Reliability and Storage in Sensor Networks

ABSTRACT

A large class of delay tolerant sensor-net applications require reliable deliver of *every* data point. The nature of sensor network deployments makes providing reliability a challenge. Harsh environments and unreliable wireless communication can cause long periods of poor to no connectivity. Meanwhile, energy and resource constraints on sensor platforms limit retransmissions and buffer sizes. This paper presents an architecture designed for these challenged networks. The architecture provides packet-level, hop-by-hop reliability for delay-tolerant data using sequential storage for buffering during long queue delays. The concepts discussed in this paper are implemented as services in the Extensible Sensing System, a deployment at the James Reserve as part of the Cold Air Flow Project.

General Terms

Sensor Networks, Reliability, Storage, Deployment

Keywords

Wireless, sensor networks, reliability, storage, Deployment

1. Introduction

Wireless sensor networks (WSN) provide a distributed, sensing and computing platform for monitoring environments in which conventional networks are impractical. The benefit of a WSN comes from embedding resource-constrained, wireless sensor nodes in the environment at a scale which provides fine grained readings over a large area. There exists a class of sensor-net applications which require reliable delivery of *every* data point collected across the network.

The nature of sensor networks makes reliability a challenging problem. Traditional network protocols such as TCP [1] provide reliability using unrestricted end-to-end retransmissions and acknowledgements while assuming large data buffers to store undelivered packets. Current research [2,3,4] has shown that end-to-end reliability in sensor networks is impractical given energy constraints and unreliable wireless links. At the same time, resource constrained sensor nodes only provide limited storage space to buffer data while waiting for end-to-end acknowledgements.

Existing solutions employ a hop-by-hop reliability mechanism at either the link or transport layer to increase the probability of packet delivery to the destination. These solutions assume high node connectivity and unvarying, low-error wireless links. Recent deployments such as the Cold Air Flow Project at James Reserve have shown that real environments rarely exhibit such well defined characteristics. Instead time varying link quality and sparse node density cause long periods of poor to no connectivity. In light of this, a new architecture is needed to provide reliable data delivery for these challenged networks.

The main contribution of this paper is the design, implementation and measurement of a sensor network system which provides reliability in the face of poor link quality and frequent disconnects. Our deployment measurements show wireless networks exhibit time varying link quality which can cause periodic deviations from average path quality. Our design utilizes the delay tolerant attribute of sensor-net applications to store data locally during these periods of deviation. The result is increased energy efficiency due to the elimination of wasted retransmissions. We provide this service by implementing a packet-level, hop-by-hop routing protocol which buffers data using sequential storage over flash memory. We show that this protocol utilizes sensor resources more efficiently by increasing storage capacity and reducing ineffective retransmissions.

2. Cold Air Flow Project

Our motivation for implementing an improved reliability architecture for sensor networks stems from the deployment of the Cold Air Flow Project at the James Reserve [9]. In this study, researchers are attempting to determine cold air flow and humidity patterns along mountainous ravines. The information is essential in modeling vegetation and wildlife patterns in these regions. Previous attempts to model cold air flow used sparsely placed weather stations and extrapolated data points over large areas. This method of data collection was ineffective for explaining ecological patterns in ravines and valleys.

The aim of the Cold Air Flow Project is to develop a more accurate model based on fine grained temperature and humidity readings along the James Reserve Valley in the San Jacinto Mountains. The project has deployed 40 nodes in 2 transects perpendicular to the valley to measure both cold air gradients across a single transect and cold air flow between multiple transects.

To build a correct cold air model, researchers require a valley wide collection of fine grained readings as a function of time. Lost data points from any part of network will require extrapolation between points and will revert the analysis back to previous inaccurate results. In this deployment reliable delivery of each data point is essential for correct analysis.

This application is an example of a larger class of sensor-net applications. In this class of application all data is essential but delivery is tolerant to relatively long delays.

3. Related Work

Previous research is partitioned into two different areas: persistent storage and reliable network protocols.

3.1 Persistent Storage

Most sensor network platforms use flash memory as a means of persistent storage. Flash memory is the appropriate choice for resource constrained sensor nodes due to its high reliability, high density, and relative low cost. Though advantageous, the

characteristics of flash memory are very different from conventional hard disks.

Flash memory is divided into sectors or blocks which are then subdivided into pages. Page sizes are device specific and can vary from hundreds of bytes up to a few kilobytes. All read/write operations are executed on a per-page basis with writes consuming more energy than reads. Flash memory writes also exhibit another unique property in that each page can endure a maximum number of writes. Repeated writes to the same page will exhaust the lifetime of the flash page. To ensure that no single page reaches its lifetime limit before the rest of the flash memory pages it is important to ensure writes are evenly distributed. This process is called ware-leveling [5].

The benefits of using flash memory for compact devices have lead to an abundance of literature on flash based file system design. The design presented in [5] as well as other current approaches emulate block devices over flash pages. Traditional files systems are then ported over the emulated block device providing compatibility with traditional implementations. This approach of emulating a block device does not utilize the unique properties of flash memory and causes fragmented data and inefficient ware-leveling.

Current sensor network research in flash based file systems has lead to the implementation of full fledged journaling file systems [6,7]. As with traditional journaling file systems these implementations create a new log entry for each write operation that occurs. To optimize this technique for flash memory each entry is placed on a separate page. Though this produces natural wear-leveling, it can cause fragmentation if the sensor data is not a multiple of the page size. Given the limited storage available on sensor nodes this fragmentation can lead to large portions of unusable space.

The nature of sensor readings requires sequentially written records without the need to modify previous data. Current implementations such as Matchbox and ELF provide a rich set of file system operations, many of which are unnecessary for applications wishing to only store sensor data. The resource overhead of a full fledged file system may not be appropriate for these applications.

Previous work on flash storage provides generalized solutions which may not fit the needs of sensor-net applications. For this reason we have created a new sequential access file system optimized for sensor data. The primary objectives of this storage layout is to maximize the write lifetime of the device using ware-leveling and overcome limited local storage capacity by distributing storage over the network.

3.2 Reliable Network Protocols

Previous work on reliability in sensor networks has focused on the network stack. The result has been a set of transport layer protocols which attempt to provide message level reliability on a hop-by-hop basis.

In [3], the authors determined the biggest problem with end-to-end recovery was the unreliable characteristics of the wireless communication link. The error accumulates exponentially over multiple hops, causing a high probability of packet loss. This work produced Pump Slowly, Fetch Quickly (PFSQ), a hop-by-hop NACK transport protocol. Message fragments are sequentially sent one hop, where the receiver will selectively request missing fragments based on sequence number. Single packet messages default to an end-to-end recovery mechanism.

In [2], the authors present RMST. RMST, much like PSFQ, is a hop by hop selective NACK transport protocol. RMST improves on PSFQ by including link layer recovery via Automatic Repeat Requests (ARQ). The link layer implements a stop-and-wait protocol using explicit acknowledgements for each sent packet. The number of ARQ retransmissions attempted before giving up is configurable but statically set. The advantage delivered by RMST is in providing added reliability at the link level instead of just relying on explicit NACKs from the transport layer to recover lost packets.

The authors in [4] present a sensor network implementation of Delay Tolerant Networking (DTN) [8]. The work in this paper is similar to RMST and PSFQ by providing a hop-by-hop transport layer protocol. In addition the implementation presents a persistent storage mechanism to handle situations in which the message is not delivered to the next hop.

Both RMST and PSFQ provide reliability by attempting to *increase the probability* of packet delivery using intelligent retransmissions on a hop-by-hop basis. This mechanism works well in systems with high connectivity and low error link quality, but in real systems where disconnects are possible both RMST and PSFQ will begin to lose packets.

DTN provides added reliability over RMST and PSFQ by storing messages locally during disconnects. In storing at the message level, DTN introduces fragmentation in the storage device since variable length messages are not multiples of the storage capacity. Fragment sizes are proportional to message sizes thus larger messages lead to larger areas of unusable storage. In addition DTN does not distinguish between full message losses due to disconnect and single packet losses due to link error. In either case the message is stored for future retransmission. In the latter case successful packet transmissions are wasted since the full message was not transferred to the next hop.

These limitations in currently available solutions require a new network protocol for reliability. In our new protocol we provide reliability at the packet level, using hop-by-hop acknowledgements. Reliability at the packet level offers minimal fragmentation and allows distributed storage of messages over multiple nodes.

4. Reliable Staged Transport

The main components of any reliable architecture are retransmissions and storage (both volatile and non-volatile). The goal of our architecture is to use both components intelligently to provide an energy and resource efficient solution.

4.1 Design Space

When designing a reliability solution, the design space can be divided into 3 dimensions:

- End-to-end vs hop-by-hop acknowledgments
- Forwarding vs store-and-forward routing
- Real-time vs delay tolerant data

4.1.1 End-to-end vs Hop-by-Hop Acknowledgments

Established network protocols, like TCP, provide reliability through end-to-end acknowledgements and retransmissions. In

these networks, the end hosts are responsible for handling error recovery with no responsibility delegated to intermediary nodes. The implicit assumption in these protocols is that packet loss is the result of a single congested queue along the route. Since queue overflow is directly correlated with transmission rate, congestion can be alleviated by regulating packet flows. In other words packet loss is a quantifiable event, which can be controlled by direct action from the end host. In sensor networks this assumption is not true. The physical characteristics of the wireless medium provide probabilistically unreliable links. Packet loss is not the result of a single quantifiable failure point but instead is a probabilistic accumulation of error over multiple hops.

To demonstrate this, assume a packet loss rate of p over a wireless channel. Over n hops the chances of a successful transmission decreases to (1-p)^n. Figure 1 shows the probability of successful packet deliver over a given number of hops for different average link error rates. This shows that for large networks with multiple hops it is almost impossible to deliver a packet using end-to-end delivery with no intermediary error recovery.

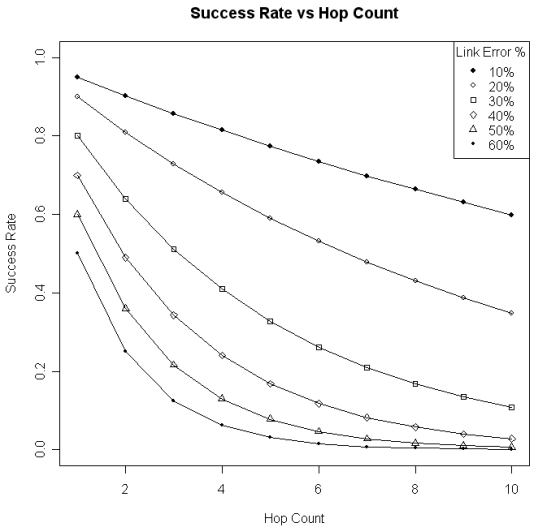

Figure 1 Probability of successful packet delivery over multiple hops given different average link error rates.

Unlike wired networks, packet loss in wireless networks is not correlated to transmission rates but instead is related to individual link qualities. Actions by the end host are ineffective at correcting packet loss multiple hops away. Instead intermediary nodes, which are directly connected to the error producing link, can immediately detect a loss and retransmit. Moving the responsibility for error recovery to each individual hop along the path reduces the total number of transmissions required to successful deliver a packet. A retransmission due to failure at node k in an n hop path has a (1-p)^n-k probability of success vs. an end-to-end recovery which has a (1-p)^n probability of success.

4.1.2 Forwarding vs. Store-and-forward Routing

The difference between forwarding and store-and-forward routing is the granularity at which the network stack views data.

In store-and-forward networks, routing is performed on complete messages. Messages are broken into a series of fragments, each tagged with a sequence number to be used in reassembly. The network stack attempts to transmit an entire message one hop by sending each individual fragment as a packet. The receiver reassembles the packets into a message, using sequence number gaps to selectively request missing packets. In this manner messages are sent and reassembled hop by hop until they reach their destination.

Forwarding networks view data on a packet level. The network stack is given a packet (messages are fragmented at a higher level), and is responsible for attempting to send the packet one hop. In these networks packet delivery is explicitly acknowledged by the one hop receiver. The relationship between packets and messages is ignored and it is the end hosts responsibility to reassembly the fragments.

Current implementations for reliability in sensor networks [2,3,4] perform store-and-forward routing using transport layer NACKs for missing packets. The largest cost for using this type of routing is storage space. At each hop the packet must be reassembled before it can be forwarded again toward the destination. On sensor nodes storage (both volatile and non-volatile) is a constrained resource. Mica 2 motes [10], a sensor node platform, provide only 4KB of RAM and 512KB of flash storage. When programmed using the TinyOS [11] operating system, the 4KB of RAM is statically allocated at compile time. This means buffer space must be pre-allocated in RAM. Unused buffer space during run time is not reallocated and is essentially wasted.

In a store-and-forward network buffer space is allocated as a multiple of message size, where in the forwarding network buffer space is allocated as a multiple of packet size. If message sizes are much larger than packet sizes then the magnitude of unused buffer space is large. In sensor networks, where resources are limited this inefficient use of storage can limit the number and complexity of services provided by the sensor net application.

4.1.3 Real-time vs Delay Tolerant Data

The time sensitive nature of the data is application specific, but is an important dimension in designing a reliable architecture.

Real time data has a hard deadline on the order of seconds or minutes and must arrive at the sink before that period. Data that is still present in the network beyond its deadline is useless and can be safely discarded. Most real time applications fall into the "sense and react" class of sensor-net applications. In these applications the network must react to events detected, within a specified time. The timely arrival of data is crucial.

Delay Tolerant networks have either a loose deadline on the order of hours or days or have no deadline at all. Delay tolerant applications usually fall into the "sense and analyze" class of sensor-net application. In these applications data is collected at a centralized point for detailed analysis of time varying phenomenon. In most delay tolerant applications each data point is essential for analysis, but no guarantees need to be met as to the time of arrival.

Reliability is defined differently in real time applications versus delay tolerant applications. In real time applications reliability is associated with arrival times and meeting the specified deadline. On the other hand, in delay tolerant applications, reliability is related to data loss and the guaranteed delivery of all data points.

When designing reliability for sensor networks the time sensitive characteristics of the data is important to consider. In real time applications the hard deadline limits the architectural choices available. In most cases the brute force design using continuous hop-by-hop retransmissions give the best results. In these networks each packets is pushed toward the sink hop by hop until the packet arrives at the sink or the packets deadline has past. Retransmissions are used to push data past unreliable links with throughput taking priority over energy efficiency. Buffer space can be relatively small since data does not have a long lifetime and buffers are not used for long term storage.

Delay tolerant applications have much more flexibility in design decisions. Since data has no hard deadline for arrival, the network stack can prioritize energy efficiency over throughput. Also storage allocation becomes a main concern since each data point must be stored in the network until its arrival at the sink. This paper focuses on the appropriate design decisions for reliability in delay tolerant sensor net applications.

4.2 Challenged Networks

Conventional network protocols make implicit assumptions about high quality links, end-to-end connectivity, and low latency responsiveness. Sensor networks do not have these qualities and therefore a new network classification must be introduced to characterize the properties sensor-net protocols must account for. In [12], the author introduces a definition for challenged networks. The following are characteristics of challenged networks:

High Latency, Low Bandwidth Communication: Sensor network platforms provide low cost, low power wireless communication. To achieve this goal, radio quality and bit rate is sacrificed. The Mica2 motes operate at 56 Kbs bit rate under optimal conditions. In addition, the combination of harsh environments and low radio quality make highly error prone links. This poor link quality can cause numerous retransmissions, which can further delay communication. Finally low cost production can yield asymmetric links due to a lower quality of manufacturing. Link asymmetry can skew round trip estimations since each leg of the path has a widely different delay.

Disconnections: In sensor networks, complete end to end path connectivity at any given point in time can be unlikely. Disconnects can arise from two sources: time-varying unreliable links and low-duty cycle node operation. Wireless link connectivity is unreliable and the quality of the link may vary over time. This can cause unpredictable periods of disconnect as links appear and disappear in the network. In addition, even as radio qualities increase, energy conservation techniques such as duty cycling and topology control will cause nodes to periodically leave the network. In these cases nodes which are critical connection points for communication may periodically shutdown to conserve energy. These shutdowns may be scheduled and predictable but will still cause periodic disconnections in the network.

Long Queue Delays: Disconnections and low latency, low bandwidth communication can lead to long queue delays at intermediary nodes. Data traffic is usually dependent on external events and cannot be regulated or throttled during periods of poor connectivity. In these situations nodes must be able to reliable store data locally until connectivity improves.

4.3 Delay Tolerant Design for Challenged Networks

Having outlined the design space and network characteristics we can analyze each design decision given the network properties.

End-to-end vs. hop-by-hop acknowledgements: Any protocol dependent on end-to-end communication requires predictable roundtrip times and stable connectivity between end points. High latency communication and frequent disconnects found in challenged networks can adversely effect end-to-end communication. As shown previously, the additive affects of unreliable link quality make the probability of successful communication low. On the other hand, hop-by-hop communication makes no assumptions about path connectivity and is only concerned about single hop link quality. Work in [2] has shown that link quality can be controlled using link layer ARQs. *Given the characteristics of challenged networks hop-by-hop acknowledgements are the appropriate choice.*

Forwarding vs store-and-forward routing: The choice between forwarding and store-and-forward routing is matter of available storage capacity. Long queue delays mean intermediary nodes must reliable store data locally until packets can be transferred. Message sizes being larger than individual packets means that buffer allocations in RAM must also be larger to cope with multiple messages. Additionally, stored data may not be a multiple of storage capacity leading to fragmentation. Fragment sizes are proportional to the size of the data. Storing data at a packet level leads to smaller fragment sizes. It also has the added benefit of allowing distributing storage of a single message over multiple nodes. *Given the constrained storage capacity on sensor nodes it is better to handle data at a packet level making forwarding the appropriate design choice.*

It is apparent that the correct reliability design for delay tolerant applications in challenged networks is a hop-by-hop, forwarding architecture. Previous work corresponds with our choice to use hop-by-hop acknowledgement, but our choice of forwarding differs from others. Current implementations provide store-and-forward routing placing the appropriate intelligence in the transport layer. Since forwarding deals with individual packets, it is appropriate to push reliability intelligence lower in the network stack to the routing layer. Providing reliability at the routing layer presents further benefits including intelligent forwarding based on path connectivity and instantaneous path quality, as explained in Section 6. Additionally, work in Delay Tolerant Networking has shown that storage plays an important role in providing reliability. Previous work has either used volatile RAM or a bulky journaling file system over external flash memory. In section 5 we present a flash based storage layout which is optimized for the sequential nature of sensor data and can overcome limited storage capacity by distributing storage over local neighborhoods.

5. Sequential Storage

Flash memory provides persistent storage for small devices but at the cost of limited write lifetime. The primary goal is to provide energy-efficient storage which maximizes the lifetime of the device. A secondary consideration is to extend the limited storage capacity available on sensor network platforms by utilizing network wide storage capacity. Section 5.1 defines a layout for flash memory which provides ware-leveling optimized for the sequential nature of sensor data. Section 5.2 presents an

extension to the sequential storage layout which expands storage capacity by distributing storage evenly over a network neighborhood.

The Sequential Storage Layout presented in this section is implemented on the Mica2 mote. The mote provides 512 KB of external flash memory. The flash on the mote is segmented in 264B pages and contains two internal page buffers to act as temporary storage during reads and writes.

5.1 Sequential Storage Layout

Sensor readings are usually variable length and time ordered. Subsequently, storage of sensor readings should sequentially write records providing a First In, First Out paradigm for reads and writes. The most appropriate data structure to handle these requirements is a variable-length queue. Queues provide FIFO reads and writes handling the time ordered aspects of data. Variable-length queues are pointer-based storing a chain of variable length data. Finally queues are naturally optimized for flash memory, since all modifications are done at either the head or the tail. This provides even ware leveling as the head and tail move sequentially through flash, continuously wrapping around the specified storage area.

A queue data structure contains two components. The Queue and a Record Locator Table (RLT). The RLT is a data structure containing a pointer to both the head and tail of the Queue.

The Queue is implemented using a chain of pointers. Each record is written to the tail of Queue. Before each record is written, a 2 byte length field is added to the head of the record detailing the length of the data. This acts as a pointer to the next record location in the Queue. The Queue is implemented to use the pages of flash memory as a circular buffer, wrapping around as it meets the end of memory.

The Queue is naturally optimized to ware-leveling but the RLT is not. The RLT must be stored in flash to maintain a persistent record of the location of the Queue and must be updated for every read and write. If RLT is located on a single page in flash, the write lifetime of the page will quickly become exhausted. This solution is to provide a distributed RLT data structure. Distributing the RLT, distributes the cost of updating over multiple pages. A further optimization is to distribute the RLT in such a way that modifying the Queue and updating the RLT can occur in a single page write instead of two separate page writes.

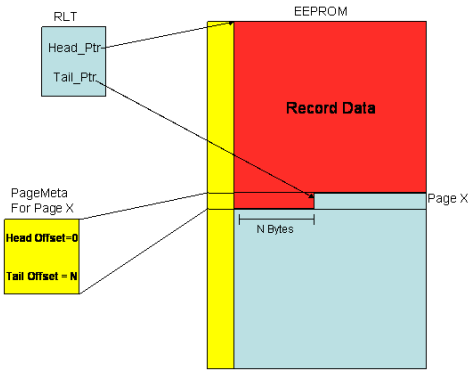

Figure 2 A logical and physical layout of flash memory showing the projection of the queue to individual pages.

As shown in Figure 2, flash memory is split into Queue Space (Red and Blue) and RLT Space (Yellow). The RLT contains pointers to the current head and tail of the Queue. The Queue currently occupies the space colored in Red. At the page level each page is split into Record Space and Page Meta space. Record space is the page level projection of the Queue Space and the Page Meta space is the page level projection of the RLT. If a page contains either the head or tail of the Queue, the Page Meta contains the byte offset of either the head or tail in the given page. If the page does not contain the head or tail the page meta byte offsets are set to a special UNSET value.

As the head or tail of the Queue is read or written, the affected page is loaded into the temporary flash buffer. Once the queue data is read/written the RLT is updated by setting the appropriate byte offset in the page meta to point to either the new head or tail of the Queue. The page is then written back to memory. Both the Queue data and the RLT are modified in a single write.

The exception to this is when the head or tail moves across a page boundary. In this case the current head or tail offset is recorded in the new page and then previous offsets are unset in the old page. Ordering the operations in this manner allows us to reconstruct the Queue even if node fails between the set and unset operation. When the node starts, it scans the page metas looking for the head and tail pointers for the Queue. If multiple head or tail pointers are found (due to previous failure), the first head pointer found is used and/or the last tail pointer found is used. This guarantees that a failure during a read will set the head to reread the data being read during failure. Consequently for the write, this will set the tail to point to the location in memory behind the record

written during the failure.

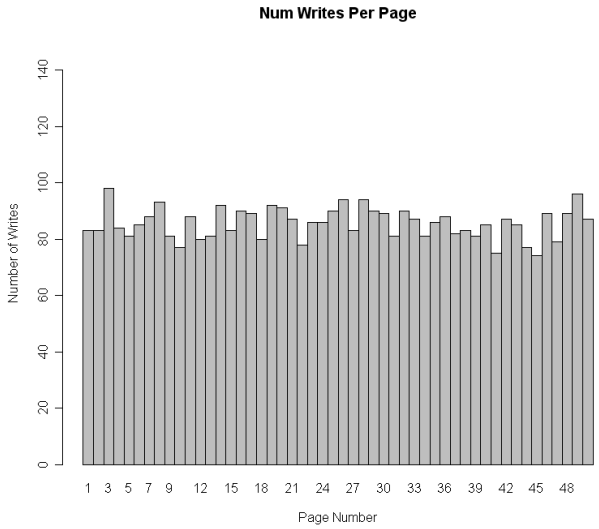

Figure 3 Ware-leveling results using a random read/write sequence with the read to write ration of 1:1

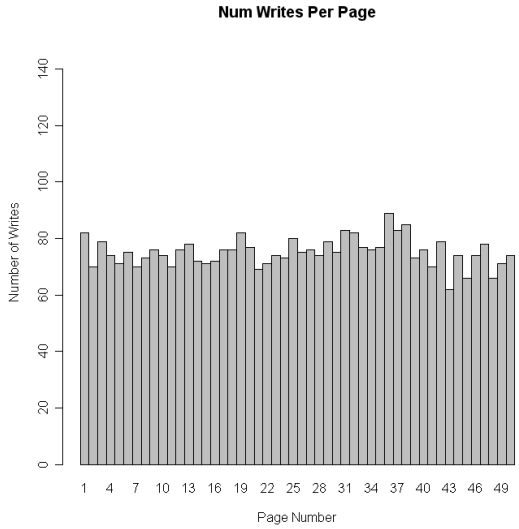

Figure 4 Ware-leveling results using a random read/write sequence with a read to write ratio varying between 3:1 to 1:3

The primary purpose of our Sequential Storage Layout was to extend flash memory lifetime by efficiently ware-leveling writes to storage. Figure 3 shows the result of randomly reading and writing data to storage. The results were produced with a 1:1 read to write ratio. The graph shows the number of flash writes for each page. As you can see the writes are fairly even with a maximum difference between page writes as 19.

In most applications reads and writes usually come in batches, ie due to loss or regaining of connectivity. Figure 4 shows the result of randomly reading and writing data to storage produced with varying 3:1 and 1:3 read to write ratio. The system started with a 1:3 write ratio until the queue was full. At which point it moved to a 3:1 read to write ratio until the queue was empty. It cycled between these two extremes until the end of the simulation. As you can see the writes are again fairly even. The major drop off of writes at page 40 is due to the simulation ending with the queue tail at page 39.

5.2 Distributed Storage

The Mica2 mote only provides 512 KB of persistent memory. Large data sizes, high data rates and long periods of poor or no path connectivity can consume the local storage capacity on a mote. In the situation where a single node or a neighborhood of nodes is overburdened, it is useful to distribute data over neighboring nodes to alleviate storage constraints. This section presents a design to evenly and efficiently distribute data over local neighborhoods.

The distribution of data works on a gradient descent policy where data attempts to find the lowest "cost" location for storage. The storage cost is a function of both available space and the cost of transmission.

The first step is to communicate neighborhood storage capacity. This is accomplished by a beaconing mechanism which periodically broadcasts a nodes available storage space in bytes. The node can also use the beacons to maintain an estimate of link quality between itself and its neighbors. This link quality provides a measure of the transmission cost over that link.

Each node keeps a neighbor list which contains each neighbor's available storage and link quality. When replacing an entry in the neighbor list storage capacity has the highest priority. The replacement policy removes the lowest capacity node from the list if a higher capacity neighbor is discovered. When a neighbor is chosen for distribution, the neighbor with the highest quality link is chosen. In essence, the algorithm chooses the best possible link quality from the list of highest capacity nodes. This mechanism weights capacity as a higher priority than link quality since this will evenly distribute the data and will not over run the capacity of a neighbor who has exception link quality.

The next important design point is to decide when to distribute data and when to store locally. The objective is to have a smooth distribution instead of waiting to exhaust a node's storage capacity before attempting to distribute the data. If the device has a relatively large amount of remaining data capacity (in actual bytes not in percentage) it should decide to store locally all the time. On the other hand if the device has little or no storage it should decide to always distribute the data. Between the two extremes the decision to store locally or distribute should be proportional to the remaining storage on the device relative to its neighbors.

To accomplish this, the distribution decision function is based on picking a random number between 1 and the total size of storage space available. If the number is greater than the storage remaining, then the node should distribute the data, otherwise it should store it locally. When full storage capacity is remaining the node will always store locally. As the remaining bytes available goes to zero, the decision to distribute will become more and more probable. This algorithm guarantees the decision to store is based on the percentage of storage on the device.

The next step is to consider the "relative" storage space available in the local neighborhood as compared to the storage available locally. Given the previous algorithm, a node may decide to distribute due to a low percentage of remaining storage, but its neighbors may have even a lower number of remaining bytes. In other words a large capacity device should not distribute proportionally to its own storage capacity if all its neighbors have relatively small storage capacity. This functionality is provided by preventing the node from distributing data, if its neighbors have less available capacity.

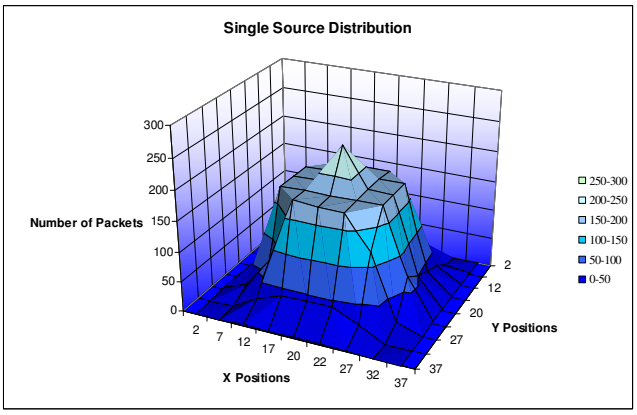

Figure 5 Distribution patter for a single node

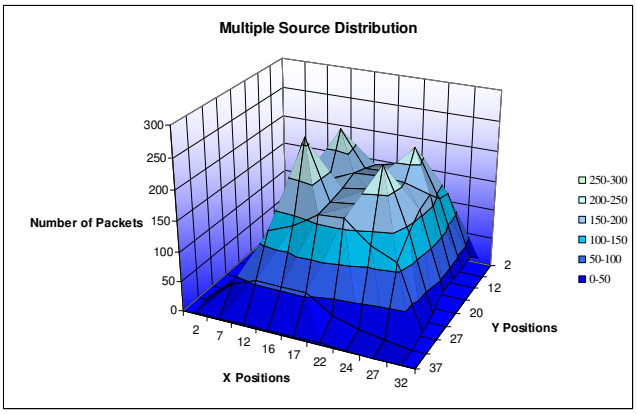

Figure 6 Distribution pattern from a cluster of nodes

The goal of our distribution mechanism is to evenly distribute data over the network by focusing on local neighborhood storage capacity. Figure 5 and 6 show the distribution of data over a network for both single source and multiple source configurations. In both cases the graphs show an even distribution around the sources as the network attempts to alleviate storage constraints at the source nodes. The graphs are a result of a 64 node simulation using the EmStar Simulation tool described in Section 7. The nodes were arranged evenly in a grid. The simulated radio channel is a circular radio model with error rates based on a normal distribution as a function of distance from the source.

In Figure 5 a single source at (20,20) produced 5000 packets of data. Each node has the capacity to store 300 packets. The distribution pattern shows data was first moved to the one hop neighbors. As the one hop neighbors became full, they also began distributing data to their one hop neighbors pushing data away from the source.

Figure 6 shows the same result with a cluster of 4 sources located at (16,16), (16,24), (24,24), and (24,16). Once again the data is sent from each source to their one hop neighbor. The data sent from each source is actually pushed at a gradient away from the other sources. This is visible by the dip in packets stored at the node in between all 4 sources. This shows that even in a clustered source environment the algorithm attempts to alleviate storage constraints by distributing data to a higher capacity location in the network.

6. Reliable Network Protocol

Currently available reliability solutions focus on implementing a network protocol which increases the probability of a successful transmission over a single hop. Without an appropriate storage mechanism, these solutions fail to provide reliability when transmissions are not successful. With the addition of a sequential storage architecture to buffer larger amounts of data over longer periods of time, we can provide a more intelligent network protocol. Our network protocol can exploit the delay tolerant attribute of sensor-net applications to store data during extreme deviations in path quality including possible network disconnection.

The correct reliability design for delay tolerant applications in challenged networks is a hop-by-hop, forwarding architecture. Our network stack provides this service using the link layer and routing layer. The link layer is responsible for improving unreliability over the physical medium, where as the routing layer is accountable for path level connectivity.

6.1 Link Layer

Wireless communication is prone to errors from reflections, interference, and path loss. The authors in [2] attempted to compensate for an unreliable physical medium by adding a stop-and-wait ARQ mechanism to the link layer. The addition of link level retransmissions improved the probability of success for a single hop transmission. Figure 7 shows the probability of success for a single packet over 25 hops given a static number of retries.

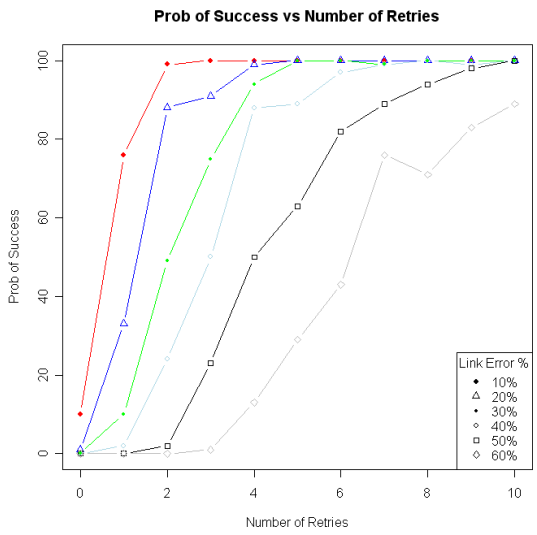

Figure 7 Probability of successful packet delivery over 25 hops given a maximum number of retries per hop.

As the average error rate increases the number of retries required to successful send the packet increases to compensate. The work in [2] statically configured the number of retries, implicitly assuming a uniform and unvarying link quality across the network. Deployments, such as the Cold Air Flow Project, have shown that actual deployments do not follow these characteristics (as seen in Section 8). Setting the retry value too low may artificially create disconnects over links with lower than expected link quality. On the other hand, if the retry value is set too high then energy is wasted transmitting over links which are experiencing an unexpected deviation from their average link quality or have become disconnected for a period. Using both the information in Figure 7 and estimated link quality, it is possible to dynamically set the retry value based on the average link quality of the specified link.

Estimation of link quality is implemented using a beaconing mechanism. Each node periodically broadcasts a beacon containing its node id and an increasing sequence number. Nodes receiving the beacon can record the number of beacons received/missed and estimate an inbound link quality. After a configured number of beacons have been sent, a node will compile a digest of all the inbound values it has recorded and broadcasts the information to its neighborhood. Nodes receiving the digest will search for their own node id in the digest and record the outbound value of their link to that node. In this manner each node has an estimate of both inbound and outbound link quality to its neighbors.

A dynamic ARQ mechanism at the link layer improves the probability of success of transmission but does not guarantee it. In the situation where an acknowledgement is not received after the allotted number of retries, the link layer flags the packet as failed and it is stored in the Sequential Storage device for future retransmission.

6.2 Routing Layer

Information in a sensor network usually flows from a distributed set of sensors to a few selectively placed sinks. Not every node will be in radio distance of a sink, so it is the responsibility of intermediary nodes to forward data multiple hops to a sink.

Our routing implementation is a multi-sink, tree based routing algorithm. Each sink periodically sends a path advertisement declaring itself as the sink. As each node receives the advertisement, they integrate they're link quality to the path quality received in the advertisement and broadcast their new path quality. In this manner a tree is formed where the path quality is based on the link qualities along the path.

Each node in the network is aware of the sink advertisement period and for every missed advertisement the path quality is exponentially decreased until after three missed advertisements the path is considered disconnected.

Varying link quality can effect overall path quality. by causing disconnects or creating periods of unexpected deviation from the average path quality. In both of these cases the routing layer can use path information to provide reliability in an energy efficient manner.

6.2.1 Path Connectivity

Harsh environments and unreliable links can cause networks to become disconnected. As packets are being routed through the network, the routing layer has the opportunity to check network connectivity before attempting to send data.

Packets enter the routing layer from one of two manners. Packets which are local arrive at the routing layer from a higher layer in the network stack. Packets which are external and are being routed along the path arrive from a lower layer in the network stack. As packets are sent into the routing layer to be forwarded, the path connectivity for specified source is first examined. If the routing layer determines the path to the source has become disconnected it will not attempt to send the packet and will instead store the packet in Sequential Storage for future retransmission. Since path connectivity information is not instantaneously propagated it is possible for data start along a path and then be stored at an external node as it enters an area in which connectivity information is fresher.

Additionally queue overflows at the routing layer must be connected to the Sequential Storage. During periods of high traffic or poor link quality it is possible for nodes to experience queue overflow. To provide complete reliability, overflows must be stored locally in Sequential Storage. If the time required to write a packet to flash takes longer than the time required to receive a packet over the radio, packets can still be lost since a queue overflow can happen while the flash is writing a previous packet. On the mica2 motes the flash writes take X cycles while the radio takes X cycles (redo calc). Based on this calculation all queue overflows can be stored locally without fear of overflows occurring faster than storage.

6.2.2 Time Varying Path Quality

Variations in individual link quality can cause large swings in total path quality. When calculating path quality these large fluctuations can cause the routing algorithm to continually choose new paths. This phenomenon is known as route flapping. Route flapping can cause network instability such as routing loops.

Averaging individual link quality estimates stabilizes path quality calculations. Figure 8 shows the path quality cost using both averaged link costs and instantaneous link costs. This was taken from a real deployment discussed in Section 7. As you can see there are periods when the instantaneous path quality greatly exceeds the average quality. In this case the period lasts for 45 minutes (from 5000 seconds to 7000 seconds). During these periods it may be beneficial to store the packet rather than attempting to send. In order to make that decision it is necessary to quantify differences in path cost in terms of energy.

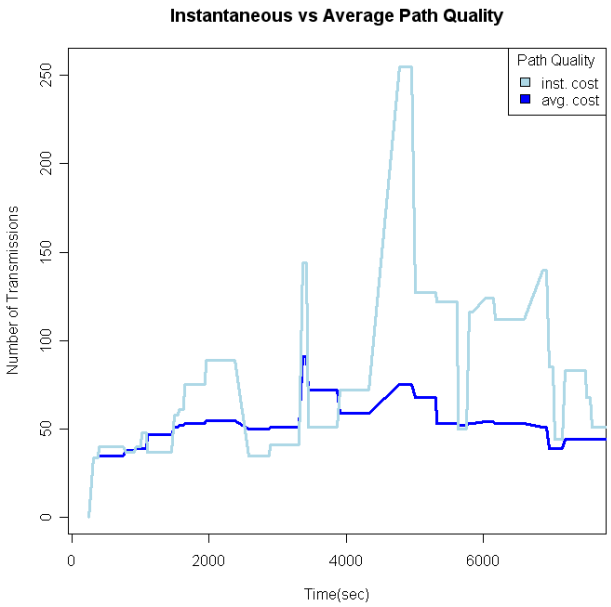

Figure 8 Path quality calculations using the instantaneous link quality metrics versus using averaged link quality metrics.

The Expected Transmission Cost (ETX) [13] estimates path quality such that the calculated value is the expected number of transmissions required to successfully transmit the packet over the path. Using ETX we are able to quantify the difference between the average path cost and the instantaneous path cost in terms of retransmissions, i.e. energy.

Next we must compare this cost with the cost of storing the packet and resending at a later time. On Mica2 motes, writing a byte to flash is considered twice as more expensive than transmitting the byte over the radio. If the difference between the average path cost and the instantaneous path cost is greater than two extra transmissions, then it is more efficient to store the packet and retransmit when the instantaneous cost has decreased.

6.2.3 Packet Re-injection.

Once packets are stored, they must be reinjected back into the network. A disconnection in the network can affect as little as one node or as many as all the nodes and can last for long periods of time relative to the sensing period. When connectivity is regained it is possible for a large number of packets to be injected into the system all at once. A reinjection policy is needed to avoid congesting the network.

Dynamic congestion control is well studied in conventional networks. For the solution to this problem we turn to TCP congestion control. TCP initially attempts to transmit packets slowly, additively increasing the number of packets sent into the network after each successful transmission. If congestion is detected TCP will multiplicatively decrease the number of packets transmitted to alleviate any congestion. The same argument can be made for packet injection rates. Initially a large period between injections is specified. The data rate increases for each successfully sent packet. As the data rate increases the bandwidth along the path decreases. If the path becomes saturated, injected packets will begin being lost. This unsuccessful send will cause a multiplicative back off in data rate. Since not all loss is related to congestion, a back off may occur due to probabilistic errors over the wireless link. In this case we are conservative in our injection policy, choosing reduced throughput over possible congestion. Authors in [14] provide alternate solutions to balance throughput and congestion control in wireless networks.

When reinjecting a packet, packet selection is important. Our Sequential Storage architecture is based on a queue which provides FIFO reads. At any point in time only the head of the queue is accessible. In a single sink environment FIFO works appropriately. Path connectivity information for the head packet applies to all subsequently stored packets since all packets need to be delivered to the same sink. This is not the case in a multiple sink scenario. It is possible for the head packet to be destined for a sink with is currently disconnected; yet, subsequent packets may be destined to a sink which is connected. To handle these situations, a round robin mechanism is implemented when handling packet reinjection. If the head packet fails to send, it is popped off the head of the queue and reinserted to the back of the queue. This functionality comes at a cost. By reinserting the data at the back of the Sequential Storage, it is being rewritten which requires additional energy and hastens the write exhaustion of the flash.

7. Experimental Results

The concepts presented in this paper have been implemented and deployed as part of the Extensible Sensing System (ESS). ESS provides the "sensor network in a box" paradigm. It is appropriate for applications requiring a distributed set of mote class nodes embedded in the environment, sending sensor data to one of many linux class sinks.

On the mote side, the components of ESS can be divided into Multihop Networking and the Data Service Engine. The Data Service Engine provides both querying and dynamic configuration of sensors in the mote field. Multihop Networking forms a multi-sink tree network as described in Section 6. Reliability and Storage are services provided by the Multihop Networking component.

On the sink side, ESS is implemented using EmStar [15]. EmStar is a set of services for building and deploying sensor-net applications. EmStar provides both simulation and emulation tools to easily analyze system performance. The experimental results of our reliability and storage architecture are part of a complete ESS deployment at the James Reserve using EmStar.

7.1 Emulation

A unique feature in EmStar is the ability to run in emulation mode. In emulation mode, a simulation on a centralized server

uses real deployed motes as radios, in place of a simulated radio model. In our emulation, we used the ceiling array at the Center for Embedded Network Sensing (CENS) for our mote deployment. Figure 9 shows the layout of the motes in the ceiling array.

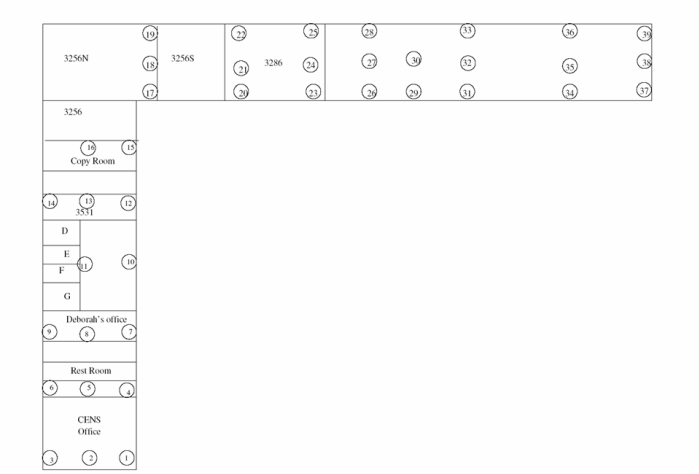

Figure 9 Ceiling Array layout at CENS.

In this experiment we ran a 24 node emulation over the ceiling topology. A Data Service Engine query was sent to the entire network requesting periodic sensor data at a 1 minute internal. In this experiment Node 2 acted as the sink.

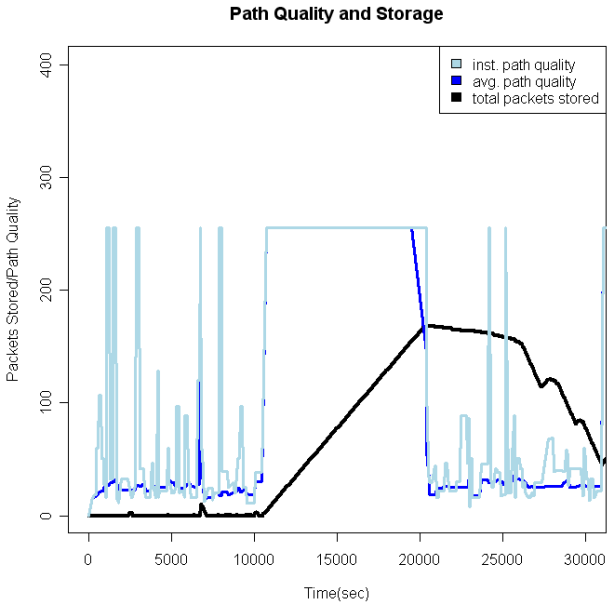

Figure 10 An overlay of both path quality and used storage capacity as a function of time for a one hop neighbor (Node 5).

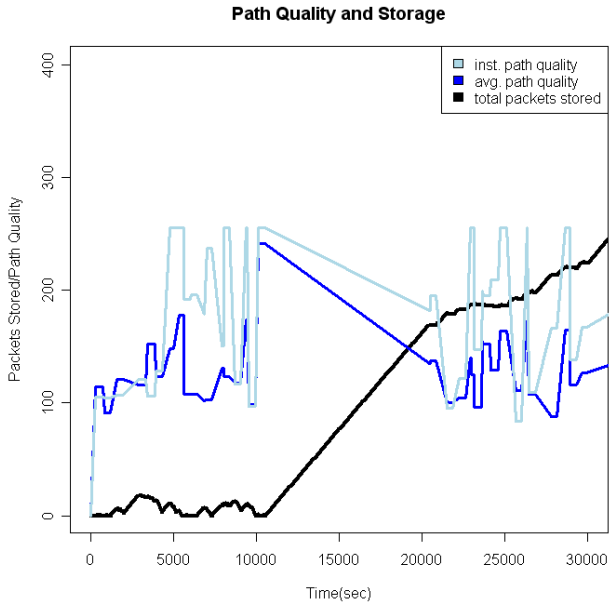

Figure 11 An overlay of both path quality and used storage capacity as a function of time for a 5 hop neighbor (Node 38).

7.1.1 Evaluation

Both graphs 10 and 11 show the path and storage metrics for a single hop node and a multiple hop node. The y-axis represents both the path quality and the number of packets stored. The path quality is measured as the number of expected transmissions to successfully send a packet to the sink, including retransmissions. A measurement of 255 means a complete disconnection. Instantaneous path quality is represented by the light blue line, average path quality is represented by the dark blue line, and the number of packets stored is represented by the black line.

Using these graphs we are able to analyze instantaneous path quality vs average path quality, storage as a function of path quality, and congestion control for the both the single hop case and multihop case.

Single Hop Analysis

In the one hop case (Figure 10), a majority of the instantaneous fluctuations are short lived relative to the data generation period resulting in minimal packet storage. Packets which are stored during these periods are randomly distributed.

At approximately t=10000 a disconnection occurs for this one hop node. As expected environmental conditions and low cost radios have temporal variations in quality. In this case the combination has caused a disconnect between this node and the sink. During the network disconnect packets are stored locally as represented by the steadily increasing packet count. Once network

connectivity is regained, packets are again drained from the network. The decreasing packet count is a result of packet reinjection.

After regaining connectivity, the initial rate of packet reinjection is slow as indicated by the shallow slope of the decreasing packet count. This is a result of our congestion control mechanism. As packets are successfully reinjected the data rate increases as indicated by the increased slope.

Multihop Analysis

In the multihop case (Figure 11), again we see instantaneous fluctuations in path quality. In this case the difference between instantaneous and average path quality is more pronounced and longer lived. Differences in quality last in the order of tens of minutes. This is due to the accumulated affect of small variations in link quality over the multiple hops. The result of there is a greater number of packets stored during the periods of large deviation. It is interesting to note that the packets stored during these fluctuations are actually sequential packets where as in the one hop case packets were randomly distributed. Applications able to handle random packet loss would not need reliability in the single hop case, but would fail without reliability in the multihop case.

At approximately t=10000, this node also experiences a decrease in path quality as indicated by the pronounced spike in both instantaneous and average path quality. Once again the node stores packets locally during this period as indicated by the increased packet count.

Congestion control attempts to reinject packets into the network once the path quality has returned to its previous value, but relatively poor path quality causes the reinjection policy to stay in slow start. Eventually worsening connectivity causes the number of stored packets to increase. This case is a good example of a situation in which poor link quality will always require longer term storage as a mechanism to handle consistently long queue delays.

This experiment shows the relationship between path quality and storage. The reliability architecture was able to react to poor link quality and long periods of disconnect by storing and reinjecting packets at appropriate times. Without reliability and storage large consecutive portions of data would be lost for analysis.

7.2 Deployment: Cold Air Flow Project

As stated earlier, the motivation for this research was the Cold Air Flow Project at James Reserve. At the time of writing, the current implementation of ESS has been collecting data at the James Reserve for over 3 weeks. Data is queried at an interval of 5 minutes, resulting in approximately 6000 data points per node over the 3 week period. The metrics collected are from a single 18 node transect along the valley. Results were returned using the Sympathy debugging service provided by ESS. Sympathy is an in-band debugging mechanism which returns system metrics over the multihop routing tree. Since Sympathy is in-band, disconnects will cause gaps in Sympathy data. The storage metrics collected by Sympathy represent a total aggregate number of packets collected so storage activity can be extrapolated over gaps.

Node Id	Total Pkts Stored	External Pkts Stored	Num Hops
39	860	0	3
106	1205	172	3
120	986	61	4
74	863	0	1
98	1020	0	5
76	836	0	2
11	811	0	2
111	4783	1913	2
118	2245	1408	3
100	886	17	4
13	856	5	3
79	1187	292	4
19	903	10	4
7	939	7	3
155	886	23	1
25	1338	488	1
97	1527	674	2
114	860	0	3

Table 1 Storage information collected from sympathy over a 3 week period

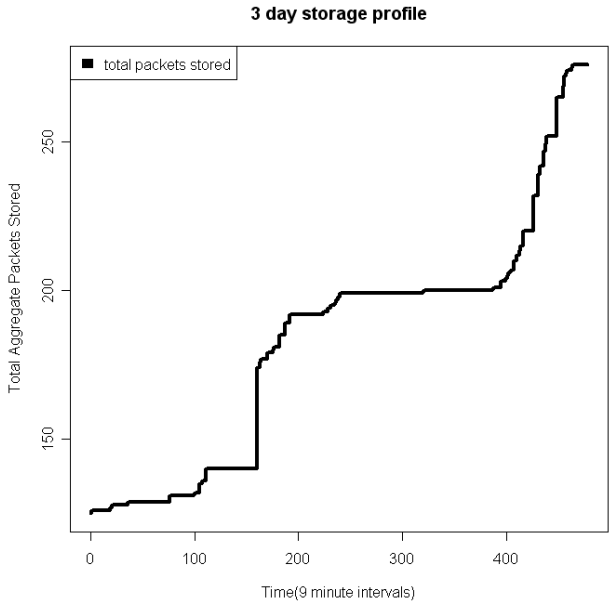

Figure 12 A 3 day storage profile for a single node.

Table 1 shows the total number of packets stored on each node over its lifetime. These represent packets which would have been lost if not for the reliability service. Each node has stored a minimum of approximately 800 packets (14% of the total packets generated at each node). In some cases nodes have stored external

packets from upstream children, in cases where routing information was slow to propagate.

Figure 12 shows a 3 day profile of storage for node 79. When examining this profile we see storage proceeds in a step like manner. This indicates that storage occurs in periodic batches. Previous implementation [2,3] assumed that loss occurred randomly and that retransmissions would be enough to mask these random losses. This profile shows that data is lost in consecutive batches where local storage is needed to buffer data during these periods.

Figure 12 also shows a period of complete disconnect. The disconnection can be seen from approximately t=125 to t=175. During this time Sympathy was not able to collect data for this node, which results in the sudden jump in total packets stored.

Results from our deployment at James Reserve have shown that poor link quality and frequent disconnects can cause batches of data loss. Our architecture was able to successfully store data during these periods guaranteeing data was not lost.

8. Conclusion

Harsh environments and wireless communication make reliability in sensor networks an interesting problem. In these challenged networks poor path quality and frequent disconnects make conventional network protocols unfeasible. In this paper we propose a hop-by-hop, forwarding protocol. We show that by moving reliability out of the end hosts and into the intermediary nodes we can increase the probability of successful packet transfer over multiple hops. In addition, we show that long queuing delays at forwarding nodes require persistent storage to buffer data during long periods of disconnection.

We have integrated our architecture as a service in ESS and have deployed an implementation for the Cold Air Project at James Reserve. Deployment results have validated our assumptions about challenged networks and have quantified the benefit of our reliability architecture in the number of saved packets.

9. Future Work

The network stack described in this architecture provides packet by packet reliable routing. In the future we would like to implement an intelligent transport layer which provides redundancy.

The architecture presented provides reliability in the face multiple failures such as of disconnected networks or node resets. A source of failure not covered by this architecture is unrecoverable node failure. In this all the data present on the failed node is lost. Reconstruction of the complete message at the end host becomes impossible.

The solution to this issue requires data redundancy provided in the network. There are two possible redundancy mechanisms to follow up in the future.

The first mechanism provides redundancy at the packet level. This mechanism can use a variety of existing hashing techniques such as Forward Error Correcting (FEC) or existing techniques in multimedia delivery. In these cases lost packets are reconstructed from other packets.

The second mechanism uses the transport layer to initially store the entire message to persistent storage at the source before sending. As messages from various sources are assembled at the sink, the sink keeps a journal of packets assembled. Periodically the sink floods a digest of the most current packet assembled from each node. As nodes receive the digest they can begin to release locally stored data.

Both mechanisms have their merits. Future work will examine these techniques to determine which is most appropriate for sensor networks.

10. References

[1] V. Jacobson. Congestion avoidance and control. *ACM ComputerCommunication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, 18, 4:314–329, 1988.

[2] Fred Stann and John Heidemann. RMST: Reliable Data Transport in Sensor Networks. *1st IEEE International Workshop on Sensor Net Protocols and Applications*, 2003

[3] C. Wan, A. Campbell, L. Krishnahmurthy. PSFQ: A Reliable Transport Mechanism for Wireless Sensor Networks. *ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, Georgia, Sept 2002*.

[4] Serqiu Nedenshi, Rabin Patra. DTNLite: A Reliable Data Transfer Architecture for Sensor Networks. *8th International Conference on Intelligent Engineering Systems, 2004*

[5] A. Kawaguchi, S. Nishioka, and H. Motoda. A flash-memory

based file system. *In USENIX Winter, pages 155–164, 1995*.

[6] Hui Dai, Richard Han, Michael Neufeld, ELF: An Efficient Log-structured Flash File System for Micro Sensor Nodes. *In the Proceedings of the 2nd international conference on Embedded networked sensor systems, 2004.*

[7] D. Gay. The Matchbox File System. *http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/matchbox-design.pdf*, 2003.

[8] Kevin Fall et al. DTN Implementation. *http://www.dtnrg.org*, 2003.

[9] James Reserve. *http://www.jamesreserve.edu*

[10] Crossbow Technology Inc. Mica2 wireless measurement system datasheet, *http://www.xbow.com*

[11] Philip Levis et al. David Culler. TinyOS: A Component based OS for wireless sensor networks. *http://webs.cs.berkeley.edu/tos/*, 2003.

[12] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets.*ACM SIGCOMM*, 2003.

[13] Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks, A. Woo, T. Tong, D. Culler. *ACM Sensys 2003, Los Angeles, November 2003*

[14] Rahul Kapur, Thanos Stathopoulos, Deborah Estrin, John Heidemann, Lixia Zhang. Application-Based Collision Avoidance in Wireless Sensor Networks. *In IEEE Workshop of Embedded Networked Sensors (EmNet) 2004*

[15] Em*: A Software Environment for Developing and Deploying Wireless Sensor Netwroks, L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, D. Estrin, *CENS Technical report #34*