



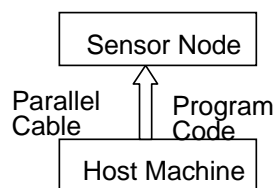
Network Reprogramming at Scale

NEST Retreat January, 2004

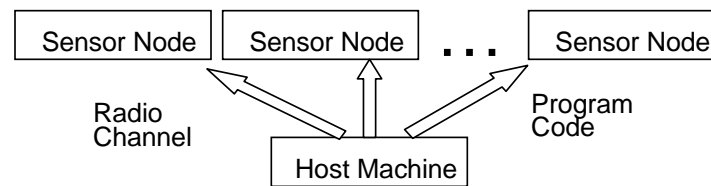
Jonathan Hui
Jaein Jeong
Gilman Tolle

Programming wireless sensor nodes

- In-System-Programming (ISP)
 - Code is loaded to a sensor node through parallel or serial cable directly connected to the host.
 - Can be a problem with a large number of sensor nodes.
- Network Reprogramming
 - Code is sent to multiple nodes through radio.
 - Add/change functionality for large number of nodes over a wide area.
 - Increased productivity for debugging/testing



Parallel programming



Network programming



Organization

- Network reprogramming implementation in TinyOS 1.1 is limited.
- Node-level Representation and System Support
 - Supporting incremental updates
 - Jaein Jeong
- Deluge: Data Dissemination Protocol
 - Multihop code delivery
 - Jonathan Hui, Gilman Tolle

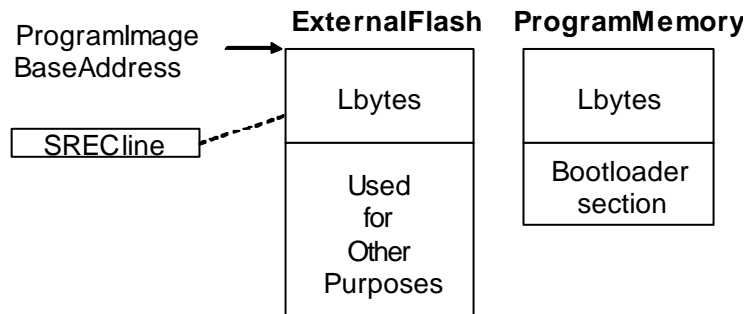


Node-level Representation and System Support

Jaein Jeong

Network Programming Concept

- Unlike ISP, in network programming, a sensor node cannot write program code directly to the program memory.
- Network programming is done in two steps:
 - Code Delivery
 - Program code is saved in external storage (external flash in XNP).
 - Possible retransmission of lost packets.
 - Program Loading
 - The boot loader loads program code to program memory.
 - Program starts after reboot.



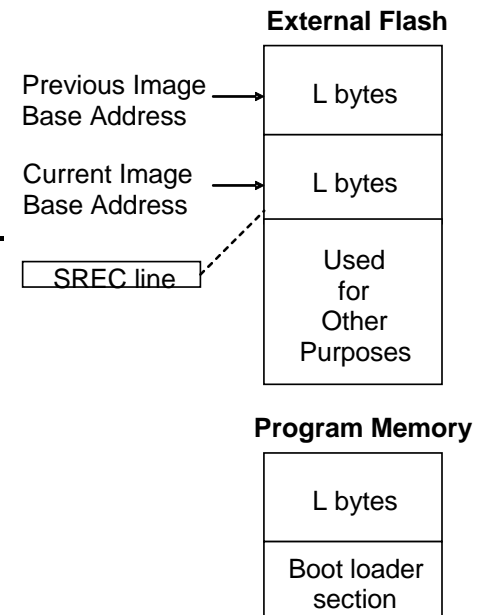


Design considerations for incremental update

- Simple node operations
 - The program history is maintained in the host.
 - The host compares two program images in fixed sized blocks and sends a block or sends a “copy” msg as update script.
 - Sensor nodes builds program image based on update script.
- Reuse the existing infrastructure
 - A contiguous memory space is allocated for the new and the previous program images
 - Supports non-incremental network programming with minimum efforts.

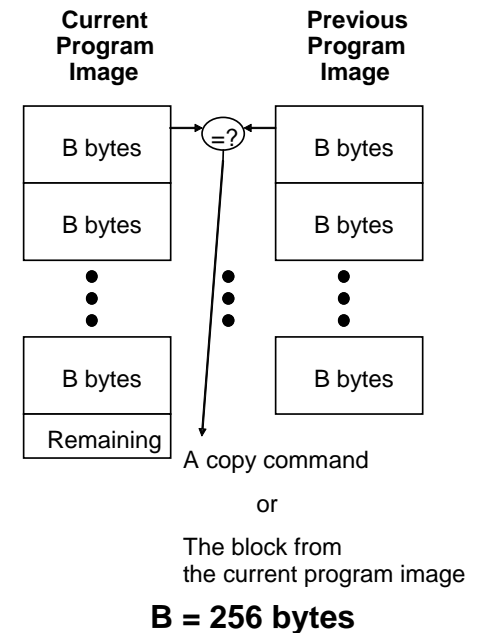
Design - Memory Layout

- The program storage (external flash) is organized in two sections: current and previous sections.
- In each section, a line of the program binary code (SREC file) is stored as a 32-byte record.
- Boot loader is modified so that it can copy code from any section on external flash.



Design - Generating update

- The host generates update script by comparing program images in fixed size blocks.
 - Current and previous image blocks are compared.
 - If the two blocks match, the host sends “copy” command to sensor nodes.
 - Otherwise, the host sends the data bytes for the current program block.





Experiments – sample applications

- We used the two sample applications:
 - XnpBlink and XnpCount.
 - simple applications to demonstrate the use of network programming.
 - Application specific code takes less than 10% of the total source code.
 - XnpBlink: 157 lines out of 2049
 - XnpCount: 198 lines out of 2049



Experiments – test scenarios

- For evaluation, we considered the following cases:
- Case 1 (Minimum program change)
 - Two copies of XnpBlink only different in the timestamp embedded during compilation time.
- Case 2 (Small change)
 - Modified a line in XnpBlink source code to change its LED blinking interval.
- Case 3 (Major change)
 - We sent the difference in the code for XnpCount from XnpBlink application.



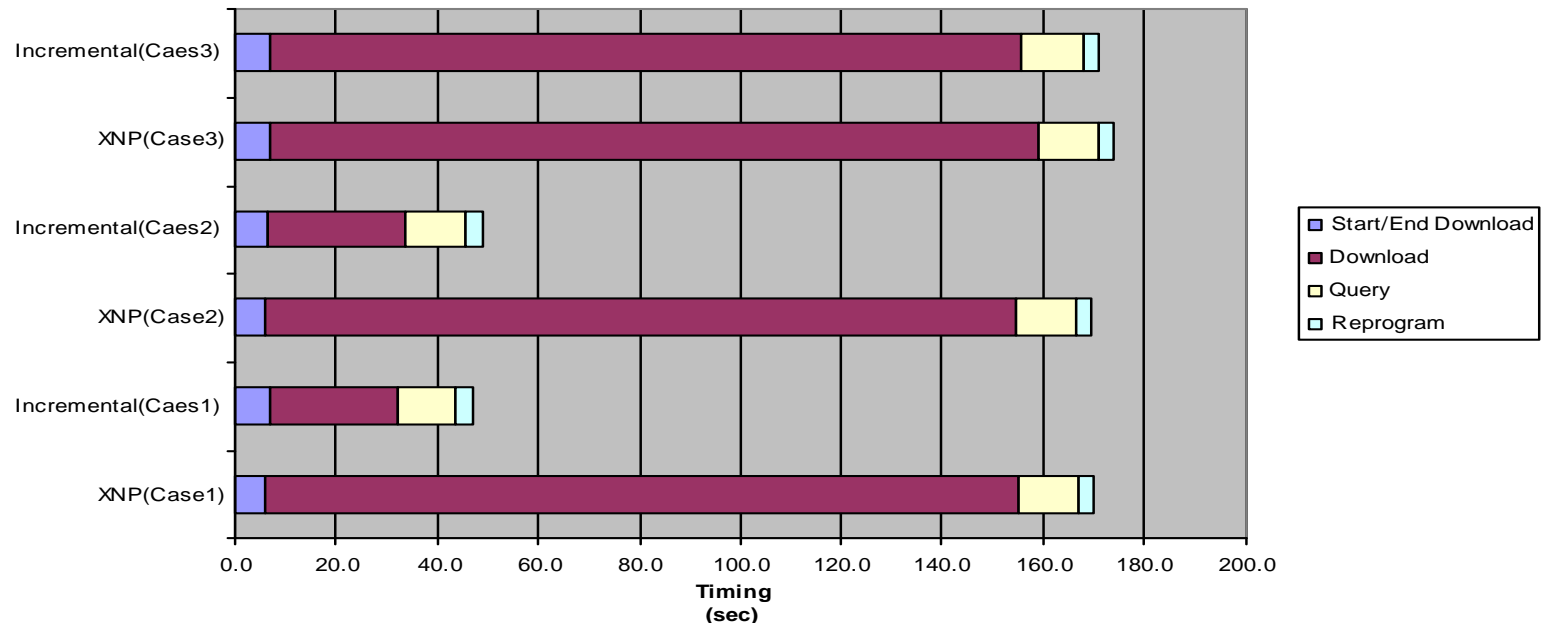
Experiments

- The transmission time depends on number of blocks common between two program images.
- The following table shows how many lines (1 block = 16 lines) are different between the previous and the current program image for three test scenarios.

	Case 1	Case 2	Case 3
Lines in SREC file (previous image)	1139	1139	1139
Lines in SREC file (new image)	1139	1139	1166
Different Lines	1	2	1066
Common Lines	1138	1137	100
Blocks to transmit	1	2	71
Blocks to copy on the sensor node	71	70	2

Experiments – Results

- For case 1 and 2, where most blocks between the two programs are the same, the incremental network programming ran much faster than XNP (3.6 times for case 1 and 3.5 for case 2).
- For case 3, where most blocks are not common, the programming time was close to that of XNP.





Discussion and Future Works

- Not much of code is shared in binary files.
 - Our approach reduced code propagation traffic for small change of code, but was not effective for more general cases.
 - Needs better compiler support for maximizing binary code reuse.
- Separate code blocks for the current and the previous versions.
 - Programming time increases due to overhead of copying blocks.
 - Needs a level of indirection for the program storage.



Deluge:

Dissemination Protocol

Jonathan Hui

Gilman Tolle



Deluge: Goals

- Create a multi-hop dissemination protocol for program data
 - Program size (128k) \gg RAM (4k)
 - 100% Reliability required
- Minimize energy
- Minimize time-to-completion
 - Time for all nodes in a connected network to receive all required data
 - Useful for debugging/testing applications
- Minimize RAM requirements

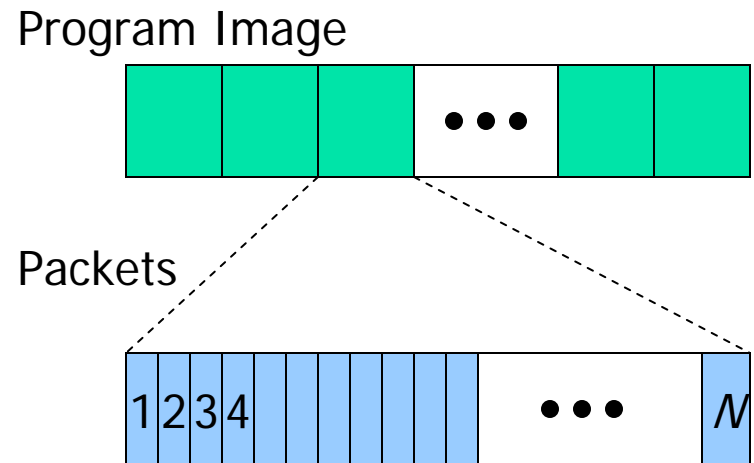


Deluge: Assumptions

- Connected network
- All nodes execute same code
- Received packets from network stack are not corrupted
- Most nodes have already received version n before $n+1$
 - For efficient incremental upgrades

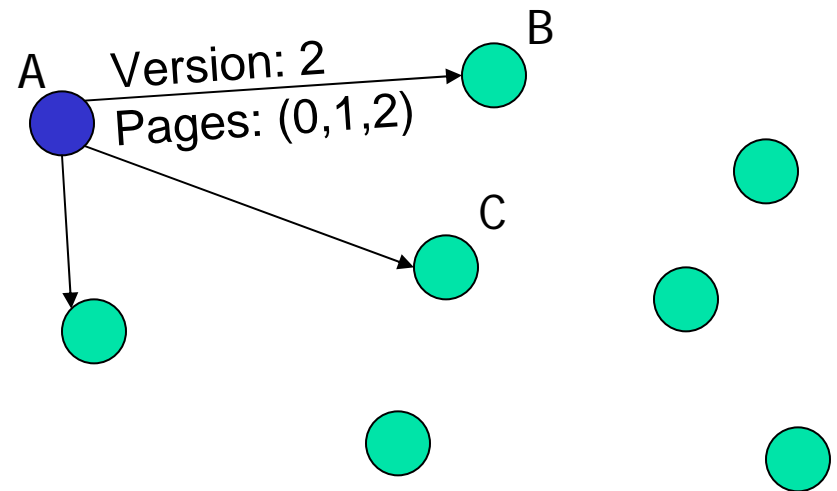
Deluge: Data Representation

- Program image divided into contiguous *pages*, each consisting of N *packets*.
 - Page-size buffers reasonable to allocate in RAM
- *Packets* are of fixed size k
 - k chosen to fit in packet supported by network.



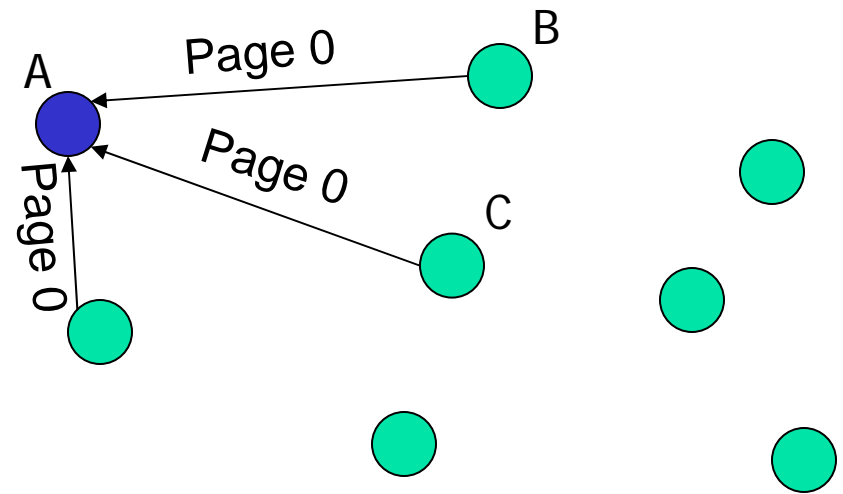
Deluge: Basic Protocol (1)

- Periodically Advertises
 - Version Number, bit-vector describing set of pages available to send
 - Delay between advertisements varied to conserve energy and lower channel utilization



Deluge: Basic Protocol (2)

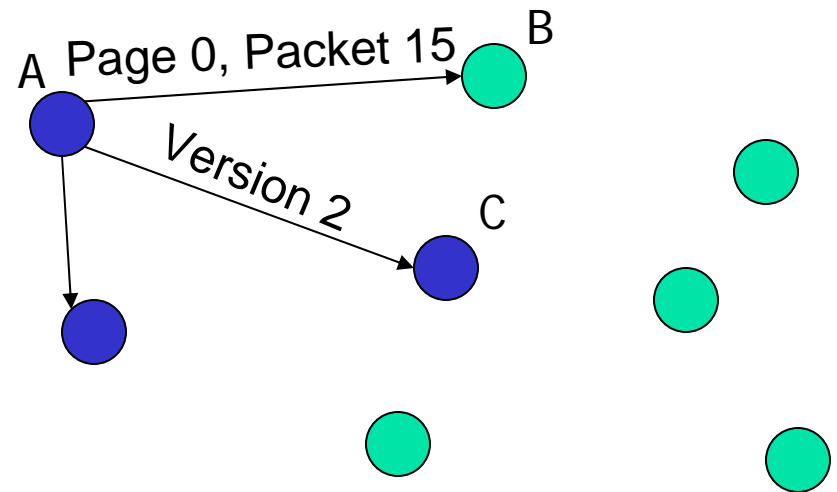
- Request
 - When a node needs to upgrade the image to match a newer version
 - Requests lowest numbered page required
 - Version number, page number, bit-vector of required packets



Deluge: Basic Protocol (3)

■ Broadcast Data

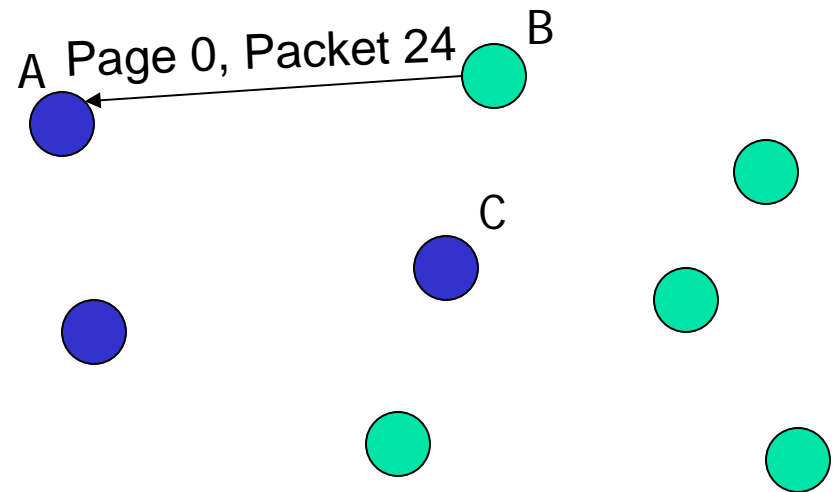
- All requested packets lowest numbered requested page sent
- Version number, page number, packet number, data.
- Allows for snooping



Deluge: Basic Protocol (4)

■ NACK

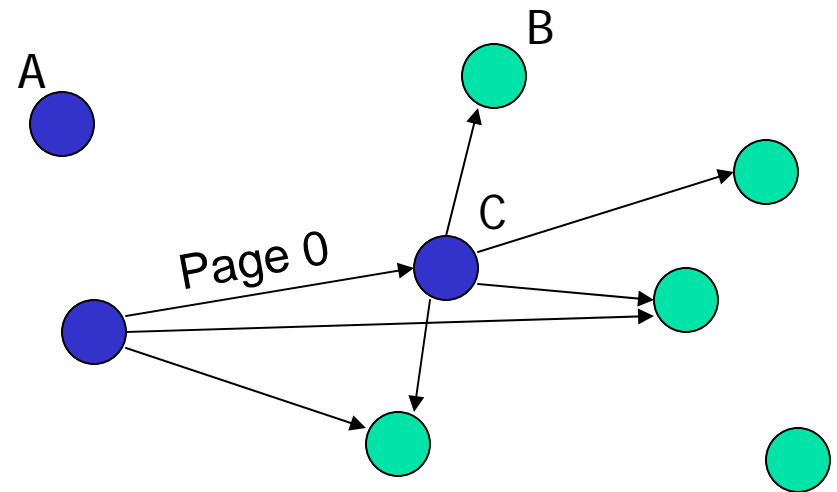
- Notify sender of dropped packets
- Essentially the same as any other request
- Gives up after a few requests in case of asymmetric links



Deluge: Basic Protocol (5)

- Advertise New Data

- Basic approach:
advertise after
receiving entire image
- Optimization:
advertise after
receiving each page





Deluge: Enhancements

- Basic protocol is similar to SPIN-RL
 - SPIN-RL does not consider large multi-page data objects
- Transferring entire image only between two nodes at a time can be improved upon
- We will consider many new optimizations to the basic protocol described earlier
- Maintain philosophy of strictly local decisions and avoid the need for neighbor tables



Deluge: Evaluation Strategy

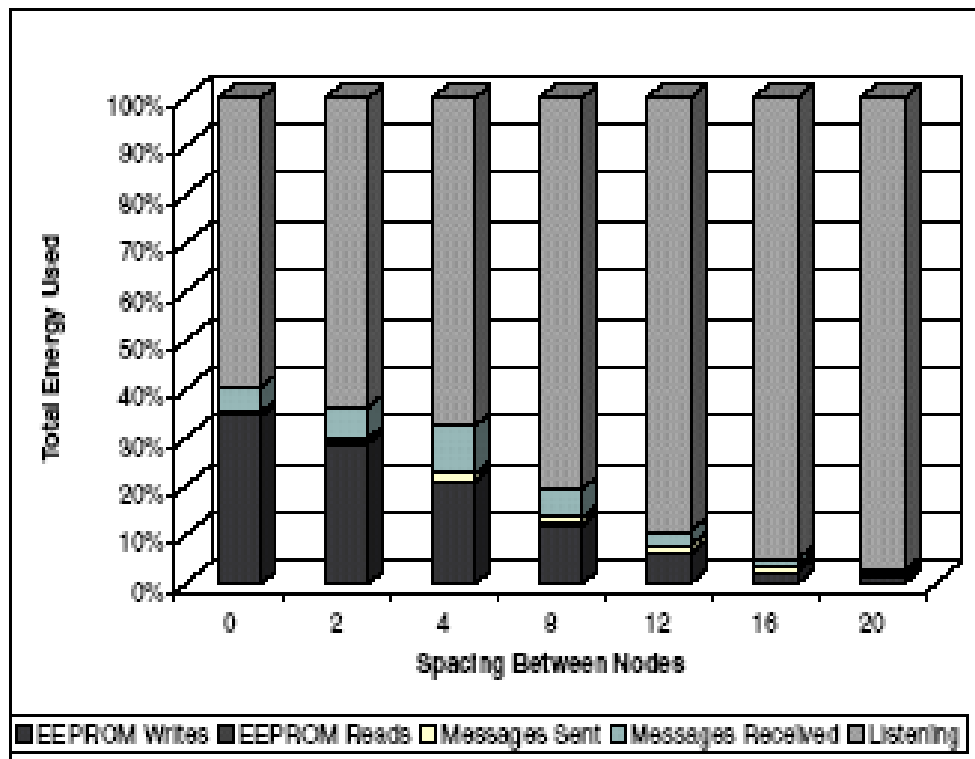
- Test basic protocol in simulation
- Examine effects of individual optimizations to protocol
- Choose the most effective optimizations
- Examine protocol performance as network scale increases



Deluge: Simulation Setup

- Network represented as a grid
- New pages originate at corner node
- Parameters:
 - Number of nodes (25)
 - Distance between each node pair (ranges from 0 to 20)
 - Pages in image (3)
 - Size of pages (512B)
- TOSSIM Empirical radio model

Deluge: Time vs. Energy



- Measured total energy used by the nodes (based on WSNA '02)
- No explicit power management enabled
- Idle listening dominates
- Reducing runtime will provide the most drastic reduction in energy costs



Deluge: Optimization Tests

- More effectively utilize the full bandwidth of the network
- Manage contention
- Lessen the effect of lossy communications
- Optimizations:
 - Transmit Data Rate
 - Spatial Multiplexing
 - Sender Selection
 - Sender Suppression
 - Forward Error Correction

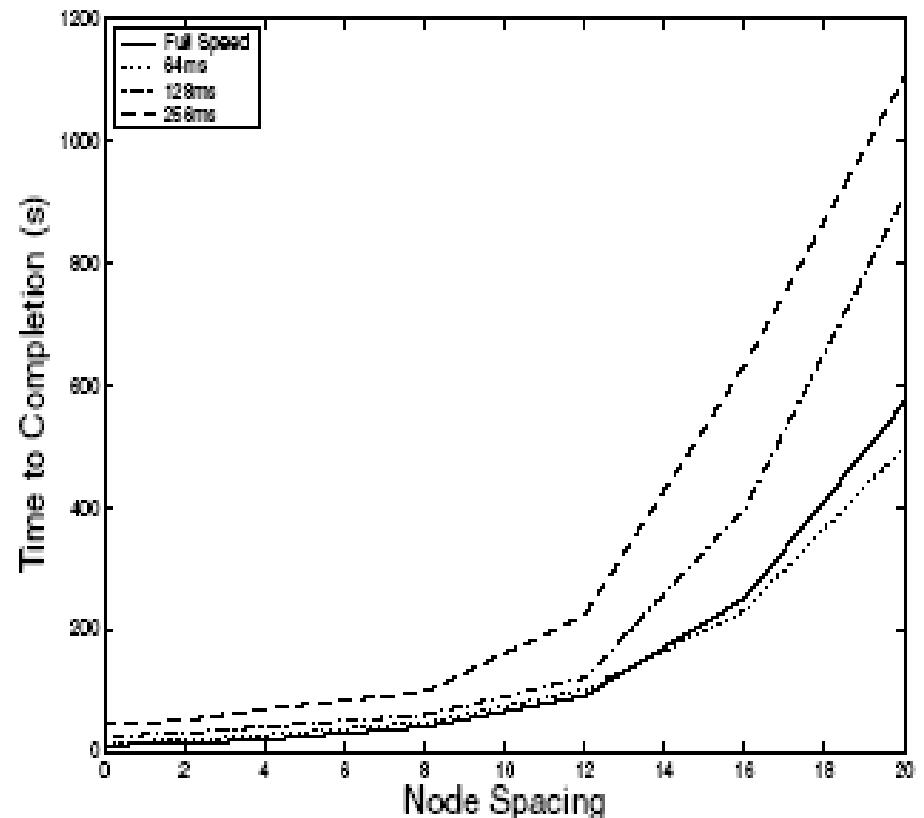


Deluge: Transmit Data Rate

- Varying Transmit Data Rate
 - Slower rates lessen contention caused by neighboring senders, but increase total runtime
 - Faster rates decrease the time to completion, but may increase contention
 - Will increased rate result in faster time to completion, or will effects of contention negate benefits?
 - Tested 4, 8, and 16 pkt/sec, and full speed

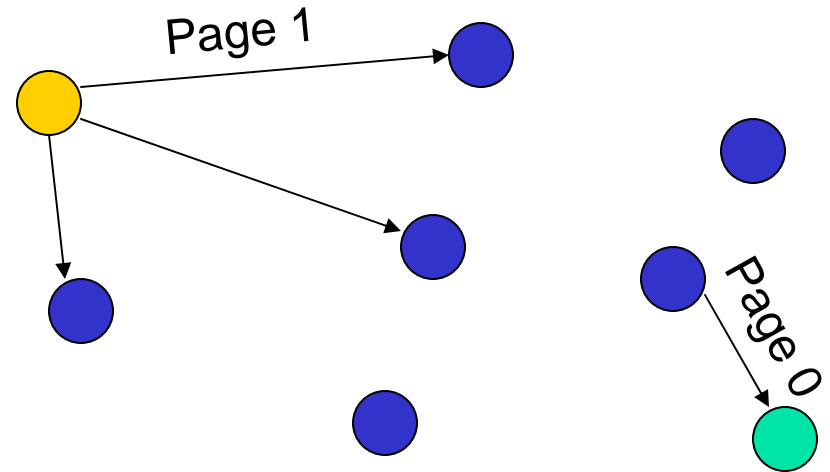
Deluge: Results

- Higher-speed send saves time (and thus energy)
- Sending at channel capacity required more data messages because of retransmissions
- High-speed sending is the best choice for our purposes

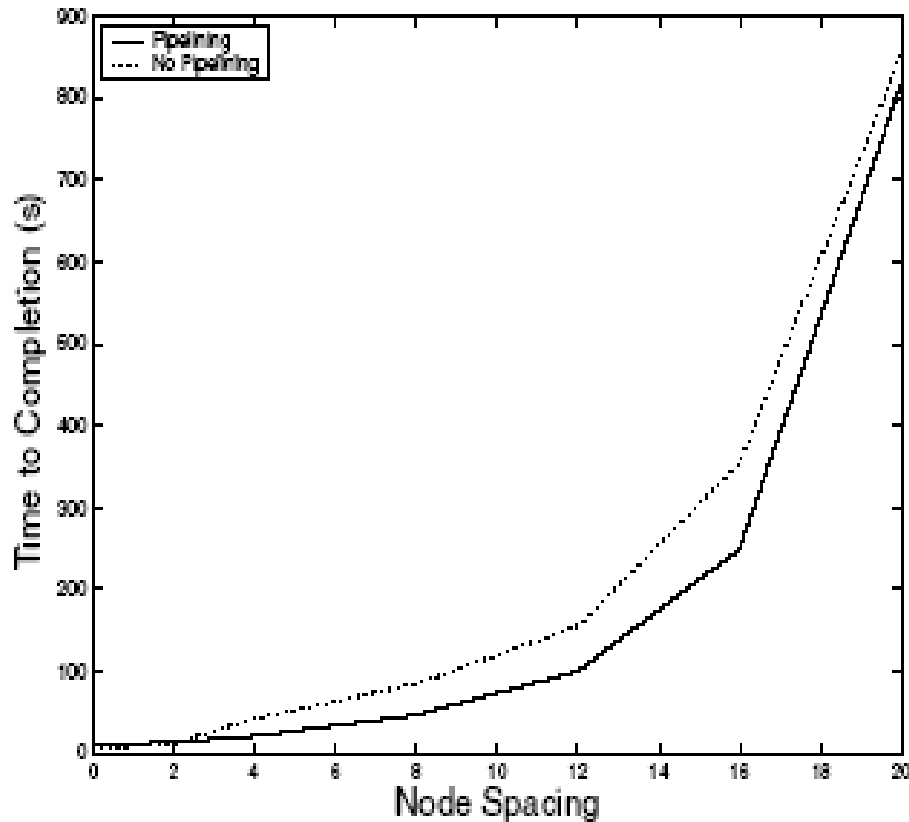


Deluge: Spatial Multiplexing

- Spatial Multiplexing
 - Take advantage of limited range radios
 - Advertise available pages immediately after receiving each page
 - Allow completed pages to propagate even if entire program image is incomplete
 - Pipelining pages should decrease time to completion, and effectively use the multi-hop network



Deluge: Results

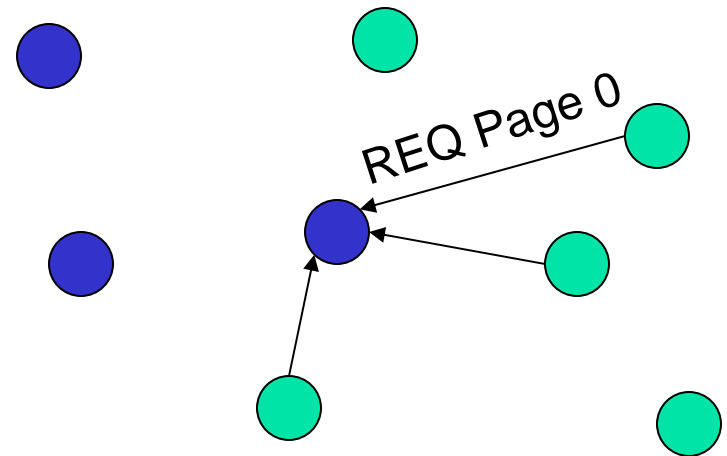


- Saved time and required sending fewer messages at all densities
- Per-page pipelining is a definite gain

Deluge: Sender Selection

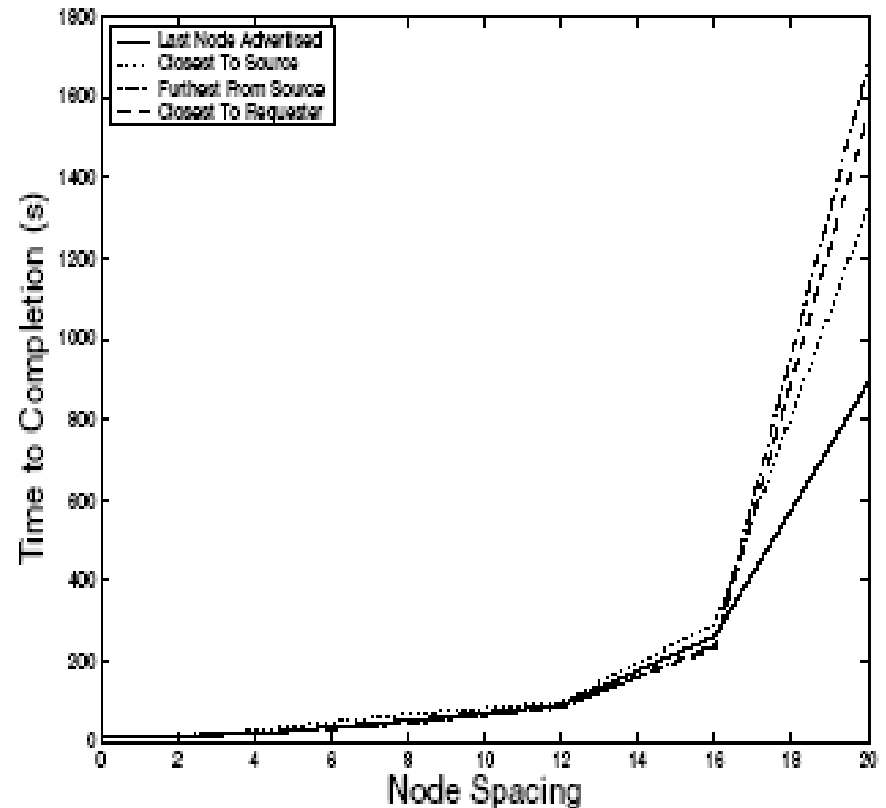
■ Sender Selection

- Want to minimize number of senders while still providing coverage for the requesters
- Four heuristics explored:
 - Request from node most recently advertised ("memoryless")
 - Request from closest neighbor based on hop count estimate
 - Request from node furthest from source
 - Request from node closest to source



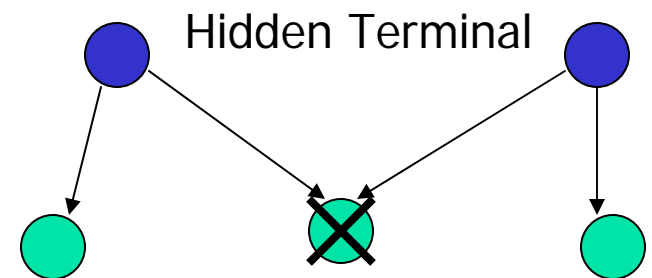
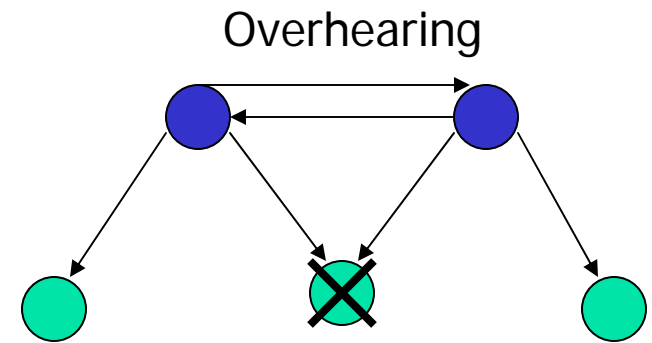
Deluge: Results

- No effects in moderately dense networks
- “Memoryless” sender selection was best in sparse networks
- Decreased ratio of senders to receivers is more important as the cost of collisions grows
- May minimize total senders by preventing divergence of “best sender” for different nodes

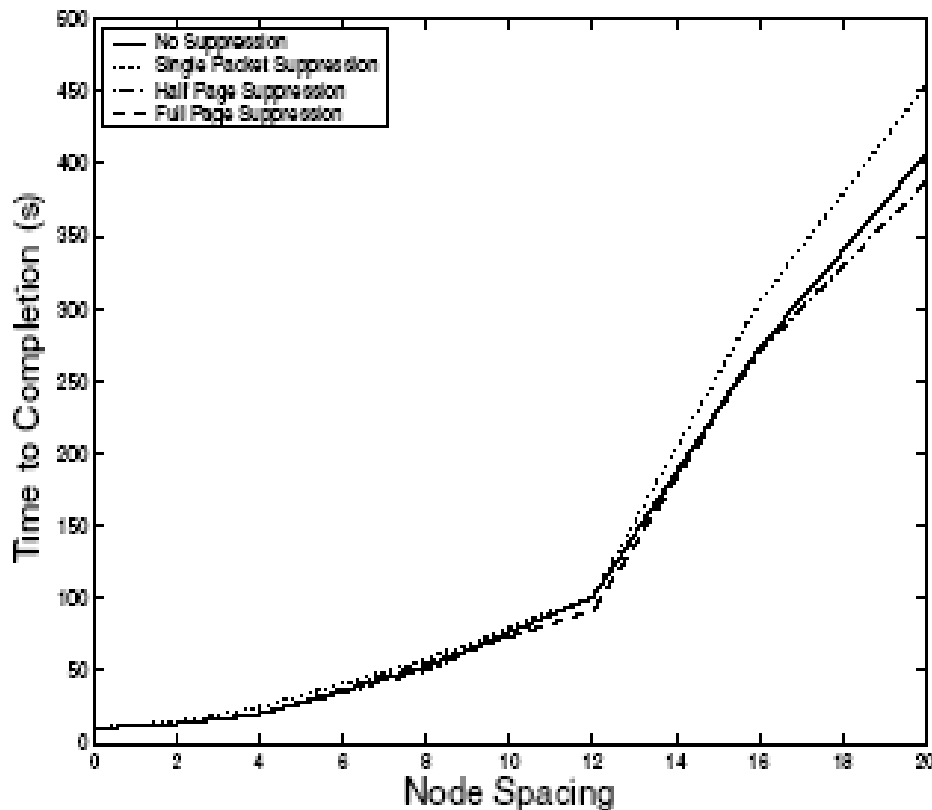


Deluge: Sender Suppression

- Sender Suppression
 - Sending node aborts the transmission of a page if neighbors are also sending
 - Tradeoff between allowing senders to reach more nodes at a time, and preventing contention.
 - Tested suppression after single packet heard, half page, full page of packets
 - Did not explicitly consider methods for dealing with hidden terminal problem



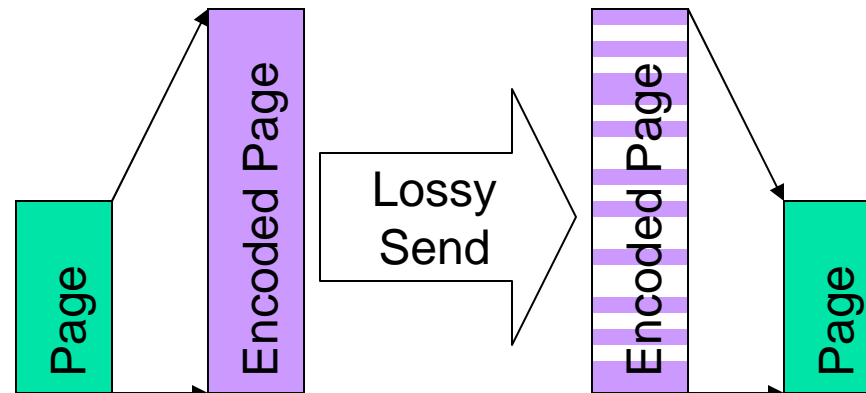
Deluge: Results



- No effects at moderate densities
- Single overheard packet impairs performance with sparse networks
- MAC collision avoidance may reduce benefits of proactive sender suppression

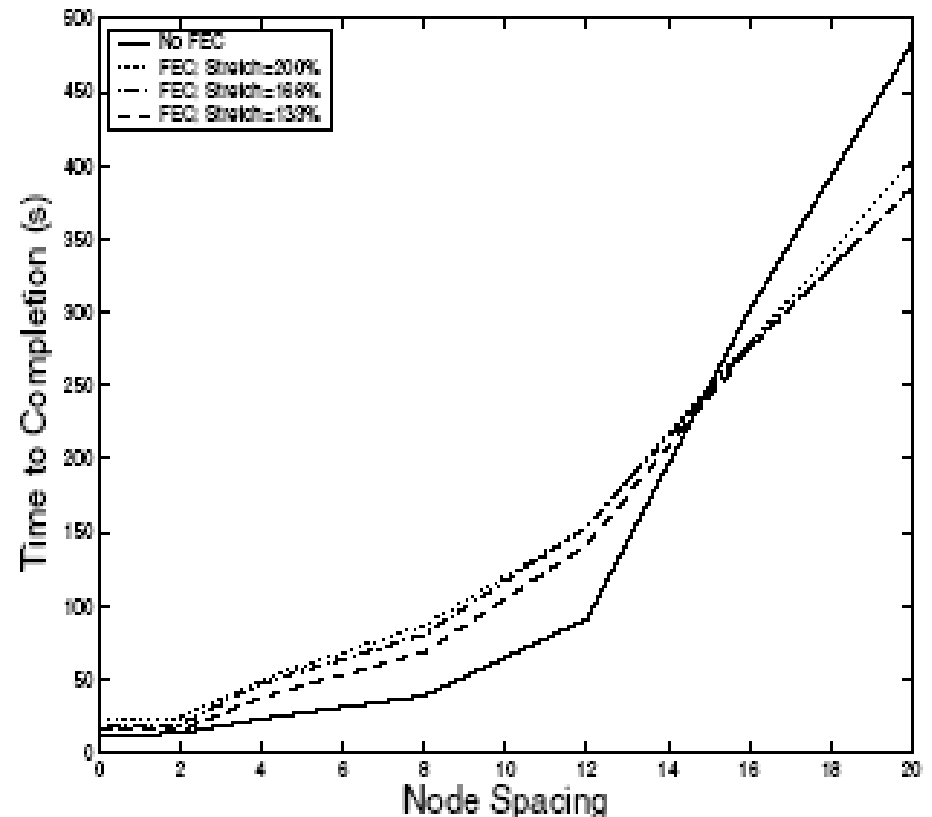
Deluge: Forward Error Correction

- Forward Error Correction
 - Add extra redundant information such that data can be reconstructed from subset of packets
 - Should lessen the need for retransmit requests, at the expense of more data transmission
 - Tested 4, 5, and 6 packets required to reconstruct 3 packets



Deluge: Results

- Much more useful in sparse networks, due to retransmit cost
- Not much difference between stretch factors
- Overall message cost is still higher with FEC



Deluge:

Optimization Conclusions

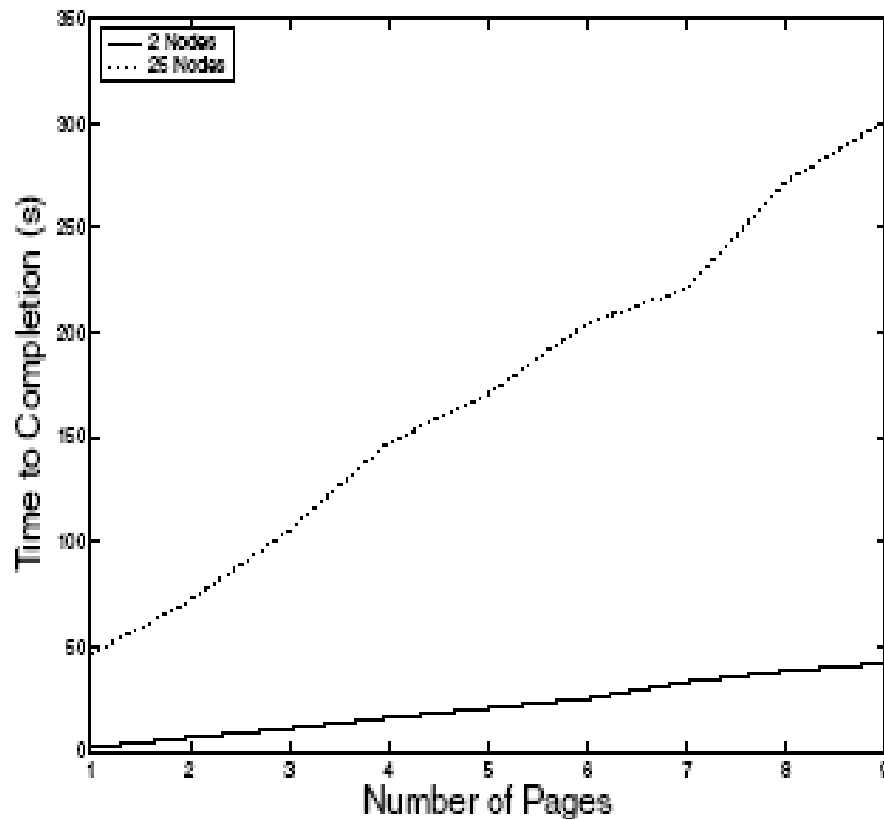
- High-speed send and spatial multiplexing are advantageous at all densities
 - Power scheduling may reduce the advantage of high-speed sending
- “Memoryless” sender selection is good enough and maintains the least state
- Forward error correction is very important in sparse networks
- Many optimizations had little effect
- Simple approach works



Deluge: Scalability Tests

- Tested scaling performance as
 - Image size increases
 - Network becomes less dense while maintaining fixed number of nodes
 - Network becomes larger with a fixed density

Deluge: Image Scaling



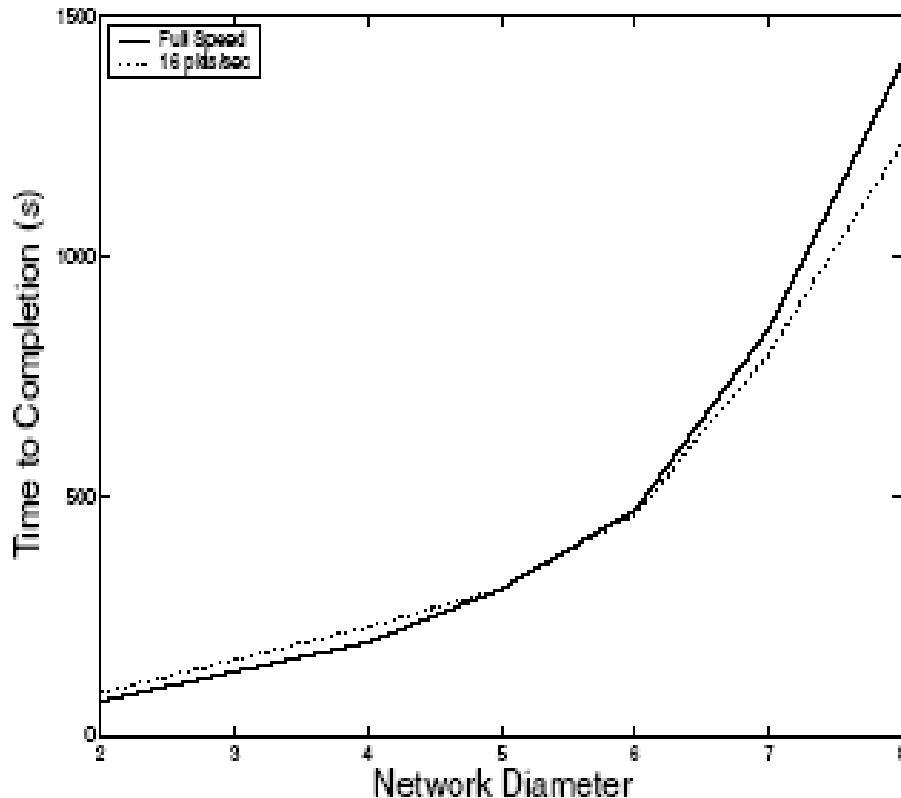
- Should be linear in image size
- Expected a sub-linear increase in multi-hop network due to pipelining
- Results are at least linear



Deluge: Density Scaling

- Examined the effect of decreasing density in earlier tests
- Increasing lossiness should leads to retransmit requests and retransmissions
- Each lost packet requires multiple packets for retransmission
 - Should see time grow exponentially as lossiness increases and density decreases
- Saw the expected exponential scaling as the grid grew more sparse

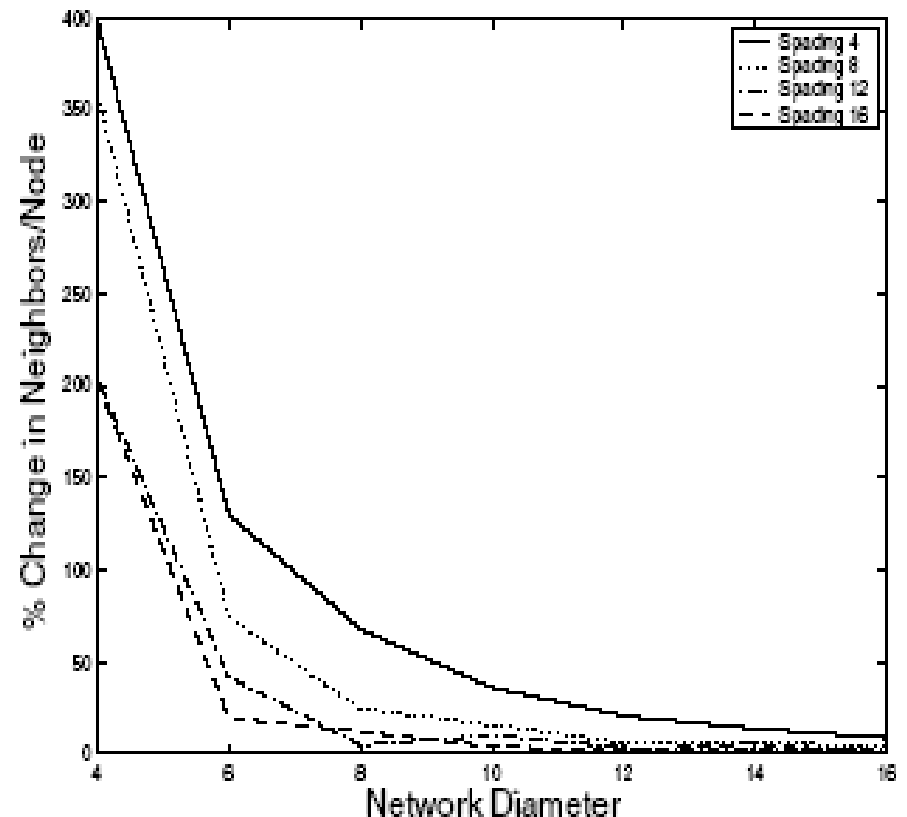
Deluge: Size Scaling



- Time should increase linearly with network diameter
- Saw super-linear increase in our tests
- Indicates that we need to manage contention more effectively
- This result may be partially due to increasing number of neighbors as network grows

Deluge: Contention Analysis

- At diameters tested, average neighbors per node grows as network grows (edge effect)
- Expected to level off as network grows larger
- May account for increasing contention
- Need to test at increasing diameters with constant neighbor count





Deluge: Hardware Results

- Successful transmission of 3 pages across 3-node multi-hop network (mica1)
- Deluge requires at least 1KB of RAM for 2 page-sized buffers
 - May be able to reduce this by using EEPROM buffer
- Deluge requires 16 KB of ROM
 - Includes all required TinyOS components
 - Much room for program size optimization



Deluge: Future Work

- Control message suppression
 - Reduce contention, robustness to varying node densities
- Concurrently running applications
 - How to transition from action to reprogramming
- Reducing energy consumption
 - Possibly utilize some form of power scheduling
- Multiple programs/versions
 - Allow specific nodes to be upgraded with different programs
 - Efficient incremental upgrades between different versions



Deluge: Demonstration

- TOSSIM Demo!