DTNLite: A Reliable Data Transfer Architecture for Sensor Networks

Sergiu Nedevschi
University of California, Berkeley
sergiu@cs.berkeley.edu

Rabin Patra
University of California, Berkeley
rkpatra@cs.berkeley.edu

***Abstract*— We present a network architecture, *DTNLite*, for reliable message delivery in sensor networks facing problems of high mobility, frequent disconnections and unreliable nodes . It is based on the DTN(Delay Tolerant Networking) architecture and its main features are asynchronous message delivery combined with custody transfer on an overlay network of sensor motes. We present an implementation of this architecture for the TinyOS platform targeting data collection applications. We also explore the various issues in reliable custody transfer and investigate the particular issue of *querying and selection of custody hops* in detail. Our simulation results show that selection criteria that use energy or delay as metrics are able to profitably exploit asymmetries in the network.**

I. Introduction and Motivation

Wireless sensor networks are a cost effective, distributed solution, providing sensing and computing solutions in various environments where conventional networks are impractical. This paper addresses the design of system support for reliable data delivery in sensor networks facing challenges such as sparse connectivity, high degree of mobility, flaky nodes and unreliable links. However the key questions are why reliability, why challenged sensor-nets, and why reliability in challenged sensor-nets?

Unlike traditional networks, reliability in sensor networks is still an open research question. There has been little amount of work on the design of reliable delivery protocols, and most of the existing solutions are application-specific. However, as sensor networks become ubiquitously deployed, we can imagine a large class of applications where reliable delivery is required. A good example is network reprogramming of sensor nodes, where the reliable delivery of every single byte of code is necessary. Reliable and timely delivery of emergency events is another.

The majority of current solutions for sensor nets assume high connectivity degrees, manageable mobility and low error rates. However, few real environments can have such well-controlled parameters, and providing these properties requires large numbers of nodes, and important energy consumption. Covering extended sensing areas is achievable by tolerating smaller node density, and maintaining a long lifetime recommends small on-times. All these are reasons for exploring solutions for challenged networks.

II. Related Work

In this section we provide a brief survey or related work in the areas of reliable data delivery in sensor and delay tolerant networking.

Efficient transport protocols to provide reliable data delivery in sensor networks have been proposed in [9] and [8].

In [8], authors introduce *RMST*, a transport protocol that provides guaranteed delivery and fragmentation/reassembly for applications that require them. *RMST* is a selective NACK-based protocol that can be configured for in-network caching and repair. Some number perform message reassembly, issuing repair request to the previous nodes in the path.

In [9], the *Pump Slow Fetch Quickly* protocol is presented, where each node performs a special type of message reassembly. That is, nodes can immediately forward to the next hop fragments if they are received in order. However, as soon as they receive an out-of-order fragment, they issue a repair request, and buffer the out-of-order fragment until the missing fragment is obtained and relayed. Nodes are thus performing fragment ordering. The repair requests are answered by previous nodes in the path that use fragment caches. PSFQ targets a small delay, comparable to forwarding approaches, combined to the reliability and small number of retransmissions specific to hop-to-hop store-and-forward.

Delay Tolerant Networking is an emerging field that attempts to develop a networking architecture[2] and philosophy revolving around asynchronous message delivery with custody transfer for networks that are subject to long delays and/or frequent disconnections that rule out contemporaneous end-to-end connections. The architecture operates as an overlay above the transport layers of the networks it interconnects, and provides key services such as in-network data storage and retransmission, inter-operable naming, authenticated forwarding and a coarse-grained class of services. Some of the issues involved in the custody transfer handshake and duplicate generation are discussed in [3].

III. Design Decisions

In this work, we explore achieving reliable delivery using acknowledgments and retransmissions. The best possible solution is the one that makes the most efficient use of retransmissions and storage(in volatile and non-volatile memory). The end-to-end argument dictates end-to-end acknowledgment as the only true answers for reliability. However, adding functionality at the intermediary hops can significantly increase efficiency.

Packets can be retransmitted at each hop, at some number of intermediate hops, or only at the source. Each

packet can be acknowledged independently, or selective acknowledgment approaches can be used.

However, traditional acknowledgment-based reliability mechanisms exhibit several common difficulties when faced with unusual challenges. These challenges are not exclusive, and usually happen simultaneously (e.g. disconnection and high round trip delay), however we will present their effects one at a time:

- **High round trip delay:** We assume the round trip delays between sources and destinations on the order of hours if not days, mainly due to disconnections. In the end-to-end acknowledgment schemes, the full message content needs to be stored at the data source, until every fragment is delivered to the destination. Since the round trip delay is so large, the acknowledgments will arrive much subsequent to the moment when data is actually delivered to the destination, and data sources will thus have to store large numbers of such messages; generation of new data may be impaired.
- **Disconnections:** We assume end-to-end connected paths between source and destination do not always exist, or they might not exist at any single moment in time. As no stable storage buffering is used, the node where the connected path is interrupted will be overflowed with data it cannot handle.
- **Large messages:** If the message size exceeds the available memory capacity, the packet reassembly cannot be performed at the intermediary nodes.
- **High mobility:** High mobility leads to routing instability and the underlying routing might not be able to maintain updated state. If the end-to-end paths are long, the message transfers might be frequently aborted especially if the links are not always symmetrical.

Taking into account these problems, we propose some mechanisms to address them. These ideas are inspired from the mechanisms underlying the Delay Tolerant Networking architecture presented in [2].

- **Store-and-forward using stable storage:** The mechanism is intended to alleviate buffer overflow problems associated with disconnections. Since buffered messages might be stored for long periods of time, buffers are likely to grow beyond the node volatile memory capacity. Moreover, buffering using stable storage increases reliability, and unreliable nodes cease to be a problem, since data is not lost during power-downs and resets.
- **Custody transfer as an alternative to end-to-end reliability:** Keeping a full copy of the data at the source until an end acknowledgment is received comes with great storage utilization penalties. A possible solution is an alternative reliability paradigm, called **custody transfer**. A custody transfer refers to the acknowledged delivery of a message from one hop to the next and the corresponding passing of reliable delivery responsibility. In other words, the custodian node, after storing the message in stable storage, becomes responsible for its successful delivery, and the previous custodian, which might be the source, can delete its own copy.

IV. DTNLite for Sensor Networks

This section discusses the design issues for implementing a custody based reliable transfer mechanism(*DTNLite*) for sensor networks and then presents an implementation for the TinyOS platform. For a background on the Berkeley TinyOS platform, the reader should consult [1].

The proposed mechanism is based on the abstraction of message switching. Messages (or bundles) are transferred on an overlay network formed of nodes that are ready to perform store and forward functions. The actual node-to-node bundle transfer between overlay nodes is done using the **custody transfer** mechanism.

A. Design Issues

- **Custody transfer with reliability:** The most important problem is the mechanism implementing the one hop transfer of a bundle from one overlay DTNLite hop to another. This is especially difficult since the bundles are *application data units*(ADUs) which means they are usually larger than the underlying network packets and have to be deliver in-order. The underlying networking layer is also rudimentary and ordinarily does not provide multiple hop reliability. The available options include multi-path sending and packet replication.
- **Persistent storage management:** Reliability demands that the sensor network nodes persistently store bundles until they are successfully able to delegate responsibility to another node. The first requirement is for the nodes to posses non-volatile storage such as flash. Making this assumption, the easiest solution is to use database operations for writing and reading bundles. Unfortunately, sensor motes are very resource constrained and they usually lack full-fledged file systems. In such a situation we have to make sure that all bundle writes are forced (flushed to storage) before a custody transfer can be acknowledged as completed. The flash storage system should be capable of supporting some low level atomic operations that would make sure that the record of stored bundles is consistent.
- **Duplicate management:** Since frequent disconnections are assumed, complete elimination of duplicate bundles is not possible. The simplest scenario where a duplicate bundle can be created is when a custody transfer acknowledgment is lost, resulting in both the sender and receiver maintaining custody. In such situations we can either assume a *deliver at least one* model or need to include some mechanisms to detect duplicates at the base-station.
- **Application awareness:** This issue concerns the degree to which applications should be aware of the network conditions. Awareness is desirable because giving applications the ability to adapt to changing network conditions can increase the network's efficiency. Solutions for long term storage of data in sensor motes can be provided by using mechanisms such as in-network aging and summarizing and compression of data in case of communication challenges.

B. Architecture and Implementation

Considering the design issues for a generic custody transfer framework, we propose a network stack architecture for the TinyOS platform. The design is loosely based on the DTN overlay architecture proposed in [2]. We have particularly targeted our implementation for data collection

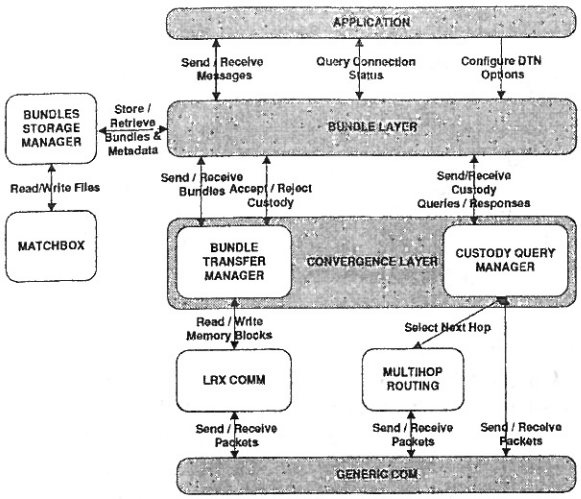


Fig. 1. DTNLite architecture

applications where there is a central base-station. Nodes generate data having the base station as the destination. We choose to cover this particular case because the overlay routing in DTNLite needs an underlying multi-hop routing protocol, and in this respect tree-based routing represents a viable alternative. However, as generic multi-hop protocols for any-to-any routing in sensor networks become available, the architecture can be easily adapted.

Addressing is an important issue in disconnected networks, but we make the simplifying assumption that all nodes have a unique static address, and that all packets share the same destination. Figure 1 presents the layering of the important components of the proposed architecture.

1) *Bundle Storage Manager:* This component (`BundleStorageMgr`) is responsible for providing persistent storage for bundles. For the Berkeley Mica motes, the flash(about 512K bytes) is the available storage medium. We use the *Matchbox* file system([4]), and its implementation for Mica motes. The Matchbox filesystem uses atomic write operations for operating with file meta-data, and provides support for data corruption detection.

2) *Bundle Agent:* This component (`BundleAgent`) corresponds to the overlay routing layer. The *Bundle Agent* provides to the application an interface for sending and receiving bundles. It is responsible for implementing the custody transfer handshake with the corresponding *Bundle Agent* on the next overlay hop. In addition, it is responsible for querying the network for an available next custody hop, and for selecting the best among the candidates. It uses the convergence layer for the actual transfer of a bundle to the next custody hop. It also relies on the convergence layer for performing and getting responses for custody queries, and for sending out custody transfer acknowledgment messages.

3) *Convergence Layer:* This component(`ConvLayer`) is analogous to the components that provide neighborhood discovery and maintenance in traditional sensor networks. It provides the *Bundle Agent* with basic primitives for transferring a bundle to another custody hop, for sending and receiving custody query requests and custody acknowledgments. It uses the following components:

1) *LRX:* This component(`LRX`) is used for the reliable multi-hop transfer of a bundle from one custody hop to another. It basically provides a primitive for high-speed transfer of a bundle over one hop. It uses a basic windowing scheme along with selective NACKs . However the *Convergence Layer* uses source routing for specifying the route of a bundle over multiple network hops to the next custody hop.
2) *MultiHop:* This component is used by *Convergence Layer* for sending/receiving custody queries and acknowledgments. The custody responses from potential custody hops contain the route of the path to the respondent.

Table I illustrates the mapping of analogous components between traditional sensor networking and DTNLite.

Sensor network function	DTNLite function	DTNLite Component
Packets	Messages(in-order)	-
Neighborhood discovery and maintenance	Custody Query and Discovery	`ConvLayer`
Multihop packet transfer	Custody hop bundle transfer	`BundleAgent`
Network Hop packet transfer `GenericComm`	Network hop bundle transfer	`LRX`

TABLE I
MAPPING BETWEEN COMPONENTS

We have implemented and tested DTNLite on TOSSIM([6]), the simulator for TinyOS applications.

V. CUSTODY TRANSFER

This section discusses an important aspect of the DTNLite framework - the custody transfer mechanism. The discussion covers the discovery and selection of potential custody nodes and the mechanics of the custody handshake. There is a more detailed discussion of these issues in [3].

The reliable custody transfer of a message requires a handshaking protocol between the source and destination nodes. Unfortunately, handshaking cannot insure both *no message loss* and *no message duplication.* In other words, we need to choose between having reliable transfer or duplicate-free transfer. Since reliability is more important in our design, message duplication must be accepted and dealt with.

Since writing to flash is an expensive operation in terms of energy consumption, the number of times a message needs to be written to flash, and thus the number of custody transfers, must be minimized. Some concerns are also related to network disconnections. If because to temporary disconnection, a route making forward progress toward the destination does not exist, the message must remain in custody until a valid route toward destination is discovered.

In order to address these concerns, the routing protocol must choose the best neighbor in the overlay to transfer custody of a message. After having decided on the best neighbor, it must decide whether transferring custody to this node is worthy at the current time.

The routing strategy can be targeted toward achieving one or more of the following global optimizations:

- **Minimizing the overall energy consumption** Very roughly, energy consumption in the network can be modeled as the sum between the energy spent for packet transmissions, retransmissions and acknowledgments, and the energy consumed by writing/reading the messages to flash. On one hand, minimizing the number of custody hops reduces the energy spent for flash writing/reading. This recommends always taking long leaps toward the destination, by choosing the neighbor in the overlay that is closest to the destination. On the other hand, failure to transfer custody happens more often if done over longer underlying routes (takes longer and due to dynamic conditions, the network topology can change during the process).
- **Obtaining a uniform distribution of the energy levels of nodes**. This optimization has the effect of improving the overall network lifetime, preventing some of the nodes from dying prematurely.
- **Minimizing the delay in message delivery**, and the number of undelivered messages due to unavailable storage capacity in the network. Minimizing the delivery delay has the effect of minimizing the storage used in the whole network. If we regard the network as a queuing system, the average delivery time of messages is the average time spent in the queue. Applying Little's Law:

$$Length_{queue} = Arrival\ Rate \times Time_{queue} \quad (1)$$

we notice that the amount of used storage is directly proportional to delivery time.

While global routing information is preferable to local information, in these conditions maintaining global information is expensive because of limited storage, and is slow to propagate. This generates lot of network overhead and even then does not capture the latest network changes. Thus, we choose to use local routing information.

We propose simple schemes, relying on local self-assessments for making the routing decision. Choosing the neighbor in the overlay means choosing the node with the best local metric. Any combination of the following metrics can be employed:
- **Energy level remaining:** Choosing the node with the highest energy level remaining is intended to yield a uniform energy distribution.
- **Average delivery time:** This represents the average time in which messages owned by the node are delivered to their final destination. A choice based on the average delivery delay is intended to minimize data loss.
- **Average energy consumption for message delivery:** This metric represents the average energy consumed by messages sent by the node until they are delivered. Choosing the neighbor with the minimum average energy consumption will decrease the overall energy consumption in the network.

A. A custody transfer mechanism implementation

In our initial implementation of the DTN architecture we propose a simple custody transfer mechanism, relying on the principles defined previously. Our implementation assumes the existence of an underlying multi-hop routing protocol. Since our system is intended for data collection, with messages sent to a single destination - the base-station, tree-routing can be assumed. However, this mechanism can also be used for applications with multiple message destinations and any-to-any multi-hop routing. The protocol relies on the fact that nodes maintain local estimates of the metric used. Quality estimates of links between nodes are also maintained at both link ends.

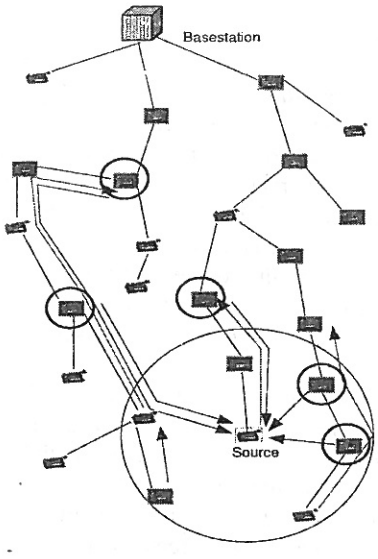


Fig. 2. Example Custody Query

1) Querying mechanisms: The querying mechanism works as follows: the custodian of a message sends a query, asking for nodes that are able to accept custody for the given message, and that have a better local metric estimate than itself. The query packet contains the estimate of its originator as well as characteristics of the message to be transferred. The query can be sent in several ways:
- **Unicast** toward destination, using the route provided by the underlying tree-based routing.
- **Flood limited** to a given number of hops, continuing with *unicast* toward destination.
- **Full flood**

Each of these techniques has advantages and disadvantages that will become evident once we discuss the entire mechanism. The query packet advances as far as possible toward destination, and the path traversed is added to the packet. All hops on the way are queried, and the ones that are willing to accept custody and have a better metric than the current custodian send a response to the query. In order for a node to decide whether it can accept custody, it needs to estimate if it has enough available storage. The response is sent back to the custodian using the reverse of the path recorded in the query. Please note that symmetry of links is assumed. The response contains the local metric estimate of the respondent. Quality estimates of the traversed links are recorded in the response as well. The custodian selects the best candidate among query respondents, using the metric of the candidates, as well as the quality estimate of the paths to these candidates.

Based on the quality of the best candidate, the custodian estimates whether the custody transfer is worth doing at the current time. If it decides to attempt a custody transfer, it sends the message to the best candidate, using source routing on the path recorded in the query response.

The message advances one hop at a time, using a reliable hop-to-hop message delivery mechanism based on selective nacks and retransmissions, implemented in [7].

Figure 2 presents an example of the query packet propagation. The source broadcasts the query to all its neighbors in the real network, and they all forward it toward the base-station. The boxed nodes are the ones that can accept custody, and among those, the circled have better metrics than the source, and consequently represent candidates for the next custody hop. Candidates respond to the query; the query and response packets are represented by arrows.

2) Metric estimates: The reliability of a link can be expressed as the ratio of packet retransmissions to the total number of transmissions on the given link. Every time a message is transferred on the link, using the hop-to-hop reliable delivery mechanism, this number is updated. Estimating the energy level remaining in a node is trivial. The average delivery time and the average energy consumption can be maintained either using a distance vector like approach, or by piggybacking information on messages and let the base-station compute the estimate. We implement the first approach.

VI. Evaluation of Custody Transfer Mechanisms

In this section we investigate the various custody transfer query and selection mechanisms presented in more detail in section V. In particular, we study the following custody query/discovery policies:

1) *Unicast* : The query is sent along the routing tree up to the destination.

2) *Limitedflood* : The query is flooded up to some levels and then sent along the routing tree to the destination.

and the following custody hop selection policies:

1) *NEAREST - Nearest Hop:* Expected to lead to more custody transfers than necessary, thus more energy consumption.

2) *FARTHEST - Farthest Hop (Closest to destination):* Minimizes the number of custody transfers, but each transfer is expected to take longer, since the probability of reliable transfer decreases with distance.

3) *ENERGY LEVEL - Node with Most Remaining Energy :* Expected to maximize overall network lifetime, minimizing the variations in the energy levels.

4) *AVG DELAY - Least Average Delay:* Every node maintains an estimate of the average delivery delay, and the node having the smallest delay is chosen.

5) *ENERGY CONS - Least Average Energy Consumption:* The policy chooses the node that on average consumes the least energy for sending a message to the destination.

All the policies mentioned above rely of completely local information for maintain estimates.

A. Simulation Setup

We are using a discrete event based simulator [5] developed for simulating routing algorithms for networks with scheduled disconnections. We made the appropriate modifications for handling unscheduled disconnections and packet collisions. For the connectivity model, we assume that all nodes within a certain radius are connected by a link. To model the intermittent nature of the network, we assume that a link can go up or down at exponentially separated intervals - the time that a link stays up is determined by the distance between the nodes. The radio collision model is simplified; packets collide only if they are destined for the same node. The queries are flooded for three levels and are then forwarded to the base along the routing tree.

Parameter	Range in simulation	Range for MICA
Time	1	1 sec
Packet size	1	25 B
Bundle size	25	625 B
Bandwidth	40/unit	1KB/s
Flash storage	5K-20K	125KB-500KB
Average degree(density)	8	-
Num of nodes	10-100	-

TABLE II
Simulation setup

For executing the custody queries, a routing tree is maintained by the nodes. The tree is formed by performing a breadth first search starting from the base-station (root node). We assume that the effective cumulative bandwidth that the base station node can handle is constant - but to normalize for all network sizes, the rate of data generation for the nodes is determined according to the number of nodes. All the custody selection policies are modeled using the approaches mentioned in section V. However we do not model the query mechanisms at packet level - we assume full knowledge of the network for selecting potential custody nodes. We choose to do so because we are more interested in comparing the custody hop selection policies per se. To make the parameters more realistic, we use the Berkeley Mica motes as the basis for parameter choosing. The mapping between Mica motes and simulation parameters is shown in table II.

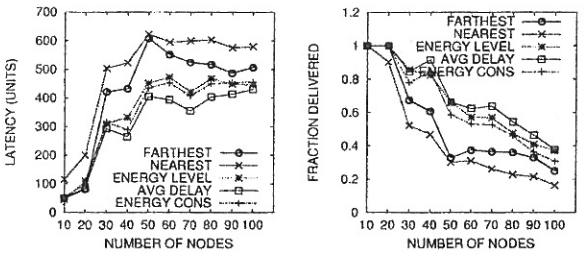
B. Evaluation

We compare the different metrics by considering a number of parameters. We perform all experiments for both unicast and limited flood queries, but since the policies have very similar behaviors, we only present the results for the limited flood queries.

1) Delivery Time*:* The first set of graphs compare the different custody transfer policies with respect to the delivery time of messages (Fig.3). We can clearly see that the *AVG_DELAY* policy expectedly performs better than the other policies, though none of them is able to deliver all the messages within the (fixed) simulation interval. Not surprisingly the *NEAREST* and the *FARTHEST* policies have the worst average latency. The *FARTHEST* policy loses out because it takes more time to complete successful custody transfers over longer hops, while the *NEAREST* policy undergoes longer delays because of the larger of number of hops that it takes.

Fig. 3 (b) shows the fraction of messages that are delivered at the end of simulation time for varying network sizes. An interesting trend that can be observed is that delivery ratios decrease as the network size gets larger - this is because of the congestion at the single base-station

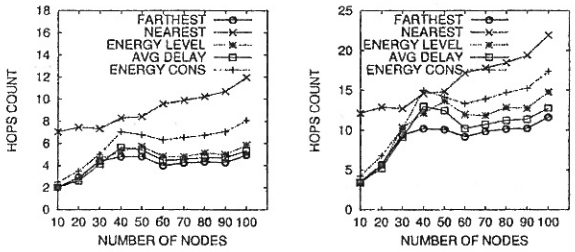
sink and suggests that the data generation rates should be lower for reliable delivery.



(a) (b)

Fig. 3. (a) Average latency for message delivery; (b) Fraction of messages delivered

2) Hops per message*:* Here we compare the number of hops needed for messages to reach the base-station for each of the observed policies. Fig. 4 (a) plots the average number of custody hops needed for each successful message received at the base. We see that the *FARTHEST* policy minimizes this criteria, although, as we see in Fig. 3 (b), it sacrifices on the delivery ratio. As expected the *NEAREST* policy maximizes the number of custody hops as is always chooses the node that is closest to the source.



(a) (b)

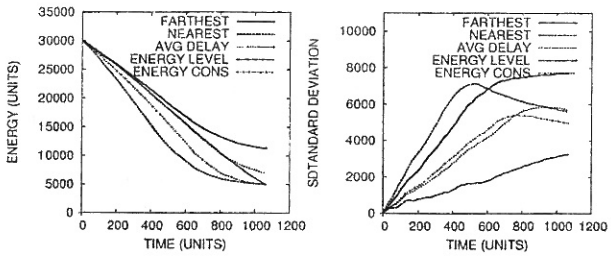
Fig. 4. (a) Custody hops per message; (b) Networks hops per message

Fig. 4 (b) plots the average number of network (not custody) hops needed for each successful message, and we can observe that the *NEAREST* policy uses the maximum number of network hops for delivering a message.

3) Energy*:* In these set of experiments, we compare the energy efficiency of different policies. Fig. 5 (a) shows the average node energy level over time for all the policies, while Fig. 5 (b) shows the standard deviation of the node energy levels with time. While we see that the *FARTHEST* policy uses the least energy overall(since it has to perform fewer custody transfers and therefore fewer writes to stable storage), it has far worse delivery performance. However the *ENERGY_LEVEL* policy is able to equalize the energy level differences among the nodes.

C. Results

We can conclude from the above results that the selection policies based on energy are able to minimize



(a) (b)

Fig. 5. (a) Average Node energy level over time(nodes=100); (b) Standard deviation of energy level with time(nodes=100)

the energy usage variation among the nodes while those based on delay are able to minimize latency of message delivery. All the policies use purely local information and the estimates about link qualities and node metrics are updated using information from neighboring nodes. This entails that, if there is a systematic variation in the link qualities, these variations can be learned over time by maintaining local estimates that are then aggregated by other nodes.

VII. Conclusion and Future Work

We have implemented DTNLite, a custody transfer based reliable transfer mechanism for sensor networks that face challenges like low memory resources, high mobility, frequent disconnections and flaky nodes. We have discussed some of the important issues in custody transfer and investigated one of them, namely *querying and selecting of the next best custody hop* in more detail. The evaluations show that our scheme can achieve reliability, while optimizing for a particular objective like energy or delay, using purely local information. In the future, we intend to perform a comparative evaluation of our architecture, relying on the custody transfer mechanism, with other proposed reliable transfer protocols(section II).

References

[1] Philip Levis David Culler. TinyOS: A Component based OS for Wireless Sensor Networks. *http://webs.cs.berkeley.edu/tos/*, 2003.

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

[3] K. Fall, W. Hong, and S. Madden. Custody Transfer for Reliable Delivery in Delay Tolerant Networks. *Intel Technical Report:IRB-TR-03-030*, 2003.

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

[5] Sushant Jain. DTN Simulator. *http://www.dtnrg.org*, 2003.

[6] Matt Welsh Philip Levis, Nelson Lee and David Culler. TOSSIM: Accurate and Scalable Simulation of entire TinyOS applications. In *Embedded Networked Sensor Systems*, pages 126–137. ACM Press, 2003.

[7] S.Kim. Structure Monitoring using Wireless Sensor Networks. *CS294-1 Deeply Embedded Network Systems class project*, 2003.

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

[9] Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy. PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks. *First Workshop on Sensor Networks and Applications (WSNA), September 28, 2002, Atlanta, GA.*