Pump-Slowly, Fetch-Quickly (PSFQ): A Reliable Transport Protocol for Sensor Networks

Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy

***Abstract*— There is a growing need to support reliable data communications in sensor networks that are capable of supporting new applications, such as, assured delivery of high priority events to sinks, reliable control and management of sensor networks, and remotely programming/re-tasking sensor nodes over-the-air. We present the design, implementation, and evaluation of *PSFQ (Pump Slowly, Fetch Quickly)*, a simple, scalable, and robust transport protocol that is customizable to meet the needs of emerging reliable data applications in sensor networks. PSFQ represents a simple approach because it makes minimum assumptions about the underlying routing infrastructure, it is scalable and energy-efficient because it supports minimum signaling thereby reducing the communication cost for data reliability, and importantly, it is robust because it is responsive to a wide range of operational error conditions found in sensor network, allowing for the successful operation of the protocol even under highly error-prone conditions. 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 their local immediate neighbors aggressively ("fetch quickly"). 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 and the TinyOS operating system. We show that PSFQ can out perform existing related techniques and is highly responsive to the various error conditions experienced in sensor networks. The source code [20] for PSFQ is freely available for experimentation.**

***Index Terms*— Energy-efficient reliable transport protocols, error control, rate control, sensor networks**

I. 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 over-the-air. This is one new application in sensor networks that requires the 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 work on the design of reliable transport protocols for sensor networks. This is expected because 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, data that flows from sinks to sources for the purpose of control or management (e.g., re-tasking sensors, actuation) 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

Manuscript received December 15, 2003. This work is supported in part by the NSF WIRELESS TECHNOLOGY Award ANI-9979439, ARO Award DAAD19-99-1-0287, and with support from Intel Corporation.

C-Y. Wan, A. T. Campbell are with the Electrical Engineering Department, Columbia University, New York, NY 10027 USA (e-mail: [wan, campbell]@ comet.columbia.edu).

L. Krishnamurthy is with the Intel Laboratory, Intel Corporation, Hillsboro, OR USA, (e-mail: lakshman.krishnamurthy@intel.com).

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 increasingly 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 or thousands of nodes need to be reprogrammed in a controlled, reliable, robust and scalable manner? Such a reliable transport protocol must be lightweight and energy-efficient to be realized on low-end sensor nodes, such as, the Berkeley mote series of sensors, and 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.

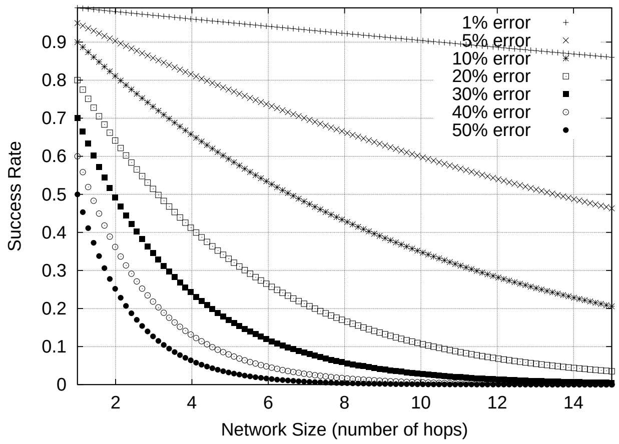

Fig. 1. Probability of successful delivery of a message using an end-to-end model across a multi-hop network.

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 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.

This paper represents an extended version of the work that first appeared in [19] and is organized as follows. Section II presents the PSFQ model and discusses important design choices. Section III details the design of the PSFQ pump, fetch and report mechanisms. Section IV presents an evaluation of the protocol and comparison to existing related techniques such as Scalable Reliable Multicast (SRM) [4] using the ns-2 simulator. We show that PSFQ can out perform an idealized SRM scheme and is highly responsive to the various error conditions experienced in sensor networks. Section V discussed experimental results from the implementation of PSFQ in a wireless sensor testbed based on Berkeley motes. Section VI discusses related work, and finally, we present some concluding remarks in Section VII.

II. 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"). Messages that are lost are detected when a higher sequence number than expected is received at a node triggering the fetch operation, (i.e., an energy-efficient negative acknowledgement system that PSFQ is based on). The motivation behind our simple model is to achieve loose delay bounds while minimizing the lost recovery cost by using localized recovery of data among immediate neighbors.

A. *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. 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 multihop forwarding techniques to exchange messages. Error accumulates exponentially over multi-hops, therefore, packet loss and reordering is more likely. 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 n hops decreases quickly to $(1-p)^n$. Fig. 1 illustrates this problem numerically. Fig. 1 plots the success rate as a function of the network size in number of hops, and shows that for larger networks 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 10%. In [21]-[22] the authors show that it is not unusual to experience error rates of 10% or above in dense wireless sensor networks. We believe that the error rate could be even higher in many cases, such as, military applications, industrial process monitoring, and disaster recovery activities. This observation suggests that end-to-end error recovery is not a good candidate for reliable transport in wireless sensor networks.

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 basis rather than end-to-end. This approach essentially segments multihop forwarding operations into a series of single hop transmission processes that eliminate error accumulation. The hop-by-hop approach thus scales better and is more tolerable to errors while reducing the likelihood of packet reordering in comparison to end-to-end approaches.

B. Fetch/Pump Relationship

For a negative acknowledgement system, the data delivery latency would be dependent on the expected number of retransmissions for successful delivery. To reduce the latency, 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 *i* (therefore increasing the chances of successful delivery) before the next packet *i*+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, (i.e., 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. Assume that the packet loss rate p stays constant 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 k retransmissions can be expressed recursively as:

$$\begin{aligned}&(1-p)+p\times\Omega(k) \qquad (k\geq 1)\\&\Omega(k)=\Phi(1)+\Phi(2)+\cdots\Phi(k)\\&\Phi(k)=(1-p)^2\times[1-p-\Phi(1)-\cdots-\Phi(k-1)](\Phi(0)=0)\end{aligned} \qquad (1)$$

Where $\Omega(k)$ is the probability of a successful recovery of a missing segment within k retransmission, $\Phi(k)$ is the probability of a successful recovery of the missing segment at kth retransmission.

The above expressions are evaluated numerically against the packet loss rate p, as shown in Fig. 2, demonstrating the impact of increasing the number of retransmissions up to k 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 retransmissions diminishes quickly and becomes negligible when k 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.

C. Multi-modal Operations

In a negative acknowledgement system, a local loss event could propagate to downstream nodes if higher sequence number packets are continuously forwarded. The propagation of a loss event could cause a serious waste of energy because a loss event will trigger error recovery operations that attempt to fetch the missing packet quickly from immediate neighbors, whereas none of their (downstream nodes) neighbors would 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.

The use of a data cache is required to buffer messages to ensure in-sequence data forwarding and the complete recovery for any fetch operations from downstream nodes. Note that the cache size effect is not investigated here but for our reference application (i.e., re-tasking) 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 conditions. 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.

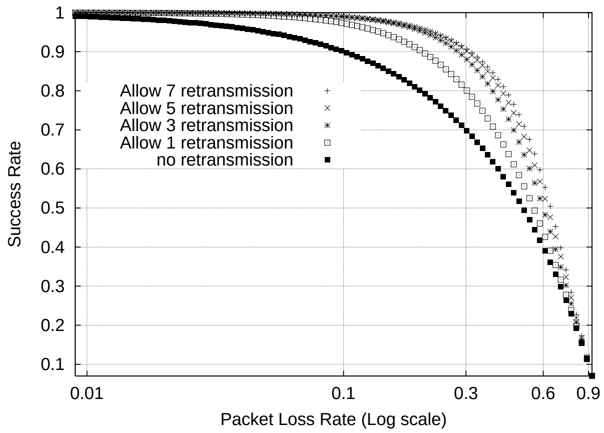

Fig. 2. Probability of successful delivery of a message over one hop when the mechanism allows multiple retransmissions before the next packet arrival.

PSFQ benefits from the following tradeoff between store-and-forward and packet switching. The pump operation operates in a multihop packet-switching 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, PSFQ exhibits a novel multi-modal communications property that provides a graceful tradeoff between the packet switching and store-and-forward paradigms, depending on the channel conditions encountered.

III. Protocol Description

PSFQ comprises three protocol functions: message relaying (pump operation), relay-initiated error recovery (fetch operation), and selective status reporting (report operation). A user 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. It is important for the user to obtain statistics about the dissemination status in the network as a basis for subsequent decision-making, such as the correct time to switch over to the new task in the case of re-tasking/ programming sensors over-the-air. 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., minimizes the overhead).

In what follows, we describe the main PSFQ operations (viz. pump, fetch and report) with specific reference to a re-tasking application -- one in which a user needs to re-task a set of sensor nodes by distributing control scripts or binary code segments into the targeted sensor nodes.

A. Pump Operation

Recall that PSFQ is not a routing solution but a transport scheme. In a case where a specific node needs to be addressed directly, instead of a whole group of sensors, which is the norm, then PSFQ can operate on top of existing routing schemes (e.g., DSDV [17]) to support reliable data transport[1]. A user node can use TTL-based as well as group address filtering [5] methods to control the scope of its re-tasking operation. Note however, that this method does not provide accurate scope control because in many cases the intended receivers cannot be neatly defined by a limit of TTL. To enable local loss recovery and in-sequence data delivery, a data cache is created and maintained at intermediate nodes.

The pump operation is important in controlling the timely dissemination of code segments to all target nodes, and providing basic flow control so that the re-tasking operation does not overwhelm the regular operations of the sensor network. This requires proper scheduling for data forwarding. We adopt a simple scheduling scheme, which use two timers T_{min} and T_{max} for scheduling purposes.

1) Pump Timers

A user node broadcasts a packet to its neighbors every T_{min} until all the data fragments have been sent out. 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 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 rebroadcasts the packet. 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.

[1] To support reliable transport for any-to-any communication scenarios, PSFQ is layered upon a routing scheme and uses the unicast address of the destination node instead of a broadcast address in data packets. In order to support hop-by-hop error recovery, a "snoop" component is needed to copy packets from the routing agent to the PSFQ agent. In this case, only nodes en-route to the destination node, as determined by the routing algorithm, participate in the PSFQ operations.

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 comes from its upstream neighbors, since a node must wait at least T_{min} before forwarding a packet as part of the pump operation. 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. In [6], the authors show that a rebroadcast system can provide only 0-61% additional coverage over what 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.

B. Fetch Operation

A node goes into the PSFQ fetch mode once a sequence number gap in a file's 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.

1) Loss Aggregation

Data loss is often correlated in time because of fading conditions and other channel impairments. As a result loss usually occurs in batches (bursty loss). PSFQ aggregates loss such that the fetch operation deals with a “window” of lost packets instead of a single packet loss. 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. Other nodes that have the data and scheduled retransmissions will cancel their timers if they hear the same "repair" from a neighboring node. 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 the 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 then all the missing segments could be resent by this neighbor.

2) *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, as discussed earlier) 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 ($\ll T_r$). The first NACK is cancelled (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 [6] 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 cancels 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 tradeoffs.

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.

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 ($\frac{1}{4}T_r$, $\frac{1}{2}T_r$). Neighbors will cancel this event whenever a reply to the same NACK for the same segment 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 $\frac{1}{4}$ T_r.

3) *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 a 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 since no packet with a 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 timer-based "proactive fetch" operation such that a node can also enter the fetch mode proactively and send a NACK message for the next segment or the remaining segments if the last segment has not been received and no new packet is delivered after a period of time T_{pro}.

The proactive fetch mechanism is designed to automatically trigger the fetch mode at the proper time. If the 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 the 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 these two cases. In our reference application, where each segment of a file needs to be kept in a 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., $T_{pro} = \alpha * (S_{max} - S_{last}) * T_{max}$ ($\alpha \geq 1$). α is a scaling factor to adjust the delay in triggering the 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. This enables a node to start the 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.

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 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 operation so that the probability of finding missing segments in the data cache of upstream nodes is maximized.

The proactive fetch operation would ensure all intended receivers eventually receive all of the data. But like any protocols that try to a maximum number before giving up, PSFQ proactive fetch could stop after reaching a threshold, which is an application-specific choice.

4) *Signal Strength based Fetch*

Recent studies [21]-[23] show that in sensor networks that use low-power radios without frequency diversity, there exists very high variability in the packet delivery performance that is both spatial and temporal dependent. Because of intermittent packet reception from nodes that are more than a single hop away (however weak the signal is) can cause nodes to send unnecessary NACK messages and retransmissions, PSFQ also takes into consideration the received signal strength of a packet during the fetch and repair operations. A node maintains a table of parent nodes (i.e., nodes from which it receives messages) with their associated average signal quality measurements. When a node detects a gap in the sequence number upon receiving a packet it only respond and send out a NACK if this packet comes from a parent with the strongest average signal quality measurement. This effectively suppresses unnecessary NACK messages triggered by the reception of packets that come from nodes that are multiple hops away.

Similarly, when a node transmits a NACK message it includes the preferred parent with the strongest average signal in the message. Nodes that receive this NACK will determine if they are the preferred parent/neighbor. All non-preferred neighbors double their response time delay in sending repair packets so that they have a greater chance of hearing the repair packet from a better candidate node (i.e., preferred parent/neighbor), allowing the node to cancel a repair whenever a response is heard before sending. This approach prevents nodes sending redundant retransmissions when they do not have a good chance of delivering theses message to a fetching node.

C. *Report Operation*

PSFQ supports an optional report operation designed specifically to feedback data delivery status 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 [12]. 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 increases 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.

A node enters the report mode when it receives a data message with the "report bit" set in the message header. The user node sets the report bit in its injected 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 nodes, (i.e., TTL=1) will respond immediately by initiating a report message by sending it to its parent node, where the previous segment came from, at a random time between (0, Δ). Each node along the path toward the source node will piggyback their report message by adding their own status information 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 the 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 piggyback its state information. 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 received the original report with no space for piggybacking additional status.

D. *Single-Packet Message Delivery*

There is need to support the reliably deliver of single-shot atomic messages in sensor networks, for example, in support of reliable control and management of sensors. For messages that fit into a single packet (e.g., smaller than the network MTU) delivery failure is undetectable using PSFQ's NACK-based protocol without the addition of explicit signaling. This is because PSFQ detects loss by observing sequence number gaps or timeouts. To address this service need PSFQ makes use of its reporting primitive to acquire application-specific feedback at the sink. PSFQ sets the report bit at the sink in every single-packet message that requires reliable delivery. Based on the feedback status the sink resends the packet until all receivers confirm reception. This essentially turns PSFQ into a positive aggregated-ACK protocol used in an on-demand manner by the sink for these special case messages. The use of the report mechanism to support reliable data delivery of single-shot atomic messages highlights the flexible use of PSFQ mechanisms to meet application specific needs.

IV. 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.

A. Simulation Approach

We implemented PSFQ as part of our reference re-tasking application using the ns-2 network simulator [11]. In order to highlight the different design choices made we compare the performance of PSFQ to an idealized version of Scalable Reliable Multicast (SRM) [4], 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 [7]-[8] in mobile ad hoc networks and some initial work on reliable multicast support [9]-[10], 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. Because SRM is designed to operate on top of an IP multicasting substrate, it assumes an environment where there is a single path from a source to an individual receiver, and each node receives each multicast packet at most once. SRM is also intended for a topology where routers are not active members of the group and do not maintain state, except for that needed for multicast routing. SRM represents a scheme that uses 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. We therefore only compare the reliable delivery portions of the SRM and PSFQ protocols. Since PSFQ uses a simple broadcast mechanism as a means 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. The SRM 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 that underpin the major motivation behind the design of PSFQ:

- *Average Delivery Ratio*, which measures the ratio of the number of messages a target node received to the number of messages a user node injects into the network. This metric indicates the error tolerance of a 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 between 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 the 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 where the sensor nodes are deployed along the hallway on each floor of a building. Fig. 3 shows such a simple sensor network in a space of dimensions 100m x 100m. Each sensor node is located 20m from each other. Nodes use radios with 2 Mbps bandwidth with nominal radio range of 25m. The channel access is the simple CSMA/CA and we use a uniformly distributed channel error model. A user node at location 0 attempts to inject a program image file of size equal to 2.5KB into every node on the floor for the purposes of re-tasking. The packet size is 50 bytes. Packets are generated from the user node and transmitted at a rate of one packet every 10ms. For PSFQ, the timer parameters are set conservatively to follow the PSFQ paradigm: T_{max} is 100ms, T_{min} is 50ms, and T_r is 20ms. Therefore, the fetch operation is 5 times faster than pump operation. Each experiment is run 10 times and the results shown are an average of these runs.

B. 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.

In Fig. 4, we present the results for PSFQ and SRM-I under various channel error conditions as we increase the network size in terms of the number of hops. As one might expect, the average delivery ratio of both schemes decreases as the channel error rate increases. For larger error rates, the delivery ratio decreases rapidly when the network size increases. Notice that the user node starts sending data packets into the network at a constant rate of one packet every 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. Observe from Fig. 4, SRM-I (dotted line) can achieve 100% delivery 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 up to two hops away from the user node. In contrast, PSFQ (solid line) achieves 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 conditions of 70% channel error rate, 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 up to 2 hops.

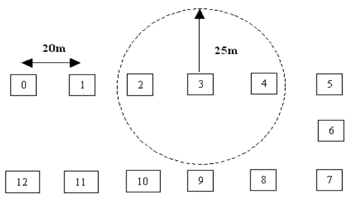

Fig. 3. Sensor network in a building. A user node at location 0 injects 50 packets into the network within 0.5 seconds.

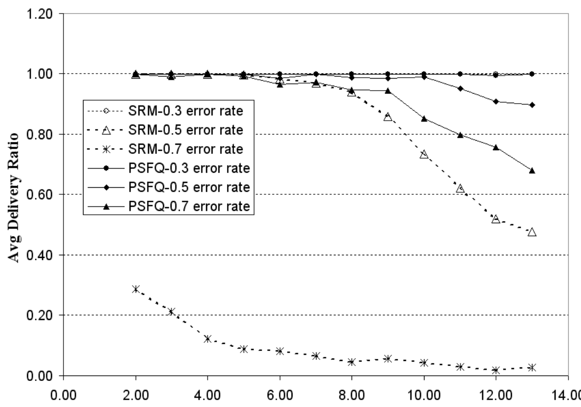

Fig. 4. Error tolerance comparison - average delivery ratio as a function of the number of hops under various channel condition for different packet error rates.

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 II.C. SRM-I does not attempt to provide ordered delivery of data and loss events are propagated along the multicast tree. In contrast, PSFQ's aggressive fetch operation and loss aggregation techniques support multiple loss windows in a single control message. One high-level design lesson here is the ineffectiveness of control messages under high-loss rate scenarios. SRM relies on the underlying MAC layer to reliably deliver explicit and periodic control messages between members of a multicast group. The failure of the virtual carrier sense in IEEE 802.11 MAC under high-loss rate environments cause SRM-I to fail, whereas PSFQ's minimalist approach enables it to do efficient control broadcasting, even under high-loss conditions.

Our second experiment examines the data delivery latency of both schemes under various channel conditions. The results are shown in Fig. 5. The delivery latency is determined only when all the intended target nodes have received all of the 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 properties. Fig. 5 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 lower error rate region is due to the "pump slowly" mechanism, where each node delays a random period of time between T_{min} and T_{max} before forwarding packets. Despite this small penalty in the lower error rate region the coupling of this mechanism with the "fetch quickly" operation proves to be very effective. As shown in Fig. 5, PSFQ can provide delay assurances even at very high error rates.

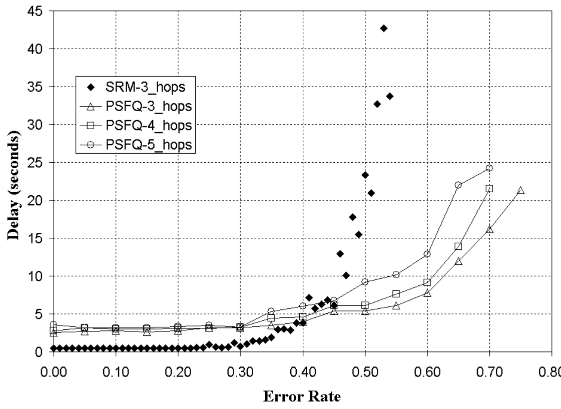

Fig. 5. Comparison of average latency as a function of channel error rate.

In the next experiment, we study the communication cost for reliability in both schemes under various channel conditions using a 3-hop network, including a 16-node (4x4)

3-hop grid network to explore PSFQ performance in a dense network where nodes can have up to four neighbors. 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). Fig. 6 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. We can observe from Fig. 6 that the communication cost in a denser grid network closely matches but is lower than its chain-network counterpart, indicating that PSFQ can exploit neighbor redundancy while suppressing unnecessary redundant transmissions. Fig. 6 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 operational 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.

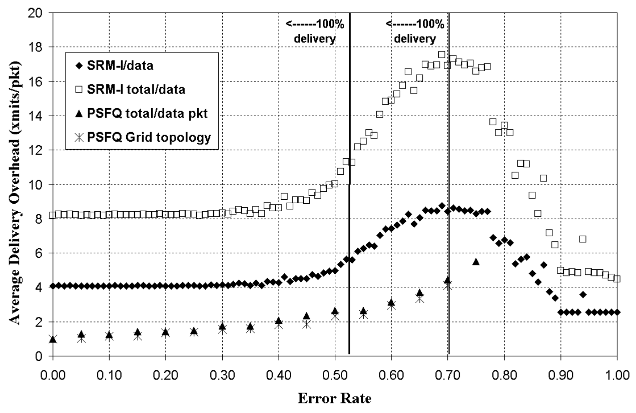

Fig. 6. Average delivery overhead as a function of channel error rate.

V. Experimental Testbed Results

In what follows, we discuss experiences implementing PSFQ in an experimental wireless sensor testbed using the TinyOS platform [1] [5] and Rene2 motes [1]. The Rene2 sensor device has an ATMEL 4MHz, low power, 8-bit micro-controller with 16K bytes of program memory and 1K bytes of data memory, 32KB EEPROM serves as secondary storage. The radio is a single channel RF transceiver operating at 916MHz and capable of transmitting at 10 kbps using on-off-keying encoding. TinyOS [5] is an event-based operating system employing a CSMA MAC. The packet size is 36 bytes. With a link speed of 10 kbps the channel can delivers at most 16 packets per second. Tuning the transmission power can change the radio transmission range of the motes.

We implemented the PSFQ pump, fetch, and report operations as a component of TinyOS that interfaces with the lower layer radio components. In the implementation, every data fragment that is received correctly is stored in 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.

A. PSFQ Parameter Space and Timer Bounds

Among the various PSFQ operations, the most aggressive timer is the fetch timer, as defined in Section III.B.2. A successful recovery after a sequence number gap has been detected relies on two successful packet receptions being accomplished within T_r, (i.e., one for receiving the NACK at the neighbors and another for receiving the repair packet at the fetching node). Since the transmission time of a single packet is non-negligible in low bandwidth environments (i.e., approximately 67ms for Rene2 mote), T_r should be long enough to accommodate at least two packet transmissions. There exists a lower bound for T_r that is defined at the granularity of the transmission time of a single packet; assume this is T_{pkt}. Recall that upon receiving a NACK, a node schedules a repair to be sent at a random time $\in [\frac{1}{4}T_r, \frac{1}{2} T_r]$. Therefore, T_r must be long enough to wait for the largest delayed repair from the neighborhood to avoid unnecessary retransmission of NACK messages, (i.e., $T_r \geq \frac{1}{2} T_r + 2T_{pkt}$). Therefore, $T_r \geq 4T_{pkt}$. In reality, to avoid using up all the available channel bandwidth during fetch operations, we increase the lower bound by one or two T_{pkt} times to allow at least one packet transmission for other operations or applications, and to accommodate other possible processing delays. For example, a reasonable bound is: $T_r \geq 6T_{pkt}$ and $T_{max} \geq 5T_r \geq 30T_{pkt}$. These values are used in all of our testbed experiments discussed in the remainder of this section.

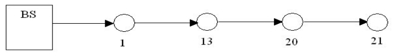

Fig. 7. A 4-hop network physically arranged in a string/chain topology.

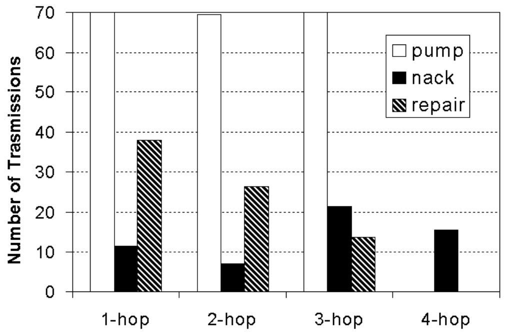

Fig. 8. Breakdown of PSFQ messages. Average delivery overhead is 1.2±0.13.

B. Messaging Overhead

In our experiments, we manipulate the radio transmission power of the motes to create multi-hop networks such that motes that are separated by 5 inches can maintain 90%~100% reception rate, while motes that are separated by 10 inches can hardly hear each other, (i.e., the reception rate is between 0%~15%). Fig. 7 shows a 4-hop network in a string/chain topology in which each node is separated by 5 inches. Here we refer to a "hop distance" as the distance between nodes that can maintain excellent communication, (i.e., more than a 90% packet reception rate). Our test scenario sends a new execution image (i.e., image file of the TinyOS BLINK [5] application segmented into 70 over-the-air packets) from the base station (BS) connected to a PC to all the sensor nodes using PSFQ. When the base station confirms the 100% reception of the image by all sensors (using the PSFQ report operation) then it sends a single control message that propagates to all the sensor nodes to initiate the process of transferring the new image from external EEPROM to the internal Flash to complete the reprogramming of the application. Note that we use PSFQ's single packet reliable service to do this controlled application switchover at sensors, as discussed in Section III.D.

Fig. 8 shows the result of our experiments in terms of communication overhead with the breakdown of the PSFQ messages. Each data point in the figure is an average of 10 independent experiments and the 95% confidence intervals are all within 10% of the average value. The overall average delivery overhead is 1.2 transmissions per received packet.

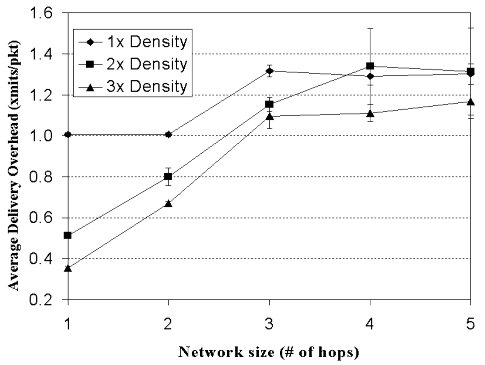

Fig. 9. Average delivery overhead as a function of network size and density.

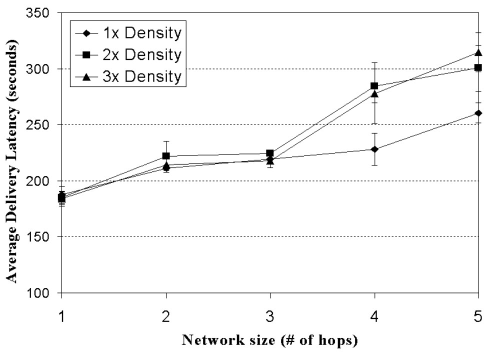

Fig. 10. Average delivery latency as a function of network size and density.

C. Network Size versus Network Density

In what follows, we examine the impact of the network density and the network size on the performance of PSFQ in terms of delivery latency and average delivery overhead.

Using the same test scenario described in Section V.B, we measure both the communication cost and delivery latency of PSFQ with various network sizes as well as various node densities in our Rene2 testbed, in which motes are arranged in string/chain topology. Fig. 9 and 10 present the results of these experiments. Each data point is an average of 10 independent experiments and the corresponding 95% confidence intervals are plotted as y-axis error bars in the figures, respectively.

Fig. 9 shows that the communication cost for reliable delivery increases rapidly when the network size increases from 1 hop to multiple hops, but it also levels off and stabilizes quickly for a network size of 4 to 5 hops. The reason for the rapid rise of the communication cost is due to the well-known hidden terminals problem in CSMA networks, which becomes evident only in multi-hop environments and creates collisions that force packet drops. Nevertheless, PSFQ's pump/fetch operations can effectively prevent loss propagation along the distribution chain, and therefore is able to maintain relatively low overhead (~ 1.3) as the network size increases. Interestingly, we can observe from Fig. 9 that as we increase the network density the communication cost actually decreases. This indicates that PSFQ effectively suppress redundant transmissions and takes advantage of overhearing transmissions from a dense neighborhood to reduce a node's transmissions, and hence, reduces the overall delivery overhead. Fig. 10 shows that the delivery latencies of PSFQ increase almost linearly with the network size but they are rather independent of the network density, which indicates that PSFQ can adapt well in a high-density environment.

VI. Related Work

To our best knowledge PSFQ represents the first reliable transport for sensor networks. In what follows, we contrast more recent contributions [13] [15] [16] for reliable data delivery in sensor networks that followed the initial publication of PSFQ [19].

RMST (Reliable Multi-Segment Transport) [13] is a transport layer paradigm for sensor networks that is closest to our work. RMST is designed to complement Directed Diffusion [3] by adding a reliable data transport service on top of it. RMST is a NACK-based protocol like PSFQ, which has primarily timer driven loss detection and repair mechanisms. The authors analyze the tradeoff between hop-by-hop vs. end-to-end repair and conclude the importance of hop-by-hop recovery, which is consistent with our analysis and simulation results. In contrast to PSFQ, which provides reliability purely at the transport layer, RMST involves both the transport and MAC layers [14] to provide reliable delivery. In ESRT [15] the authors propose using an event-to-sink reliability model in providing reliable event detection that embeds a congestion control component. In contrast to PSFQ, ESRT does not deal with data flows that require strict delivery guarantees; rather, the authors define the "desired event reliability" as the number of data packets required for reliable event detection that is determined by the application. A sink-to-sensor reliability solution is presented in [16] that focus on communication reliability from the sink to the sensors in a static network. The authors propose using a two-radio approach where each node is equipped with a low frequency "busy-tone" radio in addition to the default radio that is used for data transmission and reception. The "busy-tone" radio is used to ensure delivery of single-packet messages or the first packet of a longer message. A NACK-based recovery core is constructed from the minimum dominating set of the underlying graph.

VII. Conclusion

We have presented PSFQ, a reliable transport protocol for wireless sensor networks. Based on our reference application for remotely programming sensors over-the-air, we have discussed a number of important design goals that underpin the protocol's development, including, the correct and efficient operation under high packet error rate conditions and support for loose delay bounds for reliable data delivery. We evaluated PSFQ using simulation and through implementation in an experimental motes testbed. We found that PSFQ outperforms SRM-I in terms of error tolerance, communication overhead, and delivery latency. Nevertheless, there remain a number of open questions concerning the cost (e.g., data cache) and limits (e.g., mobility) of PSFQ. A more comprehensive discussion of PSFQ and set of results can be found in [24]. Our on-going work in the Armstrong Project [20] at Columbia University, which is more broadly investigating the development of energy-efficient control and transport mechanisms for resilient sensor networks, is focused toward the design of suitable energy-efficient congestion control and load balancing mechanisms. Our initial results in this area include CODA [18] and support for dual radio "virtual sinks" [20]. Finally, the PSFQ source code for TinyOS implementations is freely available [20] for experimentation.

Acknowledgment

We would like to thank Shane Eisenman for helping with the data measurement presented in this paper.

References

[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister, "System architecture directions for network sensors", *Proc. of the 9th International Conf. on Architectural Support for Programming Languages and Operating Systems*, Nov 2000, pp. 93-104.

[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 and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks", *Proc. of the sixth annual international conf. on Mobile computing and networking*, Aug. 2000, pp. 56-67.

[4] 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.

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

[6] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen and J.-P. Sheu, "The broadcast storm problem in a mobile adhoc network", *Proc. of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, Aug. 1999, pp. 151-162.

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

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

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

[10] E. Pagani and G. Rossi, "Reliable Broadcast in Mobile Multihop Packet Networks", *Proc. of the third annual ACM/IEEE international conference on Mobile computing and networking*, Sept. 1997, pp. 34-42.

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

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

[13] R. Stann and J. Heidemann, "RMST: Reliable Data Transport in Sensor Networks", Appearing in *1st IEEE International Workshop on Sensor Net Protocols and Applications (SNPA)*, Anchorage, Alaska, USA, May 2003.

[14] W. Ye, J. Heidemann and D. Estrin. "An Energy Efficient MAC Protocol for Wireless Sensor Networks", *In Proc. of the 21st Intl. Annual Joint Conf. of the IEEE Comp. & Comm. Soc.* (INFOCOM 2002), New York, NY, USA, June 2002.

[15] Y. Sankarasubramaniam, O.B. Akan, and I.F. Akyildiz, "ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks", To appear in *Proceedings of ACM MobiHoc*, Annapolis, MD, USA, June 2003.

[16] S-J. Park and R. Sivakumar, "Sink-to-Sensors Reliability in Sensor Networks", Extended Abstract to appear in *Proceedings of ACM MobiHoc*, Annapolis, MD, June 2003.

[17] C.E. Perkins and P. Bhagwat. "Highly dynamic Destination- Sequenced Distance-Vector routing (DSDV) for Mobile Computers", in *SIGCOMM Symposium on Communications Architectures and Protocols*, pp. 212-225, September 1994.

[18] C-Y. Wan, S. B. Eisenman and A. T. Campbell, "CODA: Congestion Detection and Avoidance in Sensor Networks", *Proc. of First ACM Conference on Embedded Networked Sensor Systems* (SenSys 2003), pp. 266-279, Los Angeles, November 5-7, 2003.

[19] C-Y. Wan, A. T. Campbell and L. Krishnamurthy, "PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks", in *Proc. Of the First ACM International Workshop on Wireless Sensor Networks and Applications* (WSNA 2002), pp. 1-11, Atlanta, September 28, 2002.

[20] Armstrong Project homepage, http://comet.columbia.edu/armstrong/.

[21] J. Zhao, R. Govindan, and D. Estrin. "Computing Aggregates for Monitoring Wireless Sensor Networks", In *Proc. Of the IEEE ICC Workshop on Sensor Network Protocols and Applications*, Anchorage, AK, May 2003.

[22] J. Zhao and R. Govindan. "Understanding Packet Delivery Performance In Dense Wireless Sensor Networks", *First ACM Conference on Embedded Networked Sensor Systems* (SenSys 2003), November 2003.

[23] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. "Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks", *Technical Report UCLA/CSD-TR02-0013*, Computer Science Department, UCLA, July 2002.

[24] Chieh-Yih Wan, "Resilient Control and Transport Mechanisms for Sensor Networks", PhD. Thesis, Department of Electrical Engineering, September 2004.

Chieh-Yih Wan is a Ph.D. candidate of Electrical Engineering at Columbia University and a member of the COMET Group. He is working on resilient control and transports for sensor networks. Chieh-Yih received his M.S. in Electrical Engineering from Columbia University in 1999, his M.S. in Applied Physics from National TsingHua University, Taiwan in 1996, and the bachelor degree in Physics from National TsingHua University, Taiwan in 1994.

Andrew T. Campbell (M'93) is an Associate Professor of Electrical Engineering at Columbia University and a member of the COMET Group. He is working on emerging architectures and programmability for wireless networks. Andrew received his Ph.D. in Computer Science in 1996 and the NSF CAREER Award for his research in programmable mobile networking in 1999. During 2003-2004 is was an EPSRC Visiting Fellow at the Computer Laboratory, Cambridge University.

Lakshman Krishnamurthy is a Principal Engineer at Intel Corporation in the communications technology lab, where he manages Intel's mesh networking program. He leads research efforts into new wireless mesh protocols, techniques to provide ease of use and improve performance of wireless networks. He is also the principal investigator of EcoSense wireless sensor network project at Intel Research. Lakshman received a Ph.D in computer science from the University of Kentucky and a BE in instrumentation technology from the University of Mysore, India.