PROACTIVE RELIABLE BULK DATA DISSEMINATION IN SENSOR NETWORKS[1]

Limin Wang Sandeep S. Kulkarni
Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing MI 48824 USA

Abstract

One of the problems in network reprogramming is to guarantee 100 percent delivery of a large amount of data (the entire binary image) in a lossy wireless channel. All the existing protocols on network reprogramming [1–6] use automatic repeat request (ARQ) scheme to recover from packet losses. We propose a new reliability scheme which is a hybrid approach of forward error correction (FEC) and ARQ. We perform a case study on MNP, a multihop network reprogramming protocol, and study the effect of adding two different FEC codes: simple XOR code and Reed-Solomon (RS) codes [7], to MNP. We evaluate the new reliability approach using TOSSIM. The simulation results show that adding simple XOR code to MNP can achieve up to 10% improvement on reprogramming speed and up to 18% reduction on active radio time (the major part of energy consumption). And we show that RS codes perform even better.

Keywords: Sensor networks, Forward error correction, Data dissemination

1. Introduction

Sensor networks have been proposed for a wide variety of applications. A sensor network needs to operate unattended for long periods of time. This requirement introduces several difficulties. First, the environment evolves over time. Predicting the whole set of actions that a sensor node might need to perform is impossible in most applications. Second, requirements are also likely to change. For example, with growing understanding of the environment or with new technological advances, some assumptions are found to be incorrect, and, hence, the specification needs to be modified accordingly. Thus, reprogramming sensor nodes, i.e., changing the software running on sensor nodes after deployment, is necessary for sensor networks.

[1]Email: {wanglim1,sandeep}@cse.msu.edu.
Web: http://www.cse.msu.edu/~{wanglim1,sandeep}.
Tel: +1-517-355-2387, Fax: +1-517-432-1061.

This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF equipment grant EIA-0130724, and a grant from Michigan State University.

Network reprogramming requires 100 percent delivery of the entire binary image (on the order of kilobytes), and hence consumes significant communication bandwidth. However, the transmissions are performed over the radio, which is known as a low-bandwidth and lossy medium. Therefore the reliability issues need to be addressed.

There are two basic methods to recover lost packets. One way is to use automatic repeat request (ARQ). In ARQ schemes, a receiver detects its own losses, and informs the sender of the missing packets, either by sending requests (NACK) or acknowledgements (pure ACK, implicit ACK, selective ACK, etc.). The sender retransmits the repair packet if it knows that a packet is lost. Another way to recover errors is to use forward error correction (FEC). FEC provides reliability by transmitting redundant packets in a proactive manner. The most commonly used FEC scheme is (n, k) FEC. The fundamental of (n, k) FEC is to add $n - k$ additional packets to a group of k source packets (called a *transmission group*) so that the receipt of any k packets at the receiver permits recovery of the original k ones. There are different levels of FEC schemes: packet-level, byte-level, bit-level. In the context of reprogramming, we only examine the packet-level FEC.

The existing protocols on network reprogramming include the single-hop reprogramming protocol XNP [1], and multihop reprogramming protocols MOAP [2], Deluge [3], MNP [4], Infuse [5], and Sprinkler [6]. All these protocols use automatic repeat request (ARQ) scheme to recover from packet losses. ARQ is an effective reliability scheme, as an error can always be recovered as long as the network is connected. However, if the error rate is high, the requests and retransmissions for the missing packets consume significant energy. In this paper, we examine the issue of adding FEC to the ARQ-based reliability scheme of a reprogramming protocol. We perform a case study on our previous work, MNP [4], a multihop network reprogramming service for sensor networks. The proposed reliability scheme is a hybrid approach of FEC and ARQ. By adding FEC, we expect to reduce the error probability experienced at the receivers, and ARQ scheme performs the remaining error corrections.

Organization of the paper. In Section 2, we give a brief overview of the MNP protocol, with emphasis on the reliability issues. In Section 3, we propose a hybrid reliabil-

ity scheme using FEC and ARQ, and describe the implementation details of adding simple XOR code and Reed-Solomon (RS) codes [7] to MNP. In Section 4, we present the simulation results on the performance of (MNP+XOR) and (MNP+RS codes). We review related work in Section 5 and conclude in Section 6.

2. A Brief Overview of MNP

In [4], we presented MNP, a multihop network reprogramming protocol, which provides a reliable and energy efficient service to propagate new program images to all the sensor nodes in the network over the radio. MNP applies an advertise-request-data handshake interface. To achieve pipelining, a program is divided into *segments*, each of which contains a fixed number of *packets*. A source node advertises the availability of the segments of a new program. Once a node receives an advertisement, if it needs the segment, it sends a request. To reduce the message collision problem, MNP uses a sender selection algorithm, in which source nodes compete with each other based on the number of distinct requests they receive. The node that has received the most number of requests is selected as the sender, and all of its neighbors either receive the segment from the sender, or go to "sleep" state to save energy. Since the focus of this paper is the reliability issues, we refer the readers to [4] for the details of the sender selection algorithm and pipelining scheme.

In MNP, each packet has a unique ID, from 1 to the size of the segment, *SegSize*. Each receiver is responsible for detecting its own loss. Since the size of the segment is small and pre-determined, a node maintains a bitmap (which we call *MissingVector*) of the current segment it is receiving in memory. Each bit in *MissingVector* corresponds to a packet. All the bits are initially set to 1. When a node receives a packet for the first time, it stores that packet in EEPROM and sets the corresponding bit in *MissingVector* to 0.

A node that is advertising maintains a *ForwardVector*, which is a bitmap of the advertised segment, and is an indicator of the packets the node needs to send if it becomes a sender. When a node sends a *request* message, it puts the loss information (its *MissingVector*) in the *request* message. When the advertising node receives the request, it marks its *ForwardVector* according to the loss information. Therefore, the *ForwardVector* of an advertising node is the union of the *MissingVectors* in the *request* messages that the node has received. A node only sends the packets indicated in the *ForwardVector*. We restrict the length of the segment to be no longer than 128 packets, so that the maximal size of *MissingVector* is only 16 bytes, and thus fits into a radio packet.

We implemented MNP on TinyOS Mica-2 [8] and XSM [9] mote platforms. In the remaining part of this paper, we use TOSSIM [10], a discrete event simulator for TinyOS wireless networks, to investigate the effect of adding FEC to the current ARQ scheme. Before adding FEC, we ran a simulation to see the packet loss pattern, which is shown in Figure 1. The simulation was conducted in a 10x10 network with 10 feet inter-node distance. The program size is 8.4KB (3 segments, 384 packets). The x-axis is the number of missing packets indicated by the *MissingVector* contained in a request message. The y-axis is the number of request messages. Since the segment size is 128 packets, the number of missing packets ranges from 1 to 128 packets. The peak at the right is the number of requests that ask for the whole segment (128 packets). These are *protocol requests*, used in the sender selection algorithm. We only consider *repair requests*, the requests that ask for *less than* 128 packets. We note that most of the repair requests only ask for a few number of packets. For example, about 50% of the requests are asking for less than 8 missing packets, 70% of the requests are asking for less than 16 missing packets, 83% of the requests are asking for less than 32 missing packets. The fact that the majority of the losses are small losses (involving a few number of missing packets) suggests that a better link is desirable to reduce the number of requests and retransmissions. FEC can be used to provide an abstraction of an enhanced link at the cost of transmitting additional parity packets.

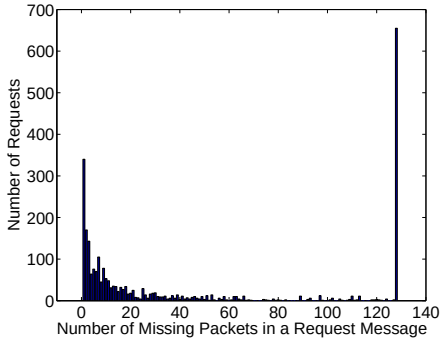


Figure 1. Packet loss pattern in MNP. 10x10 network, inter-node distance: 10 feet, program size: 8.4KB (384 packets).

3. A Hybrid Reliability Scheme for MNP

In Section 3.1, we briefly introduce the two (n, k) FEC coding schemes: simple XOR code and Reed-Solomon (RS) codes. In Section 3.2, we present the new reliability scheme for MNP, that is, adding XOR/RS FEC codes to MNP.

3.1 (n, k) FEC Coding Schemes

We consider (n, k) FEC-based approaches. There are two commonly used (n, k) FEC codes: XOR code, and Reed-

Solomon (RS) codes [7]. For simple XOR code, each transmission group has only one parity packet, which is the XOR of all the source packets in the group. Therefore, simple XOR code is a $(k+1, k)$ code. XOR code is very simple to implement. However, it can only repair a single packet loss in a transmission group.

RS codes are more flexible. RS codes are based on algebraic methods using finite fields. A transmission group can have multiple parity packets (i.e., n can be any number that is larger than k). Thus RS codes provide better protection against losses. However, the flexibility of RS codes is achieved at high processing costs, in terms of computation complexity and memory space.

3.2 Adding FEC to MNP

There are two approaches regarding the calculation of the parity packets. The first approach is to calculate the parity packets based on the actual packets that are sent in the current transmission. In this case, for every transmission, the parity packets are computed and sent. This incurs a lot of encoding and decoding overhead. Moreover, the receivers have to be aware of the *ForwardVector* of the current sender (cf. Section 2), that is, the packets that are sent in the current transmission, in order to decode the parity packets. The sender might need to send the *ForwardVector* several times in order to make sure that it is received by all the receivers. The second approach is to compute the parity packets based on the full segment. Compared to the first approach, this approach is more efficient because the encoding process is needed only once for each segment, rather than performed for each transmission; and the senders do not need to send *ForwardVector* to the receivers. Therefore, we use the second approach.

For each segment, there are a set of parity packets. The parity packets are assigned unique IDs that start from *SegSize*+1. A receiver keeps a bitmap of the parity packets, *ParityMissingVector*, in memory. The *ParityMissingVector* is operated just as the *MissingVector* in the original ARQ-based approach (cf. Section 2). All the bits in *ParityMissingVector* are set to 1 initially. When a node receives a parity packet for the first time, it set the corresponding bit to 0. We limit the size of the *ParityMissingVector* to be no larger than 4 bytes, thus the number of parity packets ranges from 1 to 32.

When a receiver sends a request, it puts *MissingVector*, as well as *ParityMissingVector*, in the request message. Correspondingly, a source node maintains a *ParityForwardVector* (in addition to the *ForwardVector* in the original algorithm), as an indicator of the *parity* packets that need to be sent if the node becomes a sender. The *ParityForwardVector* of a source node is the union of the *ParityMissingVector* in the request messages the node has received. Whenever a receiver receives a packet (either data or parity), it checks to see if the number of packets it has received is enough to recover all the missing data packets. If so, the receiver has received the entire segment, otherwise, it asks for the missing data and parity packets, and a sender sends the requested packets.

The difference between adding XOR code and Reed-Solomon codes is the capability of loss recovery and the complexity of encoding and decoding, as we mentioned in Section 3.1. Because XOR code can only have one parity packet and repair a single loss in a transmission group, in order to repair more than one loss in a segment, we divide a segment into t transmission groups. Each group has a fixed number ($SegSize/t$) of data packets and one parity packet. The parity packet that has ID *SegSize*$+i$ ($1 \leq i \leq t$) corresponds to the i^{th} transmission group. In each transmission, the sender sends all the requested data packets, followed by the requested parity packets. Whenever a node has received enough number of packets to recover the whole group, it sets all the bits of this group in *MissingVector* and *ParityMissingVector* to 0, so that the node will not request for packets within this group any more.

For Reed-Solomon codes, we consider a full segment as a transmission group, which can have multiple parity packets. As long as the receiver has received *SegSize* (or more) number of packets (either data packets or parity packets), the whole segment is received. Further optimization is possible. For example, consider one scenario where a receiver has 11 missing data packets, and has received 8 parity packets. In this case, the parity packets received are not enough to recover the losses. In order to recover the whole segment, the receiver only needs 3 more packets, rather than 11. Taking message losses into account, we allow the receivers to request for twice the *required* number of missing packets (k packets in (n, k) FEC schemes). In the previous example, the receiver will request for $2 \times 3 = 6$ packets. This feature is expected to reduce the number of retransmissions, especially when the number of parity packets is large. However, because in MNP, the packets a sender transmits is the combination (union) of the packets that are requested, it is likely that different receivers request for different sets of packets, thus the combination of them virtually covers the whole segment. In this case, this optimization is not effective. We add one restriction that if a node requests for a subset of the packets it is missing, it always requests for those packets that have the lowest IDs, so that it is more possible that the sets of packets requested by different receivers are overlapping.

4. Evaluation Results

We added simple XOR code and RS codes to MNP source code, as described in Section 3.2, and used TOSSIM [10], to evaluate the effectiveness of the new reliability scheme.

In TOSSIM, the network is modelled as a directed graph. Each vertex in the graph is a sensor node. Each edge has a bit-error rate, representing the probability with which a bit

can be corrupted if it is sent along this link. Asymmetric links exist in this model since the bit-error rate for each edge is decided independently. We decide the bit-error rate based on our experience with Mica-2 motes. Specifically, the packet loss rate on a one-hop (10 feet) link is around 5%. The loss rate increases with distance, and after 50 feet, the loss rate is 100%.

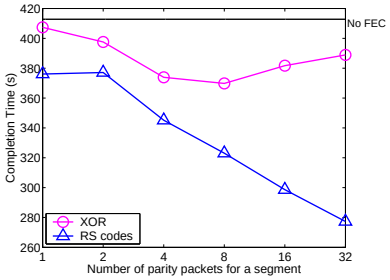
We consider two measurements: the completion time and the energy consumption. A previous study [11] shows that the time a Mica-2 mote keeps its radio on (i.e., idle listening time) contributes to the major part of energy consumption. The other parts of energy consumption include message transmissions and receptions.

The following simulations were conducted in a 10x10 network. The distance between two neighboring nodes is kept constant at 10 feet. In the current implementation, each segment has 128 data packets. The program size is 8.4KB(3 segments, 384 packets). We assume that initially only the base station, the node at a corner, has the new program.

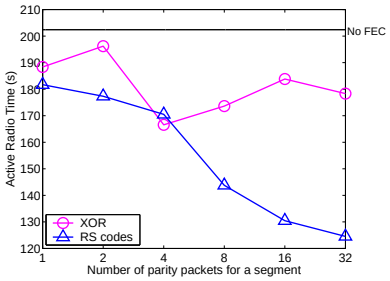
In Figure 2, we compare the performance of the original MNP protocol (marked as "No FEC" in Figure 2) with MNP protocol plus XOR/RS FEC codes. To prevent the randomness of a single simulation, we repeated each experiment three times, and presented the mean value. We found that adding either simple XOR code or RS codes to MNP helps improving performance. The completion time and the active radio time were reduced after we applied FEC schemes to MNP. For XOR code, when the number of parity packets is from 4 to 8, the reduction on completion time is about 10%, and the reduction on active radio time is about 14-17%. Increasing or reducing the number of parity packets minuses the performance gain. RS codes generally perform better than simple XOR code. As shown in Figure 2, using RS codes, the completion time is reduced by 9-33%, and the active radio time is reduced by 10-38%. For RS codes, the performance improves when more parity packets are used.

In Figure 3, we show the number of transmissions and receptions of the original MNP protocol and MNP plus XOR/RS FEC schemes. We note that, with XOR/RS FEC schemes, the number of transmissions and receptions are reduced by up to 19% and 41% respectively. In general, (MNP + XOR) scheme has lower number of transmissions and receptions than the original MNP protocol, and (MNP + RS codes) has the lowest number of transmissions and receptions among the three schemes. The number of receptions is largely decided by the average active radio time, as can be seen from Figure 2(b) and Figure 3(b).

We categorize the transmitted packets into two types: control packets and encoding packets. Control packets include the advertisements and requests. They are used for the sender selection algorithm and ARQ scheme. Encoding packets carry the information that is to be disseminated to the sensor nodes. They include data packets and parity packets. In Figure 4

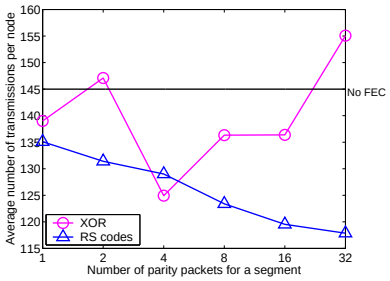


(a)

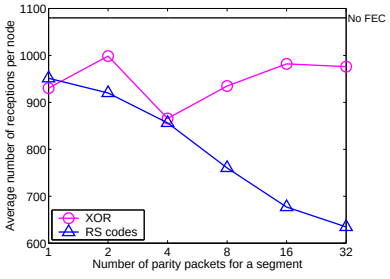


(b)

Figure 2. Completion time and active radio time of (MNP + XOR) and (MNP + RS codes), when the number of parity packets is from 1 to 32 packets per segment (128 data packets/segment). (a) Completion time (b) Average active radio time per node.



(a)



(b)

Figure 3. Average number of transmissions and receptions per node: (MNP + XOR) and (MNP + RS codes), when the number of parity packets is from 1 to 32 packets per segment (128 data packets/segment). (a) Average number of transmissions per node (b) Average number of receptions per node.

(a) and (b), we show the average number of control packets that are transmitted per node, for (MNP + XOR) scheme, and (MNP + RS codes) scheme, compared to the original MNP protocol. In Figure 4 (c) and (d), we show the corresponding results for encoding packets.

We note that using XOR/RS codes effectively reduces the number of control packets (up to 44%), especially the number of requests. For example, when we use RS codes with 32 parity packets per segment, the average number of requests per node is only 10 (Figure 4(b)), less than half of the requests transmitted per node when the original MNP is used. As to encoding packets, we note that although FEC schemes transmit additional parity packets, the number of data packets, plus the parity packets, is still lower than the number of data packets transmitted by the original MNP algorithm in general, with a few exceptions (when the number of parity packets per segment is 2 or 32 in (MNP + XOR) scheme).

From Figure 4, we can see how RS codes performance better than simple XOR code. For XOR code, when the number of parity packets per segment increases from 4 to 32, the additional parity packets do not contribute much to recovering packet losses, but incur higher transmission overhead due to more redundant packets. As we mentioned in Section 3, the limitation of XOR scheme is that, only one loss can be recovered in each transmission group. Although we can use multiple parity packets for a segment by dividing a segment into several transmission groups, only one loss from each group can be recovered. In other words, XOR code with multiple parity packets work best when the message losses are evenly distributed in the segment. However, in network reprogramming, a large part of the message losses are bursty in nature, caused by message collisions or channel errors. Therefore, dividing a segment into many tiny groups, and transmitting one parity packet for each group using XOR code, is not desirable. By contrast, RS codes can recover message losses at arbitrary locations. For RS codes, increasing the number of parity packets in a group improves its ability of recovering message losses. As shown in Figure 4 (c) and (d), for XOR code, when more than 4 parity packets are used for each segment, the number of encoding packets increases with the increase of the number of parity packets; for RS codes, the number of encoding packets remains the same although more parity packets are transmitted.

5. Related Work

Automatic repeat request (ARQ) and forward error correction (FEC) are the two basic ways to provide reliability for transmission protocols. All the existing work on network reprogramming [1–6] uses ARQ-based approaches for error recovery. In this paper, we propose adding FEC to the ARQ-based reliability scheme, and perform a case study on MNP [4], which is a multihop network reprogramming protocol. There are different types of FEC codes.

We have introduced simple XOR code and Reed-Solomon (RS) codes in Section 3.1. Both XOR code and RS codes belong to block (n, k) FEC codes. A block code has the property that any k out of the n encoding packets can reconstruct the original k source packets.

Tornado codes [12] provide an alternative to RS codes. Tornado codes have lower computation complexity than RS codes, at the small cost of reception overhead, that is, a (n, k) Tornado code requires slightly more than k out of n encoding packets to recover k source packets.

Unlike the block (n, k) codes, Luby Transform (LT) codes [13] can generate as many unique encoding packets as required, using the k source packets as input. Each encoding packet is generated randomly and independently of all other encoding packets. LT codes have the property that the receiver is able to reassemble the original k source packets as long as it receives enough number (slightly more than k) of encoding packets. LT codes are designed for delivering a large amount of data over high bandwidth internet links. They have lower computation complexity on encoding and decoding than RS codes. However, they introduce higher recovery overhead because more redundant packets are transmitted. Normally, the number of repair packets are more than 10 times the number of source packets.

6. Conclusions

Automatic repeat request (ARQ) is a commonly used technique for reliable bulk data dissemination applications in sensor networks. A typical example of this type of applications is network reprogramming. All the existing network reprogramming protocols use ARQ as the error recovery scheme. In this paper, we proposed a hybrid reliability scheme, which combines forward error correction (FEC) with ARQ schemes. The FEC provides an abstraction of a better transmission medium, and ARQ scheme takes care of the remaining error corrections. We use MNP, a multihop network reprogramming protocol, as a study case, and presented the implementation details of adding FEC schemes to MNP. Specifically, we considered two (n, k) FEC schemes: the simple XOR code, and Reed-Solomon (RS) codes. We added the two FEC schemes to MNP, and simulated them in TOSSIM.

The simulation results show that both simple XOR code and Reed-Solomon codes effectively reduce the number of request messages that ask for missing packets, thus enable faster reprogramming and less energy consumption. Adding XOR code to MNP can reduce the reprogramming time by 1-10%, and reduce the active radio time (which contributes to the major part of energy consumption) by 3-18%; while adding Reed-Solomon codes to MNP can reduce the repro-

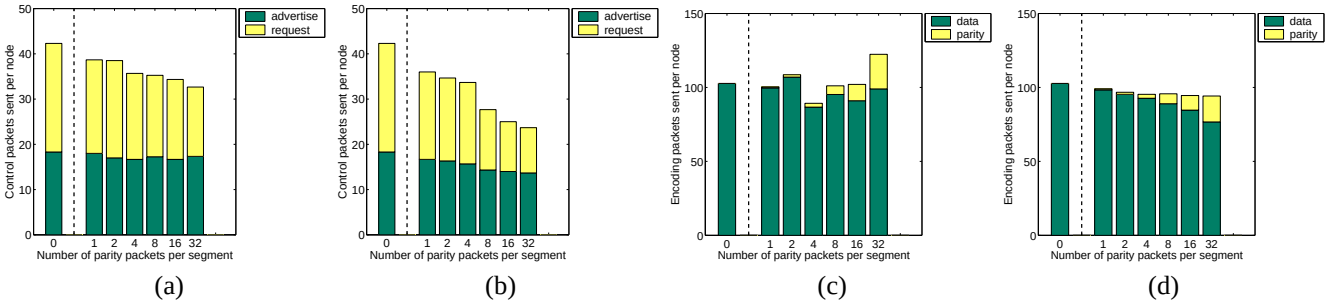


Figure 4. The average number of control/encoding packets transmitted per node. (a) control packets transmitted in (MNP + XOR) scheme (b) control packets transmitted in (MNP + RS codes) scheme (c) encoding packets transmitted in (MNP + XOR) scheme (d) encoding packets transmitted in (MNP + RS codes) scheme. * 0 - means original MNP (No FEC/parity)

gramming time by 9-33%, and reduce the active radio time by 10-38%. The message transmissions and receptions are reduced as well when the FEC schemes are used. We found that simple XOR code has limited capability of correcting errors (reducing completion time by up to 10% and reducing active radio time by up to 18%), that is, it cannot deal with bursty packet losses, which is not rare when disseminating a large amount of data in large networks. Therefore, increasing the number of parity packets does not help to improve the performance, only incurs more redundant transmissions. On the other hand, Reed-Solomon codes are more flexible. It can deal with any loss patterns. It becomes more powerful at correcting errors when more parity packets are used. XOR code is very simple to implement, while Reed-Solomon codes require higher computing resources due to the complexity of their calculation. It suggests a tradeoff between computation and performance: if more computation resources are available, a more powerful coding scheme, e.g., Reed-Solomon codes, should be used to achieve better performance; otherwise, a simple XOR FEC scheme can be used for limited improvement.

References

[1] Crossbow Technology, Inc. *Mote In-Network Programming User Reference Version 20030315*, 2003. http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/Xnp.pdf.

[2] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical report, UCLA, 2003.

[3] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the second International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, Baltimore, Maryland, 2004.

[4] S. S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks. *In Proceedings of the 25th International Conference on Distributed Computing Systems ICDCS*, pages 7–16, June 2005.

[5] S. S. Kulkarni and M. Arumugam. Infuse: A TDMA based data dissemination protocol for sensor networks. Technical Report MSU-CSE-04-46, Department of Computer Science, Michigan State University, November 2004.

[6] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy effi cient data dissemination service for wireless embedded devices. *To appear in Proceedings of the 26th IEEE Real-Time Systems Symposium*, December 2005.

[7] I. S. Reed and G. Solomon. Ploynomial codes over certain fi nite fi elds. *Journal of the Society for Industrial and Applied Mathematics*, 8(10):300–304, 1960.

[8] J. Hill and D. Culler. Mica: A wirleess platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, 2002.

[9] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. *In Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN), Special track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS)*, April 2005.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, CA, November 2003.

[11] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.

[12] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Effi cient erasure correcting codes. *IEEE Transactions on Information Theory, Special Issue: Codes on Graphs and Iterative Algorithms*, 47(2):569–584, February 2001.

[13] M. Luby. LT codes. *In Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–282, 2002.