaded design. In this section, we describe the solutions and the implementation details to address the challenges.

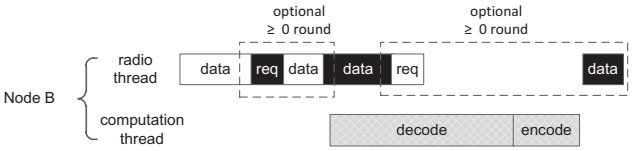


Fig. 6: Workflow of node B without incremental decoding.

A. Incremental Decoding Algorithm

In Rateless Deluge, a page is decoded after the receiver has received all encoded packets. After decoding, the native packets are obtained. Then the receiver encodes the native packets and sends them to other nodes. In MT-Deluge, the multi-threaded design allows the computation thread and radio thread to run concurrently. Therefore, an obvious benefit is that the receiver can just forward the received encoded packet to other nodes and decode the encoded packets at the same time. Figure 6 shows node B's workflow using this design.

This workflow is straightforward when multi-threaded design is employed. However, it causes the radio thread to wait a relatively long time period which is shown as a long blank area in Figure 6. We can see that there are two *black boxes* which mean two different procedures of "sending data". The first "sending data" procedure is to forward the encoded data packets to other nodes. These encoded packets are encoded by the sender (node A), thus node B can forward them without waiting for the decoding and encoding procedures. Unlike the first one, the second *black box* procedure is a data retransmission procedure. As mentioned in the Section I, a node cannot forward the encoded packets from the sender in data retransmission procedure because the encoded packets have already been received by the receivers with high probability. Thus the node has to encode native packets to retransmit. Therefore, as shown in Figure 6, the second "sending data" has to wait for the decoding and encoding procedures.

Comparing Figure 6 with Figure 5, it is easy to notice that in Figure 5, the decoding procedure starts earlier and the total time cost is lower. The reason is that an incremental decoding algorithm is employed in MT-Deluge. The basic idea is to start the decoding procedure after a small number of encoded packets have been obtained instead of starting it after *all* encoded packets have been obtained. Next, we describe the decoding procedure in Rateless Deluge generally and give the detailed description of the incremental decoding algorithm in MT-Deluge.

In Rateless Deluge, the decoding procedure consists of three parts: generating the coefficients matrix, getting the echelon matrix by Gaussian elimination, and performing back substitution. The receiver needs to recover the coefficients matrix that the sender uses by the initial seeds attached in the encoded data packets. Then Gaussian elimination is used to get the echelon matrix, followed by the back substitution which recovers the native packets. In MT-Deluge, the whole incremental decoding is divided into four procedures. Algorithm 1 presents the the four procedures in detail: INCREMENTALDECODE() is

Algorithm 1 The Incremental Decoding Algorithm

1: **procedure** INCREMENTALDECODE(A, I, K)
▷ A: the extension matrix for decoding
▷ I: number of initial packets for decoding
▷ K: number of packets in one data page
2: wait for I initial packets being received
3: generate I rows of coefficients from the I initial packets
4: **GE**(A,I,K)
5: **while** $true$ **do**
6: wait for m new packets being received
7: **if** $m \neq 0$ **then**
8: generate m new rows of coefficients
9: **SID**(A,I,K,m)
10: **if** K packets have been received **then**
11: **BS**(A,K)
12: **break**
13:
14: **procedure** GE(A, I, K) ▷ Standard Gaussian Elimination
15: $i \leftarrow 1$
16: $j \leftarrow 1$
17: **while** $i \leq I$ and $j \leq K$ **do**
18: find row max which has maximum element in column j
19: **if** $A[max, j] \neq 0$ **then**
20: swap $row\ i$ and $row\ max$
21: divide each entry in $row\ i$ by $A[i, j]$
▷ Now $A[i, j]$ equals to 1
22: **for** $u \leftarrow i + 1\ to\ I$ **do**
23: subtract $A[u, j] \times row\ i$ from $row\ u$
▷ Now $A[u, j]$ equals to 0
24: $i \leftarrow i + 1$
25: $j \leftarrow j + 1$
26:
27: **procedure** SID(A, I, K, m) ▷ One Step of Incremental Decode
▷ m: the number of newly received packets
28: $i \leftarrow 1$
29: $j \leftarrow 1$
30: **while** $i \leq I$ and $j \leq K$ **do**
31: subtract $A[i + I, j] \times row\ i$ from $row\ i + I$
▷ Now $A[i + I, j]$ equals to 0
32: $i \leftarrow i + 1$
33: $j \leftarrow j + 1$
34: **while** $i \leq I + m$ and $j \leq K$ **do**
35: divide each entry in $row\ i$ by $A[i, j]$
▷ Now $A[i, j]$ equals to 1
36: **for** $u \leftarrow i + 1\ to\ I + m$ **do**
37: subtract $A[u, j] \times row\ i$ from $row\ u$
▷ Now $A[u, j]$ equals to 0
38: $i \leftarrow i + 1$
39: $j \leftarrow j + 1$
40:
41: **procedure** BS(A, K) ▷ Back Substitution
42: **for** $i \leftarrow 2\ to\ K$ **do**
43: **for** $u \leftarrow 1\ to\ i - 1$ **do**
44: subtract $A[u, j] \times row\ i$ from $row\ u$

the outer most procedure which calls other procedures; the procedure GE() performs standard Gaussian elimination; the procedure SID() performs one step of incremental decoding; the procedure BS() performs back substitution. The algorithm starts when I initial encoded packets have been obtained at the receiver side. Then the seeds attached in these packets are used to generate the coefficients of the corresponding

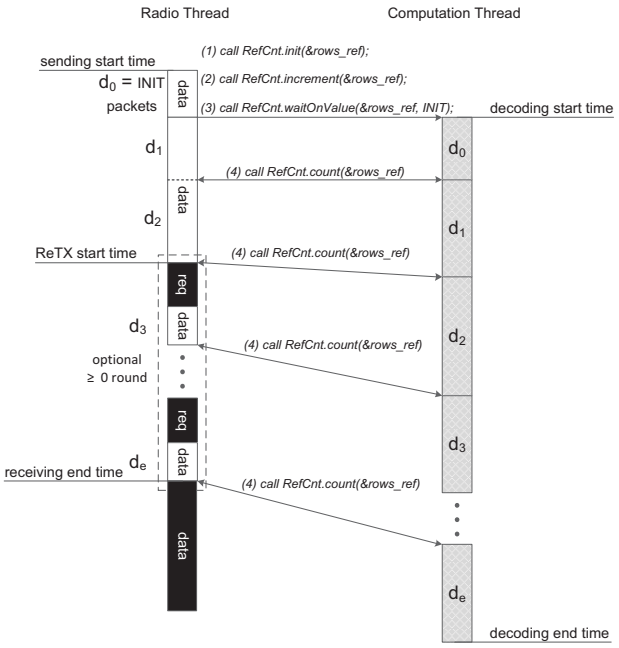


Fig. 7: Packet-level synchronization details.

rows (line 3, the coefficients of each encoded packet forms one row). Standard Gaussian elimination further calculates the echelon matrix of these initial rows (line 14-25). When the computation thread is running, the radio thread can receive new encoded packets at the same time. The corresponding rows of coefficients are generated by the seeds attached in these data packets (line 8). Then the newly received packets are inserted into the matrix and the updated matrix is calculated to be an echelon matrix (line 27-39). When enough encoded packets have been received, back substitution is used to obtain the native packets of the page (line 41-44). The complexity of incremental decoding algorithm is $O(N^3)$ given an $N \times N$ matrix, which is as same as that of original Gaussian elimination.

B. Multi-thread Synchronization

We use an adaptive packet-level synchronization mechanism to provide precise synchronization between multiple threads. The algorithm has three key features. First, packet-level synchronization between the radio thread and the computation thread provides high precision. Second, the actual workflow of each sensor node is adaptively chosen at the run-time. Third, different page sizes are chosen according to different link qualities in order to shorten the total delay.

Compared with packet-level synchronization, state-level synchronization is another choice in the implementation. In fact, we use state-level synchronization to synchronize radio thread and computation thread outside the incremental decoding phase. If incremental decoding is not performed, state-level synchronization will be sufficient. For example, as shown in Figure 3, when the sender enters state TX from state ENC, it can use synchronization primitives (e.g., a

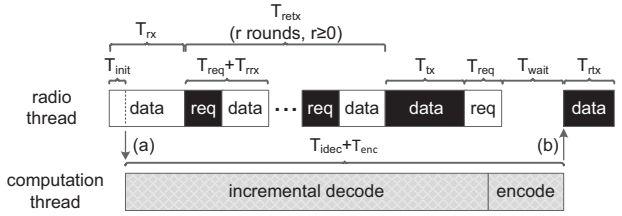


Fig. 8: One page transmission time line.

semaphore). However, when incremental decoding is used, state-level synchronization is not sufficient because it does not have any separate states anymore. The workflow and some related code of incremental decoding using packet-level synchronization are depicted in Figure 7. As same as Figure 4, the black box represents the sending operation, the white box represents the receiving operation and the grey box represents coding operations (i.e., encoding and decoding). In the current implementation, we use a synchronization primitive in TOSThreads [16] (an multi-thread extension of TinyOS) named $ReferenceCounter$ ($RefCnt$ in short) to record the received number of packets. There are four operations regarding $RefCnt$. Operation (1) initializes the value of $RefCnt$ to zero. Operation (2) increases the value of $RefCnt$ when a new encoded packet is received. Operation (3) waits till $INIT$ packets have been received. Operation (4) is used to get the current number of received packets. d_1 to d_e represent all the encoded packets received and each of them represents a segment of the page.

The $INIT$ value indicates the number of received packets before the decoding procedure starts ($INIT \in [1, k]$, k is the page size). Smaller $INIT$ can make the start time of decoding earlier while larger $INIT$ has less computation overhead. Therefore, there is a tradeoff between the start time and the overhead. In our implementation, we find out that setting $INIT$ to $k/3$ can get good performance.

d_0 (equals to $INIT$) is set before the program runs, but d_2 to d_e are chosen at run-time. Operation (4) is used to get the current number of received packets and these newly received packets are incrementally decoded subsequently. This adaptive mechanism to determine the number of the next packet segment can avoid wait time of both the radio and computation threads. In addition, link quality can affect the packet receiving progress. Retransmission is needed when link quality is poor, which makes the packet receiving progress much slower. Link quality can only be obtained after the program runs. Therefore, the adaptive mechanism can also work well under different link qualities.

C. Page Size Adjustment

The last feature is that we set different k (i.e., page size) under different link qualities. Large k means long decoding time but small number of pages. In original Deluge, k is 48 and in Rateless Deluge k is 20. The reason is that the decoding time of a page consisting of 48 packets is nearly 7 seconds which significantly impacts the dissemination delay. However, large page size (e.g., 48 in original Deluge) can be beneficial without considering the decoding cost [6]. By multi-threading, the decoding cost is amortized into the packet sending and receiving times. Thus in MT-Deluge, a large page size k is beneficial. Figure 8 shows the precise one page transmission time line on one sensor node. Note that T_{tx} equals to T_{rx}. T_{retx} consists of several T_{req} (i.e., requesting time) and T_{rrx} (i.e., re-receiving time). T_{idec} is the incremental decoding time. In Figure 8, there are two arrows which represent two constraints. Constraint (a) means that the decoding procedure starts only after $INIT$ packets have been received. Constraint (b) means that the data retransmission starts only after the decoding and encoding procedures completes. The reason is that the encoded packets have already been received by the receivers with high probability and only newly encoded packets can be used in retransmission. Given a page of k packets and r rounds of retransmission, we have:

$$T_{init} = \alpha k/3, \tag{1}$$

$$T_{rx} = T_{tx} = \alpha k, \tag{2}$$

$$T_{retx} = (T_{req} + T_{rrx})r = (\beta_1 + \beta_2)r, \tag{3}$$

$$T_{idec} = \gamma k^3, \tag{4}$$

$$T_{enc} = \delta k, \tag{5}$$

where $\alpha, \beta_1, \beta_2, \gamma, \delta$ are fixed parameters (e.g., $\delta = 0.03897$ on Tmote sky motes [6]). We set $INIT$ to $k/3$, thus we have Eq. (1). T_{req} and T_{rrx} are related to the number of lost packets. However, in most conditions, the number of lost packets are relatively small (e.g., 5 packets may be lost given a page of 24 packets when the loss rate is 20%). When the number of lost packets is small, the delay mainly depends on the $rounds$ of retransmissions which in shown in Eq. (3). T_{idec} is proportional to k^3 because it includes the time for Gaussian elimination.

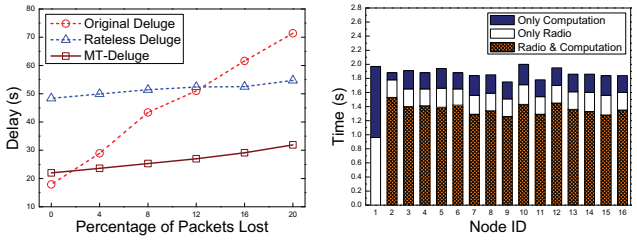
Because of the constraint(b) (i.e., arrow (b) in Figure 8), we have:

$$T_{rx} - T_{init} + T_{retx} + T_{tx} + T_{req} + T_{wait} \geq T_{idec} + T_{enc}. \tag{6}$$

We found that the total delay reaches its minimum value when T_{wait} equals to 0 and the two sides of equation (6) are equal. As a result of the simple calculation of equations (1) to (6) we obtain:

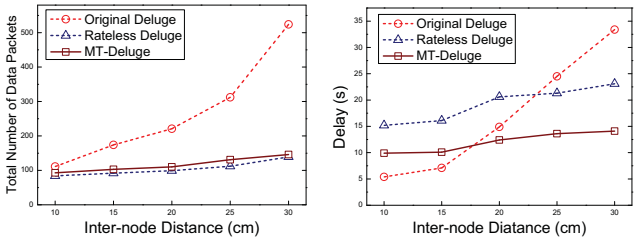
$$(2\beta_1 + \beta_2)r = \gamma k^3 + (\delta - 5\alpha/3)k. \tag{7}$$

Note that r and k are the only two variables in equation (7). We cannot choose r because it is related to the link quality, thus we should adjust k according to r. For example, r is likely to be larger in a lossy environment. That means we should set a *larger* page size k in a *lossy* environment to keep equation (7) hold. A real test-bed evaluation of this page size adjustment scheme is given in Section V.



(a) delay (b) state information

Fig. 9: Delay and state information of the line topology consisting of 16 TelosB nodes.



(a) data traffic (b) delay

Fig. 10: Data traffic and delay of the grid topology consisting of 20 TelosB nodes.

V. Experimental Study

The experimental results were obtained in a test-bed consisting of 20 TelosB [14] motes. We implemented the multi-threaded design and evaluated the one page transmission efficiency. According to the datasheet of TelosB, radio operations (e.g., send, receive, listen) cost most of the energy. Current reprogramming protocols, like Rateless Deluge and Synapse, keep the radio always on. Some other dissemination protocols [4], [17] allow the sensor nodes to sleep for saving energy. The sleep scheduling, however, has extra overhead (e.g., additional transmissions or computations). In MT-Deluge, we do not incorporate sleep scheduling and consider it as a direction of future work. Thus the dissemination delay is a direct indicator of the energy consumption and is a major evaluation metric. A series of experiments are conducted on three topologies to evaluate the performance of MT-Deluge. Two different packet losses are generated in the experiments: i) forced packet loss, the nodes transmit packets at high power level over short distances, which ensure the link qualities are good, then the nodes drop packets uniformly at a certain rate; and ii) natural packet loss, the nodes transmit packets at low power level over relatively long distances, which induces packet loss.

A. Multi-hop Line Topology

We put 16 TelosB motes in a straight line and configure the network so that each node only communicates with its one or two adjacent neighbors. One page consisting of 24 packets is disseminated from one side of the line to another using different page transfer techniques. Experiments under this topology are conducted by forced packet losses. The data traffics of MT-Deluge and Rateless Deluge are similar and is not shown here. Figure 9(a) shows the total delay using three different page transfer techniques. When the packet loss rate is zero, no retransmission is needed and network coding is not needed. Therefore, Deluge achieves the best performance. When the packet loss rate increases, network coding used in Rateless Deluge and MT-Deluge significantly improve the network performance. In our Multi-threaded design, nodes forward the encoded packets and decode them concurrently. Therefore, the dissemination delay is shortened. On average, MT-Deluge consumes 48.6% less time than Rateless Deluge. We also make the nodes perform local logging about the state information. Figure 9(b) gives detailed state information of all sensor nodes in the experiment. Node 1 is the initial sender, so it needs to encode the whole page and send the encoded packets to node 2. All other nodes receive the encoded packets and forward them to the next node. The result shows that more than 70% of the time, the radio operations and the computations run concurrently. Note that in Rateless Deluge, these operations are sequentially executed. This is the main reason that MT-Deluge can shorten the dissemination delay significantly.

B. Multi-hop Grid Topology

We put 20 TelosB motes as a 4×5 grid and evaluate the performance of one page transmission. One corner node is chosen as the initial sender to disseminate one page consisting of 24 packets. Two metrics are used: total number of data packets and dissemination delay. Three sniffer nodes are placed in the grid to passively listen packets in the network. The results are obtained by calculating the average value of the data from the three sniffer nodes. In grid experiments, we do not set packet loss rate but use natural packet loss. The transmission power of TelosB mote can be configured at different levels (i.e., 2 to 31 [14]). We set the transmission power level be 2 and the inter-node distance between $10cm$ and $30cm$. In this setting, there are both good and poor links, more similar to the real condition. Figure 10(a) and Figure 10(b) show the total number of data packets and dissemination delay of three techniques, respectively. The number of data packets of MT-Deluge is similar to Rateless Deluge in grid topology. Figure 10(b) shows that MT-Deluge can shorten the dissemination delay significantly in grid topology. Compared to the line topology, the relative delay reduction of MT-Deluge is smaller (i.e., 48.6% in line topology and 37.6% in grid topology). The reason is that grid topology has less hops than line topology.

In Section IV.C, we analyzed the relationship between page size and link qualities. We got a conclusion that it is better to choose a *larger* page size in a *lossy* environment. A series of experiments on grid topology are conducted to evaluate the performance. Again, we set different inter-node distances (i.e., dense and sparse) to get different link qualities (i.e., large

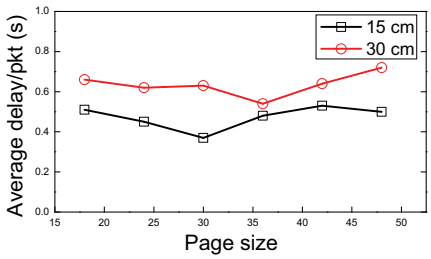
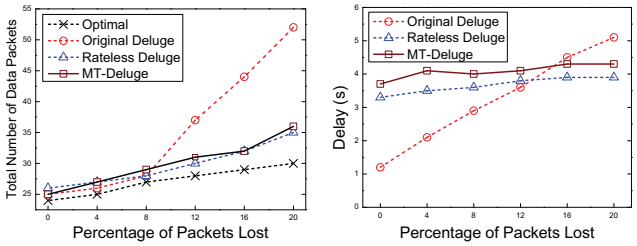


Fig. 11: Average delay per packet when the inter-node distance is set to be 15cm or 30cm.



(a) data traffic (b) delay

Fig. 12: Data traffic and delay of the clique topology consisting of 8 TelosB nodes.

inter-node distance means poor link qualities). The page size is set between 18 and 48. Because we only transmit one page, the delay is directly related to the page size. For comparison, we use average delay per packet to evaluate the efficiency of different page sizes. The result is shown in Figure 11. The optimal page size when the inter-node distance is $15cm$ is smaller. In our experiments, setting the page size near 30 can get the shortest delay in dense network. And setting the page size near 36 is optimal in sparse network.

C. Single-hop Clique Topology

Due to the limited transmission range of wireless sensor nodes, multi-hop topologies are widely used in real WSNs. We also investigate a single-hop clique topology. One sender sends a page consisting of 24 packets to other 7 one-hop nodes using Deluge, Rateless Deluge, MT-Deluge, respectively. In order to evaluate the effectiveness of network coding, the receivers are forced to randomly drop some packets. Figure 12(a) shows that MT-Deluge transmits as similar number of packets as Rateless Deluge. However, MT-Deluge has a longer delay than Rateless Deluge in the single-hop network as shown in Figure 12(b). In our experiment, MT-Deluge costs approximately 17% more time than Rateless Deluge in a single-hop network when the page size is 24. In our multi-threaded design, nodes forward the encoded packets and decode them concurrently to shorten the dissemination delay. In single-hop network, however, nodes need not to forward packets at all. Therefore, according to the analysis and our experiment results, single-threaded protocols [5], [6] are sufficient in a single-hop network.

VI. Conclusion and Future Work

In this paper, we provide a multi-threaded design for data dissemination in WSNs. Current coding-based data dissemination protocols introduce encoding or decoding overhead and this problem limits the scalability of these protocols. We propose MT-Deluge, a multi-threaded design which allows the radio operations and the computations run concurrently. We implement it based on TinyOS, and the experiments on three topologies show that MT-Deluge is able to keep the merits of network coding techniques and has much shorter dissemination delay. For future work, we will extend MT-Deluge to other protocols and leverage power management to further improve the network performance for WSNs.

Acknowledgment

This work is supported by the the National Science Foundation of China (Grant No. 61070155), Program for New Century Excellent Talents in University (NCET-09-0685).

References

[1] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. of ACM WSNA*, 2002.

[2] L. Mo, Y. He, Y. Liu, J. Zhao, S. Tang, X. Li, and G. Dai, "Canopy Closure Estimates with GreenOrbs: Sustainable Sensing in the Forest," in *Proc. of ACM SenSys*, 2009.

[3] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. of ACM SenSys*, 2004.

[4] S. S. Kulkarni and L. Wang, "MNP: Multihop Network Reprogramming Service for Sensor Networks," in *Proc. of IEEE ICDCS*, 2005.

[5] M. Rossi, G. Zanca, L. Stabellini, R. Crepaldi, A. F. H. III, and M. Zorzi, "SYNAPSE: A Network Reprogramming Protocol for Wireless Sensor Networks using Fountain Codes," in *Proc. of IEEE SECON*, 2008.

[6] A. Hagedorn, D. Starobinski, and A. Trachtenberg, "Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks using Random Linear Codes," in *Proc. of ACM/IEEE IPSN*, 2008.

[7] W. Dong, C. Chen, X. Liu, J. Bu, and Y. Gao, "A Light-Weight and Density-Aware Reprogramming Protocol for Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 99, no. PrePrints, 2010.

[8] I.-H. Hou, Y.-E. Tsai, T. F. Abdelzaher, and I. Gupta, "AdapCode: Adaptive Network Coding for Code Updates in Wireless Sensor Networks," in *Proc. of IEEE INFOCOM*, 2008.

[9] W. Li, Y. Zhang, and B. Childers, "MCP: An Energy-Efficient Code Distribution Protocol for Multi-Application WSNs," in *Proc. of IEEE DCOSS*, 2009.

[10] M. D. Krasniewski, R. K. Panta, S. Bagchi, C.-L. Yang, and W. J. Chappell, "Energy-efficient on-demand reprogramming of large-scale sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, 2008.

[11] R. K. Panta, I. Khalil, and S. Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," in *Proc. of IEEE INFOCOM*, 2007.

[12] K. Lin and P. Levis, "Data Discovery and Dissemination with DIP," in *Proc. of ACM/IEEE IPSN*, 2008.

[13] S. Katti, D. Katabi, H. Balakrishman, and M. Medard, "Symbol-level Network Coding for Wireless Mesh Networks," in *Proc. of ACM SIGCOMM*, 2008.

[14] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," in *Proc. of ACM/IEEE IPSN/SPOTS*, 2005.

[15] W. Dong, Y. Liu, X. Wu, L. Gu, and C. Chen, "Elon: enabling efficient and long-term reprogramming for wireless sensor networks," in *Proc. of ACM SIGMETRICS*, 2010.

[16] K. Klues, C.-J. M. Liang, J.-y. Paek, Musaloiu-E, P. Levis, A. Terzis, and R. Govindan, "TOSThreads: Thread-Safe and Non-Invasive Preemption in TinyOS," in *Proc. of ACM SenSys*, 2009.

[17] L. Huang and S. Setia, "CORD: Energy-efficient Reliable Bulk Data Dissemination in Sensor Networks," in *Proc. of IEEE INFOCOM*, 2008.