# Reliable and Efficient Reprogramming in Sensor Networks

CHRIS MILLER and CHRISTIAN POELLABAUER
University of Notre Dame

Retasking and remote programming of sensor networks is an essential functionality to make these networks practical and effective. As the availability of more capable sensor nodes increases and new functional implementations continue to be proposed, these large collections of wireless nodes will need the ability to update and upgrade the software packages they are running. In order to do this, the new binary file must be distributed to all nodes in the network. Making a physical connection with each individual node is impractical in large wireless networks. Standard flooding mechanisms are too energy-costly and computationally expensive and they may interfere with the network's current tasks. A reliable method for distributing new code or binary files to every node in a wireless sensor network is needed. We propose a reprogramming/retasking framework for sensor networks that is energy efficient, responsive, and reliable, while maintaining a stable network.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication, distributed networks, network topology*

General Terms: Algorithms, Design, Performance, Reliability

Additional Key Words and Phrases: Broadcast, energy efficiency, minimum energy broadcast, reliable distribution, reprogramming, retasking, sensor networks

**ACM Reference Format:**

Miller, C. and Poellabauer, C. 2010. Reliable and efficient reprogramming in sensor networks. ACM Trans. Sensor Netw. 7, 1, Article 6 (August 2010), 32 pages.
DOI = 10.1145/1806895.1806901 http://doi.acm.org/10.1145/1806895.1806901



## 1. INTRODUCTION

Wireless sensor networks are a collection of small sensor devices, typically battery powered, that may be static or mobile, and may be configured in an ad hoc fashion. Retasking sensor networks which have already been deployed presents several complications that are worthy of research. It requires the reliable distribution of a binary file or code to all of the nodes which require the

Authors' addresses: C. Miller and C. Poellabauer, Department of Computer Science and Engineering, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556; email: miller.444@nd.edu; cpoellab@cse.nd.edu.

DOI 10.1145/1806895.1806901 http://doi.acm.org/10.1145/1806895.1806901

new program, and a coordinated method of loading the new program among all of the nodes. This can be an expensive task, requiring significant battery power to fulfill all necessary radio transmissions, and consuming the computational resources of the wireless nodes. In this article, we aim to provide a reprogramming framework for sensor networks that is energy-efficient, responsive, and reliable, while maintaining a stable network autonomously.

The primary focus of efforts to make reprogramming/retasking energy-efficient is on the task of file distribution in support of remote programming. There have been a number of proposals for such a functionality [Wan 2005; Levis et al. 2004; Yu et al. 2006; Busnel et al. 2007]. Many of these have worked under the assumption of a small binary or code file, requiring a limited number of segments to be distributed. This condition was typical of early wireless sensor motes, which have limited memory capacity, but as the capability of sensor motes increases, sensor network programs will inevitably increase with them. In this article, we will explore a more efficient method of binary or code distribution for relatively large data files. The most basic method of file distribution is by flooding the segments into the sensor network. This method is very costly, though, as every node will receive and broadcast each segment of a file. Many of these broadcasts are unnecessary since sensor networks may be dense, and have highly overlapping broadcast zones. The redundant transmissions are an unnecessary expenditure of power and could lead to increased packet loss due to congestion. Another issue with the naive flooding method is that it provides no reliability. It is important to ensure that all nodes in the network receive the entire file in a timely manner, so reliability mechanisms will be needed.

The protocol chosen for binary file distribution is *Push Aggressively with Lazy Error Recovery* (PALER), a reliable transport protocol that was proposed in Miller and Poellabauer [2008]. PALER is capable of distributing a dataset or binary file to all nodes within a wireless network quickly and efficiently. It is optimized for large data files, requiring multiple packets for transmission. The primary design metrics of PALER are minimum energy cost and minimum latency. PALER reduces energy costs by limiting redundant transmissions using a neighbor elimination scheme (NES). One contribution of this work is to further enhance the energy efficiency of PALER by replacing the NES scheme used for dissemination with a transmission power control broadcast scheme. This broadcast protocol is intended to minimize total energy cost in a distributed fashion. The transmission power control scheme differs from other methods by using received signal strength measurements to calculate link cost, rather than using distance. This modification makes the scheme adaptable to realistic wireless environments, where path link properties may vary significantly. The reliability mechanisms of PALER are also modified to take advantage of the tree structure provided by this new dissemination scheme.

We propose a new autonomous sensor network reprogramming toolset called *Cascade*. Cascade will incorporate an energy-efficient distribution scheme based on PALER, using the enhanced broadcast scheme. We will show through simulation and experimentation that Cascade provides an energy-efficient distribution method for new binary files. We will also show that Cascade has a low

latency and provides reliability. A complete framework for handling remote programming will be provided. This will include any steps necessary for handling the newly distributed binary file, and managing the crossover to the new program, as well as handling nodes entering and exiting the network. These steps are crucial for maintaining a stable sensor network during a retasking event. This work is an extension of the work presented in Miller and Poellabauer [2008], and integrates the work presented in Miller and Poellabauer [2009]. The major contributions of this work are to provide an efficient, reliable, and low-latency method of distributing large binary files for sensor network programming. It also provides a method of autonomous maintenance to keep all nodes up to date, and to handle network changes, and a coordinated transition of new applications. Cascade differs from similar work in this area in that it uses a transmission power control scheme to establish a broadcast tree in a sensor network programming protocol. It also uses received signal strength measurements for the calculation of link costs rather than distance-based approaches. Results from simulation and experimentation on multiple platforms are provided to evaluate the performance of the distribution scheme on sensor motes of varying resources and capabilities.

## 2. REQUIREMENTS OF A REPROGRAMMING TOOL

A complete toolset for handling and managing remote programming on sensor networks should provide several features. It should be capable of initiating a retasking event from any location within the network. It should determine the best method for disseminating all portions of the new binary file dynamically based upon the properties of the file. It should reliably and efficiently distribute the complete binary file to all nodes which require the new version of the application. It should manage the crossover from the prior version of the application to the new version without impairing the functionality of the network. It should also autonomously handle updating new nodes to maintain the newest version of applications across the network. The following subsections will provide a brief overview of each of these features of the toolset. A more detailed description will be provided in the following sections.

### 2.1 Programming Preparation

In order to make retasking simple and easy to implement, a reprogramming toolset should be designed as a daemon which runs continuously in the background. This will allow a retasking event to be initiated at a single node without having to make a connection to all nodes to initiate the process. This daemon will require several pieces of information which must be maintained on each node to manage retasking requests. Each node will need to maintain a list of current applications loaded on the flash drive, along with the version number of the current file and an identification code which will remain constant across versions. They should also maintain a list of active applications along with any command line arguments that are needed to run the application. This information will be stored in metadata files on each node.

When a new programming event is initiated at a node, it will partition the file into chunks and initiate the distribution phase of reprogramming. Receiving nodes will compare the reprogramming request against their metadata files to determine what actions need to be taken, and if it should participate in the retasking event. The information in the metadata files will also be used to ensure nodes are operating with the latest versions of all applications.

### 2.2 Reliable Distribution

Our reprogramming toolset will make use of PALER for binary code distribution. PALER is a reliable transport protocol designed to support large dataset and binary file distribution over wireless sensor networks. It aims for energy efficiency and low latency. It uses an aggressive dissemination method to push each segment of a binary file to all nodes in the network as quickly and efficiently as possible. Recovery operations are withheld until the completion of the dissemination phase, allowing segments to be received out of order, and batching recovery messages to reduce transmissions. PALER has two primary phases: dissemination and recovery.

2.2.1 *Dissemination.* In this phase, PALER pushes all segments of a file to the network as quickly as possible. In the initial PALER proposal, energy cost of dissemination was reduced using a neighbor elimination scheme. For Cascade, PALER was modified to use Dynamic Broadcast Incremental Power (DynaBIP) [Miller and Poellabauer 2009] for the dissemination phase. DynaBIP reduces total energy cost by dynamically constructing a broadcast tree to reach all nodes using transmission power control.

2.2.2 *Recovery.* The recovery phase begins after the completion of the dissemination phase. Each node will construct a list of all segments which it did not receive during the first phase. This information will be broadcast in a negative acknowledgment (Nack) packet to its neighbors. Neighboring nodes which receive the Nack will rebroadcast any missing segments which they possess. All recovery operations are handled locally, meaning that Nacks are not propagated beyond single-hop neighbors, and all missing segments are recovered from immediate neighbors. This reduces the cost of redundant control messages during recovery.

### 2.3 Version Transition

One important feature of the reprogramming toolset is that it should maintain a stable network during the retasking process. It is not known what the application of the sensor network may be. It may be performing a critical task, and this should not be interrupted or interfered with for reprogramming. The most crucial aspect of maintaining stable operation of the network during the retasking event is management of the crossover from an old version of an active application to a new version. This hand-off should be performed with minimal impact to each node locally, and coordinated among all nodes to ensure there is no conflict.

### 2.4 Maintenance

Some sensor network applications may consist of a very large number of small wireless nodes, which may be expected to expand over time to provide broader or more precise coverage. This may include networks with very inexpensive nodes, where small numbers of new nodes may be periodically added to the network to handle expected node failures. For these applications, it is preferred that new nodes may be randomly inserted into the network with little setup, and the network will autonomously bring all nodes up to date. There may also be situations where nodes are temporarily disconnected from the network, and also need to be brought up to date. Therefore, the reprogramming toolset will need to include methods for detecting new nodes or nodes with outdated versions of applications, and an efficient method of updating the nodes.

## 3. RELATED WORK

### 3.1 File Distribution

Code distribution has many similarities with reliable multicast, reliable broadcasting, and energy-efficient broadcasting research. Some techniques for making multicast reliable in wired networks, such as proposed by Rizzo and Vicisano [1997], are computationally expensive for the limited resources of a wireless node. Multicast protocols which have been developed for wireless networks, such as those by Lee et al. [2002] and Ho et al. [1999], tend to favor robustness to provide reliability. This built-in redundancy comes at the expense of energy efficiency. Wireless broadcasting protocols focused on reliability typically require significant overhead in control packets [Hsu et al. 2007; Pagani and Rossi 1997].

Energy-efficient wireless broadcast protocols typically focus in two areas: minimizing forwarding nodes [Tseng et al. 2002; Wu and Dai 2003; Stojmenovic et al. 2002; Wu and Li 2001], or minimizing transmission power [Wieselthier et al. 2002b; Bian et al. 2002; Cheng et al. 2003]. Tseng et al. [2002] offered several methods of reducing forwarding nodes, such as probabilistic, counter, and cluster methods, each with differing levels of reliability. Wu and Li [2001] developed an algorithm to identify a dominating set in a network, which would make up the intermediate nodes in all broadcasts. Stojmenovic et al. [2002] and Wu and Dai [2003] built upon this method by reducing the size of the dominating set. Selection of a dominating set may reduce total transmissions in a broadcast; however, it does not balance the load, as the nodes in the dominating set will incur all of the cost. An interesting extension presented in Stojmenovic et al. [2002] is a neighbor elimination scheme which reduces forwarding nodes. Park et al. [2004] also used an approximation of the minimum dominating set to achieve reliability and reduce the cost of recovery. Their scheme is capable of providing reliable broadcast of a single packet, and limits the cost of dropped packets with local recovery, but the focus of this work is on reliability rather than the energy efficiency of the downstream propagation. Broadcast protocols aimed at minimizing transmission power are typically based upon a set cover [Bian et al. 2002] or minimum spanning [Cheng et al. 2003; Wieselthier et al.

2002b] problem. These tend to provide very efficient distribution trees and balance loads evenly; however, they require a knowledge of the complete network topology, and are best used in a static environment where optimal routes can be predetermined.

Other methods of achieving reliability or efficiency include FEC and network coding techniques. FEC was used in conjunction with probabilistic forwarding by Rahnavard and Fekri [2006] to add reliability to an efficiency scheme. The FEC technique offers flexibility in the propagation and recovery methods. Lossy environments such as wireless sensor networks could require an undesirable number of overhead packets produced by the encoding method to provide reliability. The forwarding probability used for their simulations is optimized for the network topology, which provides efficient propagation results. This would make implementations sensitive to alterations in the network size and density, however. Network coding was used in Lun et al. [2005] to improve efficiency by reducing the number of transmissions. They provided an alternative algorithm in their approach for multicast in wireless networks to exploit the wireless multicast advantage. Their algorithms provide a minimum-cost multicast tree, without requiring global knowledge.

### 3.2 Reprogramming/Retasking

PSFQ [Wan 2005] is a transport protocol designed for the support of wireless network reprogramming. It distributes data from a single source to a network of sinks by slowly pacing the propagation of packets. It uses in-order forwarding to reduce the propagation of control messages, that is, regardless of the order of reception, nodes will only rebroadcast packets in-order. This prevents the propagation of a loss event to the downstream nodes. Dropped packets are fetched aggressively at the time of discovery. Requiring in-order broadcasts also ensures that lost packets can be retrieved from at least one immediate neighbor. If a node receives a packet with a higher sequence number than expected, it can assume the source of that packet must possess the missing sequence numbers. By localizing recovery, it reduces loss recovery cost by suppressing the propagation of loss events and negative acknowledgments, and reducing recovery to a single-hop transmission. PSFQ uses the counter method to reduce redundant transmissions and avoid the broadcast storm problem. Due to the aggressive nature of the recovery operations, lossy environments may lead to a high number of recovery messages and collisions. The in-order forwarding mechanism also places a lower bound on latency, which scales linearly with network size.

Trickle [Levis et al. 2004] is a popular recent method of code propagation and maintenance. It uses a gossiping protocol with periodic metadata broadcasts to identify nodes which require an update to a new version of code. It uses a counter method to limit the number of gossip messages broadcast during an interval, which makes Trickle a very energy-efficient method of maintaining a sensor network. Trickle is not greatly concerned with latency, and is based upon an expectation of a very small code segment or binary file, one which will fit in a small number of packets. Another gossip-based code propagation

protocol is GCP [Busnel et al. 2007]. It uses periodic beacons to detect outdated code versions, similar to Trickle. However, it also includes a forwarding control mechanism to balance the load of distribution. Each node has a limited number of tokens that it may use for distributing each new version of code.

Melete [Yu et al. 2006] builds upon Trickle to support dynamic grouping and concurrent applications in sensor networks. It uses a periodic metadata broadcast to maintain the network. It also supports group-based code propagation. Nodes may dynamically enter and exit a group, and must broadcast a request for the new group code. Melete avoids broadcast implosion by pacing requests through a probabilistic and progressive flooding mechanism. Because of the dynamic grouping nature of their system, code propagation is accomplished through multihop unicast in many situations. Deluge [Hui and Culler 2004] also builds upon Trickle to extend its functionality to relatively large program images. Deluge divides a program object into multiple pages, each made of multiple packets. Nodes only perform receive or transmit operations on a single page at a time, limiting the amount of metadata needed to ensure reliable delivery of a large program image. MOAP [Stathopoulos et al. 2003] shares many design aspects with Deluge, but does not partition the program image into pages. As a result, nodes must receive the complete program image before becoming a source to other nodes.

MNP [Kulkarni and Wang 2005] performs a segmentation of relatively large program images similar to that performed by Deluge. It also must receive a complete segment before a node can become a source for that segment. It uses a count of requests received to prioritize selection of sending nodes. Infuse [Kulkarni and Arumugam 2006] uses TDMA to avoid collisions while disseminating a data file over a multihop network. However, this protocol requires that every node transmit every packet of the file. Message receiving is reduced by allowing nodes to select preferred predecessors so that nodes only have to listen during specific time slots. Sprinkler [Naik et al. 2007] combines a connected dominating set with TDMA to reduce the forwarder set while avoiding collisions. It assumes an extreme scale and a dense network, and is capable of greatly reducing the number of forwarders and latency in such an environment.

## 4. CASCADE

The Cascade reprogramming toolset consists of a daemon which runs continuously in the background of every node in the sensor network. This allows a retasking event to be initiated at any node without having to make a connection to all nodes to initiate the process. The daemon will include a beaconing mechanism, and a listening thread which will capture and process all beacon messages and retasking messages. When a new retasking event message is received, it will initiate threads to handle broadcast tree construction, packet forwarding, and recovery mechanisms. Several pieces of information must be maintained to support these processes. Each node will need to maintain a list of current applications loaded on the flash drive, along with the version number of the current file and an identification code which will remain constant across

versions. They should also maintain a list of active applications along with any command line arguments that are needed to run the application. This information will be needed to handle the switch to the new version if the prior version of the application is currently active. This information is maintained on two files local to each node, which will be referred to, respectively, as *applications.dat* and *active.dat* from this point forward.

When a new programming event is initiated at a node, it will check *applications.dat* for a current version of the file. If there exists an entry for the application, it will update the version number to that of the new file and initiate the distribution. If an entry does not exist, it will create a new entry using a hash function to generate a new identification code for the application, and will then initiate a distribution of the new application. The header of packets in the tree construction phase will include information about the file or application that will allow receiving nodes to allocate the necessary resources. The header will include the identification code, the version number, the file size, the number of chunks the file will be broken into, and the destination path and filename. Each receiver will compare the identification code and version to its *applications.dat* file to determine if this is a new version of the file. If so, it will allocate the necessary space for the file size. Once all chunks have been successfully received, it will save the completed file to the destination path and filename and update the version data in *applications.dat*.

Cascade utilizes periodic beacon messages. The beacon messages serve two purposes: providing local topology information used for broadcast tree construction, and verifying that each node has the latest version of all applications. Each beacon message contains the following information: the radio transmit power, a list of immediate neighbors, and a listing of all applications with current version.

The radio transmit power is the power that the node's radio is set to, in dBm, for the transmission of beacon messages. This would typically be near maximum power, or the desired maximum range for all nodes. This data is used for calculating the link cost by receiving nodes. Each receiving node will compare the received signal strength to the transmit power to determine the path loss of this link. The path loss can be used to calculate the transmit power needed to reach this node, this will be considered the link cost. This value could be stored in a single byte.

The list of neighbors is a list of the node IDs of all immediate neighbors of a node, ordered by cost of each link, with the node providing the minimum cost link listed first. The node IDs may be determined by any method which provides unique identifiers, such as a simple hash function on the IP or MAC address. Typically assigning it to the last 2 bytes of either of these values will be sufficient to avoid duplicates in a large network. The total size of the neighbor list is dynamic, depending on the density of the network, and the limit of neighbors for which metadata is maintained. This limit may be adjusted to fit the memory resources available for the target device.

The list of applications will include an identification code and version number for each application stored on the device. This will allow receiving nodes to compare the application list to their local copy of *applications.dat* to determine

if there is an application they are missing, or if they have an old version of one of the applications. This will allow nodes that recently joined the network, or that were temporarily disconnected during retasking, to request an update. Each application entry can be coded with a 4-byte identification code and a 2-byte version number.

The beacon messages for Cascade may be piggybacked on to beacon messages utilized by a sensor network, so that the only overhead is the incremental cost of the additional bytes. If a sensor network is not utilizing beacon messages, or if the devices are resource constrained and have small packet sizes, the overhead of the Cascade beacon messages may be undesirable. In such a case, the beacon function of Cascade may be split into two phases. In the maintenance phase, nodes would periodically broadcast advertisement messages, similar to the process used by Deluge. The backoff delay between advertisements may be exponentially increasing as in Deluge to reduce energy cost during inactive periods. Once a new program image is detected, an initialization phase may begin, during which all nodes would send beacons containing their neighbor list. This would allow nodes the opportunity to calculate link costs with their neighbors prior to dissemination. The reduced steady state energy usage would come at the cost of increased latency for file dissemination, but this additional delay would be relatively small compared to the dissemination period.

## 5. RELIABLE DISTRIBUTION

Cascade uses a distribution scheme based upon PALER for reliable and efficient distribution of new binary files for reprogramming. PALER is a reliable transport protocol aimed toward large code distribution in wireless sensor networks. It pushes all segments in a binary file aggressively and efficiently to minimize latency and energy cost. Recovery options are handled in a second phase in order to provide reliability with minimal impact to latency and energy cost. To avoid the contention and collisions resulting from an aggressive recovery mechanism, PALER allows out-of-order reception and maintains a list of missing segments. The first phase of PALER is a dissemination phase where all segments of a file are pushed to the nodes in a fast, nonreliable manner. In the second phase, a recovery mechanism ensures that each node receives the binary file completely.

In the first phase of PALER, the source node divides the program image into packet-sized segments and disseminates all of the segments to the network. Any node that is designated as a sending node will forward the segments as they are received, using random delays to reduce the impact of collisions. The designation of the sending nodes will be explained further in the following section. Each node allocates enough memory space for the program image, and copies the data packets into the memory allocated as it is received. A bit vector is maintained to track which segments have been received. This allows nodes to easily access received segments to handle requests from other nodes. For resource-constrained sensor motes, it may not be possible to allocate the memory space for the complete program image. In this case, each node may allocate enough memory for $N$ pages, where the page size is the optimal size

of write buffers to the flash memory. Each page may be made up of $M$ blocks, which are the size of data packets. Each node would then have a $N \times M$ buffer of packets to service requests from other nodes. The values of $N$ and $M$ would depend on the available memory, the optimal write buffer size, and the packet size.

After all packets have been broadcast, each node will broadcast a Nack to its neighbors. An important aspect of PALER is that it maintains a local recovery mechanism, which reduces control messages. The reason this is possible is because, if a neighbor receives a Nack, it can first check its own cache for the missing segments. Any segments that are present in the local cache may be sent to the requester in a single-hop transmission. As for segments that are not present in the local cache, these segments must also be among the list of missing segments on the local node, which means that they will be included in the local node's Nack. Therefore, it does not need to propagate the Nack from its neighbor, because the missing segments will be redundant. This maintains local recovery with single-hop Nack transmissions, while still ensuring that requests for missing segments will be propagated until the missing segments are found.

Following is a more detailed description of each phase of PALER.

### 5.1 Flooding Mechanism

With the lazy error recovery of PALER, recovery messages are aggregated until after the dissemination. This allows nodes to send a smaller number of Nack messages containing the list of segments they require, and also provides nodes an opportunity to overhear neighbor requests, so that duplicate Nack requests can be suppressed. However, the push operation still requires a relatively large number of transmissions, even in fairly dense networks, and many of these transmissions are redundant. The originally proposed PALER used a NES scheme to reduce redundant transmissions. The reliable distribution method for Cascade uses DynaBIP to dynamically alter radio transmit powers to minimize total energy costs.

5.1.1 *DynaBIP.* For this implementation in Cascade, a distributed version of Broadcast Incremental Power (BIP) [Wieselthier et al. 2000] is used to provide energy-efficient dissemination for the flooding phase. DynaBIP constructs a broadcast tree based on the principles of BIP in a single sweep across the network. It approximates the minimum-energy broadcast tree to allow the dissemination of a packet to all nodes in the network with the least energy cost possible. This process also greatly reduces contention, allowing the flooding phase to operate very quickly and with minimal packet loss due to collision. A more detailed description of how the broadcast tree is constructed will be provided in Section 6.

### 5.2 Recovery

When a node receives the last segment of a distribution, it schedules a broadcast of a Nack to its parent following a delay proportional to the number of hops it is

from the source in the constructed broadcast tree. This delay allows the Nacks to propagate gradually toward the leaf nodes, giving the intermediate nodes an opportunity to fulfill any missing segments beforehand. The Nack message will include a list of segments it is missing. If a node receives a segment that it was missing prior to broadcasting its Nack, it will remove that segment number from its list of missing segments. Also, if a node overhears another child of its parent sending a Nack, it will remove any common segments from its list of missing segments. This reduces redundant Nack transmissions from child nodes which may be missing the same segment. When a parent node receives a Nack, it checks the list of missing segments against its own cache. If it owns any of the missing segments, it will schedule a reply with the missing segment following a random delay between 10 and 20 ms. If it owns more than one segment, it will schedule each additional segment for a reply at 5- to 10-ms intervals. The delay prior to replies reduces contention and allows the nodes to overhear other Nack requests. These requests are aggregated so that only one response transmission per segment is used.

Since this recovery mechanism is dependent upon reception of the last segment, a timeout period is used in case of loss of the last segment. The timeout period is continuously updated to be $T_{last}+\alpha*(T_{avg}*[Seg_{tot}-Seg_{rec}])+HOP\_DELAY$. Here, $T_{last}$ is the time the last segment was received, $T_{avg}$ is the average interval between reception of each segment, $Seg_{tot}$ is the total number of segments, $Seg_{rec}$ is the number of segments received, $HOP\_DELAY$ is the proportional delay based on hop length from source, and $\alpha \geq 1$ is a small multiplier that determines how aggressive the timeout value should be, typically less than 1.5. Following the transmission of each Nack, a new timeout value is set to specify a maximum time expected before all missing packets are recovered. If any segments are still missing at the end of this timeout period, another Nack will be transmitted, and a new timeout value will be set. Each additional timeout period will be doubled to avoid a Nack implosion resulting from downstream nodes waiting for a multihop recovery to propagate.

In the case of an intermediate node failure, children of the failed node will need to acquire a new parent node to recover all missing segments. If a node broadcasts two Nack messages without a response from its parent node, it will assume the parent has failed, and will acquire a new parent. The link cost for the previous parent will be set to a value greater than the maximum transmit power, to avoid reacquiring the parent. The child node will choose the node which provides the lowest energy cost link among the remaining nodes, and will send a Nack to this new parent node. When a node receives a Nack, destined for itself, from a node that is not one of its children, it will assume that it has been assigned as a new parent for this node. It will add the node to it's child list, and adjust its transmission power as necessary to reach the new child. It will then respond to the Nack as would typically be done, sending any missing segments which it has available. Prior to sending the missing segments, it will schedule a brief delay to allow time to hear from other nodes which may select it as a new parent. If the failed node had multiple children, they may all select the same new parent, and this will allow a single rebroadcast of each missing segment for all of them.

## 6. DYNAMIC BROADCAST INCREMENTAL POWER PROTOCOL

Power efficiency and total latency are the primary design goals of Cascade, so it is important that the broadcast method used in the flooding phase be very energy efficient and reduce contention to allow fast propagation. What is desired here is the minimum-energy broadcast tree, that, is the tree rooted at the source which reaches all nodes in the network with the minimum total energy cost possible. A true calculation of this tree is NP-complete [Liang 2002; Zagalj et al. 2002], but several approximations have been proposed. One of the most well-known approximations is Broadcast Incremental Power (BIP) [Wieselthier et al. 2000], a centralized algorithm which uses a variation of the minimum spanning tree to construct a broadcast tree with an approximation ratio within 12 [Wan et al. 2001]. BIP, and distributed or localized variations of BIP [Ingelrest and Simplot-Ryl 2005; Wieselthier et al. 2002a; Xu et al. 2007; Chronopoulos et al. 2004; Zagalj et al. 2002], provide a good approximation of the minimum-energy broadcast tree; however, they all use location information for determining link cost. Location information may be obtained with a GPS device, or by using a localization algorithm, but these can be expensive in energy cost. In addition, there may be additional energy costs associated with propagating the locality information to neighboring nodes, or a centralized node.

Of greater concern, though, is the unreliability of computing link cost based on distance. It has been shown that link quality is not consistent with distance [Souryal et al. 2006; Erdogan and Hussain 2007], and that signal propagation can vary widely in different environments. Estimating the path loss exponent that should be used to model the environment is difficult, since this property differs among implementations and can vary among links within the same network. If the degree of path loss is underestimated, then the calculated transmit powers will be insufficient, and the broadcast tree will become disconnected and incomplete. If the degree of path loss is overestimated, the calculated transmit powers will be greater than necessary, resulting in reduced energy efficiency and possibly contention.

Distributed implementations of BIP have difficulty approximating the centralized broadcast tree because the nodes constructing local broadcast trees have limited knowledge of alternative paths to their neighbors. This can result in redundant or inefficient selections, since a node which may designate itself as a parent is out of immediate range of another potential parent of the child node, and it is therefore unaware of its actions.

Dynamic Broadcast Incremental Power (DynaBIP) is a new distributed approach to this problem which addresses these issues. It performs a distributed construction of a broadcast tree using received signal strength measurements to estimate link costs. This allows each node to adapt individually to the signal propagation properties of each link. The link cost estimations are thereby more accurate than can be achieved by distance-based link cost calculations, and adaptable to changes. Construction of the tree is managed by the child nodes, instead of the parent nodes, to provide a more complete vision of the available routes. Since child nodes are aware of all potential routes to themselves, they

can determine the optimal path better than a parent node working with partial information.

BIP constructs a broadcast tree by selecting the link which provides the minimum incremental cost at each iteration. For each step, the node and link which provide the smallest incremental cost among the remaining nodes are added to the tree. The best method of duplicating this process is to approximate the order in which nodes are added to the tree. Without global information, there is no way for nodes to determine which node should be selected for the current iteration. With the use of carefully selected delays based upon overheard signals, though, nodes may be able to approximate the order in which they should be included in the tree. DynaBIP uses the observed cost of each link to determine in what order it should add nodes to approximate BIP. The remainder of this section will describe the construction algorithm for DynaBIP. Following is a list of terms that describe values or data structures needed by each node. RSS refers to the received signal strength, or the power measured at the receiving node's radio upon reception. $P_i$ is the initial transmit power used by node $i$ in the construction phase. This may be assumed to be the maximum transmit power, but alternative options will be discussed later in this section. $C_{ij}$ is the cost of link $i \rightarrow j$, or the transmit power needed by $i$ to reach $j$. $T_i$ is the transmit power for node $i$ as determined by the DynaBIP algorithm. $R_{Thresh}$ is the receiving threshold necessary to ensure reception. The *Link_Cost* table is a table of all $C_{ij}$ values, stored on node $j$ and indexed by $i$. The *Transmit_Power* table is a table of $T_i$ values, indexed by $i$. Each node will also maintain its parent node, a list of its children, and a *Route_Cache* that is indexed by the last hop in the route.

### 6.1 DynaBIP Tree Construction Algorithm

The tree construction of DynaBIP can be completed with a single sweeping flood across the network. The source node will initiate the tree construction by broadcasting the first packet. It will include in the header of the packet its transmit power, $P_{src}$ and the route to this node (a null array for the source). Each node that receives this broadcast will compare the RSS to the transmit power, $P_{src}$, to determine the path loss for this link. The path loss for a link $i \rightarrow j$ may be depicted as

$$PathLoss_{ij} = P_i - RSS. \tag{1}$$

Adding this value to the radio receiving threshold will provide the transmit power necessary to maintain this link. The link cost can then be represented as

$$C_{ij} = R_{Thresh} + PathLoss_{ij}. \tag{2}$$

$C_{ij}$ will be saved in the *Link_Cost* table. Each node will store the route in the header in the *Route_Cache*, indexed by the sending node's ID, in this case the source's ID. It will then schedule a rebroadcast of this packet at a delay proportional to $C_{ij}$ in milliWatts. If it receives another broadcast of this packet from a closer node (a lower-cost link), it will cancel the previously scheduled

rebroadcast, and schedule a new rebroadcast with a delay proportional to the new link cost. The formula for the delay is

$$ForwardingDelay = \beta \times C_{ij}[mW], \tag{3}$$

where $\beta$ is the delay multiplier. There is a tradeoff between contention and latency of tree construction for the multiplier. In our trials, we found that $\beta$ values between the range 10 to 100 ms/mW provided a good balance between contention and latency. If a collision does occur due to closely scheduled rebroadcasts, a resend will be scheduled at a random backoff. This process is described further in Section 6.4.

The node which has the lowest-cost link from the source will rebroadcast first. It will include in its header its transmit power, $P_i$, the route to this node, which will include only the source node, and the transmit power needed by the source to maintain this link, $C_{si}$. When the source node overhears this broadcast, it will recognize that is has been selected as the parent by the previous node, and will update $T_{src}$ to the link cost provided in the header if $C_{si}$ is greater than the current $T_{src}$. It will also add the previous hop to its list of children, and send an acknowledgment to the new child node. Other nodes which overhear the broadcast will store the link cost to the new node, $C_{ij}$, in their *Link_Cost* table, and cache the new route. They will also update $T_{src}$ in the *Transmit_Power* table to the link cost value in the header, $C_{si}$. If $C_{ij}$ has a lower cost than the previous link to the source, or if this is the first copy of the packet it has received, it will schedule a rebroadcast at a delay proportional to the cost of the new link. This process is then repeated by each node that receives a copy of the broadcast.

When a node reaches its scheduled rebroadcast time, it will perform a calculation to determine which link among the nodes currently in the broadcast tree will provide the minimum incremental cost. It does this by scanning through the routing cache. For each entry in the routing cache, it compares the current transmit power of the last hop to the link cost from the last hop in the route to itself. The minimum difference among these entries is the route that will be selected as the link for this node's addition to the broadcast tree. It will set its parent to the last hop node in this route, and will include this route in the packet header, along with the link cost needed by its parent, $C_{ij}$. Once it rebroadcasts the packet, it will have added itself and the link to its parent to the distribution tree, just as BIP does in each iteration. The algorithm for selecting the new link to add at node j is shown in Algorithm 1, where $r_i$ is an entry in the routing cache with last hop node $i$.

For small sensor motes with limited memory and packet size, the routing cache and tables required by the DynaBIP algorithm may be too robust. In this case, the metadata may be replaced by a struct for each neighbor which includes the cost of the link to the neighbor, and the current transmit power for this neighbor. The route included in the packet header may be replaced simply with the parent node. The remainder of the process is unchanged, since the link cost and current transmit power of neighboring nodes are all that is needed to compute the forwarding delay and the minimum incremental cost link. The complete routing cache is only necessary for avoiding disconnected

**Algorithm 1** Minimum Incremental Cost Link Selection.

```
minCost ← ∞
for all r_i ∈ RouteCache do                    ▷ r_i is entry with last hop node i
    incrementalCost ← C_ij − T_i
    if incrementalCost < minCost then
        minCost ← incrementalCost
        parent ← i
        minRoute ← r_i
    end if
end for
```

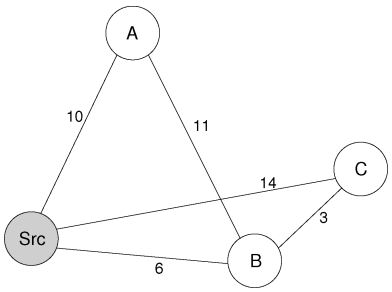



Fig. 1. Example of node selection order of DynaBIP.

loops following sweep operations, which is described in Section 6.3. The number of nodes that metadata is maintained for may be capped to limit the memory space requirements. This limit may be kept relatively low without impacting the full connectivity of the network, since a fully connected network can typically be guaranteed with just a few connections per node, as will be explained further in the next section.

The proportional delay is what allows DynaBIP to emulate the tree construction process of BIP. In each case, the link with the minimum additional energy cost will be the node which is added to the tree next. This concept will continue to work over links of varying hop counts. Consider the topology shown in Figure 1. The source initiates a broadcast at time unit 0. Node B has the lowest link cost, and will schedule a rebroadcast at time unit 6, representing an incremental cost of 6. Node A will schedule its broadcast at time unit 10, and node C at time unit 14. When node B rebroadcasts at time unit 6, node C will cancel its scheduled broadcast at time unit 14, and will reschedule it for time unit 9, since it has a link cost of 3 from node B. Node A will not alter its scheduled rebroadcast time since the link to B is not less than the link to the source. Node C will then be the second node to rebroadcast, with an incremental link cost of 3. Finally, node A will rebroadcast at time unit 10, with an incremental link cost of $C_{SrcA} - C_{SrcB} = 4$. This is the same ordering of adding nodes and links to the broadcast tree as would be observed by BIP. The ordering in which nodes are added to the broadcast tree globally may not be precisely by incremental link cost, but locally it is a close approximation.

### 6.2 Flooding Power Selection

As mentioned in the previous section, the construction of the DynaBIP tree can be completed in a single flood. The cost of tree construction will then be dependent upon the initial transmit power used by the nodes during the flooding phase. The simplest method of implementing the flooding phase of the tree construction is to have each node broadcast at full power. This incurs the highest cost of construction, but requires no information of local topology. As long as the network is fully connected, the broadcast tree will be complete. This method will provide the closest approximation of the BIP broadcast tree for low- to medium-density networks, since all links will be explored. For high-density networks, however, flooding at maximum power can lead to excessive contention and collisions, which could hamper the quality of the broadcast tree and lead to additional construction costs for resends. A lower power may be used for the standard flood method to reduce construction cost and collisions, but without a more intelligent algorithm, full connectivity cannot be guaranteed.

The relative neighborhood graph (RNG) [Toussaint 1980] provides a fully connected graph with a slightly higher level of redundancy than an MST. By setting the transmit power of each node in the flooding phase to the minimum power needed to cover their RNG neighbors, a near minimal power is used to maintain full connectivity. The slightly higher level of redundancy provided by RNG versus MST leads to a fairly high-quality broadcast tree. Though not as optimal as a standard flood in a collision-free environment, it performed very well in comparison to a standard flood in an environment with contention and collisions. Due to the quality of the broadcast tree constructed from this method, and the energy savings in the construction phase, the RNG flood was selected as the method of construction.

The determination of the RNG neighborhood is performed locally. This requires 2-hop neighborhood information at each node. This information may be obtained by including information about each node's immediate neighbors in beacon or "HELLO" messages, similar to methods used by LBIP [Ingelrest and Simplot-Ryl 2005] and Dist-BIP [Wieselthier et al. 2002a]. Instead of including the location of each of their neighbors, however, nodes will include the link cost to each of their neighbors. As with the tree construction algorithm, the construction of the RNG is based upon the energy cost of a link rather than distance.

### 6.3 Sweep Method

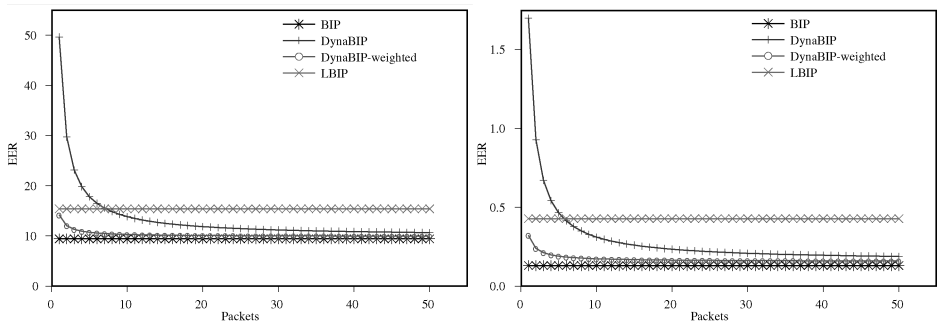
The BIP algorithm includes a method of performing a sweep following the initial construction of the broadcast tree in order to remove redundant links due to the wireless broadcast advantage. The sweep operation may be performed an unlimited number of times, but practice has shown that only miminal gains may be achieved following two iterations [Wieselthier et al. 2000]. To mimic this operation in a distributed fashion, each node will perform a sweep to determine if an alternative parent may be chosen to reduce transmission cost. After determining a parent and rebroadcasting the initial packet, each node

will schedule a sweep operation following a fixed delay period. This delay period will be long enough to allow the initial tree construction process to be completed among its immediate neighborhood. Following this delay, it will perform the same scan of its routing cache as it did during the initial construction phase. If it finds that another potential parent has increased its transmit power to a level high enough to reach itself, it will change its parent to the new candidate and broadcast this selection at a transmit power high enough for the new and prior parent to receive it. Upon receiving this broadcast, the prior parent will remove the sending node from its child list, and adjust its transmit power to the minimum needed to cover its remaining children, if any. The sweep operation is only possible when the implementation includes a routing cache, and the complete route in the headers of the tree construction packets. Without these, it would not be possible to avoid creating disconnected loops when nodes select a new parent, since nodes would not be able to determine if their neighbors were upstream or downstream. While the sweep operation can improve the energy efficiency of the broadcast tree, it may be desirable in some cases to omit this step for simplicity of implementation, or due to resource constraints on the device.

### 6.4 Reliability

DynaBIP is in itself an unreliable broadcast protocol. To support the goal of the Cascade reprogramming application, DynaBIP has been extended with several reliability features. Ensuring that the broadcast tree constructed includes all nodes in the network is an important qualification for our protocol. In order to accomplish this, acknowledgment messages are used. When a node overhears a message that indicates it has been selected as a parent, it will send an acknowledgment packet to the child node to verify that it has added it to its child list. When a node rebroadcasts the initial packet, it will schedule a resend of the packet at a designated backoff time. During this time interval, it will listen for an acknowledgment from its parent node indicating that it has received the packet and added the node as one of its children. If an Ack from the parent is not received during this time, the packet will be resent. Resends will take place after random delays in order to avoid synchronization of contending nodes. This will ensure that all links are added as expected in case of a dropped packet or collision. An implicit acknowledgment process is also used to ensure that all nodes are included in the construction phase. Each node monitors its RNG neighbors for broadcasts. If any RNG neighbor does not rebroadcast the message, then it may assume this neighbor did not receive a copy of the message, and will resend the message to them. This will ensure that all nodes are included in the tree construction.

During the dissemination, piggy-backed Nacks are used to reduce the accumulation of missing segments. When a forwarding node receives a new segment, it will check its bit vector to determine if any segments prior to that segment number are missing. If any are missing, it will include the first missing segment number in the header when it forwards the packet. When a node overhears a data packet broadcast from its child, it will check the missing




(a) Low density (b) High density

Fig. 2. Comparison of total transmission cost for a broadcast session, measured across the number of packets in the session. This graph shows that, as the number of packets in the session increases, the performance of DynaBIP converges to near that of BIP, and exceeds the performance of LBIP.

segment field of the header. If the field is not empty, and it possesses the missing segment, it will broadcast the missing segment again. This process will only help forwarding nodes update missing segments, but those are the most critical, since their losses are propagated to all downstream nodes. Additionally, each node will maintain a timer to monitor activity from their parent. Each node knows the broadcast interval of the dissemination and can expect this to be the average delay between reception of new segments. If no broadcasts are received from the parent over the period of five intervals, the node will increase its link cost for the parent, and send a request to the parent to update its transmit power. The timer will be reset, and this process will continue until packets are successfully received from the parent once again, or the node timeout is reached and the node requests a new parent to service its Nacks.

### 6.5 Performance

The graphs in Figure 2 show how DynaBIP performs in comparison to the centralized protocol BIP [Wieselthier et al. 2000], and a localized implementation of BIP, LBIP [Ingelrest and Simplot-Ryl 2005]. LBIP is a localized broadcast protocol that constructs a local BIP tree at each node based on neighborhood information and information stored in packet headers. The results are from simulations which measured the total energy cost for a transmission session, plotted by the number of packets in the session. Each plot is averaged over 100 iterations, with nodes picked uniformly at random over a square of 1000 m $\times$ 1000 m for each iteration. The average node connectivity is 5 for the low density graph, and 50 for the high-density graph. The results are represented as EER, the Expended Energy Ratio, which is the ratio of energy consumed in comparison to a blind flood. These results show that for session-based broadcast, DynaBIP performs closely to that of the ideal BIP results, while performing favorably in comparison to LBIP. These results assume that packets in the tree

construction phase of DynaBIP will be the same size as the data packets. However, construction phase packets may be much smaller, since the construction phase packets need only include the header information needed for construction. The plot of DynaBIP-weighted shows how DynaBIP would compare in a broadcast session if the size of each chunk were 1 kB.

## 7. VERSION TRANSITION AND NETWORK MAINTENANCE

### 7.1 Version Transition

Since versioning is supported in Cascade, there may be situations where a distributed application is a new version of an existing application which is currently running on the node. In these situations, Cascade should manage the transition to the new version to ensure a smooth and reliable handoff of the functions being performed to maintain network stability and reliability. In order to support this, all applications used on the network should be designed to handle the version transition process we have incorporated.

To manage version transitions, Cascade will first check for an active instance of an application anytime a new distribution is received. If an active instance of the application is found, it will send the application a SIGUSR1 signal. All applications used on the network should capture this signal, and save all necessary intermediate data to a file named *process_name.dat*. This file may be formatted however the application chooses. It will be provided as an input file to the new version of the application, so it may use any format it prefers for the data that it requires to define a stable state. When a signal is received, the application should enter a stable state as soon as is possible, save the necessary intermediate data to the *dat* file, and then terminate. When the versioning manager detects the termination of the process, it will initiate the new version of the application, providing the *dat* file as command line input. The new version of the application will use this file to reestablish the stable state of the previous application and continue execution.

As new versions are loaded onto a network, there may be instances when the network is in a transitional phase, with nodes running different versions of an application. To maintain stable operation, all applications should include the version number in the header of packets. Applications may define how they handle packets received from nodes running an alternate version. If an application is defined to be backwards compatible, it may accept the packet. Otherwise, it may simply drop packets from any node running an alternative version.

### 7.2 Network Maintenance

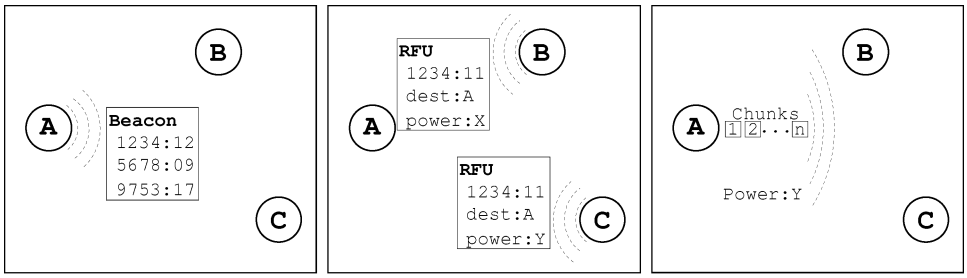
While it is necessary to interact with a sensor network in order to deploy it or to retask or reprogram the network, it is desirable for the network to operate autonomously otherwise. Sensor nodes may be cheap, unreliable, and expendable, so an operator would not want to have to monitor the network for malfunctions on individual nodes. The network should be capable of adapting to any changes

in the network, and still performing the desired task uninterrupted. To do so, each node should be working uniformly, and therefore all nodes must be using the same version of all applications. If nodes become temporarily disconnected, or experience failure during a retasking event, they need to be brought up to date once they reconnect with the network. Similarly, if a set of nodes are added to the network to increase coverage, density, redundancy, etc., the new nodes should be brought quickly up to date with the current application, without manual interaction from the operator.

The autonomous maintenance features of Cascade are enabled by periodic beacon messages, as described in Section 4. Each beacon will include a list of the applications stored on the node, provided as *(applicationID,version)* pairs. If a node hears a beacon from a neighbor with a newer version of an application, it will send a request for the newest version. There may be multiple nodes that require an update from the autonomous maintenance mechanism. To provide the most efficient method of updating multiple nodes, a variation of the DynaBIP tree construction is invoked.

When a node hears a beacon with a newer version available, it first waits for one beacon period to allow all neighbors with the newest version to broadcast their beacon. If it receives a beacon from its minimum-link-cost neighbor with the newest version, it may immediately send a Request for Update (RFU) packet to that node. Otherwise, following a one beacon period delay, it will send a RFU to the node with the minimum link-cost among those with the newest version. The RFU packet will specify which node it is requesting the update from, and what radio transmit power will be needed to reach itself. If a node overhears a RFU during its delay, it will save the transmit power specified for the sending node, and add the requesting node as a potential source. When it calculates which node will provide the minimum link cost for the update, it will consider the incremental cost for all nodes which have been specified as sources.

When a node receives a RFU destined for itself, it first waits for one beacon period for other RFU packets. It then sets its radio transmit power to the maximum specified by all received RFU packets and sends the entire file with short delays between each chunk. This process will result in an update-tree construction similar to the process of the DynaBIP tree construction, thereby providing updates efficiently, even if multiple nodes require the update. The receiving nodes will use a recovery mechanism identical to that in Section 5.2 to ensure the entire file is received. An example of the update process is shown in Figure 3. Here, nodes B and C initially have version 11 of the application with ID 1234. When they receive a beacon from node A which shows the presence of a new version of this application, they will send an RFU message which indicates the requested application, the destination of the request, and the power needed for node A to transmit to them. After receiving the first RFU, node A will wait for one beacon period for other RFU requests. After the delay, it will set its transmit power to the minimum needed to cover all requesting nodes, in this case the power needed to reach node C, and will then broadcast all chunks of the requested new version.




(a) Beacon message (b) RFU message (c) Broadcast file

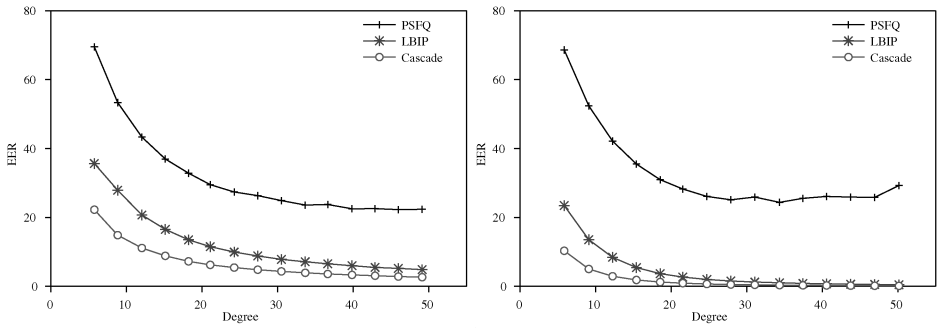
Fig. 3. Sequence of update of nodes to newer available version. Here, node A has version 12 of application ID 1234, while nodes B and C have version 11. When nodes B and C receive a beacon from A with a newer version listed, they will request an update by sending an RFU message.

## 8. RESULTS

Wireless sensor networks can be constructed of wireless devices with a broad range of capabilities and resources. Many wireless sensor nodes have very limited resources, such as the Crossbow Telos motes [Polastre et al. 2005; Sentilla 2009], which have 10 kB of RAM, 1 MB of flash memory, and an 8-MHz microcontroller. More recent sensor motes have exhibited greater processing and memory capacity. The Imote2 [Nachman et al. 2008] uses the same low-power radio as the Telos motes, but has a 416-MHz processor, as well as 256 kB SRAM along with 32 MB SDRAM. Larger gateway devices, such as the Crossbow Stargates [Crossbow Technology 2009], may also be connected with sensor devices and used in a sensor network. These devices have greater storage capacity and more powerful radios. To evaluate Cascade, we will examine its performance on both small-scale and relatively large-scale devices. The Telos motes and Stargate devices will be used for our testbeds.

### 8.1 Simulation

8.1.1 *Simulation Method.* Cascade was implemented in the Jist/Swans simulation environment [Barr and Haas 2004; Barr et al. 2005] to evaluate its performace on sensor devices with greater processing and memory capacity, such as the Crossbow Stargates. Jist/Swans is a scalable wireless ad hoc network simulator based in Java. Pump-Slowly, Fetch-Quickly (PSFQ) [Wan 2005] and LBIP [Ingelrest and Simplot-Ryl 2005] were also implemented for comparison. PSFQ is another reliable transport layer protocol designed to support the dissemination of large binary files for sensor network reprogramming. LBIP is a localized broadcast protocol which modifies radio transmission power based on an incremental power algorithm similar to BIP [Wieselthier et al. 2000]. It does not provide reliability, but will serve as a comparison of another decentralized approach to the minimum energy broadcast problem. Swans provides a full representation of the complete network layer model, with accurate representations of a wireless environment, including path loss, environmental noise and collision interference. Each node in the simulation was implemented with




(a) Path loss exponent: 2 (b) Path loss exponent: 4

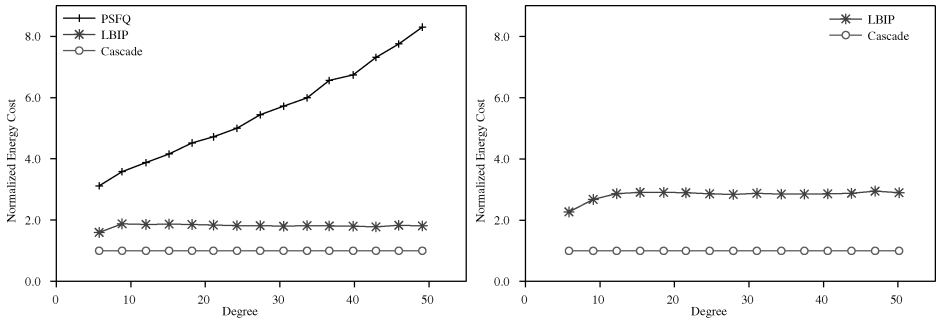
Fig. 4. Comparison of Expended Energy Ratio with no additional loss injected into the simulation.

an 802.11 radio. The environment was modeled using free space path loss with a pathloss exponent of 2 and 4. A fixed field size was used for each simulation, with an increasing number of nodes to provide an increasing density. Each iteration of simulations is performed 100 times, with nodes picked uniformly at random for each run. The results provided are arranged by average node degree. Due to the random placement of nodes in Jist/Swans, the degree of each node can differ drastically from the average node degree. For the densest network, with an average node degree of 50, the individual node degree ranges from 8 to 91. The primary metrics used to evaluate the protocols are total energy cost and latency, where latency is measured as the time needed for all nodes to receive every segment of the data file.

8.1.2 *Simulation Results.* The first group of graphs provide a comparison of the total energy cost for a broadcast session. Cascade is compared to PSFQ and LBIP. The implementation of LBIP uses the NES scheme for RNG neighbors to ensure a fully connected broadcast tree. As in [Ingelrest and Simplot-Ryl 2005], the energy cost will be represented as EER, the Expended Energy Ratio. EER is the ratio of energy consumed by a protocol in comparison to the energy that would have been consumed by a blind flood.

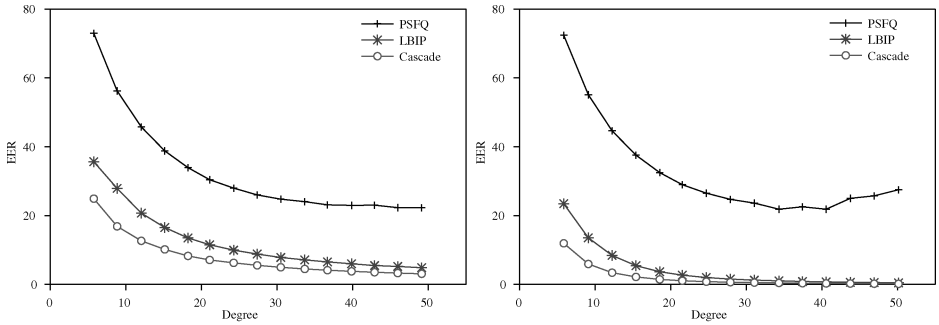
Figure 4 shows the value of EER for the broadcast of a 50 KB binary file, segmented into fifty 1-kB chunks. The results are plotted across an average node degree ranging from 5 to 50, where node degree represents the average connectivity of nodes. Figure 4(a) shows the results when the environment is modeled with a path loss exponent of 2, and Figure 4(b) shows the results for an environment with a path loss exponent of 4. As expected, PSFQ has the highest energy cost, because it does not utilize transmission power control to reduce energy cost. It does use significantly less energy than blind flooding, however, since it uses a counting method to reduce redundant transmissions. LBIP and Cascade each perform significantly better, but Cascade provides the best energy efficiency, despite requiring additional messaging to ensure reliability.

Due to the difference in scale of the values of PSFQ, it is difficult to see the difference in performance between LBIP and Cascade in Figure 4. Figure 5 provides another plot of the EER values normalized to Cascade. The plot of




(a) Path loss exponent: 2 (b) Path loss exponent: 4

Fig. 5. Comparison of Expended Energy Ratio normalized to the energy cost of the Cascade.

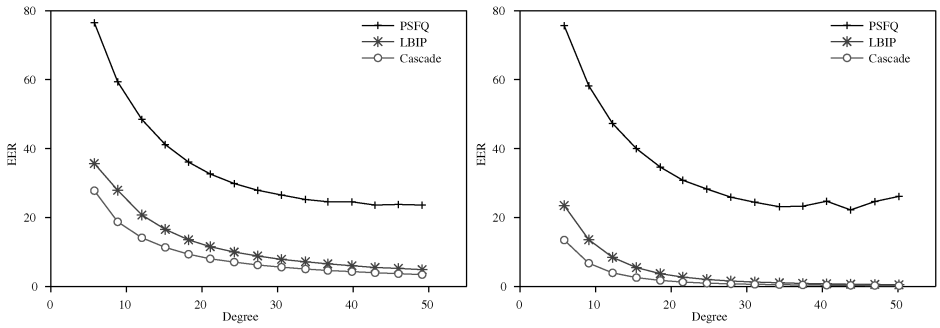



(a) Path loss exponent: 2 (b) Path loss exponent: 4

Fig. 6. Comparison of Expended Energy Ratio with a 5% loss injected into the simulation.

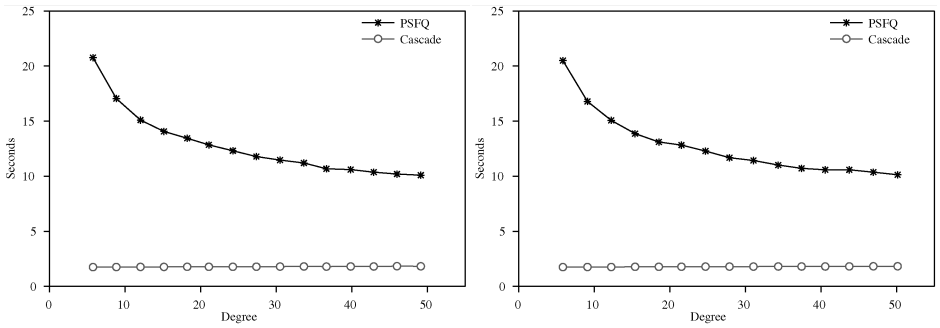
PSFQ is omitted from Figure 5(b) because it quickly scales beyond the range of the graph. It can be more easily seen here that Cascade typically incurs less than half the energy cost of LBIP for the dissemination of this 50-kB file.

The previous simulations factor in packet loss due to congestion and collision. In order to see how additional packet loss that may result from external environmental factors affect the energy cost, we modeled two additional simulations, one with a 5% packet loss, and another with a 10% packet loss. These packet losses are artificially and randomly injected, and are in addition to any losses that may result from congestion or collisions. The plot of the results for a packet loss of 5% are shown in Figure 6 and the results for a packet loss of 10% are shown in Figure 7. The LBIP plots will not increase with the additional packet loss, because it does not provide reliability, and therefore will not require additional control messages or resends due to packet loss. It is only shown here to provide a comparison of how the energy cost of Cascade compares to another distributed approach, even when Cascade requires additional messaging to ensure reliability. It can be seen here that Cascade still provides the lowest total energy cost across a broadcast session, while reliably delivering all packets in a lossy environment.




(a) Path loss exponent: 2 (b) Path loss exponent: 4

Fig. 7. Comparison of Expended Energy Ratio with a 10% loss injected into the simulation.




(a) Path loss exponent: 2 (b) Path loss exponent: 4

Fig. 8. Comparison of average latency between Cascade and PSFQ.

In addition to reliability and energy efficiency, Cascade aims to distribute the binary file to all nodes in the network quickly. We measure latency as the time needed for all nodes to receive the complete file. In Figure 8 we compare the latency of Cascade and PSFQ. LBIP is not included in this measure, because the LBIP protocol does not define any timing metrics. Figure 8 shows that Cascade provides a timely dissemination of binary files to all nodes, which scales well across all node densities. PSFQ is considerably slower for less dense networks, because it has fewer forwarders. In dense networks, the high number of forwarding nodes allows PSFQ to more closely approximate the shortest number of hops between the source and edge nodes. However, even in the extremely dense network, Cascade has a considerably lower latency. This is because PSFQ has a much slower pushing mechanism in order to reduce the number of control messages needed for recovery of dropped packets. It uses an in-order packet forwarding scheme to prevent the propagation of Nack messages to the leaf nodes. Cascade accomplishes a similar effect by delaying all recovery messages until the initial dissemination is complete and handling recovery operations locally. This allows the dissemination phase to go much faster, because it does not need to handle recovery operations between packets.

Table I. Total Energy Cost of Distribution

| Trial | mAs | EER |
|---|---|---|
| 1 | 10.85 | 11.26 |
| 2 | 9.14 | 9.49 |
| 3 | 8.0 | 8.3 |
| 4 | 8.32 | 8.64 |
| 5 | 8.31 | 8.63 |
| Average | 8.92 | 9.27 |

8.1.3 *Stargate Testbed.* To validate the simulation results for a relatively robust sensor node, we implemented Cascade on a small Crossbow Stargate [Crossbow Technology 2009] testbed. The testbed consisted of 12 SPB400CB gateway devices with 802.11b CompactFlash wireless ethernet cards. The wireless cards have a transmit power range of −10 dBM to 13 dBm. The devices were arranged in a two-story building. For each test, a 26-kB file was distributed in 27 chunks. The cost for each link was determined by calculating the necessary transmit power for the link, as described in Section 6.1, and assigning the corresponding cost in amperes.

The total energy cost of each file distribution was computed to compare to the efficiency we observed in our simulations. We logged every transmission, along with the size in bytes of the packet and the transmission power. We used this to compute the total energy cost of the distribution, including the tree construction phase and any recovery messages, in milliAmpere-seconds. We also computed the total energy cost that would have resulted from a blind flood, so that we could compute the EER (Expended Energy Ratio) as we did for the simulations. The energy cost and EER for each test are shown in Table I. The average degree of connectivity of the tests was around 9. Based on the graphs in Figure 4, the EER for degree 9 was about 14.9 for an environment modeled with a path loss exponent of 2, and 5.1 using a path loss exponent of 4. Our observed results from the implementation showed an average EER of 9.27, which falls between this range. This would indicate an environment that could be modeled with path loss exponent near 3, as may be expected from a residential indoor environment.

## 8.2 Tmote Sky Testbed

8.2.1 *Implementation Method.* To evaluate Cascade on more resource-constrain-ed devices, Cascade was implemented on Telos motes using the MoteLab Harvard Sensor Network Testbed [Werner-Allen et al. 2005]. The MoteLab testbed consists of over 100 Tmote Sky sensor motes, which have 10 kB of RAM and a 1-MB flash. They are distributed among a three-story building at Harvard University. The Deluge T2 implementation provided with TinyOS 2.x was also run on the MoteLab testbed for comparison. In each trial, a 50-kB program image was disseminated to 100 motes. The size of the payload for data packets was set to 64 bytes for both Cascade and Deluge, which resulted in 800 packets required for the full file dissemination. For each trial, we recorded the size and transmit power for all transmissions, as well as receptions, and the total number of reads and writes to the flash memory.

Table II. Output Power Configuration for the CC2420

| TXCTRL_LEVEL | Output power [dBm] | Current consumption [mA] |
|---|---|---|
| Transmissions | | |
| 31 | 0 | 17.4 |
| 27 | −1 | 16.5 |
| 23 | −3 | 15.2 |
| 19 | −5 | 13.9 |
| 15 | −7 | 12.5 |
| 11 | −10 | 11.2 |
| 7 | −15 | 9.9 |
| 3 | −25 | 8.5 |
| Receptions | | |
| N/A | N/A | 19.7 |

Since the Telos motes only have 10 kB of available RAM, a page size of 512 bytes was used in Cascade, which allows for a buffer of eight packets. The data buffers were written to the flash in 256-byte blocks, to optimize for the 256-byte page size of the flash [Numonyx 2009]. This minimizes the overhead cost of each flash write. For the Cascade implementation, a receiving threshold of −85 dBm was used in the calculation of Equation (2).

8.2.2 *Implementation Results.* Cascade was evaluated using two primary metrics: energy cost and latency. The energy cost was based on three components: transmissions, receptions, and reads and writes to flash. For the calculation of energy cost from the radio, the current consumption of transmissions and receptions was multiplied by the size of the packet in bytes, divided by the transmit rate of the CC2420 radio (250 kbps) to provide cost in milliAmpere-hours (mAh) [Sentilla 2009]. The power configuration of the CC24220 radio is shown in Table II. It can be easily observed that the energy cost of radio receptions is high, higher than the cost of a transmission at 0 dBm. Since each transmission can result in many receptions, receptions will be the dominant factor in total energy cost.

For reads and writes to flash memory, the energy cost is calculated as the energy consumption for a read or write multiplied by the bytes read/written divided by the rate of the bus interface as provided by Numonyx [2009]. For Deluge, the size of all reads and writes are the packet payload size, 64 bytes. For Cascade, it is assumed for simplicity of calculation that all reads and writes are of 256-byte pages, though some may actually only be a 64-byte block.

Table III provides an overview of the average results from the trials run on the MoteLab testbed for Cascade and Deluge. Figures include a 95% confidence interval listed as a percentage of the mean. As expected, the transmission power control scheme of the Cascade method reduces the average number of receptions per transmission for data packets. The broadcast tree constructed provides coverage of all nodes, while reducing the radio range of sending nodes to reduce the number of redundant receptions. The average completion time is the time needed for the last node to have received the complete program image. The begin time for Cascade is set at the time that the source node sends the first initiation packet. The begin time for Deluge is set when the first request packet

Table III. Averages from Trials on the MoteLab Testbed with 95% Confidence Intervals

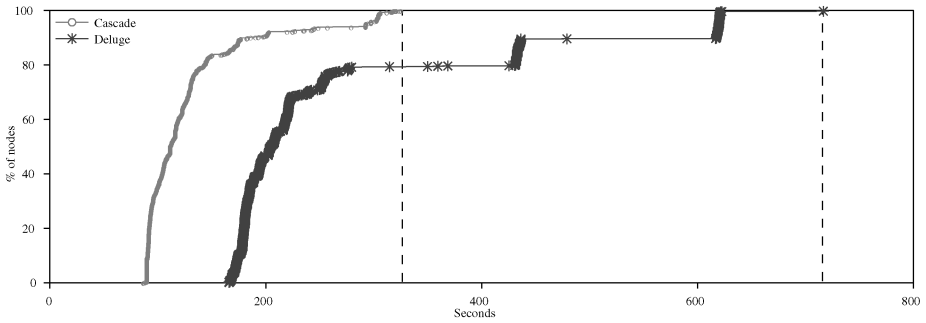
| Value | Deluge | | Cascade | |
|---|---|---|---|---|
| Number of transmissions | 33794 | ±4.8% | 20742 | ±2.9% |
| Number of receptions | 448078 | ±3.8% | 236328 | ±2.1% |
| Receptions per transmission (data packets only) | 16.5 | ±3.4% | 11.5 | ±2.8% |
| Flash reads | 18311 | ±3.3% | 1733 | ±13.7% |
| Flash writes | 80826 | ±0.4% | 33509 | ±12.2% |
| Average completion time [s] | 449.4 | ±23.3% | 251.0 | ±12.8% |
| Cost of transmissions [mAh] | 0.23 | ±3.5% | 0.17 | ±2.4% |
| Cost of receptions [mAh] | 4.05 | ±2.8% | 2.94 | ±2.1% |
| Cost of flash read/writes [mAh] | 0.06 | ±0.5% | 0.09 | ±12.2% |
| Total cost [mAh] | 4.34 | ±2.7% | 3.2 | ±2.0% |

Table IV. Breakdown of Transmissions by Cascade with 95% Confidence Intervals

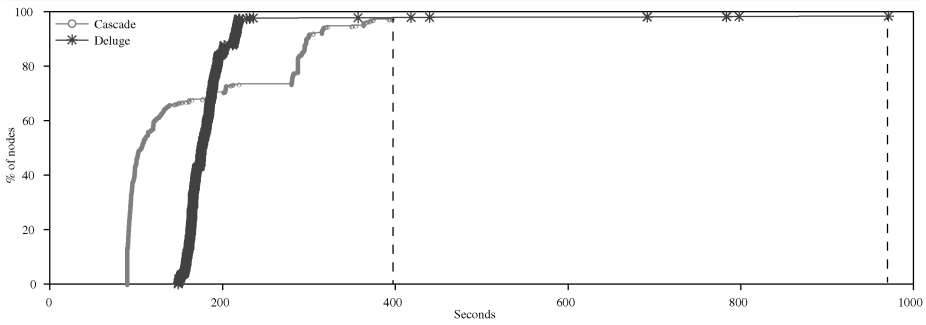
| Type | Transmissions | | % of Total |
|---|---|---|---|
| Init | 184.7 | ±5.3% | 0.89 |
| Ack | 82.8 | ±3.4% | 0.40 |
| Construction total | 267.45 | ±4.2% | 1.29 |
| Data | 18397.8 | ±3.2% | 88.70 |
| Dissemination total | 18397.8 | ±3.2% | 88.70 |
| Nack | 655.9 | ±25.4% | 3.16 |
| Data resends | 1421.2 | ±13.9% | 6.85 |
| Recovery total | 2077.1 | ±14.4% | 10.01 |
| Total | 20742.35 | | |

is sent. On average, the Cascade trials completed close to 200 s faster than in Deluge. This can be attributed to the more pipelined scheme of Cascade, which allows nodes to immediately forward received chunks. Variation for completion time was significant, particularly for Deluge, but the difference exceeds the sum of confidence intervals. While Deluge exhibited some savings in the energy cost of flash accesses, and Cascade exhibited savings in transmission costs, the dominating energy cost factor was radio receptions. The reduced radio range of Cascade transmissions helped reduce unnecessary receptions, which provided a 26% total energy reduction.

In the Cascade trials, the average number of nodes selected as parents was 27.25, with an average broadcast power of −9.5 dBm. The average number of RNG neighbors for each node was 3.13, with an average transmission power of −13.5 dBm needed to reach all RNG neighbors. The transmission values in Table III represent an average of the total number of transmissions made during the trials. A breakdown of the transmissions is provided in Table IV, again with 95% confidence intervals. The tree construction of Cascade is accomplished by the Init and Ack messages. It can be seen here that these accounted for just 1.29% of total transmissions. Therefore, the overhead cost of tree construction was easily absorbed by the reduction in energy cost provided by an efficient broadcast tree. The recovery packets included Nack messages and data packets that were sent in response to Nack messages. These messages exhibited the highest variance in the trials, since they are the most dependent on variations in the wireless environment and topology between trials. The




(a) Source node 22




(b) Source node 163

Fig. 9. Percentage of motes which have received the complete program image, plotted by time in seconds.

recovery messages accounted for just over 10% of total transmissions, leaving data packets to account for nearly 89% of all transmissions.

Latency is the second metric which was evaluated. We define latency as the time needed for all nodes in the network to have received the complete program image. The begin time for Cascade is set at the time that the source node sends the first initiation packet. The begin time for Deluge is set when the first request packet is sent. The average latency of the trials was provided in Table III. This shows that on average, for this testbed of 100 motes, the completion time for Cascade was 251 seconds, compared to 449 seconds for Deluge. In Figure 9, we plot the percentage of the network that has completed by time. Two figures are provided to show how the latency characteristics differed for two source nodes used in the trials. Figure 9(a) provides a plot of the average completion times for nodes in trials with source node 22, and Figure 9(b) shows the results for trials with source node 163. Figure 9(a) shows that 85% of motes in the Cascade trials received the complete image prior to the first completion in Deluge. The aggressive dissemination scheme of Cascade, where all segments of a program image are pushed out consecutively, rather than segmenting the dissemination in pages, allows Cascade the potential to complete quickly. Cascade also has fewer trailing nodes which extend the

total latency. The last completion for Cascade in Figure 9(a) took just 325 s, compared to the last completion for Deluge, which occurred at 716 s. In Figure 9(b), just over 65% of motes in the Cascade trials received the complete image prior to the first completion in Deluge. However, with this source node, 95% of nodes in the Deluge trials completed in just 217 s, whereas Cascade needed 350 s to reach 95% completion. We believe this is where Deluge benefits from the redundancy offered by multiple potential sources for each node. If two partitions of a network are connected only by a series of weak links, Deluge may acquire pages from multiple sources across the partition and once one node has received a full page it may become a source for other nodes within the same partition. For Cascade, nodes will attempt to resolve missing segments with their parent, which may require repeated requests if the link is weak. Despite this, Cascade still exhibited a shorter maximum latency, as 100% of nodes completed in under 400 s, compared to 970 s for Deluge.

## 9. CONCLUSIONS AND FUTURE WORK

Simple, reliable, and efficient tools for reprogramming sensor networks will play an important role in the greater adoption of sensor networks. Providing such functionality for a large set of wireless devices, which have limited available resources, is not trivial. We have presented here a framework for reprogramming sensor networks which is reliable and energy-efficient. It also provides low latency in comparison to alternative reliable transport protocols. We have verified the performance of these tools by simulation and implementation. We have also provided a toolset for maintaining a network autonomously. This will allow a network to be maintained with the newest software versions with little involvement from the administrator, including when adding new nodes to the network.

In future work, we will further explore how this framework can be extended to handling subgroup functionality. There may be some sensor networks which assign nodes to subgroups within the network, each having their own set of functions. In such a scenario, a distribution of a binary file may only need to go to a subset of nodes. It should be explored how this impacts the selection of which nodes will be included in the broadcast tree construction, and whether nodes that are not part of the subgroup should be involved in forwarding the packets.

## REFERENCES

BARR, R. AND HAAS, Z. 2004. `http://jist.ece.cornell.edu/`.

BARR, R., HAAS, Z. J., AND VAN RENESSE, R. 2005. Jist: an efficient approach to simulation using virtual machines: Research articles. *Softw.—Pract. Exper. 35*, 6, 539–576.

BIAN, F., GOEL, A., RAGHAVENDRA, C. S., AND LI, X. 2002. Energy-efficient broadcasting in wireless ad hoc networks lower bounds and algorithms. *J. Interconnect. Netw. 3*, 3-4, 149–166.

BUSNEL, Y., BERTIER, M., FLEURY, E., AND KERMARREC, A.-M. 2007. Gcp: Gossip-based code propagation for large-scale mobile wireless sensor networks. In *Autonomics* (2008-05-13), F. Davide, Ed. ACM International Conference Proceeding Series, vol. 302. ACM Press, New York, NY, 11.

CHENG, M. X., SUN, J., MIN, M., AND DU, D.-Z. 2003. Energy-efficient broadcast and multicast routing in ad hoc wireless networks. In *Proceedings of the IEEE International Performance, Computing, and Communications Conference*. 87–94.

CHRONOPOULOS, A., COTAE, P., AND PONIPIREDDY, S. 2004. Efficient power control for broadcast in wireless communication systems. In *Proceedings of the Wireless Communications and Networking Conference (WCNC)*. Vol. 3, 1330–1334.

Crossbow Technology. 2009. `http://www.xbow.com/Products/wproductsoverview.aspx`.

ERDOGAN, S. Z. AND HUSSAIN, S. 2007. Using received signal strength variation for energy efficient data dissemination in wireless sensor networks. In *Proceedings of the 18th International Conference on Database and Expert Applications (DEXA)*. IEEE Computer Society Press, Los, Alamitos, CA, 620–624.

HO, C., OBRACZKA, K., TSUDIK, G., AND VISWANATH, K. 1999. Flooding for reliable multicast in multihop ad hoc networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*. 64–71.

HSU, C.-S., TSENG, Y.-C., AND SHEU, J.-P. 2007. An efficient reliable broadcasting protocol for wireless mobile ad hoc networks. *Ad Hoc Netw. 5*, 3, 299–312.

HUI, J. W. AND CULLER, D. E. 2004. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*. J. A. Stankovic, A. Arora, and R. Govindan, Eds. ACM Pree, New York, NY, 81–94.

INGELREST, F. AND SIMPLOT-RYL, D. 2005. Localized broadcast incremental power protocol for wireless ad hoc networks. In *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC)*. IEEE Computer Society Press, Los, Alamitos, CA, 28–33.

KULKARNI, S. S. AND ARUMUGAM, M. 2006. Infuse: A TDMA based data dissemination protocol for sensor networks. *Int. J. Distrib. Sensor Netw. 2*, 1, 55–78.

KULKARNI, S. S. AND WANG, L. 2005. MNP: Multihop network reprogramming service for sensor networks. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, Los Alamitos, CA, 7–16.

LEE, S.-J., SU, W., AND GERLA, M. 2002. On-demand multicast routing protocol in multihop wireless mobile networks. *Mobile Netw. Appl. 7*, 6, 441–453.

LEVIS, P., PATEL, N., CULLER, D., AND SHENKER, S. 2004. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st Symposium on Network Systems Design and Implementation*.

LIANG, W. 2002. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *Proceedings of the 3rd International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*. ACM Press, New York, NY, 112–122.

LUN, D., RATNAKAR, N., KOETTER, R., MEDARD, M., AHMED, E., AND LEE, H. 2005. Achieving minimum-cost multicast: A decentralized approach based on network coding. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. Vol. 3. 1607–1617.

MILLER, C. AND POELLABAUER, C. 2008. Paler: A reliable transport protocol for code distribution in large sensor networks. In *Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. IEEE Computer Society Press, Los, Alamitos, CA, 206–214.

MILLER, C. AND POELLABAUER, C. 2009. A decentralized approach to minimum-energy broadcasting in static ad hoc networks. In *Proceedings of the 8th International Conference on Ad Hoc Networks and Wireless (Adhoc Now)*.

NACHMAN, L., HUANG, J., SHAHABDEEN, J., ADLER, R., AND KLING, R. 2008. Imote2: Serious computation at the edge. In *Proceedings of the 5th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 1118–1123.

NAIK, V., ARORA, A., SINHA, P., AND ZHANG, H. 2007. Sprinkler: A reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices. *IEEE Trans. Mobile Comput. 6*, 7, 777–789.

NUMONYX. 2009. M25p80: 8 mbit, low voltage, serial flash memory with 75 mhz SPI bus interface. Data sheet. Numonyx; Palle, Switzerland.

PAGANI, E. AND ROSSI, G. P. 1997. Reliable broadcast in mobile multihop packet networks. In *Proceedings of the 3rd annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*. ACM Press, New York, NY, 34–42.

PARK, S.-J., VEDANTHAM, R., SIVAKUMAR, R., AND AKYILDIZ, I. F. 2004. A scalable approach for reliable downstream data delivery in wireless sensor networks. In *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, J. Murai, C. E. Perkins, and L. Tassiulas, Eds. ACM Press, New York, NY, 78–89.

POLASTRE, J., SZEWCZYK, R., AND CULLER, D. E. 2005. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*. IEEE Computer Society Press, Los Alamitos, CA, 364–369.

RAHNAVARD, N. AND FEKRI, F. 2006. Crbcast: a collaborative rateless scheme for reliable and energy-efficient broadcasting in wireless sensor networks. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*. 276–283.

RIZZO, L. AND VICISANO, L. 1997. A reliable multicast data distribution protocol based on software FEC techniques. In *Proceedings of the 4th IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS)*.

SENTILLA. 2009. Tmote sky datasheet.
`http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf`.

SOURYAL, M., KLEIN-BERNDT, L., MILLER, L., AND MOAYERI, N. 2006. Link assessment in an indoor 802.11 network. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*. Vol. 3. 1402–1407.

STATHOPOULOS, T., HEIDEMANN, J., AND ESTRIN, D. 2003. A remote code update mechanism for wireless sensor networks. Tech. rep., UCLA, Los Angeles, CA.

STOJMENOVIC, I., SEDDIGH, M., AND ZUNIC, J. 2002. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Trans. Parall. Distrib. Syst. 13*, 1, 14–25.

TOUSSAINT, G. T. 1980. The relative neighbourhood graph of a finite planar set. *Patt. Recog. 12*, 4, 261–268.

TSENG, Y.-C., NI, S.-Y., CHEN, Y.-S., AND SHEU, J.-P. 2002. The broadcast storm problem in a mobile ad hoc network. *Wirel. Netw. 8*, 2/3, 153–167.

WAN, C.-Y. C. A. K. L. 2005. Pump-slowly, fetch-quickly (PSFG): A reliable transport protocol for sensor networks. *IEEE J. Select. Areas Comm. 23*, 4, 862–872.

WAN, P.-J., CALINESCU, G., LI, X., AND FRIEDER, O. 2001. Minimum-energy broadcast routing in static ad hoc wireless networks. In *Proceedings of the 20th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. 1162–1171.

WERNER-ALLEN, G., SWIESKOWSKI, P., AND WELSH, M. 2005. Motelab: A wireless sensor network testbed. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks* (*IPSN*). IEEE Computer Society Press, Los Alamitos, CA, 68.

WIESELTHIER, J., NGUYEN, G., AND EPHREMIDES, A. 2002a. Distributed algorithms for energy-efficient broadcasting in ad hoc networks. In *Proceedings of the Military Communications Conference* (*MILCOM*). Vol. 2. 820–825.

WIESELTHIER, J. E., NGUYEN, G. D., AND EPHREMIDES, A. 2000. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proceedings of the 19th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. 585–594.

WIESELTHIER, J. E., NGUYEN, G. D., AND EPHREMIDES, A. 2002b. Energy-efficient broadcast and multicast trees in wireless networks. *Mobile Netw. Appl. 7*, 6, 481–492.

WU, J. AND DAI, F. 2003. Broadcasting in ad hoc networks based on self-pruning. *Int. J. Foundat. Comput. Sci. 14*, 2, 201–221.

WU, J. AND LI, H. 2001. A dominating-set-based routing scheme in ad hoc wireless networks. *Telecomm. Syst. 18*, 1–3, 13–36.

XU, H., D'AURIOL, B. J., CHO, J., LEE, S., AND JEONG, B.-S. 2007. A generic localized broadcast framework in mobile ad hoc ubiquitous sensor networks. *In IEICE Trans. Comm. 90-B*, 12, 3434–3444.

YU, Y., RITTLE, L. J., BHANDARI, V., AND LEBRUN, J. B. 2006. Supporting concurrent applications in wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM Press, New York, NY, 139–152.

ZAGALJ, M., HUBAUX, J.-P., AND ENZ, C. C. 2002. Minimum-energy broadcast in all-wireless networks: : NP-completeness and distribution issues. In *Proceedings of the 8th International Conference on Mobile Computing and Networking (MobiCom)*. ACM Press, New York, NY, 172–182.

Received March 2009; revised August 2009, December 2009; accepted December 2009