Fountain Reprogramming Protocol (FRP): A Reliable Data Dissemination Scheme for Wireless Sensor Networks Using Fountain Codes

Riccardo Crepaldi, Albert Harris III
Department of Computer Science
University of Illinois at Urbana–Champaign
{rcrepal2,aharris}@cs.uiuc.edu

Michele Rossi, Giovanni Zanca, Michele Zorzi
Department of Information Engineering
University of Padova, Italy
{rossi,zancagio,zorzi}@dei.unipd.it

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design

General Terms: Design, Performance, Reliability

Keywords: Sensor networks, in-network programming

1 Introduction

Wireless sensor network technologies enable a wide variety of applications (*e.g.*, environmental monitoring). Such sensor networks are often deployed in regions that make it difficult to collect and redistribute the nodes for maintenance. However, there is often a need to reprogram all of the nodes in the network, either during application test phases on deployed networks, or to support software upgrades. Therefore, a reliable method of sending a relatively large amount of data to each node in the network is required to support these functions.

The challenge to designing such in-network node reprogramming protocols lies in the potentially large amount of energy required to successfully transmit the entire program to every node in the network. The wireless channels used by small sensor nodes are often lossy and highly variable. The use of unicast retransmissions to correct errors for each node can be prohibitive in terms of traffic generation and hence transmission cost. Additionally, such retransmission techniques are known to result in feedback implosion in dense networks [1]. Therefore, coding solutions allowing different errors at various nodes to be corrected with single packet transmissions are preferable. However, many such techniques, using forward error correction codes (FEC), tend to be inefficient for wireless sensor networks. This is mainly due to the inherent computational complexity of standard codes (e.g., Reed Solomon). In addition, standard block codes have a fixed code rate, which cannot be changed on the fly according to channel errors or number of receivers.

As a solution to the above problems, in this work we propose the Fountain Reprogramming Protocol (FRP), which uses a Fountain Code [2] designed specifically to meet the needs of sensor network reprogramming. In particular, this code is designed to maintain a high efficiency, in terms of overhead, in the face of small packet sizes and typical program lengths. In addition Fountain Codes are rateless and have a low computational complexity, as encoding and decoding are performed efficiently through XOR operations. Our fountain code has been implemented on Tmote Sky nodes and shown to execute efficiently even with the limited available processing power. Our experiments show that we achieve reliable network programming with very low overhead compared to other current in-network reprogramming techniques [3, 4].

In the rest of this paper, we briefly describe the fountain code. We additionally describe our testbed and test application. Finally, we describe the demonstration.

2 Efficient Fountain Codes

When network programming begins using FRP, the original program data is first encoded using our fountain code. Essentially, the sending node first creates a number of linear combinations of the original K (source) packets. These encoded packets are generated thanks to a pseudo-random number generator, which is used to obtain samples from a selected degree distribution. To successfully decode the original message, any receiver node needs to receive K linearly independent encoded packets. In this case, in fact, the corresponding decoding matrix has full rank and can be inverted. However, given the characteristics of typical degree distributions, the first K encoded packets received are unlikely to be independent. Thus, in practice full recovery occurs upon the reception of $K' \geq K$ encoded packets. In the most general case, K' depends on the encoding distribution in use, on the seed used for the initialization of the pseudo-random generator as well as on the error patterns introduced by the wireless channel. Nevertheless, for properly designed codes K' is only slightly larger than K. We found optimized encoding distributions for different values of K, having different performance in terms of overhead ($K' - K$) and decoding cost (XOR operations needed to decode). For our demonstration we selected a distribution leading to a good tradeoff between overhead and cost.

In case a receiver is still unable of inverting its decoding matrix after a first round of transmissions, it will ask the transmitter for additional (encoded) packets. Note that a single encoded packet can correct errors (i.e., increase

Copyright is held by the author/owner(s).
SenSys'07, November 6–9, 2007, Sydney, Australia.
ACM 1-59593-763-6/07/0011

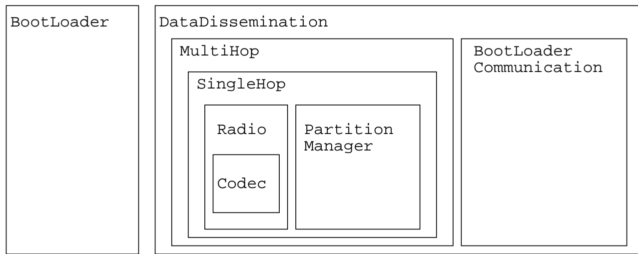

Figure 1. Software architecture

the rank of the decoding matrix) at different receivers. A special mechanism has been designed to get feedback from the receivers and retransmit the needed amount of extra-redundancy in a round based fashion. According to this scheme each receiver computes, at every transmission round, the rank r of its current decoding matrix and decides whether to send a NACK on the feedback channel (thus requesting additional packets). The process stops when all of the receivers have a *full rank* decoding matrix, i.e., they can all recover the original data. Due to the inherent scalability of our coding technique, feedback implosion is never caused, even in the face of poor channel conditions. In fact, an ARQ request from a single receiver can potentially resolve multiple losses at all receivers.

The size of applications or data files we intend to disseminate is relatively large (a simple TinyOS application that includes radio communication capabilities is typically on the order of 10 KB). Due to the small amount of RAM usually available in the sensor nodes, to transmit such an application to another node and store it in the flash memory, the data needs to be split into chunks (of K packets each), where the chunk size depends on the available RAM. Hence, a chunk at a time is sent to the receiving nodes according to the above procedure. When a chunk is received, FRP stores it in the flash memory of the node and waits for the transmission of a new block of data, which will start when all the nodes have finished decoding the current one. When all the nodes have received the whole file a command is broadcast through the network to begin the application reload process. FRP has been designed for nodes based on the MSP430 microcontroller, that provides 10 KB of RAM. Due to the memory needed for the decoding process, we chose a chunk size of 800 bytes. This leaves more than half of the RAM free for other normal operations, such as radio protocols. Each chunk is divided into $K = 32$ packets of 25 bytes each. These packets are then coded and broadcast to the network.

As Fig. 1 shows, the application is designed in a modular way that makes it very easy to alter. For example, it is simple to adapt FRP to work with other node technologies (as in fact we did with TDA5250 mounted on the *EyesIFX* nodes [5]), simply by changing the `Radio` module.

In addition to the dissemination protocol, we designed and implemented a dynamic partitioning system that allows the use of the external flash memory as a WORM (Write Once Read Many) device, without knowing in advance the number and the size of the partitions, as the components provided by TinyOS2 require. Using a hash-code, a unique ID is created for each file to be transmitted. This ID is used to tag a partition of the size of the file, that can be mounted and read at any time.

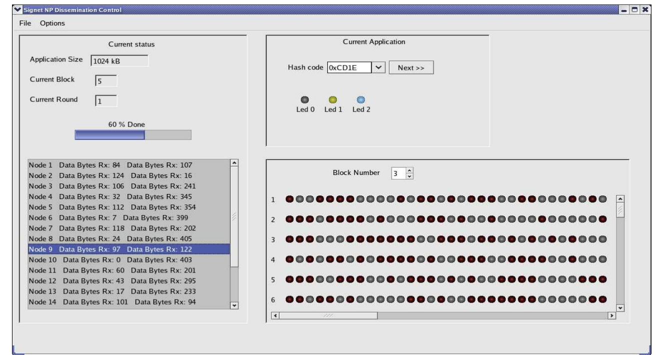

Figure 2. The monitoring application GUI

Finally we developed a bootloader that, when the node is rebooted, can copy a new application from the external flash into the program memory and run it.

3 Description of the Demonstration

The goal of this demonstration is to show FRP as it disseminates data in a reliable fashion, proving its effectiveness for network reprogramming. We will show how FRP deals with packet loss and retransmission requests, without causing feedback implosions, even in the face of high error rates. To simulate higher error rates than the ones we can obtain in the small space of the demo, our application will randomly drop packets with a given probability.

We will deploy a testbed at the conference, using 20 Tmote Sky nodes. One of these will be the transmitter, and it will disseminate data to all of the other nodes. This node will be transmitting a 20 KB program that performs a led blinking scheme with different patterns each time it is loaded. At the end of each run we will show the correctness of the dissemination, by verifying the data received and stored in the external flash. After the data is verified, we will send a command to the nodes to load and run the new application and check the led blinking pattern to confirm the new program is in fact running. Finally, a new reprogramming session will be started which will contain a program with a different blinking pattern.

While the dissemination protocol is running, we will constantly monitor the activity of the network (Fig. 2 shows the GUI of the monitoring application), showing in real-time on a laptop the number of data packets being transmitted as well as the number of overhead packets (including feedback and ARQ requests) that are received by the transmitter. We will also monitor and report the time needed for the transmission of each block of data as well as the time required for the complete reprogramming cycle.

4 References

[1] J. Nonnenmacher, E.W. Biersack, and D. Towsley, "Parity-based loss recovery for reliable multicast transmission," *IEEE/ACM Trans. on Networking*, vol. 6, no. 4, pp. 349–361, 1998.

[2] D.J.C. MacKay, "Fountain Codes," *IEE Proceedings – Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.

[3] J. Jeong, S. Kim, and A. Broad, "Network Reprogramming," Berkeley,CA,USA, Aug. 2003.

[4] D.C. Jonathan W. Hui, "The Dynamic Behaviour of a Data Dissemination Protocol for Network Programming at Scale," in *Sensys'04,*, Baltimore, Maryland, USA, Nov. 2004.

[5] Infineon, Ltd., "EyesIFXv2 version 2.0," http://www.infineon.com.