PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks

Chieh-Yih Wan
Dept. of Electrical Engineering
Columbia University
New York, NY 10027
wan@comet.columbia.edu

Andrew T. Campbell
Dept. of Electrical Engineering
Columbia University
New York, NY 10027
campbell@comet.columbia.edu

Lakshman Krishnamurthy
Network Architecture Laboratory
Intel Research and Development
Hillsboro, OR 97124
lakshman.krishnamurthy@intel.com

ABSTRACT

We propose *PSFQ (Pump Slowly, Fetch Quickly)*, a reliable transport protocol suitable for a new class of reliable data applications emerging in wireless sensor networks. For example, currently sensor networks tend to be application specific and are typically hard-wired to perform a specific task efficiently at low cost; however, there is an emerging need to be able to re-task or reprogram groups of sensors in wireless sensor networks on the fly (e.g., during disaster recovery). Due to the application-specific nature of sensor networks, it is difficult to design a single monolithic transport system that can be optimized for every application. PSFQ takes a different approach and supports a simple, robust and scalable transport that is customizable to meet the needs of different reliable data applications. To our knowledge there has been little or no work on the design of an efficient reliable transport protocol for wireless sensor networks, even though some techniques found in IP networks have some relevance to the solution space, such as, the body of work on reliable multicast. We present the design and implementation of PSFQ, and evaluate the protocol using the ns-2 simulator and an experimental wireless sensor testbed based on Berkeley motes. We show through simulation and experimentation that PSFQ can out perform existing related techniques (e.g., an idealized SRM scheme) and is highly responsive to the various error conditions experienced in wireless sensor networks, respectively.

Categories and Subject Descriptors

C.2.1. [**Computer-Communications Networks**]: Network Protocols, Wireless Communications.

General Terms: Algorithms, Design, Performance.

Keywords

Reliable transport protocols, wireless sensor networks.

This research is supported in part by the NSF WIRELESS TECHNOLOGY Award ANI-9979439 and with support from Intel Corp.

WSNA'02, September 28, 2002, Atlanta, Georgia, USA.

1. INTRODUCTION

There is a considerable amount of research in the area of wireless sensor networks ranging from real-time tracking to ubiquitous computing where users interact with potentially large numbers of embedded devices. This paper addresses the design of system support for a new class of applications emerging in wireless sensor networks that require reliable data delivery. One such application that is driving our research is the reprogramming or "re-tasking" of groups of sensors. This is one new application in sensor networks that requires underlying transport protocol to support reliable data delivery. Today, sensor networks tend to be application specific, and are typically hard-wired to perform a specific task efficiently at low cost. We believe that as the number of sensor network applications grows, there will be a need to build more powerful general-purpose hardware and software environments capable of reprogramming or "re-tasking" sensors to do a variety of tasks. These general-purpose sensors would be capable of servicing new and evolving classes of applications. Such systems are beginning to emerge. For example, the Berkeley motes [1] [2] are capable of receiving code segments from the network and assembling them into a completely new execution image in EEPROM secondary store before re-tasking a sensor.

Unlike traditional networks (e.g., IP networks), reliable data delivery is still an open research question in the context of wireless sensor networks. To our knowledge there has been little or no work on the design of reliable transport protocols for sensor networks. This is, as one would expect, since the vast majority of sensor network applications do not require reliable data delivery. For example, in applications such as temperature monitoring or animal location tracking, the occasional loss of sensor readings is tolerable, and therefore, the complex protocol machinery that would ensure the reliable delivery of data is not needed. Directed diffusion [3] is one of a representative class of data dissemination mechanisms, specifically designed for a general class of applications in sensor networks. Directed diffusion provides robust dissemination through the use of multi-path data forwarding, but the correct reception of all data messages is not assured. We observed that in the context of sensor networks, data that flows from sources to sinks is generally tolerable of loss. On the other hand, however, data that flows from sinks to sources for the purpose of control or management (e.g., re-tasking sensors) is sensitive to message loss. For example, disseminating a program image to sensor nodes is problematic. Loss of a single message associated with code segment or script would render the image useless and the re-tasking operation a failure.

There are a number of challenges associated with the development of a reliable transport protocol for sensor networks. For example, in the case of a re-tasking application there may be a need to reprogramming certain groups of sensors (e.g., within a disaster recovery area). This would require addressing groups of sensors, loading new binaries into them, and then, switching over to the new re-tasked application in a controlled manner. Another example of new reliable data requirements relates to simply injecting scripts into sensors to customize them rather than sending complete, and potentially bandwidth demanding, code segments. Such re-tasking becomes very challenging as the number of sensor nodes in the network grows. How can a transport offer suitable support for such a re-tasking application where possibly hundreds and thousands of nodes need to be reprogrammed in a controlled, reliable, robust and scalable manner?

Reliable point-to-point, or more appropriately, multicast transport mechanisms are well understood in conventional IP-style communication networks, where nodes are identified by their end-points. However, these schemes (e.g., TCP, XTP [4], SRM [5]) cannot be efficiently applied to sensor networks mainly because of the unique communication challenges presented by wireless sensor networks, including the need to support cluster-based communications, wireless multi-hop forwarding, application-specific operations, and lack of clean layering for the purposes of optimization, etc. There is a need for the development of a new reliable transport protocol, which can respond to the unique challenges posed by sensor networks. Such an approach must be lightweight enough to be realized even on low-end sensor nodes, such as, the Berkeley mote series of sensors. A reliable transport protocol must be capable of isolating applications from the unreliable nature of wireless sensor networks in an efficient and robust manner. The error rates experienced by these wireless networks can vary widely, and therefore, any reliable transport protocol must be capable of delivering reliable data to potentially large groups of sensors under such conditions. This is very challenging.

In this paper, we propose *PSFQ (Pump Slowly, Fetch Quickly)*, a new reliable transport protocol for wireless sensor networks. Due to the application-specific nature of sensor networks, it is hard to generalize a specific scheme that can be optimized for every application. Rather, the focus of this paper is the design and evaluation of a new transport system that is simple, robust, scalable, and customizable to different applications' needs. PSFQ represents a simple approach with minimum requirements on the routing infrastructure (as opposed to IP multicast/unicast routing requirements), minimum signaling thereby reducing the communication cost for data reliability, and finally, responsive to high error rates allowing successful operation even under highly error-prone conditions.

The paper is organized as follows. Section 2 presents the PSFQ model and discusses the design choices. Section 3 presents the detail design of PSFQ's pump, fetch and report operations. Section 4 presents an evaluation of the protocol and comparison to Scalable Reliable Multicast (SRM) [5] using the ns-2 simulator.

Section 5 present experimental results from the implementation of PSFQ in an experimental wireless sensor testbed based on Berkeley motes. Finally, we present some concluding remarks in Section 6.

2. PROTOCOL DESIGN SPACE

The key idea that underpins the design of PSFQ is to distribute data from a source node by pacing data at a relatively slow speed ("pump slowly"), but allowing nodes that experience data loss to fetch (i.e., recover) any missing segments from immediate neighbors very aggressively (local recovery, "fetch quickly"). We assume that message loss in sensor networks occurs because of transmission errors due to the poor quality of wireless links rather than traffic congestion since most sensor network applications generate light traffic most of the time. Messages that are lost are detected when a higher sequence number than expected is received at a node triggering the fetch operation. Such a system is equivalent to a negative acknowledgement system. The motivation behind our simple model is to achieve loose delay bounds while minimizing the lost recovery cost by localized recovery of data among immediate neighbors. PSFQ is designed to achieve the following goals:

- to ensure that all data segments are delivered to all the intended receivers with minimum[1] support from the underlying transport infrastructure;
- to minimize the number of transmissions for lost detection and recovery operations with minimal signaling;
- to operate correctly even in an environment where the radio link quality is very poor; and
- to provide loose delay bounds for data delivery to all the intended receivers.

2.1 Hop-by-Hop Error Recovery

To achieve these goals we have taken a different approach in comparison to traditional end-to-end error recovery mechanisms in which only the final destination node is responsible for detecting loss and requesting retransmission. Despite the various differences in the communication and service model, the biggest problem with end-to-end recovery has to do with the physical characteristic of the transport medium: sensor networks usually operate in harsh radio environments, and rely on multi-hop forwarding techniques to exchange messages. Error accumulates exponentially over multi-hops. To simply illustrate this, assume that the packet error rate of a wireless channel is p then the chances of exchanging a message successfully across a single hop is (1-p). The probability that a message is successfully received across n hops decrease quickly to $(1-p)^n$. For a negative acknowledgement system, at least one message has to be received correctly at the destination after a loss has happened in order to detect the loss. Figure 1 illustrates this problem numerically. The success rate denotes the probability of a successful delivery of a message in end-to-end model, which is $(1-p)^n$. Figure 1 plots the success rate as function of the network size (in terms of the number of hops) and shows that for larger network it is almost impossible to deliver a single message using an end-to-end approach in a lossy link environment when the error rate is larger than 20%.

From Figure 1, we can see that end-to-end approach performs fine even across large numbers of hops in highly reliable link environments where the channel error rate is less than 1%, (e.g., found in a wired network). Under such conditions the probability of

[1] PSFQ only requires a MAC that is capable of broadcasting operations (e.g., CSMA, TDMA).

a successful delivery is well above 90%. This requirement can be easily met in wired network and even in wireless LAN networks, such as IEEE 802.11. However, it is not the case in sensor networks. Due to the various resources and design constraints on a sensor node, sensor network operations require low-power RF communications, which cannot rely on using high power to boost the link reliability when operating under harsh radio conditions. In military applications or disaster recovery efforts, it is not unusual to have channel error rate that is in the range of 5% ~ 10% or even higher. This observation suggests that end-to-end error recovery is not a good candidate for reliable transport in wireless sensor networks, as indicated by the result shown in Figure 1.

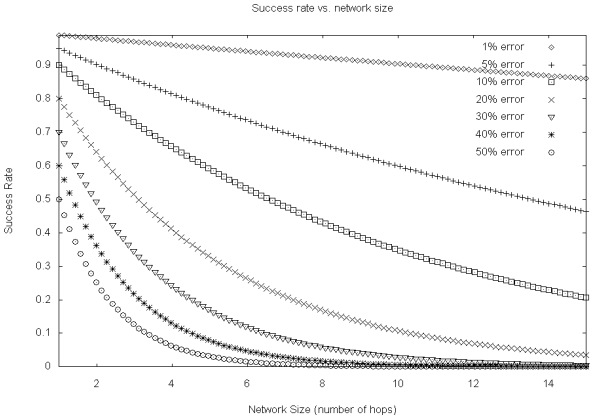

Figure 1. Probability of successful delivery of a message using an end-to-end model across a multi-hop network.

We propose hop-by-hop error recovery in which intermediate nodes also take responsibility for loss detection and recovery so reliable data exchange is done on a hop-by-hop manner rather than an end-to-end one. Several observations support this choice. First, this approach essentially segments multihop forwarding operations into a series of single hop transmission processes that eliminate error accumulation. The chances of exchanging a message successfully across a single hop is (1-p). Therefore, the probability of detecting loss in a negative acknowledgement system is proportional to (1-p) in a hop-by-hop approach (independent of network size), rather than decreasing exponentially with growing network size as in the case of end-to-end approaches. The hop-by-hop approach thus scales better and is more error tolerable. Second, the extra cost of involving intermediate nodes in the loss detection process (i.e., intermediate nodes must keep track of the data they forward, which involves allocating sufficient data cache space) can be justified in sensor networks. Typically, communication in wireless sensor networks is not individual-based but is group or cluster-based communications. Consider some of the example applications that require reliable data delivery, (e.g., re-tasking the sensor nodes, or for control or management purposes), the intended receivers are often the whole group of sensor nodes in the vicinity of a source node (a user). In this case, intermediate nodes are also the intended receiver of data, therefore there is no extra cost in transiting data through nodes.

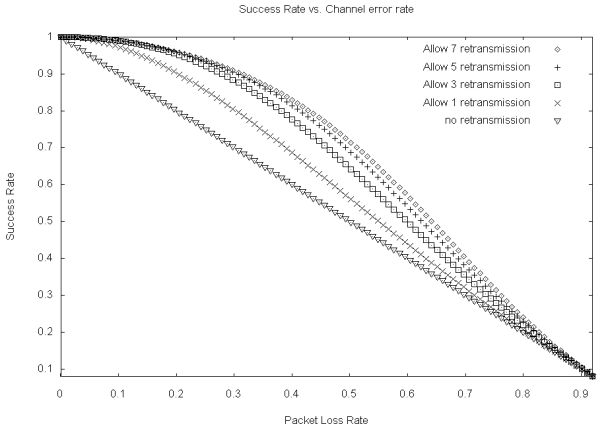

Figure 2. Probability of successful delivery of a message when the mechanism allows multiple retransmissions before the next packet arrival.

2.2 Fetch/Pump Relationship

For a negative acknowledgement system, the network latency would be dependent on the expected number of retransmissions for successful delivery. In order to achieve loose delay bound for the data delivery, it is essential to maximize the probability of successful delivery of a packet within a "controllable time frame". An intuitive approach to doing this would be to enable the possible multiple retransmissions of packet n (therefore increasing the chances of successful delivery) before the next packet n+1 arrives; in other words, clear the queue at a receiver (e.g., an intermediate sensor) before new packets arrive in order to keep the queue length small and hence reduce the delay. However, it is non-trivial to determine the optimal number of retransmissions that tradeoff the success rate (probability of successful delivery of a single message within a time frame) against wasting too much energy on retransmissions. In order to investigate and justify this design decision, we analyze a simple model, which approximates this mechanism. Let p be the packet loss rate of a wireless channel. Assume that p stays constant at least during the controllable time frame, it can be shown that in a negative acknowledgement system, the probability of a successful delivery of a packet between two nodes that allows n retransmission can be expressed recursively as:

- $(1-p) + p \times \Omega(n) \qquad (n \geq 1)$
- $\Omega(n) = \Phi(1) + \Phi(2) + \ldots + \Phi(n)$
- $\Phi(n) = (1-p)^2 \times [1 - p - \Phi(1) - \Phi(2) - \ldots - \Phi(n-1)]$
  $\Phi(0) = 0$

Where $\Omega(n)$ is the probability of successful recovery of a missing segment within n retransmission, $\Phi(n)$ is the probability of the successful recovery of the missing segment at n^{th} retransmission.

The above expressions are evaluated numerically against packet loss rate p, as shown in Figure 2. The straight line that denotes "no retransmission" is simply the probability of receiving an error free packet over the channel, which is 1-p; this line represents the case when no retransmission is attempted within a time frame before next segment is "pumped" into the channel. Figure 2 demonstrates the impact of increasing the number of retransmissions up to n equal to 7. We can see that substantial improvements in the success rate can be gained in the region where the channel error rate is between 0 and 60%. However, the additional benefit of allowing more retransmission diminishes quickly and becomes negligible when n is

larger than 5. This simple analysis implies that the optimal ratio between the timers associated with the pump and fetch operations is approximately 5. This simple model also shows that at most ≈20% gain in the success rate can be achieved with this approach, as indicated from the result shown in Figure 2.

2.3 Multi-modal Operations

There are several important considerations associated with the pump operation's ability to localize loss events while maximizing the probability of in-sequence data delivery. As a result of these considerations the PSFQ pump operation is designed for multi-modal operations, providing a graceful tradeoff between the classic "packet forwarding" and "store-and-forward" communication paradigms depending on the wireless channel conditions experienced. In what follows, we discuss the reasoning behind this key design choice.

Figure 3 illustrates an example in which a local loss event propagates to downstream nodes. The propagation of a loss event could cause a serious waste of energy. A loss event will trigger error recovery operations that attempt to fetch the missing packet quickly from immediate neighbors by broadcasting a "Nack" message. However, for nodes B and C in Figure 3, none of their neighbors have the missing packet, therefore the loss cannot be recovered and the control messages associated with the fetch operation are wasted. As a result, it is necessary to make sure that intermediate nodes only relay messages with continuous sequence numbers. In other words, node A in Figure 3 should not relay message #4 until it successfully recovers message #3.

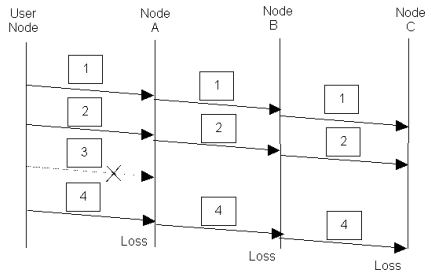

Figure 3. Propagation of a loss event. The packet with sequence number 3 sent by the user node-to-node A is lost or corrupted due to channel error. The subsequent packet with sequence number 4 received by node A triggers a loss event. If this packet is forwarded to node B, another loss event is triggered at node B. When this packet forwarded from node B to node C, it will again trigger another loss event at node C. The loss event will keep on propagating in this manner until the TTL reaches 0 and packet is dropped.

The use of data cache is required to buffer both message #3 and #4 to ensure in-sequence data forwarding and ensure complete recovery for any fetch operations from downstream nodes. Note that cache size effect is not investigated here but in our reference application, the cache keeps all code segments. This pump mechanism not only prevents propagation of loss events and the triggering of unnecessary fetch operations from downstream nodes, but it also greatly contributes toward the error tolerance of the protocol against channel quality. By localizing loss events and not relaying any higher sequence number messages until recovery has taken place, this mechanism operates in a similar fashion to a store-and-forward approach where an intermediate node relays a file only after the node has received the complete file. The store-and-forward approach is effective in highly error-prone environments because it essentially segments the multi-hop forwarding operations into a series of single hop transmission processes (errors accumulate exponentially for multi-hop communication, as discuss in Section 2.1).

However, the classic store-and-forward approach suffers from large delay even in error free environments. Therefore, store-and-forward is not a suitable choice in most cases although it could be the only choice in highly error-prone environments. PSFQ benefits from the following tradeoff between store-and-forward and multihop forwarding. The pump operation operates in a multihop packet forwarding mode during periods of low errors when lost packets can be recovered quickly, and behaves more like store-and-forwarding communications when the channel is highly error-prone. Therefore, as mentioned earlier, PSFQ exhibits a novel multi-modal property that provides a graceful tradeoff between forwarding and store-and-forward paradigms, depending on the channel conditions encountered.

The observations presented in this section motivate our "pump slowly, fetch quickly" paradigm. The fetch operation should be fast relative to the pump operation as to allow a reasonable number of retransmissions in order to maximize the success rate of receiving a data segment within a controllable time frame. In addition, these insights suggest the need for in-sequence forwarding at intermediate nodes for the pump operation.

3. PROTOCOL DESCRIPTION

PSFQ comprises three functions: message relaying (*pump operation*), relay-initiated error recovery (*fetch operation*) and selective status reporting (*report operation*). A user (source) injects messages into the network and intermediate nodes buffer and relay messages with the proper schedule to achieve loose delay bounds. A relay node maintains a data cache and uses cached information to detect data loss, initiating error recovery operations if necessary. As in many negative acknowledgement systems, there is no way for the source to know when the receivers have received the data messages. This has several drawbacks. First, the data segments must be retained indefinitely at the source for possible retransmissions. Next, it is important for the user to obtain statistics about the dissemination status (e.g., the percentage of nodes that have obtained the complete execution image for a re-tasking application) in the network as a basis for subsequent decision-making, (e.g., the correct time to switch over to the new task in the case of re-tasking). Therefore, it is necessary to incorporate a feedback and reporting mechanism into PSFQ that is flexible (i.e., adaptive to the environment) and scalable (i.e., minimize the overhead).

In what follows, we describe the main PSFQ operations (viz. pump, fetch and report) with specific reference to a re-tasking applications -- one in which a user needs to re-task a set of sensor nodes in the vicinity of its location by distributing control scripts or binary code segments into the targeted sensor nodes. A number of concerns associated with this simple PSFQ model are related to the timer issues that control the loose service properties, such as, statistical delay bounds. Important protocol parameters include message pumping speed and loss recovery speed.

3.1 Pump Operation

Recall that PSFQ is not a routing solution but a transport scheme. In the case where a specific node (instead of a whole group) needs to be addressed, PSFQ can work on top of existing routing or data dissemination scheme, (e.g. directed diffusion, DSDV, etc.), to achieve reliability. A user node uses TTL-based methods to control the scope of its re-tasking operation; note that, since the term "source" in sensor network usually denotes a sensor node which has sensed data to be sent, we use the term "user node" in this paper to refer to a node which distributes the code segments to avoid confusion. To enable local loss recovery and in-sequence data delivery, a data cache is created and maintained at intermediate nodes.

We define an "inject message" associate with the pump operation in PSFQ. The inject message has four fields in its header: *i)* file ID, *ii)* file length *iii)* sequence number, and *iv)* TTL[2]. The message payload carries the data fragment (code segment).

The pump operation is important in controlling four performance factors associated with our example re-tasking application. First, the timely dissemination of code segments to all target nodes used for re-tasking the sensor nodes. Second, to provide basic flow control so that the re-tasking operation does not overwhelm the regular operations of the sensor network, (e.g., monitoring environmental conditions). Next, for densely deployed sensor networks in which nodes are generally within transmission range of more than one neighboring node, we need to avoid redundant messaging to save power and to minimize contention/collision over the wireless channel. Finally, we want to localize loss, avoiding the propagation of loss events to downstream nodes. This requires mechanisms to ensure in-sequence data forwarding at intermediate nodes, as discussed in Section 2.3. The first two performance factors discussed above require proper scheduling for data forwarding. We adopt a simple scheduling scheme, which use two timers T_{min} and T_{max} for scheduling purposes.

3.1.1 Pump Timers

A user node broadcasts a packet to its neighbors every T_{min} until all the data fragments has been sent out. In the meantime, neighbors that receive this packet will check against their local data cache discarding any duplicates. If this is a new message, PSFQ will buffer the packet and decrease the TTL field in the header by 1. If the TTL value is not zero and there is no gap in the sequence number, then PSFQ sets a schedule to forward the message. The packet will be delayed for a random period between T_{min} and T_{max} and then relayed to its neighbors that are one or more hops away from the source. In this specific reference case, PSFQ simply rebroadcast the packet. Note that the data cache has several potential uses, one of which is loop prevention, i.e., if a received data message has a matching data cache entry then the data message is silently dropped. A packet propagates outward from the source node up to TTL hops away in this mode. The random delay before forwarding a message is necessary to avoid collisions because RTS/CTS dialogues are inappropriate in broadcasting operations when the timing of rebroadcasts among interfering nodes can be highly correlated.

T_{min} has several considerations. First, there is a need to provide a time-buffer for local packet recovery. One of the main motivations behind the PSFQ paradigm is to recover lost packets quickly among immediate neighboring nodes within a controllable time frame. T_{min} serves such a purpose in the sense that a node has an opportunity to recover any missing segment before the next segment come from its upstream neighbors, since a node must wait at least T_{min} before forwarding a packet in pump state. Next, there is a need to reduce redundant broadcasts. In a densely deployed network, it is not unusual to have multiple immediate neighbors within radio transmission range. Since we use broadcast instead of unicast for data relaying in our reference application, too many data forwarding rebroadcasts are considered to be redundant if all its neighbors already have the message. In [7], the authors show that a rebroadcast system can provide only 0 ~ 61% additional coverage[3] over that already covered by the previous transmissions. Furthermore, it is shown that if a message has been heard more than 4 times, the additional coverage is below 0.05%. T_{min} associated with the pump operation provides an opportunity for a node to hear the same message from other rebroadcasting nodes before it would actually have started to transmit the message. A counter is used to keep track of the number of times the same broadcast message is heard. If the counter reaches 4 before the scheduled rebroadcast of a message then the transmission is cancelled and the node will not relay the specific message because the expected benefit (additional coverage) is very limited in comparison to the cost of transmission. T_{max} can be used to provide a loose statistical delay bound for the last hop to successfully receive the last segment of a complete file, (e.g., a program image or script). Assuming that any missing data is recovered within one T_{max} interval using the aggressive fetch operation described in next section, then the relationship between delay bound D(n) and T_{max} is as follows:

$$D(n) = T_{max} \times n \times (Number\ of\ hops),$$

where n is the number of fragments of a file.

3.2 Fetch Operation

Since most sensor network applications generate light traffic most of the time, message loss in the sensor networks usually occurs because of transmission errors due to poor quality wireless links and not because of traffic congestion. This is not to say that congestion cannot occur but that the vast majority of loss in these networks is associated with errors. This is especially true considering the environment in which sensor networks operate in is highly unpredictable, and therefore, the quality of the communication links can vary considerably due to obstructions or hostile ambient conditions.

A node goes into fetch mode once a sequence number gap in a file fragments is detected. A fetch operation is the proactive act of requesting a retransmission from neighboring nodes once loss is detected at a receiving node. PSFQ uses the concept of "loss aggregation" whenever loss is detected; that is, it attempts to batch up all message losses in a single fetch operation whenever possible.

3.2.1 Loss Aggregation

There are several considerations associated with loss aggregation. The first consideration relates to bursty loss. Data loss is often

[2] One bit of the TTL field in the inject message is used as the "report" bit in order to solicit a report message from target nodes. The use of this bit is discussed in Section 3.3.

[3] Corresponds to the number of additional nodes that can be reached by a rebroadcast.

correlated in time because of fading conditions and other channel impairments. As a result loss usually occurs in batches. Therefore, it is possible that more than one packet is lost before a node can detect loss by receiving a packet with higher sequence numbers than expected. PSFQ aggregates loss such that the fetch operation deals with a "window" of lost packets instead of a single packet loss. Next, in a dense network where a node usually has more than one neighbor, it is possible that each of its neighbors only obtains or retains part of the missing segments in the loss window. PSFQ allows different segments of the loss window to be recovered from different neighbors. In order to reduce redundant retransmissions of the same segment, each neighbor waits for a random time before transmitting segments, (i.e., sets a retransmission timer to a random value, and sends the packet only when the timer goes off). Other nodes that have the data and scheduled retransmissions will cancel their timers if they hear the same "repair" (i.e., retransmission of a packet loss) from a neighboring node. Third, in poor radio environments successive loss could occur including loss of retransmissions and fetch control messages. Therefore, it is not unusual to have multiple gaps in sequence number of messages received by a node after several such failures. Aggregating multiple loss windows in the fetch operation increases the likelihood of successful recovery in the sense that as long as one fetch control message is heard by one neighbor all the missing segments could be resent by this neighbor.

3.2.2 Nack Messaging

We define a NACK message associate with the fetch operation as the control message that requests a retransmission from neighboring nodes. The NACK message has at least three header fields (this could be more) with no payload: *i)* file ID, *ii)* file length, and *iii)* loss window. The loss window represents a pair of sequence numbers that denote the left and right edge of a loss window (see example below). When there is more than one sequence number gap, each gap corresponds to a loss window and will be appended after the first three header fields in the NACK message. For example, if a node receives messages with sequence number (3,5,6,9,11), then computes 3 gaps and hence 3 loss windows that are (4,4), (7,8) and (10,10), respectively.

3.2.3 Fetch Timer

In fetch mode, a node aggressively sends out NACK messages to its immediate neighbors to request missing segments. If no reply is heard or only a partial set of missing segments are recovered within a period T_r ($T_r < T_{max}$, this timer defines the ratio between pump and fetch) then the node will resend the NACK every T_r interval (with slight randomization to avoid synchronization between neighbors) until all the missing segments are recovered or the number of retries exceed a preset threshold thereby ending the fetch operation. The first NACK is scheduled to be sent-out within a short delay that is randomly computed between 0 and Δ ($<<T_r$). The first NACK is cancel (to keep the number of duplicates low) in the case where a NACK for the same missing segments is overheard from another node before the NACK is sent. Since Δ is small, the chance of this happening is relatively small. In general, retransmissions in response to a NACK coming from other nodes are not guaranteed to be overheard by the node that cancelled its first NACK. In [7] the authors show that at most there is a 40% chance that the canceling node receives the retransmitted data under such conditions. Note, however that a node that cancel its NACK will eventually resend a NACK within T_r if the missing segments are not recovered, therefore, such an approach is safe and beneficial given the trade offs.

To avoid the message implosion problem, NACK messages never propagate; that is, neighbors do not relay NACK messages unless the number of times the same NACK is heard exceeds a predefined threshold while the missing segments requested by the NACK message are no longer retained in a node's data cache. In this case, the NACK is relayed once, which in effect broadens the NACK scope to one more hop to increase the chances of recovery. Such a situation should be a rare occurrence, since loss is triggered when a packet with a higher sequence number than expected is received. The upstream node that sent this packet maintains a data cache and must have obtained all the preceding segments prior to sending this higher sequence number packet, which in this scenario, failed to reach the "fetching" node. The probability that all neighbors do not have the missing segments is very low. In our reference application, since all nodes must keep the code for re-tasking purposes, all segments that have been received correctly can be pulled out of cache or external storage. Therefore, NACK messages never need to be propagated in this case.

Each neighbor that receives a NACK message checks the loss window field. If the missing segment is found in its data cache, the neighboring node schedules a reply event (sending the missing segment) at a random time between (0, T_r). Neighbors will cancel this event whenever a reply to the same NACK for the same segment (same file ID and sequence number) is overheard. In the case where the loss window in a NACK message contains more than one segment to be resent, or more than one loss window exists in the NACK message, then neighboring nodes that are capable of recovering missing segments will schedule their reply events such that packets are sent (in-sequence) at a speed that is not faster than once every T_r.

Nodes in fetch mode maintain their own loss windows to keep track of missing segments. When a fetching node receives a portion of a missing segment that only represents a partial amount of its loss window, it needs to update its loss windows accordingly, possibly splitting the loss window or creating new loss windows.

3.2.4 Proactive Fetch

As in many negative acknowledgement systems, the fetch operation described above is a reactive loss recovery scheme in the sense that a loss is detected only when a packet with higher sequence number is received. This could cause problems on rare occasions; for example, if the last segment of a file is lost there is no way for the receiving node to detect this loss[4] since no packet with higher sequence number will be sent. In addition, if the file to be injected into the network is small (e.g., a script instead of binary code), it is not unusual to lose all subsequent segments up to the last segment following bursty loss. In this case, the loss is also undetectable and thus non-recoverable with such a reactive loss detection scheme. In order to cope with these problems, PSFQ supports a "proactive fetch" operation such that a node can also enter fetch mode proactively and send a NACK message for the next segment or the remaining segments if the last segment[5] has not been received and

[4] A node knows that it has not received the last segment, but it does not know whether the last segment is lost or will be relayed at some point in the future.

[5] The last segment could be identified by the file length field in the inject message, which is essentially the sequence number of the last segment.

no new packet is delivered after a period of time T_{pro}. When a proactive fetch operation is triggered, a node will manually create a loss event and send out a NACK control message with the desired loss window.

The proactive fetch mechanism is designed to autonomously trigger the fetch mode at the proper time. If fetch mode is triggered too early, then the extra control messaging might be wasted since upstream nodes may still be relaying messages or they may not have received the necessary segments. In contrast, if fetch mode is triggered too late, then the target node might waste too much time waiting for the last segment of a file, significantly increasing the overall delivery latency of a file transfer. The correct choice of T_{pro}, must consider two issues. First, in our reference application (where each segment of a file needs to be kept in data cache or external storage for the re-tasking operation), the proactive fetch mechanism will Nack for all the remaining segments up to the last segment if the last segment has not been received and no new packet arrives after a period of time T_{pro}. T_{pro} should be proportional to the difference between last highest sequence number (S_{last}) packet received and the largest sequence number (S_{max}) of the file (the difference is equal to the number of remaining segments associated with the file), i.e. $T_{pro} = \alpha * (S_{max} - S_{last}) * T_{max}$ ($\alpha \geq 1$). S_{max} is the file length found in the header field, T_{max} is the timer defined in the previous section, α is a scaling factor to adjust the delay in triggering proactive fetch and should be set to 1 for most operational cases. This definition of T_{pro} guarantees that a node will wait long enough until all upstream nodes have received all segments before a node moves into the proactive fetch mode. In addition, this enables a node to start proactive fetch earlier when it is closer to the end of a file, and wait longer when it is further from completion. Such an approach adapts nicely to the quality of the radio environment. If the channel is in a good condition, then it is unlikely to experience successive packet loss; therefore, the reason for the reception of no new messages prior to the anticipated last segment is most likely due to the loss of the last segment, hence, it is wise to start the proactive fetch promptly. In contrast, a node is likely to suffer from successive packet loss when the channel is error-prone; therefore, it makes sense to wait longer before pumping more control messages into the channel. If the sensor network is known to be deployed in a harsh radio environment then α should be set larger than 1 so that a node waits longer before starting the proactive fetch operation. Finally, a node that starts proactive fetch will create a loss window with the left edge equal to (S_{last}+1) and right edge equal to S_{max} before sending a NACK message. The rest of the actions taken in response to a NACK message are exactly the same as normal fetch operations including the retransmission of NACKs and the handling of loss windows, as discussed earlier.

In other applications where the data cache size is small and nodes only can keep a portion of the segments that have been received, the proactive fetch mechanism will Nack for the same amount of segments (or less) that the data cache can maintain. In this case, T_{pro} should be proportional to the size of the data cache. If the data cache keeps n segments, then $T_{pro} = \alpha * n * T_{max}$ ($\alpha \geq 1$). As in discussed previously, α should be set to 1 in low error environments and to a larger value in harsher radio environments. This approach keeps the sequence number gap at any node smaller than n, (i.e., it makes sure that a node will fetch proactively after n successive missing segments). Recall that a node waits at most T_{max} before relaying a message in the pump state so that the probability of finding missing segments in the data cache of upstream nodes is maximized.

3.3 Report Operation

In addition to the pump and fetch operations, described above, PSFQ supports a report operation designed specifically to feedback data delivery status information to users in a simple and scalable manner. In wireless communication, it is well known that the communication cost of sending a long message is less than sending the same amount of data using many shorter messages [14]. Given the potential large number of target nodes in a sensor network in addition to potential long paths (i.e., longer paths through multi-hops greatly increase the delivery cost of data), the network would become overwhelmed if each node sent feedback in the form of report messages. Therefore, there is a need to minimize the number of messages used for feedback purposes. PSFQ's report message and feedback mechanisms are designed to address these issues. The report message is designed to travel from the furthest target node back to the user on a hop-by-hop basis. Each node en route toward the user is capable of piggybacking their report message in an aggregated manner. Nodes can add/append their own feedback information to the original report message sent by the most distant target node as it propagates back toward the user that initially requested the report.

3.3.1 Report Message

The report message has only one field in its header representing the destination node ID of the node that should relay this report. The payload is a chain of node IDs and sequence number pairs that feedback the current status of each node along the path from the last hop toward the source user node.

3.3.2 Report Timers

A node enters the report mode when it receives an inject data message with the "report bit" set in the TTL field. The user node sets the report bit in the inject message whenever it needs to know the latest status of the surrounding nodes. To reduce the number of report messages and to avoid report implosion, only the last hop[6] nodes will respond immediately by initiating a report message sending it to its parent[7] node at a random time between $(0, \Delta)$. Each node along the path toward the source node will piggyback their report message by adding their own node ID and sequence number pair into the report, and then propagate the aggregated report toward the user node. Each node will ignore the report if it found its own ID in the report to avoid looping. Nodes that are not last hop nodes but are in report mode will wait for a period of time ($T_{report} = T_{max} \times TTL + \Delta$) sufficient to receive a report message from a last hop node, enabling it to piggyback its state information. A node that has not received a report message after T_{report} in report mode will initiate its own report message and send it to its parent node. If the network is very large then it is possible for a node to receive a report message that has no space to append more state information. In this case a node will create a new report message and send it prior to relaying the previously received report that had no space remaining to piggybacking data. This ensures that other nodes en route toward the user node will use the newer report message rather than creating new reports because they themselves receive the original report with no space for piggybacking additional status.

---

[6] The last hop can be identified from the "TTL" field of the inject message, (i.e., nodes that receive an inject message with TTL=1).

[7] The node where the previous segment came from.

4. PERFORMANCE EVALUATION

We use packet-level simulation to study the performance of PSFQ in relation to several evaluation metrics and discuss the benefits of some of our design choices. Simulation results indicate that PSFQ is capable of delivering reliable data in wireless sensor networks even under highly error prone conditions, whereas, in contrast, other relevant approaches "retooled" to operate under such conditions cannot.

4.1 Simulation Approach

We implemented PSFQ as part of our reference re-tasking application using the ns-2 network simulator [13]. In order to highlight the different design choices made we compare the performance of PSFQ to an idealized version of Scalable Reliable Multicast (SRM) [5], which has some similar properties to PSFQ, but is designed to support reliable multicast services in IP networks. While there is growing body of work in multicast [8] [9] in mobile ad hoc networks and some initial work on reliable multicast support [11][12], we have chosen SRM as the best possible candidate that is well understood in the literature. SRM supports reliable multicast on top of IP and uses three control messages for reliable delivery, including session, request and repair messaging. Briefly, session messages are sent by each node in a multicast group to inform members of the last data message received by a node. Session messages are time-stamped and their exchange is also used to calculate the delay between pairs of nodes. Request messages are multicast by a node when it discovers that a data message is missing. Its complement is the repair message, which responds with the missing data requested in the request message. Missing data messages may be sent not only by the original source but also by any other node able to respond to the request. SRM represents a scheme that use explicit signaling for reliable data delivery while PSFQ is a more minimalist transport that can be unicast (on top of routing) or broadcast and does not require periodic signaling.

We compare PSFQ with the loss detection/recovery approach of SRM but extract out the IP multicast substrate and replace it with an idealized omniscient multicast routing scheme. In this sense, we present SRM in the best possible light. There are several considerations for doing this. First, SRM is based heavily on the group delivery model and Application Level Framing [10], which make it a good match for cluster-based communications and the application-specific nature of sensor networks. On the other hand, SRM relies heavily on an IP multicasting mechanism for data routing. It is, however, unrealistic to assume an IP substrate in the context of sensor networks, as discussed previously. In addition, PSFQ is solely a reliable data transport scheme in our reference application, it does not provide a general routing solution as in the case of SRM, and therefore it is only fair to isolate SRM from the routing cost incurred by an IP multicast substrate for the purpose of our evaluation. We therefore only compare the reliable delivery portions of the SRM and PSFQ protocols. Since PSFQ uses a simple broadcast mechanism as a mean for routing in our reference application, it makes sense to layer SRM over an ideal omniscient multicast routing layer for simulation purposes. Using omniscient multicast, the source transmits its data along the shortest-path multicast tree to all intended receivers in which the shortest path computation and the tree construction to every destination is free in term of communication cost.

The major purpose of our comparison is to highlight the impact of different design choices made. SRM represents a traditional receiver-based reliable transport solution and is designed to be highly scalable for Internet applications. SRM's service model has the closest resemblance to our reference application in sensor networks. However, SRM is designed to operate in the wired Internet in which the transport medium is highly reliable and does not suffer from the unique problems found in wireless sensor networks, such as, hidden terminal and interference. To make a fair comparison, we try to idealize the lower layer to minimize the differences of the transport medium (which SRM is designed for) for simulation purposes, and, solely focus on the reliable data delivery mechanism – we term this idealized SRM scheme as *SRM-I*.

The goal of our evaluation is also to justify the design choices of PSFQ. We choose three metrics, which underpin the major motivations behind PSFQ:

- *average delivery ratio,* which measures the ratio of number of message a target node received, to the number of message a user node injects into the network. This metric indicates the error tolerance of the scheme at the point where a scheme fails to deliver 100% of the messages injected by a user node within certain time limits.
- *average latency,* which measures the average time elapsed from the transmission of the first data packet from the user node until the reception of the last packet by the last target node in the sensor network. This metric examines the delay bound performance of a scheme.
- *average delivery overhead*, which measures the total number of messages sent per data message received by a target node. This metric examines the communication cost to achieve reliable delivery over the network.

We study these metrics as a function of channel error rate as well as the network size.

To evaluate PSFQ in a realistic scenario, we simulate the re-tasking of a simple sensor network in a disaster recovery scenario within a building. Typically, sensor nodes in a building are deployed along the hallway on each floor. Figure 4 depicts such a simple sensor network in a space of dimensions 100m x 100m. Each sensor node is located 20 meters from each other. Nodes use radios with 2 Mbps bandwidth and 25 meters nominal radio range. The channel access is the simple CSMA/CA and we used a uniformly distributed channel error model. A user node at location 0 attempts to inject a program image file with size equal to 2.5KB into every node on the floor for the purposes of re-tasking. The typical packet size used by the sensors used in this evaluation, and for which the radio is designed, is 50 bytes. This is equivalent to injecting 50 packets into the sensor network from the user node. Packets are generated from the user node and transmitted at a rate of one packet per 10ms. For PSFQ, the timer parameters were set conservatively to follow PSFQ paradigm: T_{max} is 100ms, T_{min} is 50ms and T_r is 20ms. Therefore, the fetch operation can be 5 times faster than pump operation. Each experiment is run 10 times and the results shown are an average of these runs.

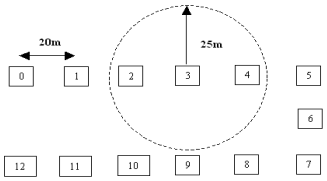

Figure 4. Sensor network in a building. A user node at location 0 injects 50 packets into the network within 0.5 seconds.

4.2 Simulation Results

One of the major goals of PSFQ is to be able to work correctly under a wide variety of wireless channel conditions. The first experiment examines the "error tolerance" of PSFQ and SRM-I, and compares their results. Following the consideration of optimizing lower layer support for SRM, SRM-I is given extra benefit in channel access by using CSMA/CA with RTS/CTS and ACK support while PSFQ only uses CSMA broadcasting. The use of omniscient multicast along with the RTS/CTS channel access greatly contributes to the error tolerance of SRM-I in two respects. First, RTS/CTS eliminates the hidden terminal problems and reduces possible interference between nodes. Second, ACK support in our simulation allows up to 4 link-layer retransmissions (note that from the simple analysis shown in Figure 2, this could provide an improvement up to 20%) after an RTS-CTS exchange, this essentially incorporates the loss recovery mechanism into lower layer (i.e., MAC-level ARQ) in addition to that offered by SRM.

In Figure 5, we present the results for PSFQ and SRM-I under various channel error conditions as we increase the number of hops in the network. As one might expect, the average delivery ratio of both schemes decrease as channel error rate increases. In addition, for larger error rates, the delivery ratio decreases rapidly when the number of hops increases. Notice that the user node starts sending data packets into the network at a constant rate of one packet per 10ms at 2 seconds into the simulation trace and finishes sending all 50 packets within 0.5 seconds. The simulation ran for 100 seconds after the user node stopped sending data packets. This arbitrary cutoff point was chosen as the time after which the delivery of data would be meaningless for a time critical re-tasking operation. Of course, this time limit is very much application-specific; in this case, consider that 100 seconds is 200 times the amount of time required by the user node to inject the entire program image file into the network. Observe from Figure 5, SRM-I (dotted line) can achieve 100% delivery at up to 13 hops away from the source node only when the channel error rate is smaller than 30%. For 50% error rate, the 100% delivery point decreases to within 5 hops; and for larger error rates, SRM-I is only able to deliver a portion of the file two hops away from the user node. In contrast, PSFQ (solid line) can achieve a much higher delivery ratio for all cases under consideration for a wide range of channel error conditions. PSFQ achieves 100% delivery up to 10 hops away from the user node even at 50% error rate and delivers more than 90% of the packet up to 13 hops away. Even under extremely error-prone channel error rates of 70%, PSFQ is still able to deliver 100% data up to 4 hops away and 70% of the packets up to 13 hops, while SRM-I can only deliver less than 30% of data even within 2 hops.

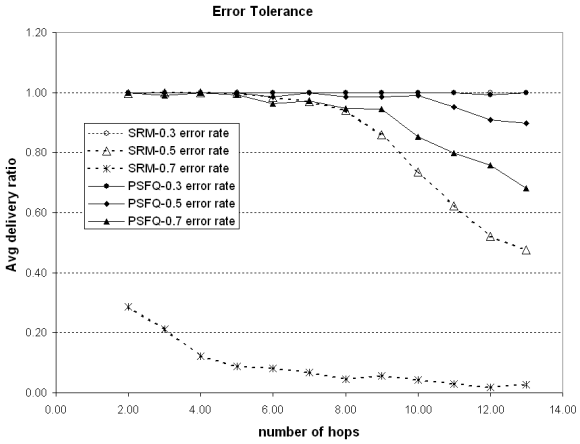

Figure 5. Error tolerance comparison - average delivery ratio as a function of the number of hops under various channel condition for different packet error rate.

The better error tolerance exhibited by PSFQ in comparison to SRM-I justifies the design paradigm of pump slowly and fetch quickly for wireless sensor networks. The in-sequence data pump operation prevents the propagation of loss events, as discuss in Section 2.3. While SRM-I does not attempt to provide ordered delivery of data and loss events are propagated along the multicast tree. PSFQ's aggressive fetch operation and loss aggregation techniques support multiple loss windows in a single control message. In contrast SRM-I is conservative in loss recovery operations. This is because SRM is intended for applications without fixed deadlines, it also does not support the aggregation of multiple loss windows in a single control message.

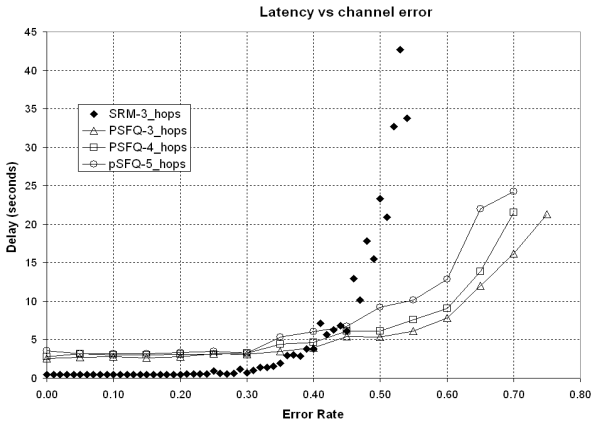

Figure 6. Comparison of average latency as a function of channel error rate.

Our second experiment examines the data delivery latency of both schemes under various channel conditions. The results are shown in Figure 6. Delivery latency is defined only when all intended target nodes receive all data packets before the simulation terminates. For SRM-I, we know that 100% delivery can be achieved only within a limited number of hops when the error rate is high. In this experiment, we compare the two schemes using a 3-hop network and investigate PSFQ's performance with a larger number of hops since PSFQ has better error tolerance. Figure 6 shows that SRM-I has a smaller delay than PSFQ when the error rate is smaller than 40%, but its delay grows exponentially as the error rate increases,

while PSFQ grows more slowly until it hits its error tolerance barrier at 70% error rate. The reason that SRM-I performs better than PSFQ in terms of delay in the small error region is due to the "pump slowly" mechanism, in which each node delays a random period of time between T_{min} and T_{max} before forwarding packets. Despite this small penalty in the smaller error region, the coupling of this mechanism with the "fetch quickly" operation proves to be very effective. As shown in Figure 6, PSFQ can provide delay assurances even at very high error rates. Figure 6 also shows that as the number of hops increases, the delay in PSFQ increases rapidly in the higher error rate region, but it is still within the anticipated delay bound.

In the next experiment, we study the communication cost for reliability in both schemes under various channel conditions using a 3-hop network. Communication cost is measured as the average number of transmissions per data packet (i.e., average delivery overhead). For SRM-I, we separate the communication cost of the SRM-specific loss recovery mechanisms from the total communication cost, which includes the cost associated with the link-layer loss recovery mechanisms (RTS/CTS/ACK). Figure 7 shows that the cost for PSFQ is consistently smaller than SRM-I by an order of magnitude even after excluding the link-layer cost of SRM-I. Figure 7 also illustrates the 100% delivery barrier of both schemes (the two vertical lines). The 52% error rate mark shows the limit for SRM-I while the 70% error rate mark shows the operation boundary for PSFQ. The different performance observed under simulation is rooted in the distinct design choices made for each protocol. PSFQ utilize a passive, on-demand loss recovery mechanism, whereas SRM employ periodic exchange of session messages for loss detection/recovery.

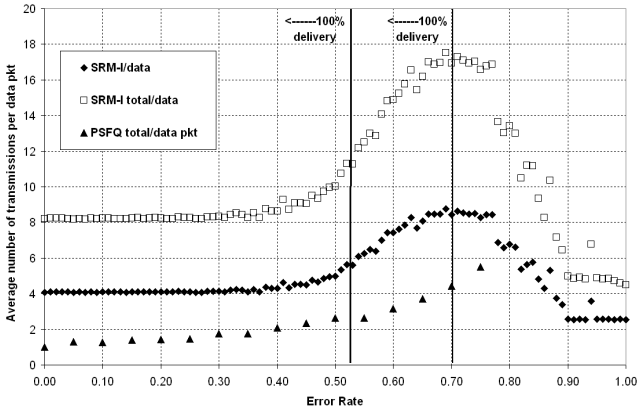

Figure 7. Average delivery overhead as a function of channel error rate

If we consider the results for SRM-I in Figure 7, we can observe four distinct portions of the curve: *i)* from 0 to 30% error rate, the curve is linear where the link-layer loss recovery mechanisms are able to recover all packet losses and there is no need for the SRM (REQ, REP) mechanisms to be used; *ii)* from 30% to 50% there is a rapid increase in messages as the packet loss overwhelms the link-layer mechanisms, and the SRM reliable delivery mechanisms must be used to maintain 100% delivery; *iii)* after that point, message growth continues, but despite SRM, not all packets can be delivered to all nodes; and *iv)* the curve declines rapidly when the channel error rate increases beyond 70%, at which point the error rate is so high that hardly any transmission can take place since the RTS-CTS exchange is rarely completed.

5. WIRELESS SENSOR TESTBED

In what follows, we discuss some early experiences implementing PSFQ in an experimental wireless sensor testbed. We implemented PSFQ using the TinyOS platform [1][6] on RENE motes [1]. The sensor devices have an ATMEL 4MHz, low power, 8-bit micro-controller with 8K bytes of program memory and 512 byte of data memory; 128KB EEPROM serves as secondary storage. The radio is a single channel RF transceiver operating at 916MHz and capable of transmitting at 10kbps using on-off-keying encoding. The radio performs transmission and bit sampling in software (TinyOS). TinyOS [6] is an event-based operating system, employing a CSMA MAC and performs encoding and decoding of the byte stream using Manchester encoding with a 16-bit CRC. The packet size is 30 bytes. With a link speed of 10kbps the channel capacity can delivers at most 20 packets per second. Tuning the transmission power can change the radio transmission range of motes.

We implemented the PSFQ pump, fetch and report operations as a component of TinyOS that interfaces with the lower layer radio components. The component code size for PSFQ is 2KB. In the implementation, every data fragment that is received correctly is stored into the external EEPROM at a predefined location based on its sequence number. The sequence number is used as an index to locate and retrieve data segments when a node receives a NACK from its neighbors.

We conducted several experiments using a simple scenario for preliminary evaluation of PSFQ. In order to emulate a wireless channel with different packet error rates, we manipulate the radio transmission power of the mote and measured the packet loss rate between every pair of motes that were separated by a fix distance before starting the experiments by calculating the number of missing packet for the transmission of 100 packets between the node pairs. Due to the irregular fading conditions in the laboratory, it was difficult to obtain accurate and fine-grained channel error rates. We were only able to obtain four different error rates that were relatively consistent over the period of our experiments. This simple experiment measures the delivery latency of sending 30 packets from a source node to multiple target nodes hops away. In order to monitor the completion time in which a mote received all the packets, additional motes connected to a laptop computer were placed close to the target motes to snoop the reports sent by each node.

Figure 8 shows the result of our experiments. Every data point in the figure is an average of 5 independent experiments. The timer parameters were set as follows: T_{max} = 0.3s and T_r = 0.1s. In the figure, the delay for a 5-hops network increases rapidly for an error rate of 15%. Whereas smaller-hop networks, the delay increases more slowly even at high error rates. The results observed from the testbed show poorer performance in comparison to the simulation results where a rapid increase occurred at much higher error rate. This discrepancy is expected because of several factors. First, channel conditions of the wireless link in the real world are highly irregular especially for motes, which uses a very simple radio. The error rates presented in Figure 8 are not accurate enough. Second, the computational overhead cannot be captured in simulation. Since the processor has to process every bit received off the link, intensive computation could overwhelm the processor forcing packets to be missed. Therefore, the actual loss rate during the experiment could be higher. Finally, the mote implementation uses a very simple

random number generator (16-bit LFSR) therefore the likelihood of collision in the testbed is higher than under simulation conditions.

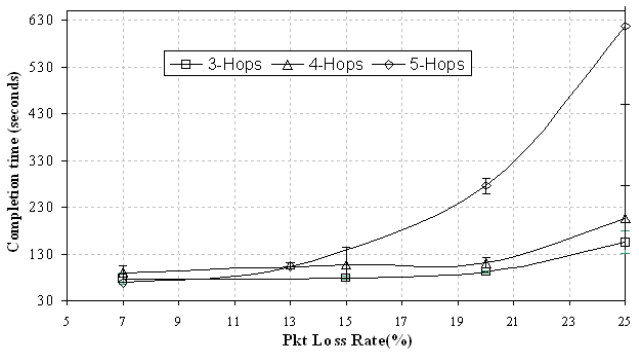

Figure 8: Delay experiments for mote testbed

6. CONCLUSION

In this paper, we have presented PSFQ, a reliable transport protocol specifically designed for wireless sensor networks. PSFQ is a lightweight, simple, mechanism that is scalable and robust making minimum assumptions about the underlying transport infrastructure. We have discussed the need for reliable data delivery in sensor network, especially the application where a user node needs to re-task a group of sensor nodes in its vicinity by injecting program images into target nodes. Base on this reference application, we have described the design of PSFQ to achieve several goals, including operation under high error rate conditions and support for loose delay bounds for data delivery. We evaluated PSFQ and compared its performance to an idealized SRM implementation under simulation. We found that PSFQ outperforms SRM-I in terms of error tolerance, communication overhead, and delivery latency. We also presented some initial results from an experimental wireless sensor network that supports PSFQ using the TinyOS platform on top of the RENE motes. Results show a basic proof-of-concept indicating that the approach looks very promising in an actual wireless testbed. Future work includes more experimentation with larger numbers of sensors. The driving force behind our work remains transport and system support for programming wireless sensor networks. Along those lines we intend to study the impact of re-tasking motes on-the-fly using PSFQ. Results from this phase of our work will be the subject of a future publication.

7. ACKNOWLEDGEMENTS

Many thanks to Shane Eisenman for helping in data measurement, and we would also like to thank the anonymous reviewers for their insightful comments.

8. REFERENCES

[1] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister. "System architecture directions for network sensors", *Proc. of the 9th International Conf. on Architectural Support for Programming Languages and Operating Systems*, pg 93-104, Nov. 2000.

[2] Cots Dust, Large Scale Models for Smart Dust. http://www-bsac.eecs.berkeley.edu/~shollar/macro_motes/macromotes.html.

[3] C. Intanagonwiwat, RC. Govindan, D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks", *Proc. of the Sixth Annual ACM International Conference on Mobile Computing and Networking*, pp. 56-67, Aug. 2000.

[4] J. Atwood, O. Catrina, J. Fenton and W. Strayer, "Reliable Multicasting in the Xpress Transport Protocol", *Proc. of 21st Conference on Local Computer Networks*, pg 202-211,Oct 1996.

[5] S. Floyd, V. Jacobson, C. Liu, S. Macanne and L. Zhang. "A Reliable Multicast Framework for Lightweight Session and Application Layer Framing". *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 784-803, Dec. 1997.

[6] TinyOS Homepage. http://webs.cs.berkeley.edu/tos/

[7] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, J.-P. Sheu, "The broadcast storm problem in a mobile adhoc network", *Proc. of the Fifth Annual ACM International Conference on Mobile Computing and Networking*, pp. 151-162, Aug. 1999.

[8] J.J. Garcia-Luna-Aceves, E. L. Madruga, "The core assisted mesh protocol", *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1380-94, Aug. 1999

[9] S.-J. Lee, M. Gerla, C.-C. Chiang, "On-demand multicast routing protocol", *Proc. IEEE Wireless Communications and Networking Conf.*, pp. 1298-1304, Sept. 21-25, 1999.

[10] D. Clark and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proceedings of ACM SIGCOMM '90*, pp. 201-208, Sept. 1990.

[11] C. Ho, K. Obraczka, G. Tsudik, K. Viswanath, "Flooding for Reliable Multicast in Multi-Hop Ad Hoc Networks", ACM *Mobicom Workshop on Discrete Algorithms & Methods for Mobility* (DialM'99), Aug. 1999.

[12] E. Pagani, G. Rossi, "Reliable Broadcast in Mobile Multihop Packet Networks", *Proc. of the Third Annual ACM International Conference on Mobile Computing and Networking*, pp. 34-42, Sept. 1997.

[13] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/

[14] D. A. Maltz, "On-Demand Routing in Multi-hop Wireless Mobile Ad Hoc Networks", *PhD Thesis*, Carnegie Mellon University, 2001.