

PALER: A Reliable Transport Protocol for Code Distribution in Large Sensor Networks

Chris Miller and Christian Poellabauer
Department of Computer Science and Engineering
University of Notre Dame
miller.444@nd.edu, cpoellab@cse.nd.edu

Abstract—Re-tasking and remote programming of sensor networks is an essential functionality to make these networks practical and effective. As the availability of more capable sensor nodes increases and new functional implementations continue to be proposed, these large collections of wireless nodes will need the ability to update and upgrade the software packages they are running. Standard flooding mechanisms are too energy-costly and computationally expensive and they may interfere with the network’s current tasks. A reliable method for distributing new code or binary files to every node in a wireless sensor network is needed. This paper proposes a more effective method, called PALER (Push Aggressively with Lazy Error Recovery), which builds upon the previously proposed PSFQ protocol [1], a reliable transport protocol which slowly paces the propagation of file segments, but uses an aggressive local recovery method to avoid packet implosion due to loss propagation. PALER uses a more aggressive pushing mechanism and reduces the recovery mechanism to a single inclusive NACK. Furthermore, PALER uses local neighbor information to reduce redundant transmissions. This paper studies this new protocol’s energy efficiency and shows that it scales well to higher densities and field sizes.

I. INTRODUCTION

Wireless sensor networks are a collection of small sensor devices, typically battery powered, that may be static or mobile, and may configure in an ad hoc fashion. They have inspired a great deal of research into the design and implementation of such networks, such as routing mechanisms, event triggering, and clustering methods, as well as methods of improving energy efficiency with radio power control mechanisms and adaptive functionality. Re-tasking sensor networks which have already been deployed is still an open area of research. It requires the reliable distribution of a binary file or code to all of the nodes which require the new program, and a coordinated method of loading the new program among all of the nodes. This can be an expensive task, requiring significant battery power to fulfill all necessary radio transmissions, and consuming the computational resources of the wireless nodes.

In this paper, we will focus on the task of file distribution in support of remote programming. There have been a number of proposals for such a functionality. Many of these have worked under the assumption of a small binary or code file, requiring a limited number of segments to be distributed. This condition was typical of early wireless sensor motes, which have limited memory capacity, but as the capability of sensor

motes increases, sensor network programs will inevitably increase with them. In this paper, we will explore a more efficient method of binary or code distribution for relatively large datafiles. The most basic method of file distribution is by flooding the segments into the sensor network. This method is very costly, though, as every node will receive and broadcast each segment of a file. Many of these broadcasts are unnecessary since sensor networks may be dense, and have highly overlapping broadcast zones. The redundant transmissions are an unnecessary expenditure of power and could lead to increased packet loss due to congestion. Another issue with the naive flooding method is that it provides no reliability. It is important to ensure that all nodes in the network receive the entire file in a timely manner, so reliability mechanisms will be needed.

Wan, et al. proposed Pump Slowly, Fetch Quickly (PSFQ) [1], a reliable transport protocol which reduces transmissions, making it energy efficient and scalable. The basic premise behind the protocol is to propagate the segments at a relatively slow pace (“pump slowly”), and use an aggressive NACK mechanism to fetch missed segments (“fetch quickly”). This protocol attempts to minimize the cost of lost packets by reducing NACKs and rebroadcasts in response to NACKs to a single hop. They also reduce the amount of transmissions through forwarding by using a counter mechanism to limit transmissions. These methods are effective in reducing the total number of transmissions necessary to flood all segments of a file throughout a sensor network, however, the aggressive timing of the fetch operation can lead to congestion in a dense network when nodes attempt to respond to NACKs. Also, the retransmission delay that is added to allow for handling NACKs places a lower bound on the latency. We propose *Push Aggressively with Lazy Error Recovery* (PALER), which is based on PSFQ with the enhancement that NACKs are reserved until the end of the transmission of all segments. This reduces the amount of NACK packets, particularly in high-loss environments. A slower response period reduces collisions in NACK responses, and further reduces the number of total transmissions for complete dissemination. This modification also eliminates the need for a retransmission delay, allowing a more aggressive forwarding method, leading to reduced latency.

PSFQ and PALER have been implemented and evaluated

through simulation to exhibit the effects of node density and hop count on total transmissions and latency. These results show that a more relaxed recovery mechanism avoids much of the contention and collision found in a more aggressive mechanism. It also provides an opportunity for improved collaboration among nodes to avoid redundant responses. In addition, withholding negative acknowledgements until all segments of a dataset have been broadcast allows a more aggressive propagation of segments. This allows PALER to improve energy efficiency by reducing transmissions while reducing latency. The results show that PALER is capable of functioning efficiently in very dense networks, and latency scales very well across an increasing field size. We compare two variations of PALER to evaluate the advantages of differing pruning techniques in the flooding phase. Version I utilizes a counter method to reduce redundant transmissions, while version II utilizes local neighbor data to prune unnecessary transmissions.

II. RELATED WORK

Code distribution has many similarities with reliable multicast, reliable broadcasting and energy-efficient broadcasting research. Many traditional techniques for making multicast reliable in wired networks, such as [2], are too computationally expensive for the limited resources of a wireless node. Multicast protocols which have been developed for wireless networks, such as [3] and [4], tend to favor robustness to provide reliability. This built-in redundancy comes at the expense of energy efficiency. Wireless broadcasting protocols focused on reliability typically require significant overhead in control packets [5, 6].

Energy-efficient wireless broadcast protocols typically focus in two areas: minimizing forwarding nodes [7, 8, 9, 10], or minimizing transmission power [11, 12, 13]. [7] offers several methods of reducing forwarding nodes, such as probabilistic, counter, and cluster methods, each with differing levels of reliability. [10] developed an algorithm to identify a dominating set in a network, which would make up the intermediate nodes in all broadcasts. [9] and [8] built upon this method by reducing the size of the dominating set. Selection of a dominating set may reduce total transmissions in a broadcast, however, it does not balance the load, as the nodes in the dominating set will incur all of the cost. An interesting extension of [9] is a neighbor elimination method which is very similar to the method used in PALER to reduce forwarding nodes. [14] also used an approximation of minimum dominating set to achieve reliability and reduce the cost of recovery. Their scheme is capable of providing reliable broadcast of a single packet, and limits the cost of dropped packets with local recovery, but the focus of this work is on reliability rather than the energy efficiency of the downstream propagation. Broadcast protocols aimed at minimizing transmission power are typically based upon a set cover [12] or minimum spanning [11, 13] problem. These tend to provide very efficient distribution trees and balance loads

evenly; however, they require a knowledge of the complete network topology, and are best used in a static environment where optimal routes can be predetermined.

Others methods of achieving reliability or efficiency include FEC and network coding techniques. FEC is used in conjunction with probabilistic forwarding by [15] to add reliability to an efficiency scheme. The FEC technique offers flexibility in the propagation and recovery methods. With a large data distribution, the number of overhead packets generated by the FEC is expected to be minimal relative to the total packets sent, but this assumes a lossless environment and ideal MAC layer. A realistic scenario could require an undesirable number of overhead packets produced by the encoding method to provide reliability. The forwarding probability used for their simulations is optimized for the network topology, which provided efficient propagation results. This would make implementations sensitive to alterations in the network size and density, however, which could lead to a loss of efficiency or reliability. Network coding is used in [16] to improve efficiency by reducing the number of transmissions. They provide an alternative algorithm in their approach for multicast in wireless networks to exploit the wireless multicast advantage. While the number of transmissions may be reduced in this method, the potential overhead cost of the control data which must be appended to each packet is not evaluated. For large data transmissions, as may be used for re-tasking, this cost could be significant. Extensions would also need to be made to this scheme to provide reliability and loss recovery.

Trickle [17] is a popular recent method of code propagation and maintenance. It uses a gossiping protocol with periodic metadata broadcasts to identify nodes which require an update to a new version of code. It uses a counter method to limit the number of gossip messages broadcast during an interval, which makes Trickle a very energy efficient method of maintaining a sensor network. Trickle is not greatly concerned with latency, and is based upon an expectation of a very small code segment or binary file, one which will fit in a small number of packets. Our protocol will be more focused on an efficient method for a single reprogramming event of a relatively large binary file. Another gossip based code propagation protocol is GCP [18]. It uses periodic beacons to detect outdated code versions, similar to Trickle. However, it also includes a forwarding control mechanism to balance the load of distribution. Each node has a limited number of tokens that it may use for distributing each new version of code.

Melete [19] builds upon Trickle to support dynamic grouping and concurrent applications in sensor networks. It uses a periodic metadata broadcast to maintain the network. It also supports group-based code propagation. Nodes may dynamically enter and exit a group, and must broadcast a request for the new group code. Melete avoids broadcast implosion by pacing requests through a probabilistic and progressive flooding mechanism. Because of the dynamic grouping nature of their system, code propagation will be accomplished through

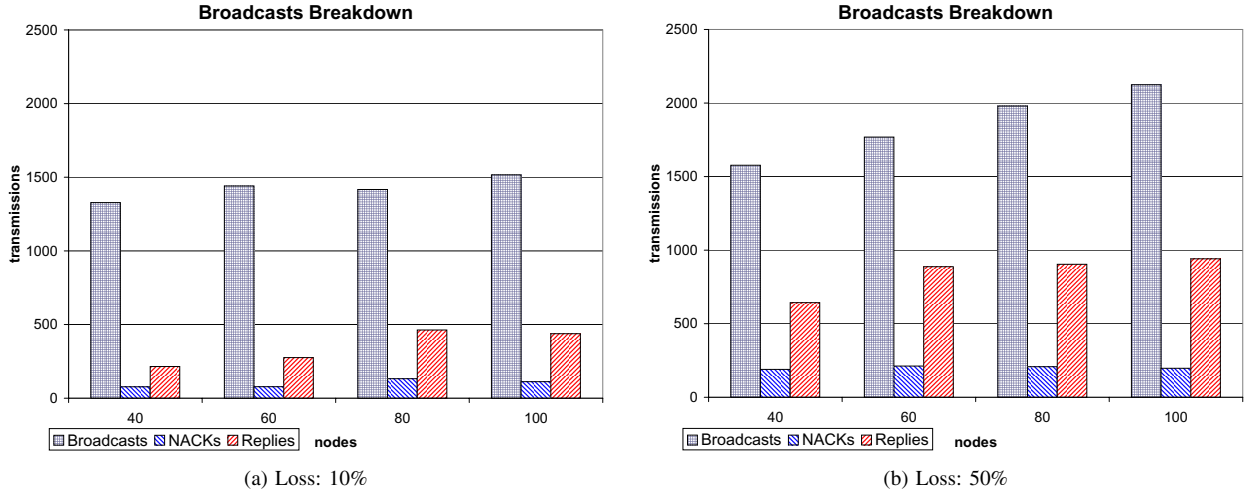


Fig. 1: Breakdown of the types of transmissions during a simulation of PSFQ. Simulation had a fixed field size with increasing node count. (a) shows the results of a low-loss environment, 10%. (b) shows the results of a high-loss environment, 50%.

multi-hop unicast in many situations. In our protocol, we will instead focus on a one-to-all distribution of a large data file.

III. PROTOCOL DESIGN

A. PSFQ Design

PSFQ [1] distributes data from a single source to a network of sinks by slowly pacing the propagation of packets. Nodes that detect a lost packet due to out-of-order packet reception will aggressively fetch the missing packet by sending a NACK to its immediate neighbors. Nodes will refrain from forwarding packets received out-of-order until all packets leading up to that one have been recovered. In other words, regardless of the order of reception, nodes will only rebroadcast packets in-order. This prevents the propagation of a loss event to the downstream nodes. Requiring in-order broadcasts also ensures that lost packets can be retrieved from at least one immediate neighbor, since the neighbor that broadcast the higher than expected sequence number must also contain the expected sequence number. By localizing recovery, it reduces lost recovery cost by suppressing the propagation of loss events and negative acknowledgements, and reducing recovery to a single hop transmission. The PSFQ protocol is built upon a tightly coupled timing between the pushing mechanisms and fetching mechanisms. The pumping mechanism relies on two timers, T_{min} and T_{max} . After reception of each in-order packet, the packet will be scheduled for re-broadcast following a random delay between T_{min} and T_{max} . A counter is maintained for the number of times each packet is heard. If a packet is received four times prior to rebroadcasting, the rebroadcast event is canceled to suppress redundant transmissions.

If a packet is received out-of-order, the node will schedule a NACK with a request for the missing segments after a short random delay. The node will continue to send a NACK every T_r until all of the missing segments have been received. The

key to PSFQ is the relation between T_r and T_{max} . The ratio between T_r and T_{max} determines how many opportunities a node has to fetch a missing segment before the next segment is expected to arrive. The higher the ratio T_{max}/T_r the greater the probability that the segment will be successfully received over multiple hops. In their implementation, they selected $T_{max} = 5 * T_r$, and $T_{min} = \frac{1}{2} T_{max}$. T_r must also be chosen to provide an adequate window for recovery. To provide a minimum necessary window, T_r should be at least four times the latency for a single packet, but they proposed that a reasonable value would be $T_r = 6 * T_p$.

When a node receives a NACK, it checks the request against its own cache to determine if it has any of the missing segments. If it does, it will schedule a reply with the missing segment at a random time within the interval $\frac{1}{4} T_r$ to $\frac{1}{2} T_r$. To reduce contention and redundant transmissions, it will cancel the reply event if it overhears any neighbors responding to the same missing segment prior to its own reply. To improve the likelihood that only one neighbor will respond to a NACK request, each node maintains a table with the average signal strengths of its parent nodes. When it sends a NACK, it includes its preferred neighbor (node with the highest average signal strength) in the header. The neighboring nodes will check to determine if it is the preferred neighbor; if not, it will double its delay interval before sending the reply, giving it more time to overhear a reply from the preferred neighbor.

The slow pumping mechanism of PSFQ along with the counter method for pruning of forwarding nodes is effective in reducing contention and collision, and efficiently propagating a file with minimal redundant transmissions. The fetching mechanism avoids the NACK implosion problem by localizing recovery. However, the aggressive nature of the fetch mechanism and the relatively short time frame of the recovery intervals leads to a great deal of contention. While

the neighboring nodes listen for duplicate replies from other neighbors, the response timeframe does not allow a large window to randomly disperse the responses and to overhear responses. Each node is generating a random delay period between $\frac{1}{4}T_r$ to $\frac{1}{2}T_r$. This interval equates to just $\frac{3}{2}T_p$, which provides little opportunity for overhearing. The inclusion of a preferred neighbor based on received signal strength helps increase the window of opportunity for overhearing other responses, but nodes that computed a small random delay will still be transmitting close to the same time as the preferred node. In a dense network, this may result in a high number of collisions, possibly resulting in more redundant responses. Figure 1 shows a breakdown of the types of transmissions observed in a simulation of PSFQ. In this simulation, the field is set to $1km \times 1km$, and the density ranges from 40 to 100 nodes per km^2 . The first graph shows the results for an average 10% packet loss, the second graph depicts the same results for a 50% loss environment. These figures show that the number of response messages are a significant percentage of the total transmissions. Many of these reply transmissions are redundant, and could be avoided with a less aggressive method. Additionally, in a high loss environment, the in-order requirement can result in a high number of NACK transmissions. Another impact of the timing constraints is that it places a lower bound on the latency that is equivalent to $T_{min} * hops$, where *hops* is the maximum number of hops needed to reach all nodes from the source.

B. PALER Design

To avoid the contention and collisions resulting from the aggressive recovery mechanism of PSFQ, PALER eliminates the in-order reception requirement and maintains a list of missing segments. After all packets have been broadcast, each node will broadcast a NACK to its neighbors. An important aspect of PALER is that, even though it does not require in-order reception, it still maintains a local recovery mechanism. The reason this is possible is because if a neighbor receives a NACK, it can first check its own cache for the missing segments. Any segments that are present in the local cache may be sent to the requester in a single hop transmission. As for segments that are not present in the local cache, these segments must also be among the list of missing segments on the local node, which means that they will be included in the local nodes NACK. Therefore, it does not need to propagate the NACK from its neighbor, because the missing segments will be redundant. This maintains local recovery with single hop NACK transmissions, while still ensuring that request for missing segments will be propagated until the missing segments are found.

When a node receives a segment, it checks if the segment is already contained in its cache; if it is, it increments a counter for that segment. If it is the first reception of this segment, it schedules a forwarding event at a random delay between 0 and 100 ms. If the counter for the segment reaches three before the segment is rebroadcast, the forwarding event

is canceled. Because NACKs are withheld until the final segment, intermediate nodes do not have to have a minimum delay period prior to rebroadcasting, so the interval can be any random period within 100 ms. Since the counter limits the rebroadcast within a region to the first three attempts, the average interval prior to a rebroadcast tends to trend closer to 0 than 100 ms, particularly in dense networks. This eliminates the lower bound on latency, and provides a much improved latency that scales well across large multi-hop environments.

When a node receives the last segment of a distribution, it schedules a broadcast of a NACK to its one-hop neighbors which includes a list of segments it is missing. The NACK is scheduled following a random delay period, which is used to reduce collisions, and to allow the node an opportunity to overhear rebroadcasted segments in response to other NACKs. If it overhears a segment that it was missing prior to broadcasting its NACK, it will remove that segment number from its list of missing segments. When a neighboring node receives a NACK, it checks the list of missing segments against its own cache. If it owns any of the missing segments, it will schedule a reply with the missing segment following a random delay between 0 to 50 ms. If it owns more than one segment, it will schedule each additional segment for a reply at 10 ms intervals. The delay prior to replies reduces contention and allows the nodes to overhear other replies. If a node overhears a reply for a segment that it has scheduled a reply for, it will cancel its reply event.

Since this recovery mechanism is dependent upon reception of the last segment, a timeout period is used in case of loss of the last segment. The timeout period is continuously updated to be $T_{begin} + \alpha * (T_{avg} * [Seg_{tot} - Seg_{rec}])$. Here, T_{begin} is the time the first segment was received, T_{avg} is the average interval between reception of each segment, Seg_{tot} is the total number of segments, Seg_{rec} is the number of segments received, and $\alpha \geq 1$ is a small multiplier that determines how aggressive the timeout value should be, typically this would be less than 1.5. Following the transmission of each NACK, a new timeout value is set to specify a maximum time expected before all missing packets are recovered. If any segments are still missing at the end of this timeout period, another NACK will be transmitted, and a new timeout value will be set. The first timeout following a NACK will be calculated using the same formula as above. If additional NACK's are necessary, each additional timeout period will be doubled to avoid a NACK implosion resulting from downstream nodes waiting for a multi-hop recovery to propagate.

C. PALER Flooding Mechanism

With the lazy error recovery of PALER, NACK and recovery transmissions are greatly reduced, leading to a very efficient broadcast with minimal wasted energy. However, the pushing operation still requires a relatively large number of transmissions, even in fairly dense networks, and many of these transmissions are redundant. PSFQ increased efficiency over standard flooding by using a counter with a cutoff of 4 to

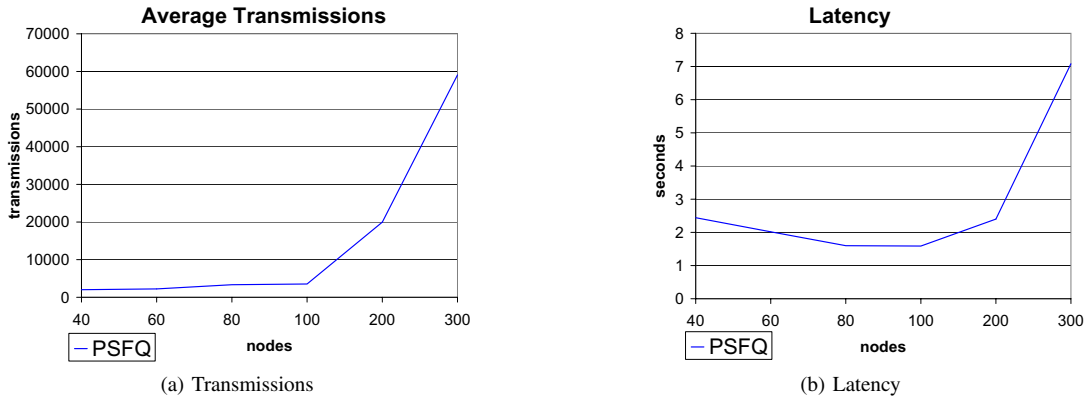


Fig. 2: Results of PSFQ simulation with $T_r = 6ms$. Simulation had a fixed field size with increasing node count. (a) shows the total number of transmissions during the simulation. (b) shows the average latency.

locally limit the number of rebroadcasts within a region. [7] showed that if a node overhears a message 4 times, the average additional coverage that can be achieved by performing its own rebroadcast of the message is just 5%. The additional coverage that can be expected after overhearing a message 3 times was shown to be about 9%. In PALER, a counter with a cutoff of 3 is used to limit redundant transmissions. The selection of 3 as the cutoff proved to reduce forwarding transmissions without significantly affecting NACKs or recovery transmissions.

To further improve the efficiency of the propagation method, a new mechanism was devised that used two-hop neighbor information that is dynamically acquired. To generate a representation of each node's two-hop neighborhood, each node will include a list of its immediate neighbors with the first n segments of a broadcast. The value n is an implementation set value that will determine the strength of the neighborhood representation; in our implementation n is 10. When a node receives a segment with this list in the header, it will add pairings of the sending node with each of its neighbors to a table that list all of the immediate neighbors of nodes within its two-hop neighborhood. To reduce the amount of contention and collision during the propagation of the first n segments, particularly in dense networks, a probabilistic method can be used to limit the number of rebroadcasts. A probabilistic method is used instead of a counter, because a counter could result in the same nodes rebroadcasting each time, resulting in an incomplete neighborhood representation. The probability selected should be high enough to ensure that each node broadcast its neighborhood information at least once, which is implementation dependent on n . It can be determined by using a cumulative probability distribution function to determine what value of p is needed for the desired confidence level.

Following the initial n segments, when a node receives a segment for the first time, it generates a list of its neighbors and schedules a rebroadcast event at a random time as before.

The list of neighbors represents all of the immediate neighbors that may need this current segment. For each transmission of that segment it receives prior to rebroadcasting, it removes the sending node from the list of neighbors associated with that packet. It then acquires the list of the sending node's immediate neighbors from its neighbor table, and removes each of these nodes (if present) from its list of neighbors for this segment. These are removed because it is able to assume that each of these nodes received the segment from the sending node, and therefore do not need this segment. If the list of neighbors for a segment becomes empty prior to forwarding the segment, the rebroadcast event is canceled, since it is assumed that all neighbors received the segment.

IV. SIMULATION METHOD

To evaluate PALER, it was implemented in the Jist/Swans simulation environment [20, 21], a scalable wireless ad hoc network simulator based in Java. PSFQ was also implemented for comparison. Swans provides a full representation of the complete network layer model, with accurate representations of a wireless environment, including path loss, environmental noise and collision interference. Each node in the simulation was implemented with an 802.11 radio, with a range of approximately 250 meters. Two sets of simulations were performed, one with a fixed field size of $1km \times 1km$, with an increasing density, and another with a fixed density and increasing field size. For each simulation, a file of size 50 KB was distributed. The file was segmented into 50 chunks, each 1 KB. The primary metrics used to evaluate the protocols are total transmissions and latency, where latency is measured as the time needed for all nodes to receive every segment of the data file. The goal of PALER is to improve on the efficiency of PSFQ by reducing transmissions, while still meeting or exceeding the latency performance of PSFQ. In our simulation results, we show that this was accomplished. Our evaluation of PALER will show that energy efficiency can be further improved using dynamic, local neighborhood

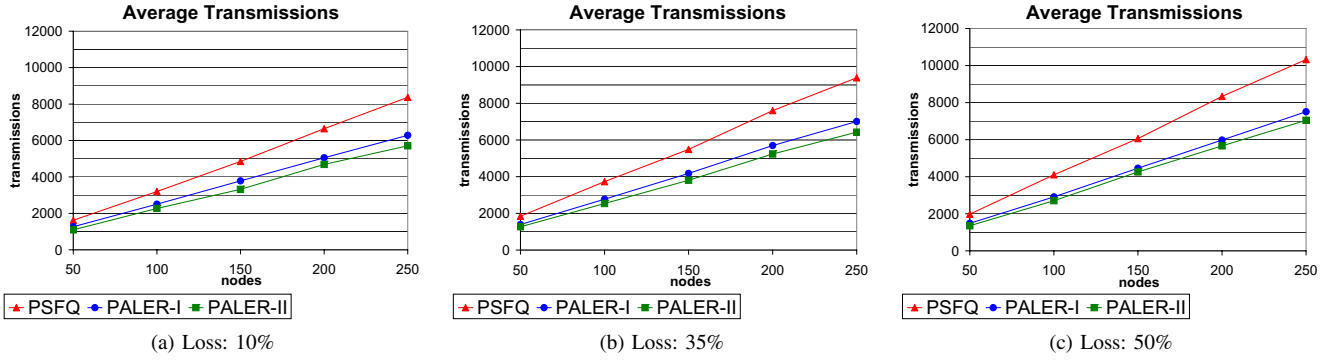


Fig. 3: Comparison of total transmissions of PSFQ, PALER-I and PALER-II, measured across increasing field size. Density is fixed at 50 nodes per km^2 . A 50 KB file is transmitted in 1 KB segments. (a) shows the results for a low-loss environment, 10%. (b) shows the results for 35% loss, and (c) for a 50% loss. PALER-I and PALER-II exhibit a significant decrease in the number of total transmissions required to complete a code distribution, with PALER-II demonstrating the best energy efficiency.

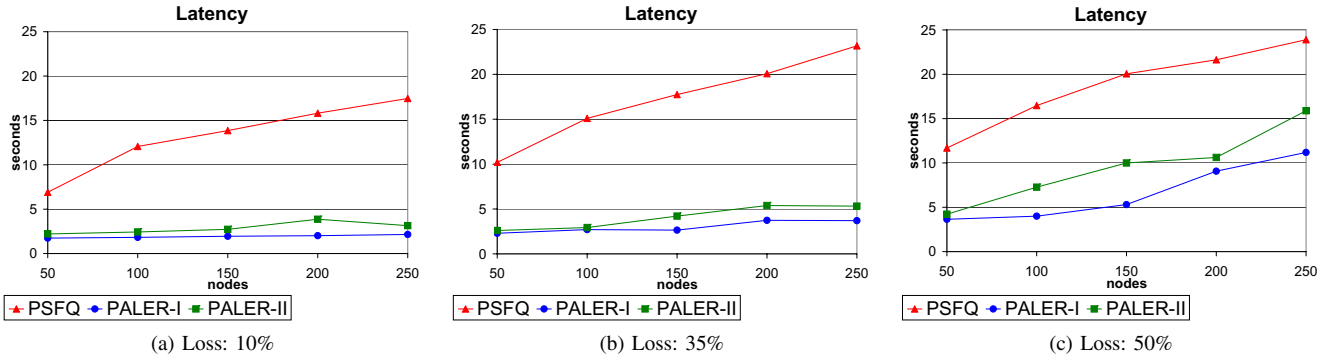


Fig. 4: Comparison of average latency of PSFQ, PALER-I and PALER-II with a fixed density and increasing field size. These latencies correspond to the simulation results in Figure 3. PALER shows an ability to scale extremely well across multiple hops, scaling below a linear rate of increase.

information, though this may slightly impact latency. For comparison, we have implemented both flooding mechanisms in PALER. The results for the PALER implementation using the counter method are presented as PALER-I, and the results for the implementation utilizing the neighbor pruning method are presented as PALER-II.

To determine the timing parameters used for the PSFQ implementation, a test scenario was implemented in Jist/Swans to determine the average time for a packet transmission with the range of densities used in our simulations. The calibration results showed that $T_p \approx 1ms$. Based on the recommendation in [1], T_r was set to $6 * T_p$. Therefore, in our initial implementation of PSFQ, the timing parameters were $T_r = 6ms$, $T_{min} = 15ms$, and $T_{max} = 30ms$. The simulation results for this implementation exhibited a very poor energy efficiency, and did not scale well with high node densities. The results of these simulations for a low loss environment are shown in Figure 2. The simulations were implemented with a field size of $1km \times 1km$, with an average packet loss of 10% and 40 to 300 randomly placed nodes, each with a range of 250m. Even

at 40 nodes, the number of transmissions is above 2000, which is what would result from a basic flooding mechanism. It can be seen that as the density increases above 100 per km^2 , PSFQ does not scale well, and the number of transmissions quickly escalates to multiple times the value of flooding. Therefore, to provide a stronger basis for comparison, the timing metrics were changed to those used in Wan et al.'s implementation [1]. For their implementation, the timing metrics were $T_r = 20ms$, $T_{min} = 50ms$, $T_{max} = 100ms$. This provided a greatly improved energy efficiency with transmissions an order of magnitude smaller, though the latency did increase as a result of the lower bound imposed by T_{min} .

V. RESULTS

The first simulation performed a comparison for varying field size of a network. The density for these simulations is fixed at 50 nodes per km^2 . The number of nodes in the simulation ranges from 50 to 250, which corresponds to a field size ranging from $1km \times 1km$ to $2.236km \times 2.236km$ (or a width of 4 hops to a width of 9 hops). The results in

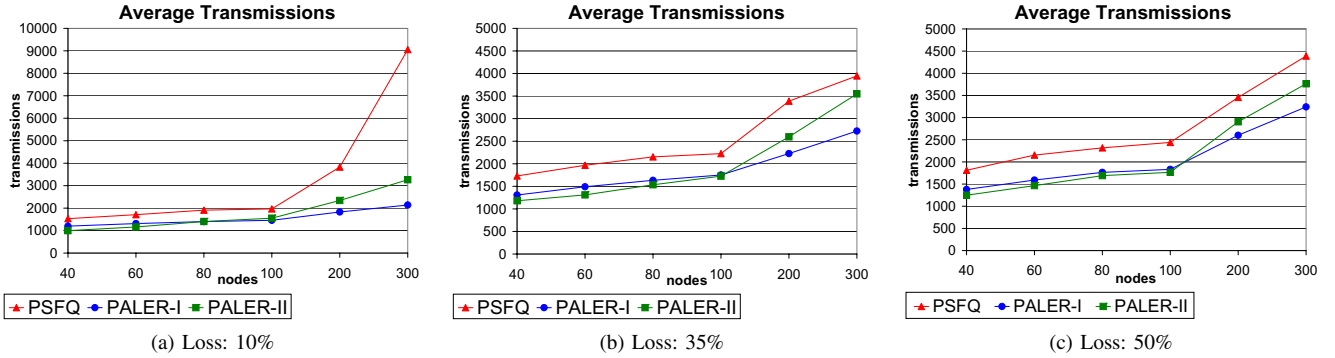


Fig. 5: Comparison of total transmissions of PSFQ, PALER-I and PALER-II measured against increasing density. The field is fixed at $1km \times 1km$, each node has a range of $250m$. A 50 KB file is transmitted in 1 KB segments. (a) shows the results for a low-loss environment, 10%. (b) shows the results for 35% loss, and (c) for a 50% loss. The benefits of PALER-II are greatest in networks with an average node connectivity less than 20.

Figure 3 show the average number of transmissions, measured by the network size, for an average loss environment of 10%, 35% and 50%. The graphs show that PALER-I exhibits a 20-30% reduction in transmissions from PSFQ, while PALER-II exhibits a 30-35% reduction. The increase in efficiency is fairly consistent across differing loss environments and differing field sizes.

The second set of graphs in Figure 4 show the corresponding latencies for the same set of simulations. Due to its lower bound on latency, PSFQ scales fairly linearly with field size (number of hops). However, it does perform rather consistently among increasing loss environments. As seen, PALER scales very well across increasing field sizes, particularly in moderate loss environments. In high loss environments, such as that of Figure 4c, latency increases in stronger relation to field size, but still performs strongly in comparison to PSFQ. PALER-I exhibited the best latency, revealing the tradeoff of energy efficiency and latency between the flooding mechanisms. This slight increase in latency in PALER-II is due in part to the random timing mechanism of PALER. In PALER-I, the nodes which do not choose the lowest random delay periods are the ones that suppress forwarding, whereas in PALER-II, the pruning selection is partially independent of the delay period chosen.

To examine how PALER performs in differing densities, another set of simulations was performed with density varying from 40 nodes per km^2 to 300 per km^2 . The results are shown in Figure 5, for loss environments of 10%, 35% and 50%. The efficiency improvement of PALER-II is approximately 30%, which is consistent across all densities when averaged over the different loss environments. PALER-I increases in efficiency improvement as density increases. It results in a reduction of transmissions of 23% for a density of 40 nodes per km^2 , but nearly 45% for densities of 300 nodes per km^2 . In extremely dense environments, PSFQ actually performed worse in a low-loss environment, due to high contention during recovery. As a result, Figure 5a is shown on a different scale to accommodate

the plot of PSFQ. These results show that PALER scales well to extremely dense networks. The overhead of generating neighborhood information in PALER-II can be seen in very dense networks. Since PALER-II requires nodes to broadcast at a certain probability for an initial period, collisions can result in very dense networks. For densities below 100 nodes per km^2 , PALER-II performs more efficiently, but at densities above 100 nodes per km^2 PALER-I performs more efficiently. This point of intersection corresponds to an average node connectivity of approximately 20. In many cases, this may be an acceptable upper bound, but if an extremely dense network is expected, PALER-I will perform efficiently.

The greatest factor in the reduction of transmissions by PALER is the reduction of contention and collisions during recovery. The lazy recovery method allows nodes to carefully pace recovery operations and avoid redundant transmissions. Figure 6 shows the breakdown of PALER transmissions between broadcasts, NACKs, and reply messages. The graphs from Figure 1 are included for comparison, it can be seen that the number of broadcasts is fairly similar, but the number of NACKs and replies make up a significantly smaller percentage of total transmissions in PALER.

VI. CONCLUSIONS AND FUTURE WORK

The results from the previous section show that PALER is able to consistently improve energy efficiency above PSFQ across a broad range of network sizes and densities. In addition, latency proved to scale very well in PALER, particularly in moderate loss environments. The main reason for the improved efficiency in the number of total broadcast is due to the more relaxed nature of the recovery mechanism, which avoided much of the contention that affected PSFQ. The comparison in Figure 6 shows that PALER performs fairly consistently with PSFQ in the total number of forwarded broadcasts. However, the percentage of NACK and reply messages is greatly reduced in PALER. PALER implemented with a neighbor pruning mechanism showed an additional

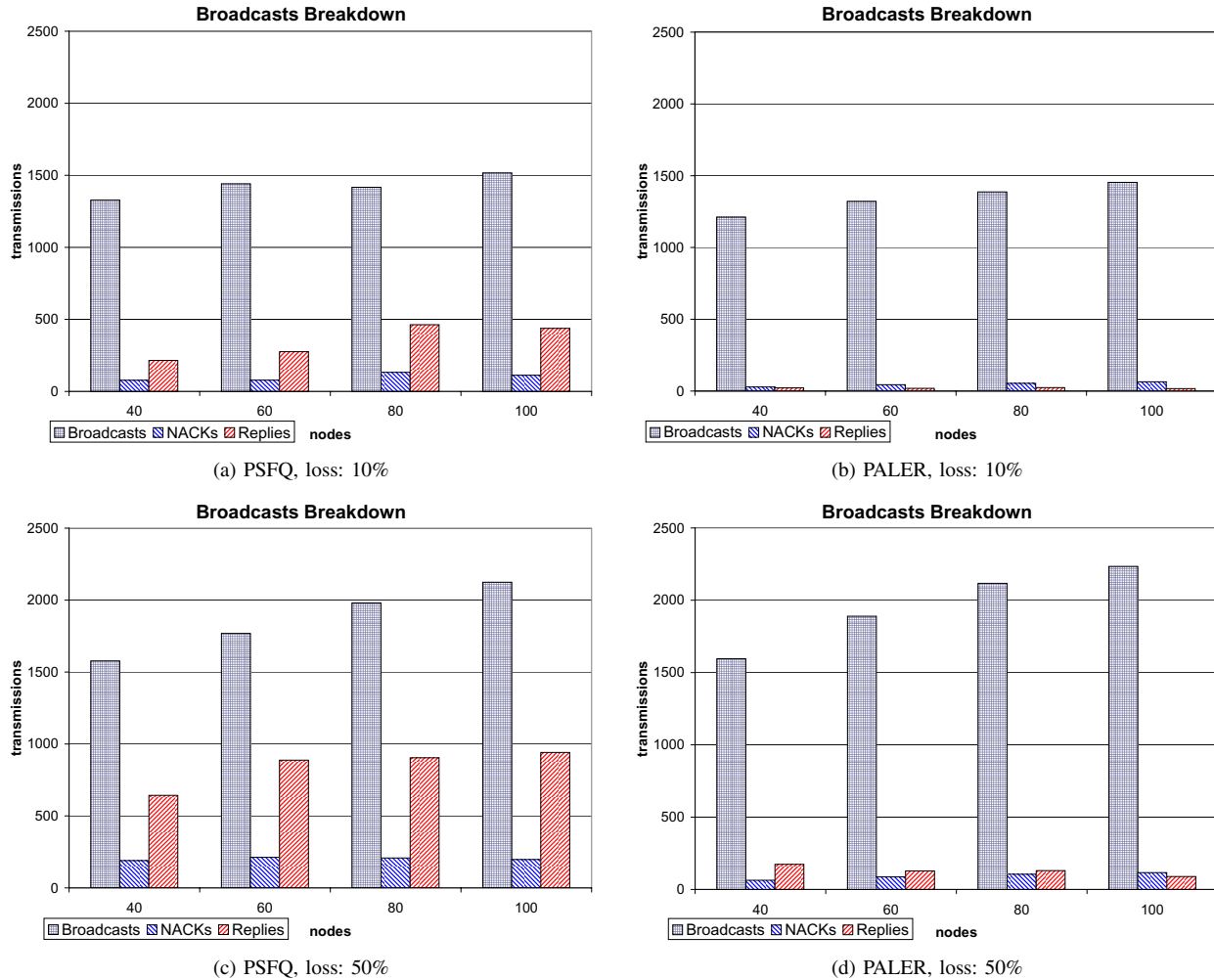


Fig. 6: Breakdown of the types of transmissions during a simulation of PALER. Simulation had a fixed field size with increasing node count. PSFQ graphs as shown in Figure 1 are included for comparison (a)&(b) shows a comparison for a low-loss environment, 10%. (c)&(d) shows a comparison for a high-loss environment 50%. PALER shows a much smaller percentage of NACK and reply messages, relative to broadcasts, in comparison to PSFQ.

improvement in energy efficiency in moderate densities. This increase in energy efficiency came at the cost of slightly higher latencies than the counter method implementation. However, the average latencies of both implementations scaled considerably better than PSFQ, and in many sensor network environments the savings in power consumption will outweigh the small increase in latency.

We plan to further develop a comprehensive sensor network reprogramming framework. Additional methods to improve energy efficiency will be explored, such as dynamically adjusting radio transmission power. Neighbor locality information may be estimated from received signal strength. This information may be used to determine the minimum transmission power needed to reach all neighbors. The neighbor pruning mechanism leads nicely in this direction, since a list of neighbors requiring the segment is maintained. If the estimated

transmission power to reach a neighbor is added to each node in the list, nodes may dynamically adjust their transmission power to the maximum of these values in the remaining list of neighbors. This has the potential to reduce the total power expended for distributing new code, and reduce contention and collisions since the transmission range will be limited to the desired coverage area.

Other areas for future work include adding version metadata to facilitate automatic updates and maintenance. A fully implemented system should be capable of ensuring that all nodes maintain the most current version of code. This will require the ability to detect new or outdated versions and update any necessary nodes in a manner that allows the network to continue to perform reliably during the update. We would also like to support group-centric re-tasking. Sensor network nodes may be organized into groups which perform

specialized functions. These group assignments may adjust dynamically to respond to the environment. We will explore methods to efficiently support such functionality.

ACKNOWLEDGMENTS

This work is supported in part by NSF under grant number CNS-0545899.

REFERENCES

- [1] A. K. L. Wan, C.-Y.; Campbell, "Pump-slowly, fetch-quickly (psfq): A reliable transport protocol for sensor networks," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 4, pp. 862–872, April 2005.
- [2] L. Rizzo and L. Vicisano, "A reliable multicast data distribution protocol based on software FEC techniques," in *The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS'97)*, Sani Beach, Chalkidiki, Greece, Jun. 1997. [Online]. Available: citeseer.ist.psu.edu/rizzo97reliable.html
- [3] Y. Yi and S. Lee, "Demand multicast routing protocol," 2002. [Online]. Available: citeseer.ist.psu.edu/yi02demand.html
- [4] C. Ho, K. Obraczka, G. Tsudik, and K. Viswanath, "Flooding for reliable multicast in multi-hop ad hoc networks," in *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Seattle, WA, 1999, pp. 64–71. [Online]. Available: citeseer.ist.psu.edu/ho99flooding.html
- [5] C.-S. Hsu, Y.-C. Tseng, and J.-P. Sheu, "An efficient reliable broadcasting protocol for wireless mobile ad hoc networks," *Ad Hoc Netw.*, vol. 5, no. 3, pp. 299–312, 2007.
- [6] E. Pagani and G. P. Rossi, "Reliable broadcast in mobile multihop packet networks," in *MobiCom '97: Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1997, pp. 34–42.
- [7] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wirel. Netw.*, vol. 8, no. 2/3, pp. 153–167, 2002.
- [8] J. Wu and F. Dai, "Broadcasting in ad hoc networks based on self-pruning." [Online]. Available: citeseer.ist.psu.edu/wu03broadcasting.html
- [9] I. Stojmenovic, M. Seddigh, and J. Zunic, "Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 14–25, 2002. [Online]. Available: citeseer.ist.psu.edu/stojmenovic01dominating.html
- [10] J. Wu and H. Li, "A dominating-set-based routing scheme in ad hoc wireless networks," *Telecommunication Systems*, vol. 18, no. 1–3, pp. 13–36, 2001. [Online]. Available: citeseer.ist.psu.edu/wu99dominatingsetbased.html
- [11] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "Energy-efficient broadcast and multicast trees in wireless networks," *Mob. Netw. Appl.*, vol. 7, no. 6, pp. 481–492, 2002.
- [12] F. Bian, A. Goel, C. S. Raghavendra, and X. Li, "Energy-efficient broadcasting in wireless ad hoc networks lower bounds and algorithms," *Journal of Interconnection Networks*, vol. 3, no. 3-4, pp. 149–166, 2002.
- [13] M. X. C. J. S. M. M. D.-Z. Du, "Energy-efficient broadcast and multicast routing in ad hoc wireless networks," *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, pp. 87–94, 9-11 April 2003.
- [14] S. Park, R. Vedantham, R. Sivakumar, and I. Akyildiz, "A scalable approach for reliable downstream data delivery in wireless sensor networks," 2004. [Online]. Available: citeseer.ist.psu.edu/park04scalable.html
- [15] N. Rahnvard and F. Fekri, "Crbcast: a collaborative rateless scheme for reliable and energy-efficient broadcasting in wireless sensor networks," *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, pp. 276–283, 19-21 April 2006.
- [16] D. Lun, N. Ratnakar, R. Koetter, M. Medard, E. Ahmed, and H. Lee, "Achieving minimum-cost multicast: a decentralized approach based on network coding," *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, pp. 1607–1617 vol. 3, 13-17 March 2005.
- [17] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," 2004. [Online]. Available: citeseer.ist.psu.edu/levis04trickle.html
- [18] Y. Busnel, M. Bertier, E. Fleury, and A.-M. Kermarrec, "Gcp: Gossip-based code propagation for large-scale mobile wireless sensor networks," 2007. [Online]. Available: <http://www.citebase.org/abstract?id=oai:arXiv.org:0707.3717>
- [19] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2006, pp. 139–152.
- [20] R. Barr and Z. Haas, "Jist/swans website," April 2004. [Online]. Available: <http://jist.ece.cornell.edu/>
- [21] R. Barr, Z. J. Haas, and R. van Renesse, "Jist: an efficient approach to simulation using virtual machines: Research articles," *Softw. Pract. Exper.*, vol. 35, no. 6, pp. 539–576, 2005.