

Deluge: Data Dissemination for Network Reprogramming at Scale *

Adam Chlipala, Jonathan Hui, Gilman Tolle[†]
University of California at Berkeley
Computer Science Division
Berkeley, CA 94720
{adamc,jwhui,get}@cs.berkeley.edu

ABSTRACT

In this paper, we present *Deluge*, a reliable data dissemination protocol for propagating large amounts of data (i.e. more than can fit in RAM) from one or more source nodes to all other nodes over a multihop, wireless sensor network. To achieve robustness to lossy communication and node failures, we adopt an epidemic approach. Representing the data object as a set of fixed-sized pages provides a manageable unit of transfer which supports spatial multiplexing and provisions for incremental upgrades. Due to the large data size, we identify a set of possible optimizations and evaluate their effectiveness.

We demonstrate that the Deluge algorithm reliably distributes data across an increasingly sized multi-hop network while maintaining a constant amount of local state. We also demonstrate that the energy required to distribute this data is within the allowable per-mote energy budget. Future directions include new methods for contention management and more effective power scheduling.

Keywords

Sensor Networks, Data Dissemination, Network Reprogramming

1. INTRODUCTION

Wireless sensor networks show great promise in their ability to provide extensive, unobtrusive monitoring for extended periods of time. These networks are composed of large numbers of small, inexpensive nodes that integrate sensing, computation, and wireless communication. In general, sensor nodes have multiple resource constraints including limited memory and computational power, low bandwidth communication, and scarce energy resources.

One of the proclaimed advantages of sensor networks is their ability to operate for extended periods of time without physical intervention by humans. For example, sensor networks may be used in remote or hostile locations too dangerous for humans to enter. In environmental applications, sensor networks can be used where the mere presence of humans during the study may disturb results. The very nature of these applications makes physical interaction with nodes for maintenance purposes unacceptable. Furthermore, the sheer number of nodes can make physical maintenance a

daunting task. Nevertheless, users must be able to add or change the functionality of a deployed network to fully utilize its capabilities.

It is clear that network reprogramming is required for the success of wireless sensor networks. In many cases, complete knowledge of an environment is not known and makes predicting what actions to perform and when to perform them a difficult, if not impossible, task when developing a sensor network application. Additionally, requirements and environments may evolve over time, making the ability to add or change functionality of a deployed network imperative.

Developers face a more immediate problem. As sensor network research matures, the number of nodes used in test beds and deployments continues to grow. Test beds sized at tens of thousands of nodes are now on the horizon, making manual reprogramming of nodes no longer sufficient to provide the productivity required by developers. Instead, they are faced with a problem where the network needs to provide greater support for debugging and testing by automating the reprogramming of nodes. To support network reprogramming, a protocol for the reliable distribution of a program image to nodes is required. In general, the program image is too large to fit in RAM and is much larger than existing protocols are designed to support. A typical sensor node may only have 4 kilobytes of RAM while program images may reach sizes up to 128 kilobytes.

While wireless sensor networks have attracted increasing research attention, protocols designed for wireless sensor networks have generally focused on the processing and communication of relatively *small* data objects, though not without reason. In general, data generated by individual nodes, such as temperature values, usually have representation sizes on the order of bytes. Additionally, the resource limitations of nodes have encouraged greater focus on designing for small data objects. For example, the low bandwidth of the radio has kept communication packets small with a typical size around tens of bytes. Small packets, combined with limited memory and computation capabilities, have provided little reason to consider *large* data objects on the order of kilobytes.

With the pressing need for network reprogramming, the design of protocols to support large data objects can no longer be ignored. We consider the problem of reliably disseminating large amounts of data to many nodes in a wireless sensor network. Because of the resource constrained nature of nodes, an effective algorithm should attempt to minimize the amount of energy required. Additionally, the

*CS262/CS294-1, Fall 2003 Class Project

[†]Authors are listed in alphabetical order.

length of time required to distribute the data should be minimized. While time may not be important for many deployments, it is useful for the purposes of debugging and testing during development. Sufficient support for incremental upgrades should also be provided since program data often evolves slowly.

In this paper, we present *Deluge*, a reliable data dissemination protocol for propagating large amounts of data (i.e. more than can fit in RAM) from one or more source nodes to all other nodes over a multihop, wireless sensor network. To achieve robustness to lossy communication and node failures, we adopt an epidemic approach. Representing the data object as a set of fixed-sized pages provides a manageable unit of transfer which supports spatial multiplexing and provisions for incremental upgrades. In Section 2 of this paper, we review related work. In Section 3 we formally define the problem we address, state our assumptions, and identify metrics for evaluation. We describe the basic protocol in Section 4 and identify a set of possible optimizations in Section 5. We provide simulation results and evaluate the basic protocol along with our proposed optimizations in Section 6, list future work in Section 7, and conclude with Section 8.

2. RELATED WORK

The problem we address can be considered a special case of reliable data dissemination. The main difference is the need to distribute relatively large amounts of data to many nodes. We build on a body of work on both wired and wireless networks.

2.1 Dissemination Protocols

A naive approach to data distribution might include the simple retransmission of broadcasts. However, early work in wireless networks showed that simple retransmission of broadcasts leads to the *broadcast storm problem* [16], where redundancy, contention, and collision impair the ability to perform well. The authors discuss the need to have a controlled retransmission scheme and propose several schemes, such as probabilistic and location-based methods. While these methods attempt to minimize the number of retransmissions, they only make a best-effort attempt for distributing data. In contrast, network reprogramming requires all data to reach the intended nodes.

The experiments conducted by Ganesan et al. [6] provide greater evidence for the need to have controlled retransmission schemes. More importantly, the results show even greater complexities caused by the wireless medium. The authors identify several interesting effects at the link-layer, notably highly irregular packet reception contours and the likeliness of asymmetric links. Because of these effects, it is important to design protocols which are self-organizing and robust to widely varying connectivity scenarios.

Demers et al. [5] have proposed an epidemic algorithm for managing replicated databases. This work showed that epidemic approaches are robust to unpredictable communication failures. Additionally, the authors note that keeping interactions between nodes and decisions based on those interactions strictly local makes algorithms straightforward to implement correctly. However, because this work is based on traditional wired network models, the algorithm makes sole use of unicast links. For efficiency, dissemination protocols for wireless networks should make use of the broadcast

medium when possible.

Deluge is similar to SPIN-RL [12] in that it makes use of a three-stage (advertisement-request-data) handshaking protocol. SPIN-RL is designed for broadcast network models and provides reliability in lossy networks by allowing nodes to make additional requests for lost data. Trickle [14] builds upon this approach by proposing suppression mechanisms for control and data messages, and methods for periodically broadcasting advertisements to increase reliability. However, because these approaches are targeted at the dissemination of small data objects, various optimizations required for efficiently supporting large data objects are not considered. For example, methods for spatial multiplexing are not considered because the latency for transferring small data objects between nodes is low. Other examples include intelligent sender selection and the possibility of forward-error correction.

Other dissemination protocols use less ad-hoc approaches by relying on an underlying topology. A reliable broadcast protocol which makes use of a clustered topology has been proposed [17]. Other approaches include the construction of minimum connected dominating sets [4, 22]. These approaches are advantageous in that fewer control messages are required for eliminating redundancies when actively disseminating the data. However, the construction of such topologies can be complex and makes protocols based on these topologies relatively inflexible to variations in connectivity.

2.2 Reliable Data Transfer

To support network reprogramming, *reliably* communicating data to all sinks is required. Low bandwidth and high loss rates common to wireless sensor networks force the use of different solutions than those used for traditional wired network models. *Pump Slowly, Fetch Quickly (PSFQ)* [21] is a reliable transport protocol design for wireless sensor networks. The protocol distributes data from a source node at a relatively slow data rate (pump slowly) but allows nodes to request missing packets from neighboring nodes aggressively (fetch quickly). The NACK-based approach also batches all requests for lost packets whenever possible. The authors also indicate the importance of hop-by-hop error recovery where loss detection and recovery should be limited to a small number of hops (ideally one hop). In lossy network models, the probability of losing packets accumulates exponentially with the number of hops, requiring exponentially increasing work.

Reliable Multi-Segment Transport (RMST) [19] is a reliable transport layer designed to complement Directed Diffusion [9] in wireless sensor networks. RMST provides reliable transmission of data from a source to all subscribed nodes (sinks). The repair mechanism is NACK-based, where the sinks unicast requests for dropped packets back to the source. Support for caching allows intermediate nodes to intercept and service the requests, allowing for faster repairs.

Scalable Reliable Multicast (SRM) [11] is a reliable multicast mechanism built for wired network models. The key contribution of SRM is the various suppression techniques it uses to minimize network congestion and request implosion at the server. In this paper, no methods for suppressing control messages are evaluated but should be considered in future work.

A *digital fountain* approach to distributing data has also been proposed [2]. It departs from the standard ACK/NACK-

based approaches by using forward error-correction to provide reliability. The basic principle behind the use of forward error-correction is to encode information by adding additional redundant data such that a receiver can reconstruct the original data from any k -sized subset of the encoded data. This approach addresses a problem known as feedback implosion, where requests for lost packets can overwhelm the network or even the server. With forward error-correction, nodes are less likely to make requests for missing packets. Originally designed for point-to-point networks, such approaches may be useful and should be evaluated on wireless sensor networks where low bandwidth, single channel radios with high loss rates are common.

2.3 Network Reprogramming

Research activity directed at network reprogramming for sensor networks has been limited. Recently, TinyOS [8] has included limited support for network reprogramming for the Mica2 [3] via *XNP* [10] with the release of version 1.0. XNP only provides a single-hop solution, requiring all nodes to be within bidirectional communication range of the source. To distribute the program image, the source broadcasts the entire image then queries nodes for packets which were lost. Nodes make requests if necessary. The source then rebroadcasts every packet requested by some node, allowing other nodes to snoop and fill their own missing slots.

A more comprehensive approach to network reprogramming is presented with *Multihop Over-the-Air Programming (MOAP)* [20]. As the title suggests, MOAP supports the distribution of the program image over a multihop network. In MOAP, the program image is divided up into segments no larger than a communication packet. Nodes with a new program image advertise and other nodes make requests as necessary. Nodes make requests to specific nodes to help minimize the number of senders. A NACK-based approach for requesting dropped packets is used to minimize state on the sender. To efficiently manage the large bitmap required for keeping track of which segments have been received, a sliding window approach is proposed.

While MOAP advances the data dissemination problem, it still ignores many design decisions. MOAP requires nodes to receive the entire code image before making advertisements. It does not allow the use of spatial multiplexing to leverage the full capabilities of the network. Methods for intelligent sender selection are not considered. While the authors mention the possibility of using forward error-correction no evaluation is provided to show its usefulness.

A *difference-based* [18] approach to reprogramming is also proposed. This code distribution scheme attempts to save energy by only sending the changes to the currently running code. Using optimizations such address shifts, padding, and address patching, code update information can be minimized. While a method for efficiently distributing code update information is not discussed, such difference-based methods are orthogonal and complement data dissemination protocols.

3. PROBLEM STATEMENT

We address the problem of reliably disseminating a large data object originating from one or more source nodes to all other nodes in a connected, wireless sensor network. Specifically, we consider data objects which can be many times larger than the amount of RAM available. Our notion of

reliability is strict in that *all* of the data must be received in its entirety.

3.1 Assumptions

We make few assumptions about the topology of the networks. For networks where nodes are static, we simply assume that it is connected. In networks composed of mobile nodes, we only require that one or more nodes from each partition periodically migrate to other partitions such that it is possible for data from one partition to eventually reach all partitions.

We currently only consider the reliable dissemination of one type of data object. In the context of network reprogramming, this implies that all nodes run the same type of program and it is not possible to selectively transmit program data to a subset of nodes. To support our reliability claims, we assume that received packets passed up by the network stack are not corrupted. Currently, TinyOS provides SECDED encoding and CRC's on communication packets, making corrupted packets passed to the application highly unlikely.

Finally, we assume that an attempt to disseminate a new version of a data object only happens after a vast majority of nodes have already received the current version. We note that this assumption is not necessary for the correctness of our algorithm. However, it is necessary for our methods of achieving high performance when considering incremental upgrades.

3.2 Evaluation Metrics

Wireless sensor nodes are strictly limited in their energy capacity. To maximize the lifetime of a sensor network, it is necessary to minimize the energy usage of operations on individual nodes. We consider total energy consumed by all nodes in the sensor network as a primary metric for evaluating data dissemination protocols.

We choose Mica as a representative platform on which to base our energy model. On the Mica, as is typical with other sensor nodes, the components that consume the most energy include the processor, radio transceiver, sensors and external flash storage (EEPROM). For the purposes of data dissemination, we do not consider the energy consumed by sensors. To evaluate the amount of energy consumed, we count the number of operations carried out by all nodes in the network and calculate the total energy required for completing those operations. The operations we consider include the number of messages sent and received by the radio transceiver, the number of reads and writes executed by the EEPROM, and the amount of time spent in the idle-listening state from the initiation of data transfer until all intended nodes have completely received the data. Empirical data for the amount of energy consumed by each of these operations are presented in [15] and are given in Table 1.

Operation	nAh
Receive a Packet	8.000
Transmit a Packet	20.000
EEPROM Read 16-bytes	1.111
EEPROM Write 16-bytes	83.333
Idle-Listen for 1 millisecond	1.250

Table 1: Energy required by selected Mica operations.

To put these energy numbers into perspective, a conservative estimate for the amount of energy a pair of AA batteries can provide at 3 volts is 2200 mAh. Given a nine month deployment, a node can afford to consume 6.9 mAh per day. For more details, refer to [15].

Because our primary motivation for this work is to support network reprogramming, we also consider time-to-completion as another primary metric. As discussed in Section 1, network reprogramming is important to assist in the development process. Developers should be able to quickly install updates or make changes when debugging or testing. However, as discussed in [14, 20], time-to-completion for many protocols are often at odds with energy consumption. A common hypothesis for this behavior is that the wireless, broadcast medium often saturates when pushing data at higher speeds, causing an increase in the number of dropped packets. It is not clear that this hypothesis holds when dealing with large amounts of data. Transmitting data at a low rate will significantly increase the time-to-completion and consequently the idle-listening time, which also wastes energy.

Another important metric to consider is the amount of RAM required to support data dissemination. Sensor nodes are severely constrained in the amount of memory available to running applications. The lack of dynamic memory allocation in TinyOS enforces strict limits on the amount of memory which can be statically allocated to support data dissemination. A minimal amount of memory should be reserved to provide the primary application more freedom for complex operations.

4. DELUGE: BASIC PROTOCOL

4.1 Names and Conventions

Each successive update to the overall data image managed by Deluge is given a greater version number. Each node knows which version it currently possesses, and a node compares version numbers in any situation where it must determine whether it should be requesting new data or offering to send data to its out-of-date neighbors.

Deluge divides a data image into *packets* of a fixed size n . The packet is the smallest unit of reliability that Deluge considers: Packets are transmitted in single messages that must either be received or dropped in their entirety by any potential receivers. n is chosen to fit within the network packets used by the TinyOS networking stack.

At a higher level, a data image is divided into contiguous *pages*, each consisting of a fixed number N of packets. A node may only be engaged in sending the packets of a single page at a given time. Similarly, a node dedicates itself to receiving the packets of one page at a time. Since RAM size is small relative to data image sizes, this form of segmentation helps minimize the sizes of buffers that must be stored in memory. Buffering of pages is necessary because the flash storage that is the eventual destination of the data is not directly addressable.

The motivation for using these divisions and the choices for the concrete values of n and N come from the limitations of the target mote platforms. The more fine-grained the data nodes keep on their progress towards receiving all of a new data image, the easier it is to plan communications and avoid transmission of redundant information. However, the amount of RAM available to a mote is small enough to

rule out many otherwise reasonable schemes a mote would use to track the states of itself and its neighbors. Also, with limited network bandwidth and many poor link qualities, the benefits of minimizing the amount of metadata to be communicated can outweigh the benefits that come from motes broadcasting more detailed information about their current states.

We choose the value of N such that it is feasible to include in periodic advertisements complete bit vectors indicating which pages of the new data image a node has not yet received fully. While the packet sizes used by TinyOS's radio network layer may be adjusted as desired, larger sizes decrease the effective bit-level transmission rate, since error detection methods are applied at the packet level to decide whether or not to drop a given packet.

Another limiting factor involves the requirements for sending portions of a node's program image. Since the EEPROM used to store a new data image as it accumulates is not directly addressable, it is necessary to keep a portion of the new image in RAM if it is to be forwarded on to other nodes. As Deluge must deal with data images many times larger than available RAM, N must be chosen such that N -sized buffers may reasonably be allocated in main memory. Larger values minimize the need for EEPROM interactions that bring relatively high energy and time costs.

The value of n is chosen similarly: A bit vector indicating which packets within a page a node has received should fit in a reasonably sized TinyOS packet. This is important in the Deluge algorithm because it is possible for many nodes to make simultaneous packet-granularity requests for transmission of data from potential senders. Limiting such requests to single packets minimizes coordination overhead, while allowing them to provide as much information as possible on a node's needs allows for better scheduling of data propagation.

4.2 The Protocol

Deluge is a NACK-based protocol that relies on periodic advertisements to keep nodes informed of their neighbors' states. The basic cycle of communication goes as follows:

1. Nodes periodically broadcast *advertisements*. Each advertisement contains a version number and a bit vector describing which pages of that version the advertiser has received completely. The delay between advertisements can be varied to conserve energy and bandwidth. Faster advertisements are used in periods of active updating, while slower advertisements can be used the majority of the time.
2. When a node has determined from advertisements that it needs to upgrade some part of its image to match a newer version, it determines the lowest numbered page it requires. It waits for a fixed amount of time, listening to advertisements to find which of its neighbors are offering to send that page. Different heuristics can be applied to select among this pool of potential senders.
3. When this period is up, the node sends a *request* to the particular sender chosen by the heuristics, indicating the page and packets within that page that it needs.
4. Upon receiving such a request, a node begins a period of waiting like the one described above. Instead of

being used for sender selection, this time is reserved to allow selection of which page and which of its packets to send. Various heuristics can be used to choose the page to send, and the set of packets to send will be the union of those found in requests to this node for the page that is chosen.

5. After this period ends, the sender broadcasts a *data packet* for every requested packet of the selected page.
6. After a node receives the last packet it needs to complete a page, it broadcasts an advertisement before attempting to request any packets it still needs. This stimulates spatial multiplexing.

The entire process avoids maintaining any kind of neighbor table or any other information on the states of other nodes that uses more than a constant amount of space. This promotes robustness in the face of a changing environment, unpredictable link strengths, and other such factors. It also simplifies implementation.

While the protocol was described as NACK-based, there is no explicit sending of NACK’s after an unsuccessful round of sending. Instead, the sender (or some other sender) will “remind” potential receivers of what it has to offer with another advertisement. Receivers will re-send requests for specific packets within the pages they didn’t receive entirely, leaving out those packets that they did receive. These re-requests play the same role as NACK’s, while being controlled by a less stateful process.

Ideally, the members of the network will elect senders that are sufficiently distant from each other to minimize network contention. Instead of using any global coordination method to achieve this, Deluge relies on sender selection heuristics designed such that nodes with overlapping neighbor sets should tend to choose the same senders, and senders that are dispreferred in one such neighborhood should be dispreferred in any others they belong to as well.

A simple form of incremental upgrades is supported using the same mechanism by which nodes advertise their progress in obtaining the image. Advertisements may be marked as “complete” to indicate that they give complete information on which pages of an image have changed since the previous version. Nodes that are only one version behind may then skip upgrading the pages that have not changed. Even greater savings are possible by using an intelligent binary differencing scheme, where the data image that is transferred is a “script” describing how to update an old image, rather than the contents of the image.

5. DELUGE: POSSIBLE OPTIMIZATIONS

The basic protocol, as described in Section 4, is similar to SPIN-RL. However, because SPIN-RL is designed for relatively small data objects, a whole set of possible optimizations have not been considered. With large data objects and low bandwidth, simply transferring the data between two neighboring nodes takes a significant amount of time. This latency opens the door to a wide range of possible optimization not previously considered. In this section, we identify a set of possible optimizations to increase the performance of Deluge.

In each of the possible optimizations we identify, we continue our philosophy of keeping decisions strictly local to a node and avoiding the need to maintain neighbor tables and

states of other nodes in forms that use more than a constant amount of space.

5.1 Packet Transmit Rate

We consider sending data packets at a wide range of rates, from the maximum possible send rate to one-tenth that rate. Transmitting data packets at the highest possible rate may not be the best solution for efficiently disseminating data. One problem is that contention is increased, causing increased probabilities in packet collisions and energy usage required to recover from collisions. Slowing the data packet transmit rate can help reduce contention and reduce the number of dropped packets. Additionally, slowing the transmit rate has the advantage of reserving channel capacity for messages control packets, as proposed in [21]. However, slowing the transmit rate also reduces the bandwidth, in turn possibly increasing the amount of time required to completely disseminate the data to all sinks. This could be more of a problem when dealing with large amounts of data since small changes in transmit rates could significantly change the time-to-completion and energy wasted in the idle-listen state.

5.2 Spatial Multiplexing

While low-power radios used in sensor networks have limited range, we can take advantage of this fact to increase performance of disseminating data. The limited range of the radios gives the network multiple communication cells, allowing the use of spatial multiplexing such that different parts of the network can communicate in parallel without interfering with each other. When the sinks are more than one hop away from the source nodes, we can utilize spatial multiplexing and pipeline the dissemination of separate pages.

Pipelining is a common trick to increasing the bandwidth of high-latency operations while possibly suffering an increase in latency of the operation itself. In CPUs, pipelining increases transistor utilization by overlapping the execution of multiple instructions. By using a similar method, we can increase the throughput of disseminating different pages to nodes more than one hop away by overlapping the transmission of different pages.

To illustrate the importance of pipelining, consider a sink node which is four hops away from the source node as shown in Figure 1. One option is to force nodes to receive all pages of the data object before rebroadcasting those pages to the next node. This method is inefficient in that only one cell of the network is utilized at all times. Given the four hop network, it is possible for nodes in different cells to broadcast without interfering. Thus, we can make use of spatial multiplexing to pipeline the transmission of pages.

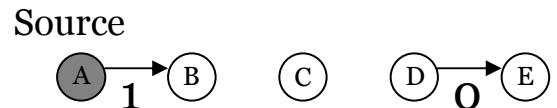


Figure 1: Example four-hop network showing the effectiveness of spatial multiplexing.

To support pipelining, we allow nodes to advertise and send fully received pages even if all pages for the data object have not yet been received. To promote the effects of

pipelining, nodes which have just received a page immediately broadcast an advertisement notifying neighboring nodes of the newly received pages. Additionally, a node which has just completed sending a page delays its next advertisement to allow neighboring nodes to propagate the page further away from the source.

Figure 1 provides a simple example of the mechanism used to promote pipelining. Node A, the source node, initiates the data transfer by sending the first page to node B. After completely receiving the page, node B immediately advertises its newly received page to notify node C of the new data. Node C then makes a request, to which node B responds by broadcasting the data. Node A delays its next advertisement to allow node B to propagate the first page. Without a delay, node B may request the next page from node A, blocking the propagation of the first page. Once node C has fully received the first page, it can then propagate that page to node D. At the same time, node B is now idle and is free to make a request for the next page from node A. This strategy does not address the hidden terminal problem where node B may be unable to receive some packets from node A due to collisions caused by node C. However, it is important to note that node D's reception of packets is unaffected by node A's transmissions. In the worst case, node B may have to wait until node C has completed its send. But if this occurs, pipelining without interference can now occur with both nodes A and D sending simultaneously.

5.3 Sender Selection

Because the sending of a page is a fairly significant operation, the problem of choosing efficient senders when making requests becomes more important. Ideally, the perfect sender selection mechanism would minimize the number of senders to decrease energy consumption and channel utilization of the radio. Additionally, a perfect sender selection method would minimize the amount of time required for completely disseminating the data to all sinks.

We consider four simple heuristics for efficiently selecting a sender when more than one node is able to supply the needed data.

1. A node requests data from the sender which most recently advertised. This simple approach relies on the random nature of when advertisements are broadcast by different nodes. Choosing a node which most recently advertised causes requests from neighboring nodes to gravitate to a single node.

The following approaches make use of a distance metric from the source nodes. The metric is an approximation of hop count from the source node, calculated using moving averages based on self reporting of estimated distance by nodes.

2. A node requests data from the node nearest to it. This approach attempts to minimize the distance between a sender and requester in order to ensure a high quality, bidirectional link between them.
3. A node requests data from the node furthest from the source. This approach attempts to promote spatial multiplexing and reduce the time required for completely disseminating the data to all sinks. One disadvantage with this approach is lack of bias given to promote high quality, bi-directional links between the sender and receiver.

4. A node requests data from the node closest to the source. The disadvantages of this approach include the impairment of spatial multiplexing and no bias to promote high quality, bi-directional links between the sender and receiver. While we see no apparent advantages, it is evaluated simply for comparison.

5.4 Sender Suppression

Many protocols [11, 14, 16] have considered mechanisms for suppressing redundant message broadcasts to reduce energy consumption and channel utilization. For this work, we evaluate methods for suppressing data transmissions of nearby senders. We leave methods for suppressing control messages for future work.

We consider an approach of eliminating neighboring senders by suppressing nodes such that only one sender within a cell is active. If a node sending data messages also overhears data messages from other nodes at the same time, it decides whether or not to suppress itself. This decision is strictly local and is based on unique node ID's, the ID of pages being sent simultaneously, and the number of overheard data messages from other nodes. Due to the dynamics of wireless communications, it is possible for nodes to occasionally receive messages from distant nodes. Suppressing nodes when overhearing just one data message may be overly aggressive. To account for this, we allow a certain number of overheard data messages to occur before nodes are suppressed. Also, to continue our philosophy of sending pages in sequential order, lower page numbers are preferred. Ties are broken by node ID in the case where the pages being sent are the same.

We do not consider any methods for dealing with the hidden terminal problem. Such methods can be fairly complex and require cooperation by the receiver [1]. It is not clear whether such approaches are effective in networks which are not based on a point-to-point model.

5.5 Forward Error-Correction

As mentioned in Section 2, a digital fountain approach which makes use of forward error-correction to reliably distribute large amounts of data has been proposed. The basic mechanism behind the use of forward error-correction is to encode information by adding additional redundant data such that a receiver can reconstruct the original data from any k -sized subset of the encoded data. Unlike more traditional ACK/NACK-based approaches, forward error-correction addresses a problem known as feedback implosion, where requests for lost packets can overwhelm the network or even the server. With forward error-correction, nodes are less likely to make requests for missing packets.

While originally designed for point-to-point networks, we evaluate the effectiveness of using forward error-correction when dealing with multihop, wireless sensor networks. The use of such approaches may also prove useful when dealing with low bandwidth, lossy radios. However, the effectiveness of such an approach may be limited when considering the highly variable characteristics of wireless networks. Unlike more traditional wired network models, packet loss rates between different nodes can vary from nearly perfect communications to more than 90%. Forward error-correction codes are best optimized when loss rates between all nodes are roughly constant and can be described by a single estimate. For example, when packet loss rates are high, additional re-

dundant information should be encoded to allow a greater number of packets to be dropped. However, this increases the amount of the encoded information to transmit.

5.6 Packet Ordering

We also consider the order in which to send packets within a page. We evaluate two approaches: (i) sequential order and (ii) random order.

In the basic protocol, nodes request pages in sequential order. This approach minimizes the range of pages that neighboring nodes may request for two reasons. First, nodes are limited to requesting the lowest numbered page that has not been received. Second, because both senders and receivers give bias to lower numbered pages, those pages are more likely to complete before higher numbered pages. One could argue that packets within a page should also be sent in sequential order for the reasons described above. However, the effectiveness of such an approach is not so clear. The reason is that the cost of transmitting all requested packets within a page is relatively small.

The digital fountain approach has suggested sending packets in random order to eliminate bias for specific packets. When sending packets in sequential order, nodes requesting packets with high packet ID's will always need to wait for requests of lower packet ID's to be fulfilled. More importantly, when considering sender suppression mechanisms, this may be important since requests for higher packet ID's are more likely to get dropped when a sender is suppressed.

6. EVALUATION

We chose to evaluate the performance of Deluge by examining the effect of the different optimizations described in Section 5 on the metrics described in Section 3.2. In addition, we compare the performance of Deluge to estimates of optimal performance based on the network topology at increasing scale.

6.1 Simulation Setup

We implemented the Deluge algorithm in the nesC programming language [7], and executed the program within the TOSSIM [13] simulation framework. We chose to use TOSSIM because of its accurate representation of real-world TinyOS sensor networks. TOSSIM models radio loss characteristics based on measurements taken of Mica mote radio performance at varying distances, and this radio model was used exclusively in our tests.

In order to consistently examine the effect of network size and density on the performance of the dissemination protocol, we chose to model a multihop network by placing nodes at points in a square grid and changing two variables: the number of nodes in the grid and the spacing between the nodes. The node in the upper-left corner was always set to be the source of the new data image. We varied the spacing to represent networks of varying density in order to examine the effect of loss rates on the algorithm. The possible values for network spacing range from 0, creating a single-cell topology, to 22, a nearly disconnected network. We examined the performance of the algorithm by placing the nodes at spacings of 0, 2, 4, 8, 12, 16, and 20 feet between nodes. The function from distance to per-link packet loss rate is based on empirical data gathered from tests of Mica motes, and embedded within TOSSIM. We generated a new set of inter-node link qualities for each separate run of the sim-

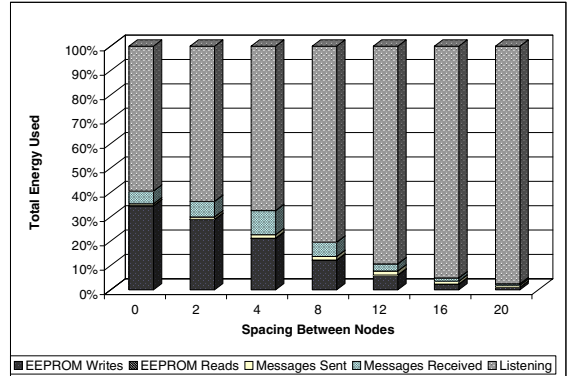


Figure 2: Breakdown of Energy Used (Full Speed Send)

ulation, in order to prevent our results from being unduly affected by particularly good or bad connectivity for certain nodes in a generated topology.

Our first preliminary test was designed to examine the effects of different energy costs as described in [15], and how they were associated with the operations performed by the algorithm. We tested the basic protocol, without any optimizations and found that the cost of idle listening overwhelms all other costs, as shown in Figure 2. This is true at all non-minimal levels of density. Therefore, in analyzing the effects of the rest of the optimizations, we focus on minimizing time to completion. Where necessary, we will also present data on messages sent and received.

6.2 Optimization Results

To accurately compare the effects of the different optimizations, we fixed the number of pages to distribute (3) and the number of nodes intended to receive the image (a 5 by 5 grid). We varied the configuration of the Deluge algorithm by selectively enabling specific combinations of the optimizations specified above. Each configuration of Deluge was executed with three different generated connectivity graphs for each scaling setting and the results were averaged, in order to prevent unusually good or bad connectivity from unduly affecting the resulting metrics. We measured time to completely distribute the three pages to the twenty-four receiver nodes; the total number of messages sent and received, broken down into data and control; the number of EEPROM reads and writes; and the total energy consumed by all nodes within the network. The following graphs present only the time metric, due to the dominating listening cost.

6.2.1 Transmit Data Rate

Our first experiment was designed to test the effect of varying the data transmission rate. We hypothesized that transmitting at a slower rate would minimize the effects of contention and result in an overall savings in energy, at the possible expense of increased time to completion. We ran the simulation with data rates of 256ms per packet, 128ms, 64ms, and at the full speed of the radio channel. Each speed was run at each of the 6 network densities. Figure 3 presents the time to completion.

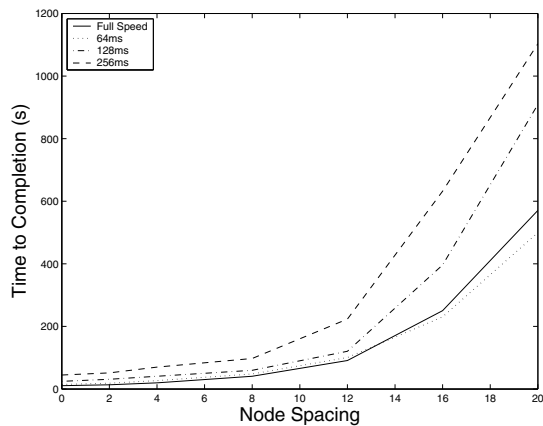


Figure 3: Effect of Send Rate on Time ($p < .01$)

At most densities, a faster send rate resulted in a faster time to completion. We found that when the network was very dense, slower send rates did indeed decrease the number of message retransmissions. However, the increased radio listening cost induced by the longer time to completion dominated the energy savings resulting from fewer transmissions, and slower send rates generally resulted in a greater usage of energy. The only case in which this was not true was in networks of density 0 and 2, presumably due to the much greater opportunity for collision present in networks of such high density.

We did find that sending at the maximum channel capacity consistently required sending more data messages to complete the same dissemination task. This is most likely due to increased collisions caused by channel contention. Unusually, the effect was most pronounced in sparser networks. We also found that sending at data rates of 8 pkt/sec and 4 pkt/sec required more control messages than sending at 16 pkt/sec. This effect was also only noticeable in sparse networks. This may result from interactions with certain timeout values that are used by the algorithm to decide when to stop requesting a page from a given sender, how quickly to readvertise, and how often to advertise. The slower send rates may result in more advertisements and more request messages intended to restart a stalled page transmission.

In terms of both message count and energy, sending at a fixed rate of 16 pkt/sec outperformed full speed sending in sparse networks. In denser networks, the two faster speeds were indifferent, but better than the slower speeds. These results suggest that Deluge should send at a data rate that is slightly slower than full channel utilization.

6.2.2 Spatial Multiplexing

The second experiment was designed to test the effect of allowing spatial multiplexing through per-page pipelining. We hypothesized that enabling pipelining would more effectively use the available bandwidth and result in a savings of time. Figure 4 presents the total time spent by the two different pipelining configurations with the transmit data rate set to full speed.

Pipelining was consistently effective in lowering the time to completion at nearly all network densities, though the difference was not statistically significant. It is possible that tests with larger networks would increase the effectiveness of

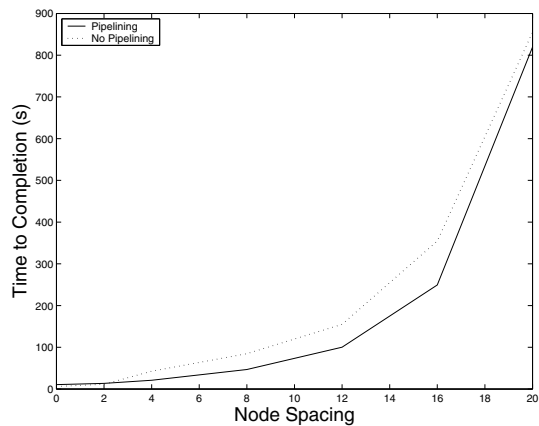


Figure 4: Effect of Pipelining on Total Time

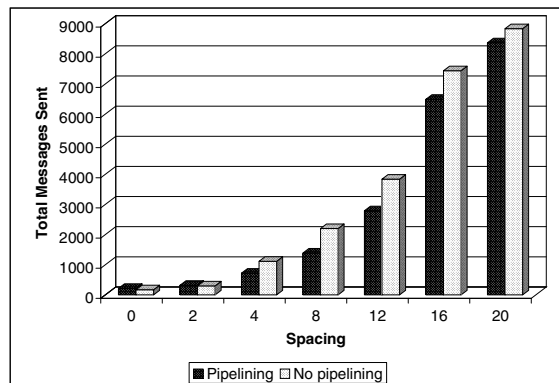


Figure 5: Effect of Pipelining on Sent Messages

pipelining since there is greater opportunity to utilize spatial multiplexing. In a single-cell network, we found that the extra control messages contributed to greater contention and a slightly greater time to completion. This result was expected, because pipelining can only be helpful when multiple non-overlapping radio neighborhoods exist in the network.

In addition, as shown in Figure 5, pipelining required fewer sent messages at all network densities less than a single cell. Both control and data messages were reduced, but the majority of the reduction was due to data messages. This can likely be accounted for by the fact that pipelining moves the data through the network more quickly. This serves to reduce the number of long distance page requests, and consequently reduces the need for retransmission of data.

6.2.3 Sender Selection

Our third experiment examined the effect of the different heuristics for sender selection on overall time to completion. We hypothesized that preferring senders with an estimated hop count close to that of the requester would minimize the negative effects of packet loss, and secondarily preferring hop counts further from the source would encourage the spread of the image. We set the send rate to full speed. Figure 6 presents our results.

At most densities, the different heuristics turned out to

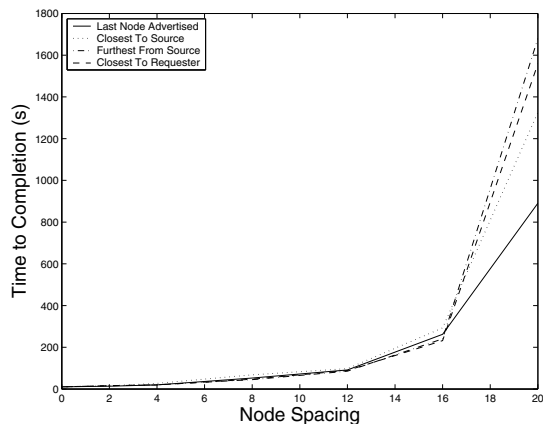


Figure 6: Effect of Sender Selection on Total Time

have no significant effect on time to completion. This suggests that future work should include a more thorough examination of sender selection. The sparsest network we tried showed a large reduction in time to completion when the sender selection method was to select the most recently heard advertiser. This is likely due to the fact that allowing nodes to store information about previous best senders, instead of responding to all advertisements, will increase the number of total senders and create excess contention.

6.2.4 Sender Suppression

We chose to compare three different sender suppression techniques. The first is aggressive suppression, such that overhearing a single data packet while a node is sending a page causes that node to back off for the length of the rest of the page. We hypothesize that this will be too sensitive to stray long links, and will unnecessarily increase the runtime of the algorithm by suppressing senders that could possibly be successful.

We also examine suppression after overhearing half of a page worth of data packets, and after hearing a full page worth of data packets. For a node to overhear a full page of data packets during the time it is sending its own, it must have started at exactly the same time as a sending neighbor, or it must have more than one sending neighbor. This seems like the worst case, and so by delaying the suppression by half a page or a full page, we hope to eliminate clusters of senders while allowing simple neighboring senders to continue. Figure 7 shows the effects on time to completion.

The results were indistinguishable in most densities. In very sparse networks, we saw that single-packet suppression took longer to complete than no suppression or multi-packet suppression. In general, no significant difference existed between the different types of suppression. This result is surprising, because most other multicast data dissemination protocols do use suppression of many types. Further work should examine control packet suppression in addition to data packet suppression.

6.2.5 Forward Error Correction

We now test for the effects of per-page forward error correction. We hypothesized that enabling forward error correction on a per-page level would result in fewer NACK requests for retransmission in sparse lossy networks, and result

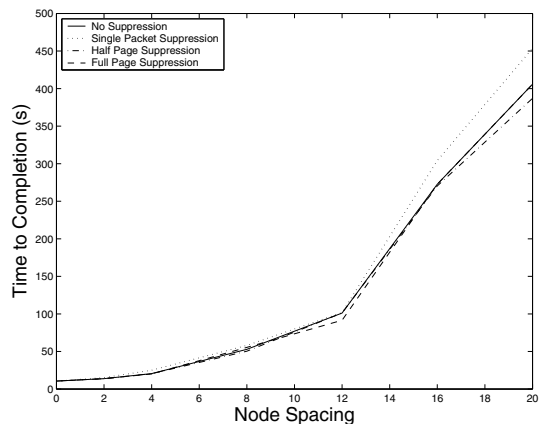


Figure 7: Effect of Sender Suppression on Total Time

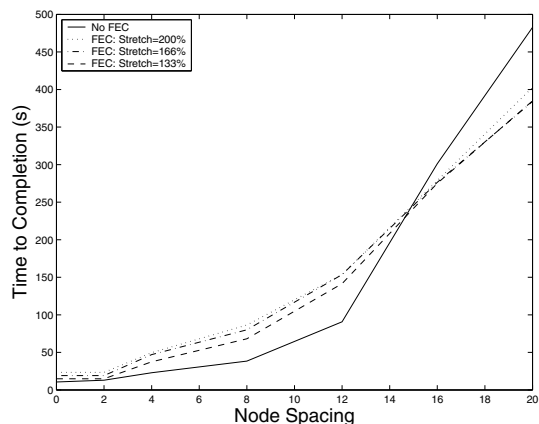


Figure 8: Forward Error Correction vs. Time

in a savings of time and therefore energy. We hoped that this would outweigh the extra cost of re-sending the entire page with each NACK, and the cost of sending more data packets overall. We tested FEC with three different stretch factors and randomly ordered packet sending against the basic Deluge algorithm with a full-speed send rate. Figure 8 shows the time to completion by the different configurations.

Forward error correction was clearly more advantageous in sparse networks than in dense networks. Because the link qualities between immediate grid neighbors in dense networks were sufficiently better than the average link quality in the network, the redundancy was not necessary for accurate transmission to immediate neighbors. Plus, the collisions caused by the sending of unnecessary extra packets likely impaired overall performance. In the two sparsest networks we tried, FEC showed a clear improvement in time. This is undoubtedly accounted for by a dramatic reduction in lost and retransmitted control messages. However, the increase in the number of data messages sent by Deluge-FEC is greater than the decrease in control messages, and Deluge consistently demonstrates fewer necessary sent messages.

6.2.6 Packet Ordering

We examined the effect of random packet ordering on both the traditional Deluge algorithm, and Deluge-FEC with a

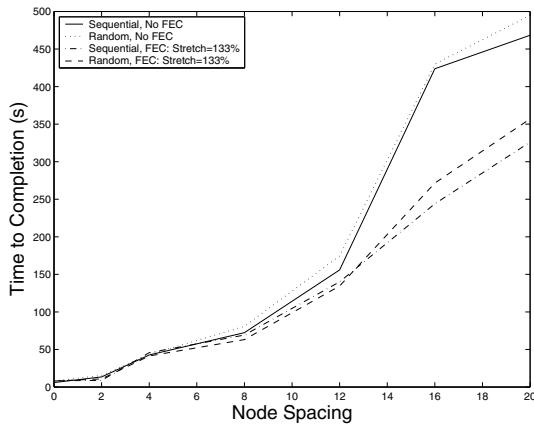


Figure 9: Effect of Random Ordering on Total Time

33% expansion. We hypothesized that random ordering would improve the performance of Deluge-FEC, and that it would have no effect on standard Deluge. We implemented the random ordering by randomly permuting the set of requested packets just prior to sending, with a new ordering for each instance of responding. Figure 9 displays the effects of random packet ordering on time to completion.

In the higher densities, we found that random ordering did not have much effect on the time to completion. As the network grew sparser, the negative effects of random ordering grew more pronounced. We found that random ordering actually increased the time to completion, while having little effect on the number of messages sent. This negative effect occurred in both Deluge and Deluge-FEC. This may result from the decorrelation between missed packets across multiple requests.

6.2.7 Optimization Conclusions

Based on the results of these tests, we will now consider the Deluge algorithm to include full-speed sending, spatial pipelining, simple most-recent-advertisement sender selection, and no forward error correction. We omit the more complex optimizations because a simple algorithm that performs no differently than a complex one is more amenable to analysis.

As a general result, we found that sparse, loosely-connected networks tend to include more variability. This may be due to differences in underlying connectivity from run to run of the simulation, but the loss of packets itself can create wide changes in the performance of the algorithm and optimizations. Lost packets in sparse networks are much more damaging, indicating a paradoxically greater need to manage contention as the network connectivity gets worse.

In all real-world network densities, the number of lost packets greatly increases the time to completion for the algorithm. This tends to make radio listening the dominating cost. This suggests that we should examine power scheduling optimizations that will allow a given mote to power down its radio when it is not directly involved in transmission of a page. Low-power listening in the MAC layer may also be a useful optimization, but the increased preamble length creates more chance for error. This may serve to increase packet loss to the point where the degraded performance of the algorithm outweighs the energy savings.

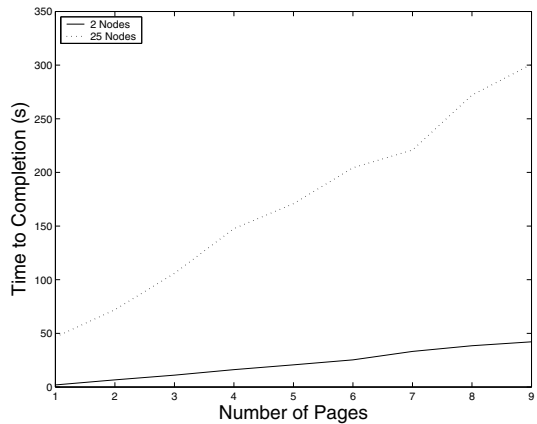


Figure 10: Page Count vs. Total Time

6.3 Scalability Results

A single-hop network of two nodes with perfect radio connectivity represents the base case for scalability. Therefore, we use the time to completion to transfer n pages in this simple network as our baseline measurement. This accounts for the effect of timer delays that cannot easily be accounted for by a raw latency and bandwidth modeling of a larger network.

We would like to examine the performance of the algorithm as the network scales. Three main dimensions of scaling can affect performance.

1. an increase in the number of pages to send
2. an increase in the lossiness of the network
3. an increase in the diameter of the network

We would like the time to completion to scale linearly with the number of pages. The addition of spatial pipelining should cause the time to increase sublinearly, due to the propagation of more than one page at a time. We created a 2-node network and a 25-node network, and charted the time required to send an increasing number of pages. Figure 10 demonstrates the effect of increasing page count on time to completion.

We expect that the packet losses resulting from an increasingly sparse network will have an exponential cost, as each packet loss will generate multiple packets for re-requesting and retransmission. In all of our previous tests, we saw a roughly exponential scaling of time to completion as the density dropped.

The third type of scaling is the most crucial, as real networks will be built with ever-increasing sizes. Ideally, we would like our algorithm's time to completion to scale linearly with increasing hop distance from the furthest node to the source. Unfortunately, we found that the runtime of Deluge increased superlinearly as the network diameter increased, as shown in Figure 11.

To further understand the neighborhood properties of the network at different spacing factors and network sizes, we examined the average number of neighbors per node as both of these factors increased. We considered a neighbor to be any node with a probability of receiving a packet from a source. We also considered a limited definition of neighbor

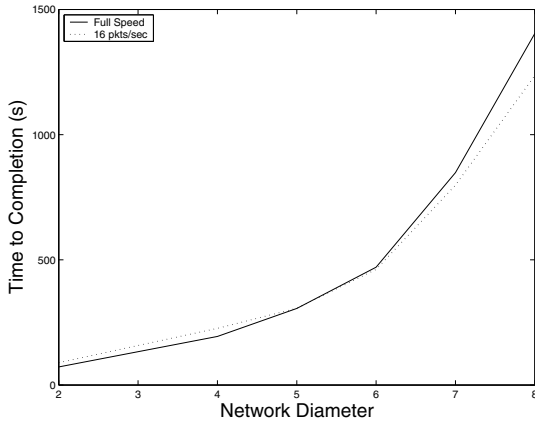


Figure 11: Network Diameter vs. Total Time

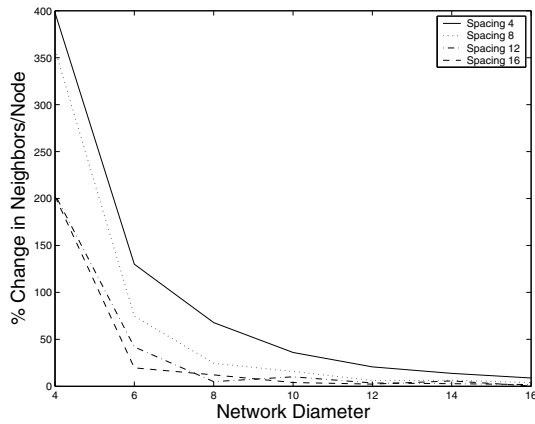


Figure 12: Neighbor Count vs. Network Diameter

based on a greater than 50% packet success rate. Figure 12 presents the percent change in average neighbor table size as the network diameter increases.

We see that in the network sizes and spacings used to test our optimizations and scaling performance, the rate of increase in the number of neighbors per node decreases as the network grows larger. This implies that edge effects due to a large portion of nodes at the edge of the network are affecting the results. Each density should have a critical size, after which any increase in network size does not increase the number of neighbors. We believe that because we ran our tests at network sizes below this critical size, contention increased as the network grew. We believe that some of the nonlinearity in network scaling is due to this increasing contention. In future work, we plan to test the scaling performance of our algorithms in networks larger than the critical size. We also plan to consider new methods for congestion control.

6.4 Preliminary Hardware Tests

Because TOSSIM executes programs written directly in nesC, we were able to compile our system and install it on a small network of Mica nodes. Preliminary tests indicate that our system successfully distributes data across a 3-node network, with placement such that the network contains two hops. Clearly, future work includes testing the system on a

larger hardware testbed, but a proof of concept installation was successful.

The requirements in RAM and the binary size of Deluge is given in Table 2. These numbers include all TinyOS components which are required for Deluge to operate correctly (i.e. GenericComm, Timer, etc). We note that the current implementation of Deluge is not optimized to reduce RAM usage and binary size. For comparison, we also include RAM requirements and binary size of CntToLedsAndRfm, a sample TinyOS application.

Application	RAM (KB)	ROM (KB)
Deluge	1.75	15.73
CntToLedsAndRfm	0.32	8.39

Table 2: RAM and ROM requirements of Deluge and CntToLedsAndRfm.

These numbers show that Deluge consumes an acceptable amount of resources, leaving enough for the main application. About 1KB of Deluge’s RAM usage can be attributed to the send and receive page buffers. This lower bound on RAM usage can be further reduced if the EEPROM buffer (512 bytes on the Mica) is properly utilized.

7. FUTURE WORK

This paper describes initial work on a data dissemination protocol for large data objects. The current Deluge implementation makes assumptions that may not be true with many sensor network applications. In this section, we identify several issues we plan to focus on in future work.

7.1 Control Message Suppression

While this work has experimented with the suppression of data packets, the suppression of control messages was not considered. SRM [11] and Trickle [14] have both considered various methods of suppression. In future work, we will consider methods for suppressing redundant or useless control messages. Eliminating these redundancies will help reduce contention yet allow useful messages to transmit at a sufficiently high rate. Suppression mechanisms also have the ability to dynamically adjust to varying densities, limiting the amount of contention caused by control messages in dense areas of the network.

In our experiments, we noticed significant tradeoffs with setting the advertisement period to short vs. long periods. To help promote quick propagation of new data, the advertisement period should be short to minimize the expected time required for neighboring nodes to learn about the newer data. However, our results show that having all nodes advertise at a high rate causes a great deal of contention and inhibits the transmission of all other messages, including data. This effect increases as the network density increases, suggesting that the period at which an individual node advertises should adjust with the density of the network. One method to dynamically adjust with changing densities is to have nodes continuously estimate the number of neighbors. However, this directly conflicts with our philosophy of keeping minimal state and avoiding the need for neighbor tables. Instead, a better approach is to make use of control message suppression.

Additionally, in dense networks, numerous request messages made by nodes after an advertisement can be lost due

to high contention. To avoid these problems, suppression of request messages should also occur. However, suppression of request messages is a more difficult problem since requests can be made for differing pages and specific packets within a page. Even if we consider requests for a single page, the number of unique requests can equal the number of packets in a page in the worst case. A possible approach to deal with requests for differing pages is to suppress requests for higher-numbered pages. To limit the number of unique requests for data within a page, we can further divide data pages into packet sets composed of two or more contiguous packets and have requests be made for these sets rather than individual packets.

7.2 Concurrently Running Applications

In its current state, Deluge gives little attention to the fact that the application may require other operations to run while data is disseminated. In some applications, it may be acceptable to pause the application while the data is being disseminated. However, due to the large amount of time required to disseminate the data, it is likely that the application will need to continue some, if not all, of its tasks. The results given in Section 6 assume that the dissemination of data is the primary application of the sensor network. For example, while the results show that sending data packets at the highest possible rate reduces both energy usage and latency, the radio channel is pushed to saturation. This leaves little room for any other communication required by the application. In future work, an evaluation of how Deluge affects various concurrently running applications is necessary.

7.3 Reducing Energy Consumption

While the energy consumption of the current Deluge implementation is acceptable, it leaves much room for improvement. As shown in Section 6, most of the energy consumed by nodes occurred in the idle-listening state. In future work, we will consider methods for turning the radio off and allowing the node to operate in a low-power state while data is being disseminated. One possible optimization is to allow a node to turn off its radio when neighboring nodes are sending a page which is not needed. Allowing the radio to power down does create difficulties with effectively receiving broadcast advertisements, and suggests a need for more detailed scheduling to ensure that on-times coincide.

Another approach is to make use of low-power listening. This approach does not require the creation and maintenance of schedules. Instead, nodes periodically turn their radios on and off to save power. Because the state-transition schedule of the radio is unknown, nodes must transmit a preamble proportional to the duty-cycle of the radio before transmitting a packet. While this approach can save significant energy, it is not clear whether such an approach would be beneficial. Low-power listening decreases the bandwidth of the radio channel and is dependent on the duty-cycle of the radio. In future work, we will evaluate the effectiveness of using low-power listening with Deluge.

7.4 Multiple Types/Versions

As stated in Section 3.1, we assumed that only one type of data object exists and is required by all nodes in the network. In the context of network reprogramming, this assumes that all nodes run on the same code base. Al-

though this may be true for many sensor network applications, other applications may have nodes run different code bases depending on their function. As the complexity of sensor network applications increases, separating required functionality among different nodes may be a useful technique for efficient development and operation. For example, in some high-speed sampling applications it may not be possible to sample multiple sensors and process the data due to limited computational power. Instead, different subsets of nodes may be responsible for sampling different sensors and processing their data.

Future work will need to consider support for multiple types of data objects where each type is only required by a subset of nodes. Describing different types of data objects is simple in that we can extend the metadata to include a unique identifier for each type. However, the dissemination protocol must now consider that nodes requiring a specific data object maybe be separated by one or more nodes which do not. The reliable transport of data over multiple hops is not a new problem. In future work, we will evaluate methods for efficient caching of data along routes, ensuring reliability, and congestion control.

Deluge allows efficient incremental upgrades by sending only those pages which differ from the previous version. However, the current implementation only handles this case when the new version number and the current version number differ by exactly one. When the version numbers differ by more than one, Deluge currently requires all pages to be sent since information describing which pages have changed is not available. There may be cases where connectivity between some nodes may fade for significant amounts of time, causing nodes to fall two or more version behind in the presence of frequent upgrades. One method to allow incremental upgrades for multiple versions existing in the network is to describe each version of a page with a unique identifier. This can be done by using minor version numbers or applying a hash to the contents of the page.

8. CONCLUSION

This paper has demonstrated that the Deluge system reliably distributes data across a multi-hop network while maintaining a constant amount of local state and making only local decisions. We keep the interactions between nodes strictly local and avoid the need to maintain neighbor tables. This property allows Deluge to be robust to widely varying connectivity scenarios. Since there is no need to maintain state about all neighboring nodes, nodes may move and connectivity can vary without requiring nodes to adapt to such changes.

We have examined the effects of multiple optimizations on the performance of Deluge in terms of runtime, energy consumption, and message transmission. We have shown that the excess contention incurred by fast message sending is outweighed by the energy savings resulting from a faster time to completion, and that enabling spatial multiplexing through per-page pipelining decreases the time to completion and transmitted messages. Forward error correction has been examined and shown to be beneficial in extremely lossy networks.

Many of the optimizations we tested showed no significant improvement in the efficiency of Deluge. This finding shows that it is possible to achieve an efficient implementation without complex algorithms. This is promising since it

allows us to keep implementations simple, making a correct implementation straightforward.

In examining the scaling performance of Deluge, we have shown that the performance scales linearly with the amount of data to be transmitted. In addition, we have identified areas for future work in contention management intended to enable Deluge to scale linearly with the radius of the network. Deluge successfully completes distribution in lossy and temporarily disconnected networks.

Finally, a proof of concept installation of the Deluge system has been tested and found to be successful on a small network of Mica motes.

9. REFERENCES

- [1] V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LAN's. In *SIGCOMM*, pages 212–225, 1994.
- [2] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of the ACM SIGCOMM Conference*, pages 56–67, 1998.
- [3] Crossbow Technology, Inc., San Jose, CA, USA. *Mica2 Wireless Measurement System*, 2003.
- [4] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *IEEE International Conference on Communications (ICC)*, pages 376–380, June 1997.
- [5] A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12. ACM Press, 1987.
- [6] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical Report UCLA/CSD-TR 02-0013, UCLA, 2002.
- [7] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *SIGPLAN Not.*, 38(5):1–11, 2003.
- [8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.
- [9] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.
- [10] J. Jeong, S. Kim, and A. Broad. *Network Reprogramming*. University of California at Berkeley, Berkeley, CA, USA, August 2003.
- [11] S. K. Kasera, G. Hjálmtýsson, D. F. Towsley, and J. F. Kurose. Scalable reliable multicast using multiple multicast channels. *IEEE/ACM Transactions on Networking*, 8(3):294–310, 2000.
- [12] J. Kulik, W. R. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 8(2-3):169–185, 2002.
- [13] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*. ACM Press, November 2003.
- [14] P. Levis, N. Patel, S. Shenker, and D. Culler. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. Technical Report UCB/CSD-03-1290, University of California at Berkeley, Berkeley, CA, USA, 2003.
- [15] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM Press, 2002.
- [16] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 151–162. ACM Press, 1999.
- [17] E. Pagani and G. P. Rossi. Reliable broadcast in mobile multihop packet networks. In *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pages 34–42. ACM Press, 1997.
- [18] N. Reijers and K. Langendoen. Efficient code distribution in wireless sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 60–67. ACM Press, 2003.
- [19] F. Stann and J. Heidemann. RMST: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pages 102–112, Anchorage, Alaska, USA, April 2003. IEEE.
- [20] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical report, UCLA, Los Angeles, CA, USA, 2003.
- [21] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: A reliable transport protocol for wireless sensor networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 1–11. ACM Press, 2002.
- [22] J. Wu and H. Li. Domination and its applications in ad hoc wireless networks with unidirectional links. In *International Conference on Parallel Processing (ICPP)*, pages 189–200, August 2000.