The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale

ABSTRACT

To support network programming, we present *Deluge*, a reliable data dissemination protocol for propagating large data objects from one or more source nodes to all other nodes over a multihop, wireless sensor network. Deluge builds from prior work in density-aware, epidemic maintenance. We show that Deluge can reliably propagate data to all nodes and characterize its overall performance. The protocol exposes interesting propagation dynamics only hinted at by previous dissemination work. We examine these dynamics and propose methods to improve Deluge and possibly take advantage of them. A simple model is also derived which describes the limits of data propagation in wireless networks.

1. INTRODUCTION

Wireless sensor networks (WSNs) represent a new class of computing with large numbers of resource-constrained computing nodes cooperating on essentially a single application. WSNs must often operate for extended periods of time unattended, where evolving analysis and environments can change application requirements, creating the need alter the network's behavior by introducing new code. Unlike the traditional method of programming a node over a dedicated link, the embedded nature of these systems requires a mechanism to propagate new code over the network. However, developers face a more immediate problem. As WSN research matures, the scale of testbeds and deployments continues to grow. Testbeds sized at tens of thousands of nodes are now on the horizon, making code propagation over the network a necessity in the debugging and testing cycle. These factors suggest that *network programming* (the programming of nodes by propagating code over the network) is required for the success of WSNs. Specifically, we consider the propagation of complete binary images. While virtual machines can provide low-cost retasking with the use of virtual programs, it is sometimes necessary to reprogram nodes with a new binary image [6]. For example, the virtual machine itself may need changes.

The core service required to enable network programming is the dissemination of the program image over a multihop WSN and presents several problems. First, program images are typically much larger than what previous dissemination protocols consider. This is an issue because a sensor node operates on a constrained storage hierarchy where a communication packet (36 bytes) $<<$ RAM (4K) $<<$ program size (128K) $<$ external flash (512K). Second, the dissemination must tolerate node densities which can vary by factors of a thousand or more. For example, collectively programming all nodes before deployment could put thousands of nodes within the same communication range. Third, complete reliability is required, meaning that every byte must be correctly received by all nodes that need to be programmed. Such reliability must be achieved in the presence of high loss rates and evolving link qualities due to the dynamic nature of wireless networks. Fourth, propagation must be a continuous effort to ensure that all nodes receive the newest code. This is necessary since network membership is not static: nodes come and go due to temporary disconnections, failure, and network repopulation. Finally, the dissemination process should require a minimal amount of time. This not only reduces any service interruptions to an deployed application, but is also useful in shortening the debugging and testing cycle.

This paper provides three main contributions. First, we present *Deluge*, a reliable data dissemination protocol for propagating a large data object (i.e. larger than can fit into RAM) from one or more source nodes to all other nodes over a multihop wireless network. Deluge's density-aware, epidemic properties help achieve reliability in unpredictable wireless environments and robustness when node densities can vary by factors of a thousand or more. Representing the data object as a set of fixed-size pages provides a manageable unit of transfer which allows for spatial multiplexing and support for efficient incremental upgrades. Second, we characterize the propagation dynamics of Deluge. While the protocol achieves the task of reliably disseminating large data objects to all nodes, it exposes interesting propagation behavior only hinted at in previous dissemination work. Third, we develop a simple model of Deluge's propagation behavior and use it to identify different factors which limit the overall bandwidth of any multihop communication protocol.

In Section 2 of this paper, we review related work. We describe the protocol formally in Section 3 and discuss our evaluation methodology in Section 4. In Section 5, we present Deluge's overall performance and characterize Deluge's propagation dynamics. Section 6 discusses future directions and conclude with Section 7.

2. RELATED WORK

The problem we address is an important special case of reliable data dissemination. The main differences include the dissemination of data over lossy links, a constrained storage hierarchy, and the need to retain the most recent copy in order to propagate data to additional nodes as they become connected over time. Deluge builds on two bodies of prior work. The first is controlled flooding for communication in wireless and multicast networks. *Scalable Reliable Multicast (SRM)* is a reliable multicast mechanism built for wired networks [4], using communication suppression techniques to minimize network congestion and request implosion at the server.

For data dissemination in wireless networks, naive retransmission of broadcasts can lead to the *broadcast storm problem*, where redundancy, contention, and collisions impair performance and reliability [9]. The authors discuss the need to have a controlled retransmission scheme and propose several schemes, such as probabilistic and location-based methods. The experiments conducted by Ganesan et al. identify several interesting effects at the link-layer, notably highly irregular packet reception contours, the likeliness of asymmetric links, and the complex propagation dynamics of simple protocols [2].

Demers et al. propose an epidemic algorithm based on strictly local interactions for managing replicated databases which is robust to unpredictable communication failures [1]. *SPIN-RL* is an epidemic algorithm designed for broadcast networks that makes use of a three-phase (advertisement-request-data) handshaking protocol between nodes to disseminate data [5]. The epidemic property is important since WSNs experience high loss rates, asymmetric connectivity, and transient links due to node failures and repopulation. However, their results show control message redundancy at over 95% as SPIN-RL considers the suppression of only redundant request messages, and for lossy network models SPIN-RL does not perform as well as naive flooding. *Trickle* builds upon this approach by proposing SRM-like suppression mechanisms to minimize redundant transmission of control messages and pseudo-periodic advertisements to increase reliability, allow for quick propagation, and consume few resources in the steady state [8]. However, Trickle only provides a mechanism for determining when nodes should propagate code. Deluge builds directly off Trickle, by adding support for the actual dissemination of large data objects with a three-phase protocol similar to SPIN-RL.

Because reliability is of top priority, Deluge also borrows ideas from prior work in reliable data transfer protocols. *Pump Slowly, Fetch Quickly (PSFQ)* [16] and *Reliable Multi-Segment Transport (RMST)* [11] are selective NACK-based reliable transport protocols designed for WSNs where low bandwidth and high loss rates are common. Because the cost of end-to-end repair is exponential with the path length, both protocols emphasize hop-by-hop error recovery where loss detection and recovery is limited to a small number of hops (ideally one). Like PSFQ and RMST, Deluge uses a selective NACK-based approach and error recovery is limited to a single hop. However, these approaches do not consider methods for adapting to spatial node density.

Research activity directed at network programming for WSNs has been limited. Recently, TinyOS [15] has included limited support for network programming via *XNP* [3]. However, XNP only provides a single-hop solution, requiring all nodes to be within bidirectional communication range of the source. Additionally, repairs are done on a whole file basis, requiring expensive scans through external flash to discover missing data.

A more comprehensive approach to network programming is presented with *Multihop Over-the-Air Programming (MOAP)*, supporting distribution of a program image over a multihop network [12]. Deluge shares many ideas with MOAP, including the use of NACKs, unicast requests, broadcast data transmission, and windowing to manage metadata required to keep track of which segments are required. However, MOAP ignores many key design options. For example, MOAP does not fragment the image, requiring nodes to receive the entire code image before making advertisements. Thus, it does not allow the use of spatial multiplexing to leverage the full capabilities of the network. Additionally, methods to deal with the adverse effects of asymmetric links are not considered.

A *difference-based* approach to programming has also been proposed [10]. This code distribution scheme attempts to save energy by sending only 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. DELUGE

Deluge is an epidemic protocol and operates as a state-machine where each node follows a set of *strictly local* rules to achieve a desired global behavior: the quick, reliable dissemination of large data objects to all nodes. In its most basic form, each node periodically advertises the most recent version of the data object it has available. If S receives an advertisement from an older node, R, S responds with its object profile. From the object profile, R determines which portions of the data need to be updated and requests them from any neighbor that advertises the availability of the needed pages, including S. Nodes receiving requests then broadcast any requested data. Periodically, nodes advertise newly received data in order to propagate it further.

While the basic form of Deluge is quite simple, many subtle issues are considered to achieve high performance. The first is its density-aware capability, where redundant messages are suppressed. In Deluge, suppression of redundant advertisements and requests is used to minimize contention. Note that while suppression can increase performance by avoiding congestion collapse, its necessary backoff delays introduce latency. For example, requests are made with a random backoff such that a single request can suppress other similar requests. Second, protocols for WSNs need to be robust to asymmetric links, where a link in one direction can have a significantly different loss rate than the other direction. In Deluge, if a node has not completely received all of its data after making a few requests, it stops making requests and searches for a new neighbor to request data. However, if sufficient progress is being made, it may continue making requests. Third, Deluge dynamically adjusts the rate of advertisements to allow for quick propagation when needed while consuming few resources in the steady state. Fourth, Deluge attempts to minimize the set of concurrent nodes broadcasting data within a given cell. Finally, Deluge emphasizes the use of spatial multiplexing to allow for parallel transfers of data.

In the remainder of this section, we describe in detail how Deluge represents the large data object and present a formal description of the Deluge protocol.

3.1 Data Representation

To manage the large size of the data object, Deluge divides the data object into fixed-size *pages*. The page is the basic unit of transfer and provides three advantages: (i) it limits the amount of state a receiver must keep while receiving data, (ii) it enables *efficient incremental upgrades* from prior versions and (iii) allows for *spatial multiplexing*. In this section, we discuss the first two in detail and save our discussion of the last for Section 3.2.

As with many reliable transfer protocol, the data object of size S_{obj} is divided into *packets* of a fixed size S_{pkt}. The packet is the smallest unit of reliability that Deluge considers and must be correctly received or dropped entirely. To ensure receipt of all packets, the node must keep track of the remaining packets required to complete the object. However, because the packet size is generally much smaller than the object, even maintaining a bit-vector of all packets consumes an unacceptably large amount of RAM. Instead, Deluge fragments the data object into P pages each of size $S_{page} = N \cdot S_{pkt}$, where N is a fixed number of packets, as shown in Figure 3. By requiring a node to dedicate itself to receiving a single page at a time, the bit-vector need only be N bits in length.

In many cases, an upgrade may only contain a few minor changes to localized areas within the data object. Thus, nodes need only request those pages which have changed in order to match the newer version. To support this, we assume that updates to the overall data object are serialized. A *version number* is used to distinguish between different updates and must be monotonically increasing to produce a total order for all updates. A node compares version numbers to determine whether it should request new data.

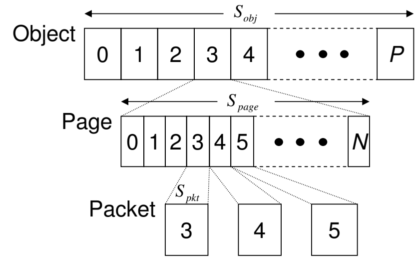


Figure 1: Data Management Hierarchy.

Upgrading from a prior version requires knowing which pages have changed. Because data from recent deployments show that nodes can fade out of connectivity for a significant amount of time [13], it is necessary to allow efficient upgrades from a version which is more than one version behind. This implies that knowledge about *when* pages were last changed is needed. To address this issue, Deluge represents the set of n pages for a given version, v, by an *age vector*, $\mathbf{a} =< a_0, a_1, \ldots, a_{n-1} >$, which describes how "old" each page is. More specifically, the contents of page i at version v last changed at version $v - a_i$. A complete description of the object is defined by the tuple $(v, \mathbf{a})$, which we call the *object profile* and is stored in non-volatile storage along with the data it represents. A node receiving an object profile for a newer version uses the age-vector to determine which pages need to be updated. In the current implementation, the page age is specified by a nibble, limiting the communication of an object profile to several packets. However, this tradeoff forces nodes sixteen or more versions behind require the transfer of all pages in the image regardless of their age.

3.2 The Protocol

A node operates in one of three states at any time: *MAINTAIN*, *RX*, or *TX*. The set of local rules an individual node follows is a function of its current state and specify what actions and state transitions to take in response to each potential event. We formally describe the set of local rules and discuss how each of these rules contribute to the desired global behavior.

3.2.1 Maintenance

The primary responsibility of a node in the *MAINTAIN* state is to ensure that all nodes within its communication range have (i) the newest version of the object profile and (ii) all available data for the newest version. To maintain this property, each node periodically advertises a summary representing the current version of its object profile and the set of pages from the object which are available for transmission. We define a page i as *complete* if every packet for that page has been correctly received. Page i is *available* only if it is complete and all pages in the range $[0, i)$ are also complete. Thus, the summary need only contain two integers $\{v, \gamma\}$, where

v is the version number and γ is the largest numbered page available for transfer.

Ideally, the transmit rate of advertisements in a given cell should be independent of the spatial node density, allow for quick propagation of summaries, and consume few resources in the steady state. This should hold even in the case where thousands of nodes are within the same communication range, a scenario which is representative of programming nodes prior to deployment. To achieve this, Deluge uses Trickle to control the transmission of potentially redundant messages. It divides time into a series of rounds and nodes choose whether or not to broadcast an advertisement during each round. The duration of round i is specified by $\tau_{m,i}$ and is bounded by τ_l and τ_h. In each round, a node maintains a random value r_i in the range $[\frac{\tau_{m,i}}{2}, \tau_{m,i}]$. The local rules for a node with summary ϕ are as follows:

M.1 During round i with start time $t_i = t_{i-1} + \tau_{m,i-1}$, broadcast an advertisement with summary ϕ at time $t_i + r_i$, only if less than k advertisements with summary $\phi' = \phi$ have been received since time t_i.

M.2 If any overheard packet indicates an inconsistency among neighboring nodes (i.e. advertisements with $\phi' \neq \phi$, any requests, or any data packets) were overheard during round i, set $\tau_{m,i}$ to τ_l and begin a new round only if $\tau_{m,i} \neq \tau_l$.

M.3 At the beginning of a round i, if no overheard packet indicates an inconsistency among neighbors during the previous round, set $\tau_{m,i}$ to $\min(2 \cdot \tau_{m,i-1}, \tau_h)$.

Trickle is density-aware in that the threshold k bounds the number of advertisements made in a given cell by suppressing the transmission of redundant advertisements. In a lossless, single-cell network model, an advertisement with a summary $\phi' = \phi$ will only be transmitted at most k times in a period $\frac{\tau_{m,i}}{2}$, independent of the density. In a lossy, multi-cell model, the number of transmissions is bounded by $O(log(n))$, where n is the number of nodes in the cell [8]. Additionally, the dynamic adjustment of $\tau_{m,i}$ in the range $[\tau_l, \tau_h]$ allows for quick propagation of summaries during an upgrade and low resource consumption in the steady state by decreasing and increasing the advertisement period respectively.

With the advertisement service, a node can determine if any of its neighbors have an old object profile. If so, the new object profile needs to be communicated. The local rule governing this process for a node with version v are as follows:

M.4 During round i, transmit the object profile for version v at time $t_i + r_i$ only if an advertisement with version $v' < v$ was received at or after time t_i and less than k attempts to update the object profile to version v have been overheard.

This method for updating object profiles is a form of controlled flood, providing a reliable approach which is density-aware. This is important since the dissemination of the object profile precedes the dissemination of the actual data and nodes must successfully upgrade the object profile before requesting any data. By following the rounds defined by the advertisement service and upgrading an object profile only if less than k redundant upgrade attempts have been transmitted, we keep Trickle's beneficial properties of low redundancy and quick propagation while its epidemic property helps to ensure eventual propagation to all nodes.

The local rules defined so far ensure eventual consistency of object profiles, thus allowing nodes to learn about a newer version and determine which pages need to be updated in order to match the newer version. We now discuss the method used to initiate the reception or transmission of new pages. We first present the remaining local rules and then discuss their contribution to the overall behavior of Deluge. In Deluge, nodes request data from a single node S at a time. The local rules for a node with with summary $\phi = \{v, \gamma\}$:

M.5 On receiving an advertisement with $v' = v$ and $\gamma' > \gamma$, transition to *RX* unless (i) a request for a page $p \leq \gamma$ was previously received within time $t = 2 \cdot \tau_{m,i}$ or (ii) a data packet for page $p \leq \gamma + 1$ was previously received within time $t = \tau_{m,i}$.

M.6 On receiving a request for data from a page $p \leq \gamma$ from version v, transition to *TX*.

We leverage the Trickle suppression mechanisms to help minimize the set of senders and simplify the decision making process of nodes at any given time. The only trigger that causes R to request data from S is the receipt of an advertisement stating the availability of a needed page. Because Trickle bounds the number of transmitted advertisements in a given cell, the set of nodes which may become senders during any period is also bounded. Deluge simply requests data from the node which most recently advertised the needed page. If R overhears a data packet of the needed page, it suppresses any requests for a full round and attempts to snoop as much as possible. This also suppresses the initiation of any additional senders which can interfere with the current set of senders.

One significant contribution of Deluge is its emphasis on *spatial multiplexing*. Deluge advertises the availability of complete pages even before all pages in the object are complete, allowing the further propagation of newly received pages. Throughput can be increased by pipelining the transfer of pages across the network. Without pipelining, propagating the object across a network of d hops would require d complete object transfers, requiring a time of $o(d \cdot S_{obj})$. Instead pipelining the transfer would approach a time of $o(d + S_{obj})$. Intuitively, it is the time for the first bit of data to traverse the network in addition to the time required to flush the pipeline.

Considering that Deluge targets both large object sizes and large scale networks, spatial multiplexing can significantly enhance performance.

In order to realize the full benefit of spatial multiplexing, Deluge takes special care to ensure that transfers of different pages do not interfere with each other. First, Deluge constrains nodes by requesting pages in sequential order, that is, a request for page i cannot be made unless data all pages in the range $[0, i)$ are also up-to-date and complete. This allows neighboring nodes take advantage of the broadcast medium by working together in receiving the same page rather than contending with each other in requesting different pages. An added advantage is that a node need not decide whether to give up an attempt to receive a specific page p and focus its efforts on a different page. Because all nodes complete pages in sequential order, any node advertising $\gamma > p$ is also able to supply page p.

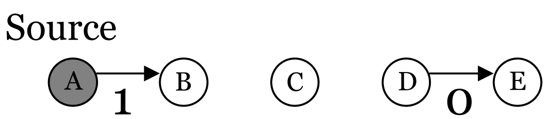


Figure 2: Pipelining. *Example four-hop network showing the effectiveness of spatial multiplexing.*

Second, a transfer of page p will always take higher priority than a transfer of a page $p' > p$. From local rule $M.5$, nodes may not issue a request for a page p if a transfer of page $p' < p$ is in progress. As shown in Figure 2, this constraint reduces interference, including the hidden terminal problem, caused by messages generated from transfers of different pages.

3.2.2 Request

The responsibility of a node in the RX state is to actively make requests for the remaining packets required to complete page $p = \gamma + 1$. Each request operates as a selective negative acknowledgment (SNACK) in which a bit-vector specifies which data packets in the page are needed. To achieve density-awareness, requests are made with a random backoff to help minimize collisions with requests from other nodes and allows for suppression if any requests or data packets are overheard during the backoff period. Utilizing the broadcast medium, responses to requests are shared by all receivers. The suppression mechanism can be considered a special case of Trickle. Specifically, there is no need to worry about the *short-listen problem* [8] since nodes in RX are initially synchronized by the advertisement which caused the transition.

After a node R makes a request, it waits for a response, and if some of the data was lost in the process, R makes a subsequent request for the needed data. Nodes delay subsequent requests until a period of silence equal to ω packet transmit times is detected in order to help ensure that any transmission of data packets has completed before requests are made. In the case of asymmetric links, we also limit a node to λ requests before returning to *MAINTAIN*. This is necessary since a node R receiving advertisements from S may not be able to communicate requests to S. However, if progress (measured as reception rate of data) is above some threshold $0 < \alpha < 1$, Deluge allows the node to continue making requests.

We now present the local rules for a node in RX. For a node n with summary $\phi = \{v, \gamma\}$, we define $p = \gamma + 1$ as the page which n dedicated to receiving and S as the node that caused n's transition into RX.

- $R.1$ After not receiving a request or a data packet for time $t = \omega \cdot T_{tx} + r$, where r is a random value in the range τ_r, transmit a request to node S.
- $R.2$ After λ requests with a packet reception rate of $\alpha' < \alpha$, transition to *MAINTAIN* even if page p is incomplete.
- $R.3$ If all packets for page p are received, mark page as complete and transition to *MAINTAIN*.

It is important to note that a node need not be in RX in order to receive data packets. Deluge takes advantage of the broadcast medium and any packets needed to complete page $\gamma + 1$ are saved regardless of which state the node is in.

3.2.3 Transmit

The responsibility of a node in TX is to broadcast all requested packets for a given page and continue to service any subsequent requests for data from the same page until all packets have been delivered and then transitions back to *MAINTAIN*. Deluge services requests by taking the union of any new requests with previous requests not yet serviced. A C-SCAN schedule is used to provide fairness among requesters. We define Π as the set of packets from page p which have been requested and is initialized to the request which caused the transition to this state. The set of local rules are as follows:

- $T.1$ On receiving a request for a packets Π'_{rx} from page p, set Π_{tx} to $\Pi_{tx} \cup \Pi'_{rx}$.
- $T.2$ Continue broadcasting packets in Π_{tx} with a C-SCAN schedule and removing each broadcast packet from Π_{tx} until $\Pi = \{\}$ then transition to *MAINTAIN*.

3.3 Design Space

The design space for data dissemination protocols is large and includes: methods for suppressing redundant control and data messages, selection of nodes to transmit needed data, use of forward error-correction (FEC), the fragmentation of data to allow for spatial multiplexing, use of link quality estimates or other metrics to improve local decisions, among others. In the early stages of designing Deluge, we experimented with several of these options in simulation. Due to space limitations, we briefly mention some of our findings.

The suppression mechanisms make up most of the complexity in Deluge. To confirm their importance, we tested Deluge without any suppression and it performed so poorly that the simulations did not complete after days of execution. With just request message suppression, Deluge is nearly identical to SPIN-RL. We again tested Deluge by keeping only the request suppression and it also performed poorly, confirming the results presented by SPIN-RL's authors that show it performing worse than naive flooding in lossy network models.

We experimented with suppressing the transmission of data packets if k redundant data packets were overheard while in TX, where lower values of k represented more aggressive suppression. Lower values of k tended to decrease the performance of Deluge. Keep in mind that Deluge already attempts to limit the number of senders. While too many senders leads to high contention and greater opportunities for the hidden terminal problem, a small number (two or three) of neighboring senders is able to cover more area without much loss in performance. This result led to the decision of not employing additional techniques for data packet suppression.

Deluge currently takes a very simple approach to selecting a sender by using the most recent advertisement. We tested more sophisticated methods based on a hop count metric (i.e. requesting data from nodes closest to the source, nodes furthest from the source, and closest neighbor), but found no significant difference in performance for any of these approaches over Deluge's current approach.

We also tested the use of FEC which allow receivers to reconstruct the original data from any k-size subset of the encoded data at the expense of transmitting redundant data. It was interesting to see that FEC improved performance in sparse networks while harming performance in dense networks. The decreased performance in dense networks was due to the existence of highly variable link qualities where nearby neighbors had high link qualities and nodes further away had poor link qualities. FEC works best in environments where loss rates are predictable and have low variance, allowing the amount of redundant data transmitted to be tuned to match the link qualities of the network.

4. EVALUATION METHODOLOGY

The metrics we use to evaluate Deluge are driven by the primary motivation for this work: network programming. We list the metrics we consider, ordered from highest to lowest priority.

1. **Complete Reliability**. Every byte of the data must be correctly received by all nodes.
2. **Completion Time**. In deployments, any interruption to their primary service caused by network programming should be minimized. In the development process, network programming should quickly install updates to shorten debugging and testing cycle.
3. **RAM Usage**. Sensor nodes are severely constrained in the amount of memory available to running application and is compounded by the lack of dynamic memory allocation in TinyOS.
4. **Energy Consumption**. Wireless senor nodes are strictly limited in their energy capacity and a minimal amount of energy should be used in order to lengthen network lifetimes.

The full Deluge protocol as described in this paper is implemented in the nesC programming language on the TinyOS platform. By executing Deluge, we first observe the overall performance of Deluge under different network diameters, densities, and object sizes. We then investigate Deluge's reaction to these parameters by examining the detailed propagation dynamics which occur. Finally, we develop a simple model to help identify factors which limit overall performance.

To evaluate and investigate the behavior of Deluge, we use two separate mechanisms. The first is a TinyOS hardware testbed of modest size, composed of Mica2-dot nodes [14]. The second is TOSSIM, a bit-level node simulator designed specifically for the TinyOS platform [7]. We use these two methods to provide varying degrees of realism and scale: the former providing the most realistic environment while the latter allows us to scale to hundreds of nodes and different node topologies.

4.1 TinyOS Hardware

To gather empirical data, we use a testbed composed Mica2-dots, a TinyOS supported hardware platform. Each node contains a 7MHz, 8-bit microcontroller as a CPU which offers 128KB of program memory and 4KB of RAM; a 512KB external flash chip used for storing application generated data; and communicate via a 433MHz radio transceiver which transmits 19.2Kbit/s [14]. In ideal conditions, the Mica2-dot can transmit about 40 packets per second at a size of 36 bytes each after encoding and media access.

We fully implemented Deluge in nesC and ran experiments using the Mica2-dot hardware platform. We deployed twenty-seven nodes non-uniformly in a 160' by 40' office environment. With the source placed at one end, the diameter of the network varies between three and four hops with a majority of nodes two hops away from the source. To instrument a back-channel, we used specialized hardware to create a UART to TCP bridge, allowing nodes to transmit and receive messages over TCP through an Ethernet adapter. By opening up sockets to each node from a desktop computer, we timestamp each UART message with precision on the order of milliseconds. This information allows us to track the propagation of each page. We note that the UART to TCP bridge is only used as a mechanism for gathering timing information from the network and does not represent any significant source of noise. At the end of each experiment, we poll each node via the TCP connection to verify the integrity of the data object.

4.2 TOSSIM

In addition to hardware experiments, we use TOSSIM, a discrete-event network simulator, to investigate and evaluate Deluge at networks of much greater scale and differing structures. TOSSIM compiles directly from unmodified TinyOS application code and simulates communication between nodes at the bit level. One advantage of TOSSIM is that it simulates the same application logic that runs on TinyOS hardware. In the rest of this section, we focus our discussion on TOSSIM's radio model since the radio is the most significant shared resource of a WSN and is the most important component when simulating Deluge.

Capturing sufficient detail when simulating the communication between nodes is essential since the behavior of dissemination protocols can be highly sensitive to low level factors. We have experimented with evaluating Deluge using high-level simulators, but they were unable to capture unique behaviors which appears when simulating low-level details. TOSSIM captures data-link level network interactions with high fidelity by simulating communication between nodes at the bit level. Simulating communication at the physical layer allows TOSSIM to capture the entire TinyOS network stack and all of its complex behaviors, including the CSMA MAC layer packet transmission delays and backoffs, link-level acknowledgments, packet CRC checks, SECDED packet encoding scheme, sender-receiver synchronization, and hardware-specific timing. The main advantage with simulating at the bit-level is that the transmission and reception of bits govern the actions of each layer, rather than modeling each layer with its own set of parameters.

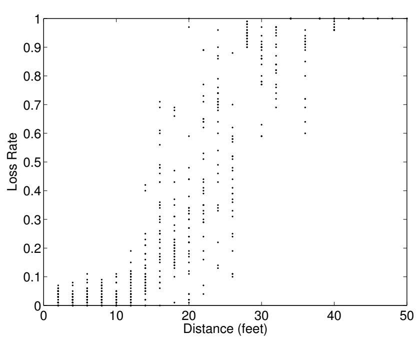


Figure 3: TOSSIM Packet Loss Rates vs. Distance

The network itself is modeled by a weighted directed graph $G = (V, E)$, which is static for the entire duration of the simulation. Each vertex $v' \in V$ represents a node and each edge $(u, v) \in E$ specifies a bit-error rate in the direction from u to v. The bit-error rate represents the probability a given bit is flipped while in transmission from u to v. Each bit-error rate is independently chosen based on the distance between u and v and a random distribution derived from empirical data collected on Mica nodes. Figure 3 shows this model's packet loss rate over distance from an experiment in TOSSIM. Sampling the bit-error rate for each edge independently allows for asymmetric links where the a bit-error rate for (u, v) is significantly different than (v, u). This is important since asymmetric links are a real problem in WSNs.

TOSSIM treats the transmission of each individual bit as an event. If a node receives bits from multiple senders at the same time, the union is taken as the result. In doing so, real-world wireless problems dealing with interference and the hidden terminal problem can be observed in simulation. TOSSIM does make some significant simplifications from the real world. For example, the transmission strength for each node is uniform within a 50 foot radius. This implies that while a close node may have a lower bit-error rate, it is unable to overpower the signal of a further node. It is important to note that this aspect of TOSSIM is overly pessimistic when compared to the real world where signal strength can fade at a polynomial rate with distance.

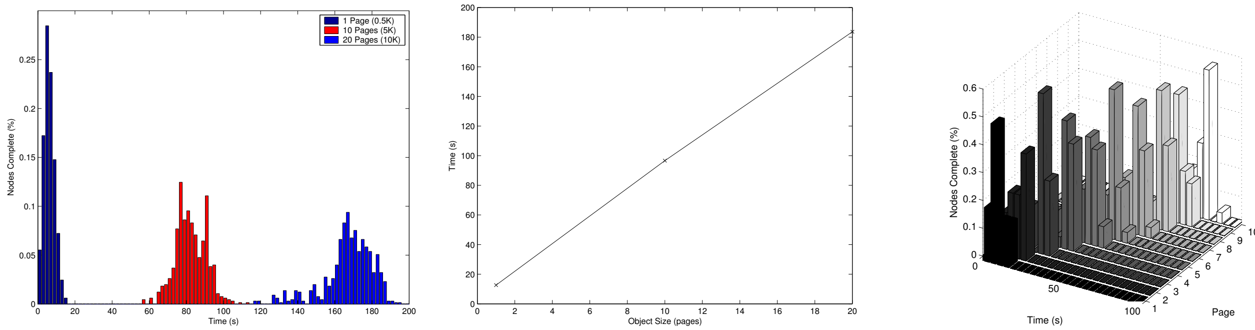
5. PROPAGATION

5.1 Empirical Results

The procedure of each experiment follows that of a normal deployment scenario. In its initial state, all but one of the nodes are deployed and operating in the steady-state, meaning that they have the same version v of the object profile and the same set of completed pages for version v. The remaining node acts as the source node. Initially disconnected from the deployment, a new image with version $v' > v$ and all page ages set to 0 is downloaded to the source via a physical connection from a desktop computer. The experiment begins by introducing the source into the twenty-six node deployment, beginning the dissemination process. We measure the time to propagate each page to individual nodes, relative to the first advertisement made by the source. No concurrent services, other than those required by Deluge, execute for the duration of each experiment. In the current implementation, each page is 528 bytes with 24 data packets per page and each data packet has a data payload of 22 bytes. For the advertisement service, we set $\tau_l = 2$ seconds, $\tau_h = 60$ seconds, and $k = 1$. For requests, we set $\tau_r = 0.5$ and $\omega = 8$.

In every test, the data was correctly and completely received by all nodes in the network. The empirical results are shown in Figure 4 and represent 25 experiments at object sizes of 1, 10, and 20 pages. Figure 4(a) shows the distribution of finishing times (the time to receive all pages) for individual nodes. The distribution tends to look like a normal distribution because the majority of nodes lie two hops away from the source in the three hop network. It is clear that as the object size increases, so does the variance in finishing times.

The increase in variance is due to the variation in link qualities between nodes, where nodes with good paths from the source receive the image quicker than those



(a) Total propagation time for entire object.

(b) Varying object size.

(c) Propagation time for each page.

Figure 4: Empirical Propagation Time. *This data represents 25 experiments at object sizes of 1, 10, and 20 pages.*

with poor paths. Some nodes three hops away had poor connectivity and took significantly more time to receive each page. At the same time, nodes within direct radio connectivity of the source received the image quickly. The leftward tail that appears in the 20 page case is attributed to the combination of nodes with poor connectivity three hops away and the mechanism to help prevent interference between transfer of different pages when pipelining. The mechanism causes those nodes with poor connectivity to prevent two-hop nodes from making requests for additional data, while the one-hop nodes are free to do so.

As shown in Figure 4(b), the average time for all nodes to receive the entire object is linear with the size of the object. In Figure 4(c), we plot the distribution of finishing times for each page for a 10 page object. The propagation time for each page is approximately equal. We do not see the full effects of pipelining because the network diameter is limited to four hops.

5.2 Simulation Results

While the empirical results are promising, we were unable to experiment with networks of large scale since the testbed configuration does not scale easily. Instead, we used TOSSIM to evaluate and investigate the behavior of Deluge with network sizes on the order of hundreds of nodes and tens of hops. This section starts by evaluating the propagation performance with different network diameters, densities, and object sizes for a square topology. We then examine the propagation dynamics in detail to understand how Deluge reacts to the different parameters. Finally, the overall performance of Deluge in a linear network is provided and a simple model is derived to help identify the different factors which limit overall performance.

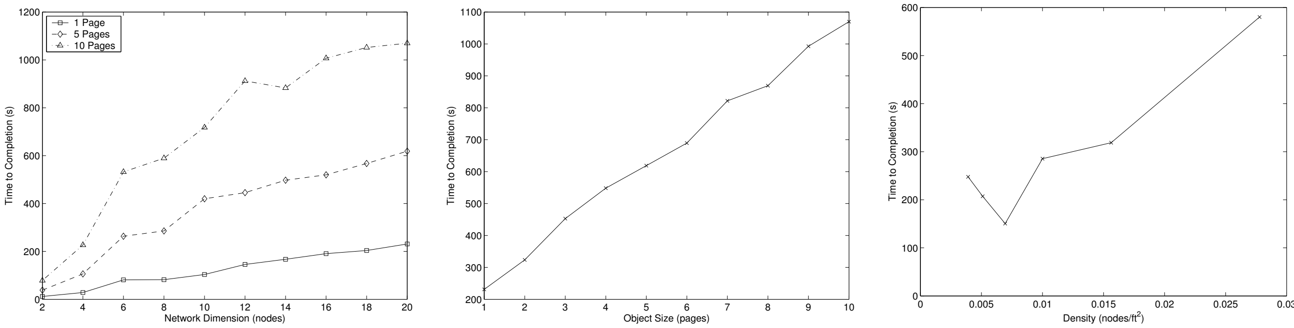
5.2.1 Overall Performance

We briefly discuss the overall performance of Deluge for a square topology. Due to execution times on the order of tens of hours for each simulation, we do not provide confidence intervals. We begin by executing application-level code identical to that used in the empirical experiments, including all parameter values. Because Deluge requires nodes to keep their radios on, the vast majority of energy is spent in the idle-listening state where the radio continuously listens to the channel. Thus, we focus on completion time as it closely resembles energy consumption.

Figure 5(a) shows the total propagation time for different object sizes in square topologies of different diameters. Node density is kept constant with nodes spaced 15 feet apart. At network diameters less than 8, the propagation time is clearly a function of the *product* of the network diameter and object size. With network diameters greater than 8, the slope of the propagation times for the multi-page objects approaches that of a single page, indicating the effectiveness of the pipelining (i.e. the propagation time is the *sum* of the network diameter and object size). Figure 5(b) plots the propagation times for various object sizes in a 20×20 grid topology, showing that propagation time is linear with object size. Figure 5(c) shows that an increase in density harms overall performance. In the next section, we investigate the propagation dynamics of Deluge to see why density affects performance.

5.2.2 Dynamic Behavior

In this section, we investigate the propagation behavior of Deluge for a square topology. Figure 6 shows the propagation of a single page from a corner node through a 20×20 network with nodes spaced 15 feet apart for a single experiment and is representative of all other experiments of similar topologies. Figures 6(a)-6(c) show a time series of the propagation while Figure 6(d) summarizes the completion time for each node. With this topology, the propagation behaves as expected: the propaga-



(a) Varying network size. (b) Varying object size. (c) Varying node density.

Figure 5: Simulated Propagation Time for Square Structures. *These results are from a $N \times N$ network where $N = 2, 4, 8, \ldots, 20$ and spacing 15. Figure (c) is for a $180' \times 180'$ field for different densities.*

tion progresses at a fairly constant rate in a nice wavefront pattern from corner to corner. The irregularities are due to the non-uniform loss rates and contention.

An interesting behavior emerges as we increase density. To show this behavior, we repeat the experiment with a 20×20 network with adjacent nodes spaced 10 feet apart, increasing the density by $2\frac{1}{4}$ times. As shown in Figure 6(e), the propagation begins as it did in the sparse case. But soon after, the propagation along the diagonal begins to slow significantly while quick propagation along the edge continues and completely wraps around the edge before filling in the middle

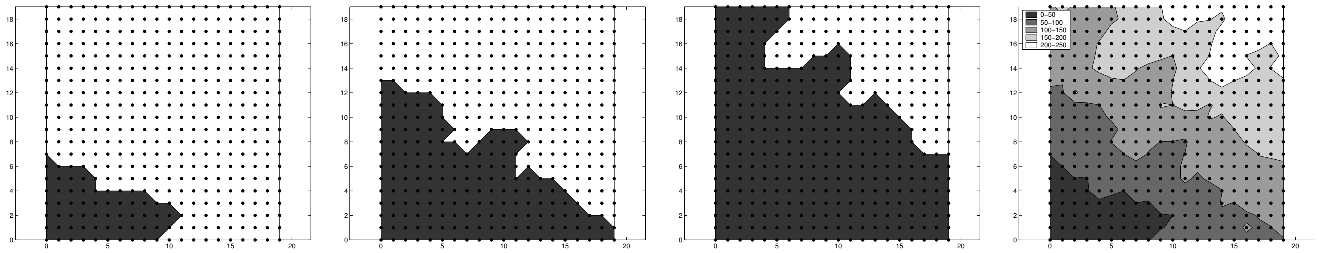
To see the behavior with pipelining, Figure 7 shows the propagation of a five page object in both the sparse and dense case. With multiple pages, the complex rate of progress becomes apparent even in the sparse case. If we take a closer look at the sparse case when propagating a single page, the propagation along the diagonal is actually slightly slower than along the edge, which accounts for the linear wavefront shape. The difference in propagation times is not enough to clearly show the behavior as experienced by the dense case. However, when propagating multiple pages, the delay in reaching the middle nodes accumulates over time and thus forms the same pattern as in the dense case. In the dense case, the behavior becomes even more pronounced.

To examine this behavior in greater detail, we look at the rate of propagation for a single page along the edge and diagonal, shown in Figure 9(a). We use the propagation times from only those nodes on the bottom edge (i.e. $(i, 0)\forall i \in [0, 19]$) and along the diagonal (i.e. $(i, i)\forall i \in [0, 19]$) to represent the propagation along the edge and diagonal respectively. The slope of the plotted times represents the propagation rate. Early in the process, the propagation rate along both the edge and diagonal are identical. However, the propagation rate along the diagonal drops to nearly 20% of the original rate once it reaches node $(5, 5)$. Note that the propagation rate remains fairly constant after the drop.

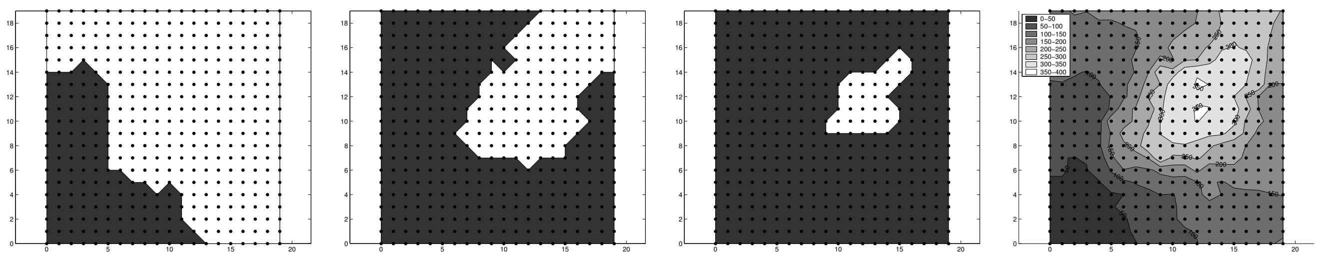
The root cause of this behavior is the *hidden terminal problem*, which occurs when two nodes A and C communicating within the range of node B are unable to coordinate their transmissions, thus causing collisions when transmitting to B. Nodes in the center of the network have more neighboring nodes and experience a greater probability of collisions than those on the edge of the network. Considering that the interference range modeled by TOSSIM is 50' and that nodes are spaced 10' apart, a significant slow down at node $(5, 5)$ is understandable since it, along with all other nodes in the center, experiences the greatest number of nodes within the interference rage.

To see the effects of interference, we count the number of transmission events within the interference range and the number of messages overheard which pass the CRC check. Figures 8(a) and 8(b) show the number of each message type sent in a 10 second window within the interference range of nodes $(2, 2)$ and $(5, 5)$ respectively. Node $(5, 5)$ clearly shows a higher number of sustained transmissions within its interference range, causing a higher likelihood of collisions. Figures 8(c) and 8(d) show the effects of the interference by plotting the number of messages overheard which pass the CRC check. The data shows that even though the rate of transmissions within node $(5, 5)$'s interference range is greater, the rate of correctly received messages is, at best, half of node $(2, 2)$.

Deluge's reaction to collisions is compounded by a more subtle problem. Recall that Deluge achieves its density-aware property by taking advantage of the broadcast medium to overhear similar packets and suppresses redundancies by employing a linear backoff scheme. Thus, it relies on overhearing packets to estimate node density. When the channel is pushed to saturation, the high number of collisions can cause such a mechanism to severely underestimate the number neighbors, stimulating transmission of more redundant messages, causing more collisions, and leading to congestion collapse. While Del-



(a) $t = 50$ (b) $t = 100$ (c) $t = 150$ (d) Summary



(e) $t = 100$ (f) $t = 200$ (g) $t = 300$ (h) Summary

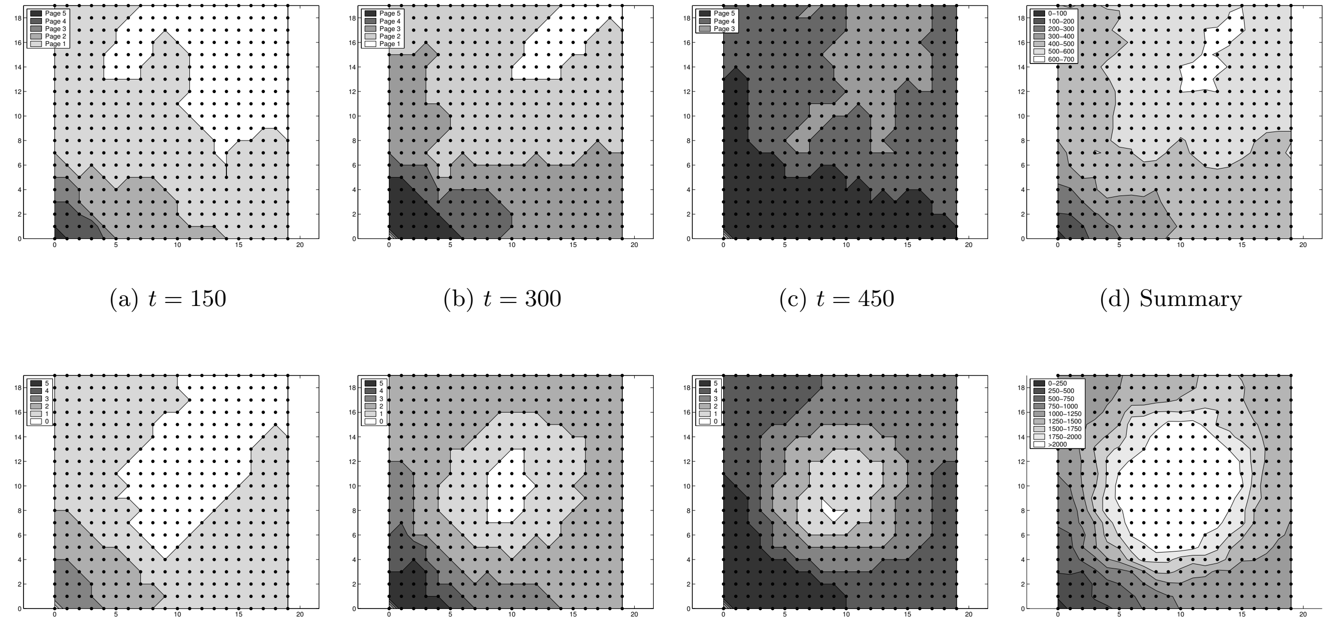
Figure 6: Simulated Propagation Time for 1 Page in a 20×20 Grid Topology. *Figures (a) - (d) and (e) - (h) are from a network with node spacings 15 and 10, respectively.*

uge borrows the suppression mechanisms from Trickle, it differs in that it operates near channel capacity to quickly disseminate large amounts of data. The authors of Trickle studied its performance with a relatively low data rate. Figure 8(b) shows that the advertisement rate increases in proportion with other activity, providing evidence that Deluge is underestimating the node density.

To test how Deluge performs when the channel is not pushed to saturation, we increase the backoff time for a request to decrease the maximum rate. We chose to vary the request backoff time for several reasons. First, decreasing the rate of requests has the effect of decreasing the rate of invoking nodes to begin transmitting data. Second, in contrast to data messages, request messages can be transmitted by any node needing data. With a large set of requesters, request messages are more likely to increase collisions due to the hidden terminal problem. Because requests are unicast to the node that most recently advertised, it is unlikely for many senders in a region to begin transmitting data. Finally, Figure 8(d) clearly shows that the amount of request traffic can significantly overcome the amount of data traffic for those nodes which experience slow propagation times. By decreasing the request traffic, useful data traffic can utilize more of the channel.

Figure 9(b) compares the propagation of a single page along the edge and diagonal for a τ_r of 1 second with the original τ_r of 0.5 seconds. By doubling τ_r, the propagation rate along the diagonal improves by about 2.7 times while the propagation rate along the edge remains nearly identical, leading to an improvement in overall propagation performance. With these improved effects, we continue to increase τ_r to 2 seconds. Figure 9(c) compares the propagation of a single page along the edge and diagonal for a τ_r of 2 seconds with the original τ_r of 0.5 seconds. While the propagation rate across the diagonal is slightly improved over the original test, it does not match Deluge's performance when τ_r is 1 second. The propagation rate along the edge is worse than either of the other tests. This leads to lowered overall performance. Note that the propagation rate across the diagonal only experiences a 28% rate reduction relative to the edge rather than the 80% reduction seen in the original test. This shows that lowering channel utilization is effective in eliminating the hidden terminal problem and minimizing the difference in propagation rates between the edge and diagonal. However, it is also at odds with lowering the overall propagation rate, a primary goal of Deluge.

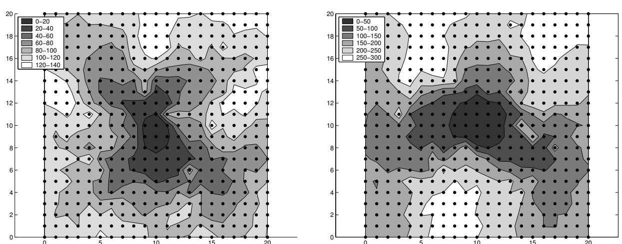
In each of the setups thus far, the propagation began at a corner node. We now examine the behavior when the source node is placed at the center of the network. One might suggest that starting the propagation in the center might help to eliminate the behavior of following the edge and also decrease the propagation time by about half. We repeat the simulations with τ_r at 0.5 seconds except with a 21×21 grid and the source node at the center of the network rather than at the corner.



(a) $t = 150$ (b) $t = 300$ (c) $t = 450$ (d) Summary

(e) $t = 200$ (f) $t = 500$ (g) $t = 1000$ (h) Summary

Figure 7: Simulated Propagation Time for 5 Pages in a 20×20 Grid Topology. *Figures (a) - (d) and (e) - (h) are from a network with node spacings 15 and 10, respectively.*



(a) Spacing 15. (b) Spacing 10.

Figure 10: Simulated Propagation for 1 Page from the Center in a 21×21 Grid Topology.

Figure 10(a) shows the propagation behavior for the sparse case (spacing 15). The time to reach all nodes is reduced by approximately 40%. Additionally, the propagation still does not behave in a nice circular manner. One cause is Deluge's depth-first tendency, where propagation of a single page along good links is not blocked by delays caused by poor links. Notice that the propagation speeds up as it approaches the edge of the network. Figure 10(b) shows the propagation behavior for the dense case (spacing 10). The time to reach all nodes is reduced by approximately 25%, significantly less than the expected 50%. Even though placing the source at the center effectively reduces the the diameter by about half, Deluge is unable to take advantage of the quick edges since nodes in the center experience a greater number of collisions. However, once the propagation reaches an edge, it begins to speed around the edge as before.

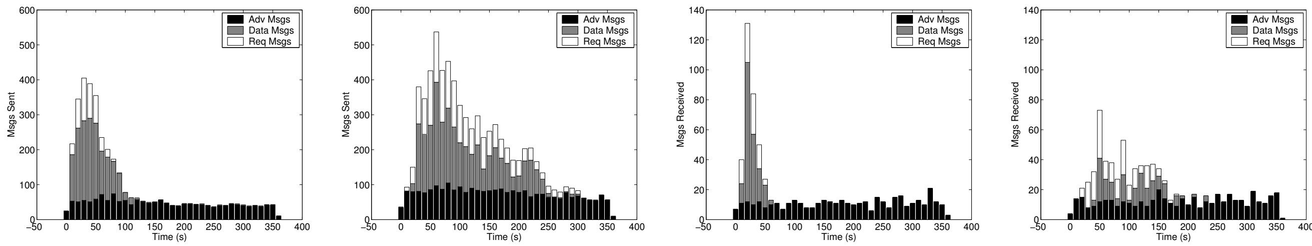
5.2.3 Linear Structures

We briefly discuss the Deluge's overall performance in linear networks. Figure 11 shows similar tests as in the square topology case but for a $2 \times N$ topology, where N ranges from 4 to 76 nodes. The linear topology can be considered a special case of the grid by representing just the edge. Figure 11(a) shows the effectiveness of the spatial multiplexing and Figure 11(b) shows that propagation time is linear with object size. Unlike the square topology, Figure 11(c) shows that an increase in density actually improves performance.

5.2.4 Model

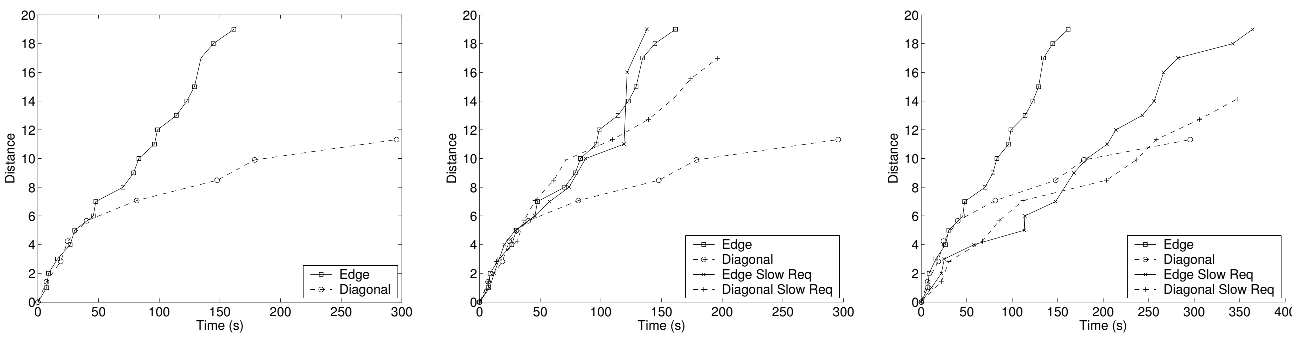
For the linear case, the simulations show that Deluge takes about 40 seconds to disseminate each page to 152 nodes across 15 hops. Considering that the link bandwidth between two nodes can reach 32 packets/second, this seems suboptimal, only achieving 4% of this rate. To help understand this difference and the various factors which contribute to it, we develop a simple model of the Deluge process for the linear case.

First, consider a hypothetical scenario where packets can be sent across a multihop network at full speed, 32 packets/second. Unlike wired networks, the broadcast



(a) Node $(2, 2)$. (b) Node $(5, 5)$. (c) Node $(2, 2)$. (d) Node $(5, 5)$.

Figure 8: Number of Messages Sent and Received. *Figures (a) and (b) show the number of messages sent within the interference range. Figures (c) and (d) show the number of messages overheard by the node.*



(a) $\tau_r = 0.5$ s. (b) $\tau_r = 1$ s. (c) $\tau_r = 2$ s.

Figure 9: Simulated Propagation Rate of a Single Page. *Figure (a) shows the propagation rate through the diagonal and edge while Figures (b) and (c) compare the propagation rate for different values of τ_r.*

nature of wireless prevents any multihop communication protocol operating on a single-channel radio network from achieving this rate. Recall from Figure 2 that a three hop spacing is required to prevent collisions from simultaneous transmissions. Thus, any multihop communication scheme is limited to 33% of the hypothetical bandwidth.

Assume that the expected time to transmit a page is $E[T_{tPage}]$ and the delay between completing a page and requesting a new page is D_{newReq}, the expected time to transmit an object of n pages across a d hop network is

$$E[T_{obj}] = \min(d \cdot n, d + 3(n-1)) \cdot (E[T_{tPage}] + D_{newReq}). \quad (1)$$

Next we define the expected time required to actively transmit all data packets. Given a packet loss rate $E[r_l]$, we define the expected number of transmission for a given packet as $E[N_{tPkt}] = \frac{1}{1-r_l}$. The expected time required to transmit just the data packets is

$$E[T_{tx}] = E[N_{tPkt}] \cdot T_{tPkt} \cdot N \quad (2)$$

where T_{tPkt} is the transmission time for a single packet. Given that $T_{tPkt} = 32.2$ ms and $E[r_l] = 0.1$, as derived from the simulation data, the value of $E[T_{tx}]$ is relatively small.

Much of the remaining difference from the hypothetical model is due to the various delays and backoffs within Deluge. The first source of delay is the random backoffs before transmitting requests, where the expected backoff delay of $E[\tau_r] = \frac{\tau_r}{2}$. When receiving a page, the expected time spent backing off while making requests is

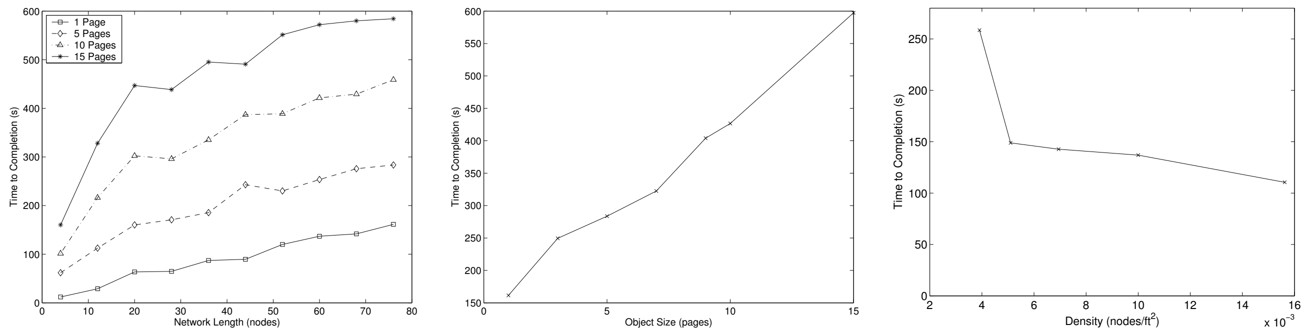
$$E[T_{req}] = E[N_{tPkt}] \cdot E[N_{reqs}] \cdot E[\tau_r], \quad (3)$$

where $E[N_{reqs}]$ is the expected number of requests a node must make to complete a given page.

The second source of delay comes from the relatively low rate of advertisements. The expected period for receiving advertisements from nodes closer to the source is given by

$$E[T_{rAdv}] = E[N_{tPkt}] \cdot \frac{\tau_l}{2} \cdot (1 + E[N_{Supp}]) \quad (4)$$

where $E[N_{Supp}]$ specifies the expected number of times that an advertisement from hop h is suppressed by hop $h-1$ before transmitting an advertisement (in the linear case, $E[N_{Supp}] = 1$). Assume node R is at hop $h+1$ while S is at hop h. After S acquires new data to



(a) Varying network length. (b) Varying object size. (c) Varying node density.

Figure 11: Simulated Propagation Time for Linear Structures. *These results are from a $2 \times N$ network where $N = 4, 12, 20, \ldots, 76$ and spacing 10.*

transmit, the expected time for R to learn that S has new data and transition to RX is $E[T_{rAdv}]$.

Additionally, when a node exceeds its limit of λ requests, it must transition to *MAINTAIN* and wait for another advertisement before making additional requests. Thus, the expected amount of time spent waiting for additional advertisements when the request limit of λ is exceeded is

$$E[T_{tGiveUp}] = E[N_{tPkt}] \cdot \left(\frac{E[N_{reqs}]}{\lambda} - 1 \right) \cdot E[T_{rAdv}]. \quad (5)$$

We now define $E[T_{tPage}]$ (the expected time required to transmit a page across a single hop) used in (1) as

$$E[T_{tPage}] = E[T_{rAdv}] + E[T_{tx}] + E[T_{req}] + E[T_{tGiveUp}]. \quad (6)$$

We test the accuracy of the model by fitting it with the simulated data from the linear case. The parameters $N_{hops} = 15$, $T_{tPkt} = 0.0322$ seconds, $r_l = 0.1$, and $E[N_{reqs}] = 5.4$ represent averages derived from the simulated data while the remaining parameters are identical to the ones used in the experiments. Figure 12 shows that the model fits fairly well the simulated data.

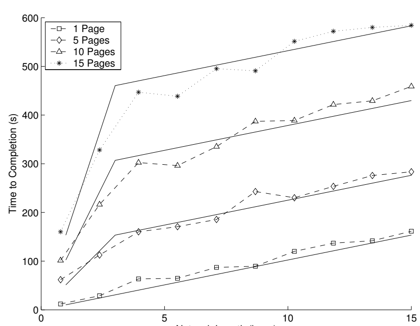


Figure 12: Comparison of Model with Simulation. *The solid lines represent the predicted times from the model.*

Pipeline	$E[T_{req}]$	$E[T_{tGiveUp}]$	$E[T_{rAdv}]$	$E[T_{tx}]$
66%	14%	11%	5%	4%

Table 1: Contributions to Reduced Bandwidth from the Hypothetical Scenario.

From the model, the items which contribute to the difference from the hypothetical scenario are shown in Table 1. As mentioned, pipelining alone accounts for 66% of the difference and is a fundamental limit of any multihop communication protocol in single-channel, wireless networks. The remaining 30% of the difference is due to the delays and backoffs in Deluge. These delays and backoffs represent a tradeoff: while we would like to minimize these delays to increase bandwidth, they are also necessary for suppression and without suppression, contention would slow propagation. The magnitude of these delays should really be a function of neighboring node density. Thus, there is an inherent limit to the dissemination rate which is much less than that of simply routing along a single path. With a greater density, a larger delay period reduces any collisions caused by the hidden terminal problem. We have tested Deluge in very sparse cases with low delay settings and it is able to achieve much higher transfer rates.

6. LOOKING FORWARD

While Deluge provides good, robust performance, there is room for potential improvement. Ideally, Deluge should operate at maximum channel capacity, but never exceed this limit. The results suggest that a more aggressive mechanism to avoid congestion collapse is required in order to accurately estimate the node density. One possibility is to employ an exponential backoff scheme rather than a linear backoff. Dynamically adjusting the backoffs may have the additional effect of minimizing wasted time caused by backoffs. In areas where nodes are sparse, the backoffs can be relatively small while contention remains low and the suppression mechanisms remain effective.

This raises an important open question: What is the true limit on the rate of dissemination in wireless networks? It is clear that this bound is much lower than is the case for simply routing a message across the network. The potential collisions between nodes when disseminating data leads to the use of selective and delayed retransmissions, implemented in Deluge through the use of suppression mechanisms. The difference in square and nearly-linear geometries suggest that these delays may need to adapt to larger topological factors.

The increased performance along the edge introduces an interesting concept for disseminating large data objects. Ideally, we would like to duplicate the beneficial edges in the center of the network. One way is to select a linear set of nodes through the center of the network to initially participate in the dissemination process. This allows for quick propagation across the center of the network, splitting the network in two. We might recursively continue this process, essentially creating a fractal structure for propagation. With this process, the propagation time for a single page should be much improved. However, it inhibits the use of pipelining since it can be difficult to enforce spatial multiplexing for transfers between different pages. Also, methods for creating these paths in a distributed manner need to be explored.

The propagation dynamics described are determined by very low level factors and hence may be influenced by the simulator itself. Because TOSSIM models a uniform signal strength within a 50' radius, the impact of interfering transmissions may be overestimated. However, we are confident that such effects should occur at some density in real deployments and that the actual density may depend heavily on the way the network behaves. While previous work has hinted at similar behaviors in real deployments, the effects are too complex to state anything conclusive and only small amounts of data are disseminated. We are currently building a testbed to examine Deluge's propagation dynamics at much larger scales.

7. CONCLUSIONS

In this paper, we presented *Deluge*, a reliable data dissemination protocol for propagating large data objects from one or more source nodes to all other nodes over a multihop WSN. With its density-aware, epidemic mechanisms, we have shown that Deluge can reliably disseminate data to all nodes. The current version exposes propagation dynamics only hinted at by previous work. These propagation dynamics show the impact of the hidden terminal problem on data dissemination. We presented a simple model of Deluge's propagation behavior, describing factors which limit its propagation performance. Unlike wired networks, data dissemination protocols cannot achieve an aggregate bandwidth equal to the link capacity due to spacing required in spatial multiplexing and delays necessary to allow for suppression. The actual bound for any dissemination protocol remains an open question.

8. REFERENCES

[1] 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.

[2] 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.

[3] J. Jeong, S. Kim, and A. Broad. *Network Reprogramming*. University of California at Berkeley, Berkeley, CA, USA, August 2003.

[4] 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.

[5] 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.

[6] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, pages 85–95. ACM Press, 2002.

[7] 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.

[8] 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, University of California at Berkeley, 2004.

[9] 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.

[10] 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.

[11] 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.

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

[13] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, Berlin, Germany, Jan. 2004.

[14] University of California, Berkeley. Mica2-dot schematics. `http://webs.cs.berkeley.edu/tos/hardware/design/ORCAD_FILES/MICA2/6310-0306-01ACLEAN.pdf`, March 2003.

[15] University of California, Berkeley. Tinyos. `http://www.tinyos.net/`, 2004.

[16] 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.