DTNLite: A Reliable Data Transfer Architecture for Sensor Networks

Rabin Patra and Sergiu Nedevschi
CS294-1: Deeply Embedded Networks(Fall 2003)
{rkpatra,sergiu}@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 on 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 the 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 sensornets, and why reliability in challenged sensornets?

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. This is mainly because in traditional sensor network applications, such as monitoring, directed diffusion or object tracking, occasional data loss is acceptable. 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. For applications with large data units, exceeding the maximum packet payload, reliable delivery of all fragments is required to make the data usable.

The majority of current solutions for sensor nets assume high connectivity degrees, manageable mobility and low error rates. On the other hand, 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 an important tool in extending coverage is mobility, i.e. moving data around. Moreover, maintaining a long lifetime recommends small on-times. All these are reasons for exploring solutions for challenged networks.

Finally, in challenged networks, reliability, though hard to achieve, is essential, because data loss opportunities are plentiful, and without reliability mechanisms, most of the data will be lost or unusable.

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.

A. Reliable Data Delivery in Sensor Networks

Efficient transport protocols to provide reliable data delivery in sensor networks have been proposed in [1] and [2].

In [2], 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. Nodes along the route of a message cache message fragments, and some number of nodes perform message reassembly, test for lost fragments, and issue repair request to the previous nodes in the path.

Another reliable transport protocol for sensornets is presented in [1]. In this approach called *Pump Slow Fetch Quickly*, 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. Most of these approaches are application-specific reliable data delivery mechanisms. In [3], the author proposes a store-and-forward hop-to-hop efficient data transfer protocol, used for high-bandwidth data collection in the structural monitoring of the *Golden Gate* bridge.

B. Delay Tolerant Networking

This is an emerging field that attempts to develop an networking architecture[4] 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. An implementation of this protocol is also publicly available at [5].

The ZebraNet [6] system includes custom tracking collars (nodes) carried by animals under study across a large, wild area; the collars operate as a peer-to-peer network to deliver logged data back to researchers. The collars are small wireless devices that include GPS, flash memory, wireless transceivers, and a small CPU.

III. Design Space

The main instruments for achieving reliable delivery are acknowledgment and retransmission. Thus, the design space for reliable solutions will be shaped by retransmissions: when are they performed and who is responsible for them. The best solution is the one that makes the most efficient use of retransmissions.

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 treated independently, and acknowledged independently, or selective negative acknowledgments can be sent for the missing fragments of a larger message.

From the point of view of the number of nodes that perform retransmissions, the extreme cases are when acknowledgment and retransmission is done at each hop, and the case when they are done only at endpoints. As illustrated in [1], the second solution is well suited for very reliable and stable networks, with less than 1% error rates. However, it is not suited for links with high error rates, because the error probability for the whole path accumulates, becoming $1 - (1 - p^n)$, where p is the link error rate and n is the number of hops in the path. Consequently, retransmissions at intermediary hops are desirable.

From the point of view of the number of fragments that need to be acknowledged, the extremes are the forwarding and store-and-forward approaches. In forwarding, each packet is acknowledged separately, and can immediately be sent to the next hop. In store-and-forward on the other hand, messages are assembled at every hop, and the missing fragments are negative-acknowledged. Forwarding is inefficient due to the high number of acknowledgments, while store-and-forward features large delays.

There are other approaches that reside in between extremes, both in terms of number of nodes performing retransmission, as well as in terms of number of acknowledgments per message. Both [1] and [2] propose hybrid approaches that rely on in-network caching, loss detection and repair. In the PSFQ approach([1]) , the solution exhibits a multi-modal property that provides a graceful trade off between forwarding and store-and-forward paradigms.

Having understood the design options for reliable data delivery in non-challenged sensor networks, let us analyze the performance of the available options when challenges such as disconnection and high mobility are added to the set of assumptions about the network. We perform the analysis on three representative alternatives:

- *Alternative 1:* Forwarding, with end-to-end acknowledgment only.
- *Alternative 2:* Store and forward, with packet reassembly at each node and selective negative acknowledgments. No end to end acknowledgments. Data is stored in volatile memory.
- *Alternative 3:* Store and forward, with packet reassembly at each node and selective negative acknowledgments. End to end acknowledgments. Data is stored in volatile memory.
- *Alternative 4:* The PSFQ hybrid approach.

Let us enunciate some of the difficulties exhibited by the above reliability mechanisms when faced with different types of 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. Alternatives 1, 3 and 4 all require final end-to-end acknowledgments. This means the full message content needs to be stored at the data source, until every fragment is delivered to the destination. However, since the round trip delay is so large, the acknowledgments will arrive much subsequent the data was actually delivered to the destination, and data sources will thus have to store a large number of such messages. This is obviously problematic, since storage is limited at the data sources, and consequently generation of new data may be impaired. Alternative 2 does not face this problem.
- *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. For all the observed alternatives, the fact that source and destination are not connected does not stop sensors from sampling and sending data toward the destination. However, since no stable storage buffering is used, the node where the connected path is interrupted will be overflowed with data it cannot handle, if it has limited volatile memory(RAM) capacity.
- *Flaky nodes:* We assume sensor nodes are unreliable and that they can go to sleep unexpectedly and wake up at a later time. Under these assumptions, alternative 3 cannot guarantee reliability, since the messages stored in volatile memory can be lost, and the data is not backed up at the

source.

- *Large messages:* If the message size exceeds the available memory capacity, the packet reassembly cannot be performed at the intermediary nodes. This renders alternatives 2 and 3 unusable.
- *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, then the message transfers might be frequently aborted especially if the links are not always symmetrical. On top of these problems, PSFQ is also very inefficient, because route changes make the caches of previous nodes in the path unreachable, and end retransmissions must be performed.

These arguments are also briefly shown in this table I. As one can observe, all the alternatives prove unsuitable when faced with some of these challenges.

Taking into account the set of difficulties they alternatives are facing, we propose a few mechanisms addressing them. These ideas were inspired from the mechanisms underlying the Delay Tolerant Networking architecture for IP networks presented in [4], mainly because IP networks featuring disconnections and large delays face a similar set of issues as the ones presented above. These ideas are:

- *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 flaky 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 in networks with large roundtrip delays. 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. More detailed issues related to custody transfers are presented in section V.

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. The reliable transfer mechanism proposed in the previous section is based 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 custody transfer refers to the acknowledged delivery of a message from one DTNLite hop to the next and the corresponding passing of reliable delivery responsibility.

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. Other factors like the flakiness of connections, unreliability of the nodes and the absence of an any-to-any mulithop routing mechanism make reliable multi-hop transfer challenging. However various options are available, some of which are discussed in the previous sections. The available options include some of the solutions discussed in the section III and include multi-path sending, packet replication etc.
- *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 system. 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 basestation. This issue is discussed in more detail in section V
- *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.

Challenge	**Alternative 1**	**Alternative 2**	**Alternative 3**	**Alternative 4**
Description	Forwarding,end-to-end acks	Store and forward, selective nacks, no end-to-end acks	Store and forward, selective nacks, end-to-end acks	PSFQ
High roundtrip delay hours or days	Storage overflow at data source	Storage overflow at data source		Storage overflow at data source
Flaky Nodes or low duty cycles		unreliable, data loss		
Frequent disconnections	RAM overflow at node with disconnection	RAM overflow at node with disconnection	RAM overflow at node with disconnection	RAM overflow at node with disconnection
Message size $\geq$ memory size		impossible reassembly at intermediary nodes	impossible reassembly at intermediary nodes	
High mobility	Frequent aborts, restarts	Frequent aborts, restarts	Frequent aborts, restarts	Frequent aborts, and inefficient - caches at pervious nodes become unavailable

TABLE I

RELIABILITY MECHANISMS IN THE FACE OF 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 [4]. We have particularly targeted our implementation for data collection applications where there is a central basestation. 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 to any-to-any delivery.

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. In these conditions, each message is identified by a combination of the the source node id and a unique bundle id (or *token*) for the originating node.

Figure 1 shows presents the layering of the important components of the proposed architecture.

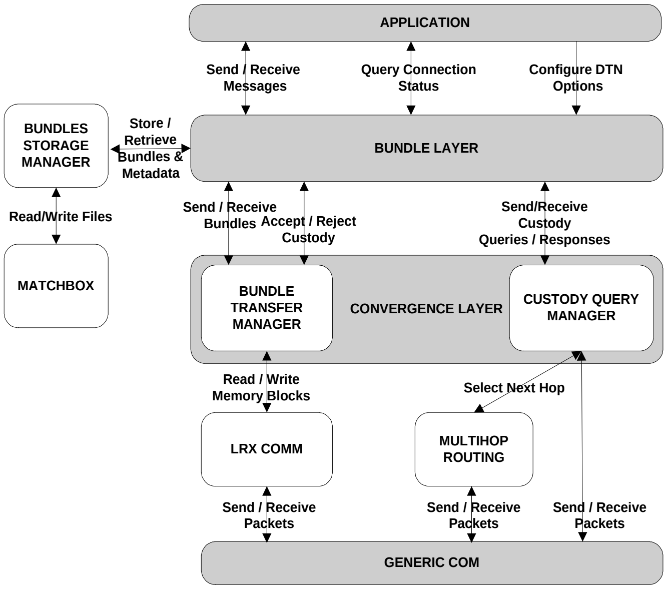

Fig. 1. DTNLite architecture

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([7]), 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. Consequently, 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. Currently, the allocation of memory for the bundles is done in the *Bundle Agent* itself, though this aspect might need to be modified if the bundle sizes exceed the available memory size. The interface it provides to the application is described in the following:

```
command token_t  sendBundle(DTNBundle *bundle);
event   result_t sendBundleDone(token_t token);
event   result_t receiveBundle(DTNBundle *bundle);
event   result_t connectionStatChanged(stat_t *st);
```

The connectionStatChanged event provides the application with information about changes in the network connectivity state.

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:

   a) *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.

   b) *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.

   The *Convergence Layer* provides the following interface to the *Bundle Agent*:

```
command result_t sendCustodyQuery(query_t *query);
event   result_t receiveCustodyQuery(query_t *query);

command result_t sendCustodyQueryResp(resp_t *resp);
event   result_t receiveCustodyQueryResp(resp_t *resp);

command result_t transferBundle(bundle_t *bnd);
event   result_t receiveBundle(bundle_t *bnd);

command result_t sendCustodyAck(ack_t *ack);
event   result_t receiveCustodyAck(ack_t *ack);
```

Table II 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 II

MAPPING BETWEEN COMPONENTS

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.

A. Custody transfer and duplicate creation

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

Let us illustrate the handshaking mechanism between a source node A and a destination B: A sends a message to B; when B receives the messages, it saves it to stable storage, it assumes custody and sends back an acknowledgment to A. Upon receiving the acknowledgment, A deletes the message from its storage and renounces its custody.

If our goal is reliability alone, the message exchange can already stop here: if the acknowledgment is successfully received by A, we have one owner, otherwise we have two owners, a duplicate being produced. However, there is another property we are interested in: the awareness of the participants in the transfer of the possibility of a duplicate being produced. More precisely, if only the message exchanges described above are performed, B can never estimate whether a duplication has happened or not. If, on the other hand, a final acknowledgment is sent from A to B, B will have more information about a possible duplication: if the final acknowledgment is successfully received, B knows it is the only owner, otherwise it knows it might share the ownership with A. This type of information can be useful when doing in-network aggregation, or when running duplication elimination algorithms.

1) Dealing with duplicates: Duplicates are problematic because they consume storage and bandwidth resources, both of them scarce in sensor networks. The extreme case when the ratio of duplicates to regular messages in the system keeps increasing until the normal network functioning is impaired must be prevented. In this sense, nodes must check for more than one copy of the same message, and eliminate them. Explicit duplicate elimination can be initiated at the destination, by broadcasting deletion packets containing lists of unique ids for messages already delivered.

However, in many cases, duplicates may pose a larger problem, because they hinder the in-network processing ability of the nodes. More precisely, *duplicate-sensitive* in-network operations, such as averaging, are problematic in the presence of duplicates, as opposed to *duplicate-insensitive* operations, such as computing the minimum value. This is one good use of information obtained during custody transfers regarding the duplication possibility: if a node is sure it has exclusive ownership of the message, it can safely use it for in-network aggregation.

2) Congestion and custody transfer: An additional problem related to custody transfer is the fact that once custody has been transferred from the source, the messages cannot be safely discarded. The only possible ways to reuse the storage allocated to these messages is either to transfer them to another custodian or to drop them after their expiration deadline passes. This makes the network very difficult to manage during congestion, when message dropping is extremely valuable. Since the only place messages can be dropped is at the source, care has to be taken when accepting new messages for transfer. A possible mechanism to alleviate this predicament involves keeping additional information with every message, that would help the intermediary custodians assess whether they are authorized to discard it. In this case, the intermediary node act as a placeholder for the source.

3) Choosing the next custody hop: In a given sensor network, some or all the nodes may act as potential custodians for messages. In effect the custodians form an overlay network on top of the underlying sensor network. The problem of finding the next custody hop is reduced to the problem of routing in the overlay network.

Choosing the next custody hop incurs specific issues that need to be addressed, and that make the routing metrics in the custodians overlay substantially different from the routing metrics used in the underlying network.

In today's sensornets, stable storage is usually provided by flash memory. 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 must be minimized, meaning that the number of custody transfers needs to be minimized.

Another concerns is related to network disconnections. If, due 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 make the following decisions:

- Choosing the best neighbor in the overlay to transfer custody to.
- After having decided on the best neighbor, establishing whether transferring custody to this node is worthy at this time.

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

- Minimizing the **overall energy consumption** in the network. 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. However, transferring the custody over a long underlying network path is not recommended for several reasons. Transferring custody at a distant node takes a significant amount of time, and because the network conditions are dynamic, the probability of the path between custody hops changing or ceasing to exist altogether is much higher. Thus, failure to transfer custody happens more often if done over longer underlying routes. Moreover, for the case where packet retransmissions are not performed at each hop, the number of end-to-end retransmissions increases with path length. Thus, when optimizing for energy consumption, we need to balance between minimizing the number of custody transfers and minimizing number of retrials and retransmissions.
- 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. These two optimizations are very closely related. 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. If the storage requirement surpasses the available storage, the arrival rate cannot be maintained, and packets must be dropped. This means either losing data altogether, or compressing data and diluting information quality. Moreover, storage usage is not uniformly distributed throughout the network, making the problem worse. Consequently, especially for application with high sampling rates, delivery delay must be minimized in order to minimize data loss.

Having identified candidate network metrics to optimize, let us establish what type of routing information would enable us to achieve these optimizations. Usually global routing information is preferable to local information. However, considering the following factors, we choose to employ local information:

- Global information is expensive to maintain due to limited storage.
- Global information is expensive and slow to deliver, generating additional protocol overhead.
- Due to dynamic network changes and high mobility, global information presents an outdated view of the system, not very useful in assessing the appropriate routing decisions.

We propose simple schemes, relying on local self-assessments, for routing decision making. 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

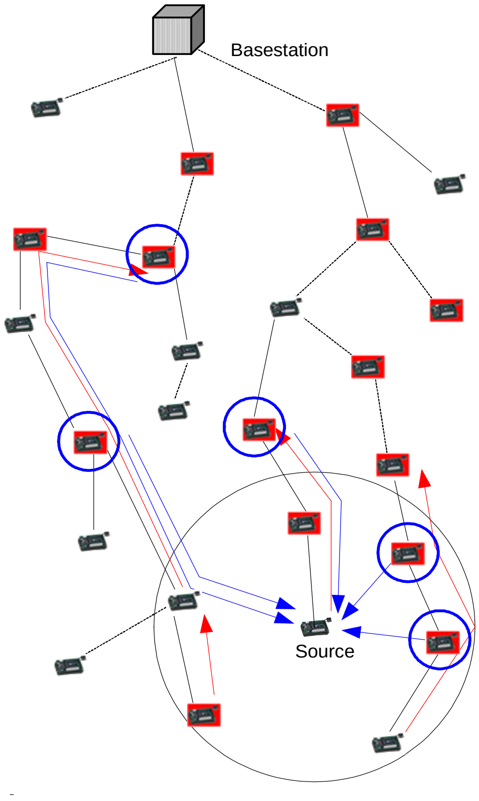

Fig. 2. Example Custody Query

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.

B. 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.

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. Sending the query can proceed 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 by Kim in [3].

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 paths of the query packets are represented by red arrows. The red nodes are the ones that can accept custody, and among those, the ones circled with blue are the ones that have better metrics than the source, and consequently represent candidates for the next custody hop. Candidates respond to the query; the response packets are represented as blue arrows arrows.

2) Metric estimates: present the mechanisms used to maintain local estimates and link quality 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, as defined in section V-A.3, 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 favor the first approach. Thus, the average delivery time of a node(Avg_delay) is updated every time a custody transfer is completed:

$$delay_{new} = Avg\_delay_{next\_hop} + Avg\_waiting\_time_{local}$$

$$Avg\_delay_{local} = \alpha \times Avg\_delay_{local} + (1-\alpha) \times Delay_{new} \quad (2)$$

$Avg\_waiting\_time_{local}$ is the average time spent by messages at the current node.

Very similarly, the average energy consumption(Avg_EC) is updated as follows:

$$EC_{new} = Avg\_E_{transfer} + Avg\_E_{store} + Avg\_EC_{next\_hop}$$

$$Avg\_EC_{local} = \alpha \times Avg\_EC_{local} + (1-\alpha) \times EC_{new} \quad (3)$$

where $Avg\_E_{transfer}$ and Avg_E_{store} are the average energies required to transfer and store a message respectively at the current node.

VI. Evaluation of Custody Transfer Mechanisms

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

1) *Unicast:* The query is sent along the routing tree up to the destination.
2) *Multicast:* 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 Hop:* This policy chooses the very nearest node among the nodes that are ready to accept custody transfer. It is expected to lead to more custody transfers than necessary, and as a result to consume more energy.
2) *Farthest Hop:* This policy chooses as the next custody hop the node that is farthest away, i.e the node that is closest to the destination. While this policy minimizes the number of custody transfers performed, each transfer is expected to take longer, especially in flaky networks where the probability of a reliable transfer decreases with distance.
3) *Most Energy Level:* This policy chooses the node with the most energy to be the next custody hop . The policy is expected to maximize the overall lifetime of the network by minimizing the variations in the energy levels of the nodes
4) *Least Average Delay:* Every node maintains an estimate of the average delivery delay, and the node having the smallest delay is chosen.
5) *Least Average Energy Consumption:* The policy chooses the node that on average consumes the least energy for sending a message to the destination.
   The following table presents the symbols used to represent the choices mentioned above on the evaluation graphs.

Term/heading	Description
MULTICAST	Use multicast for queries
UNICAST	Use unicast for queries
FARTHEST	Node closest to base
NEAREST	Node closest to source
ENERGY_LEV	Node with most energy
AVG_DELAY	Node with least delay to base
ENERGY_CONS	Node with least avg. energy per message to base

TABLE III

Legend for Graphs

A. Simulation Setup

We are using a discrete event based simulator that was developed for simulating routing algorithms for networks with scheduled disconnections. We made the appropriate modifications necessary 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 length of the link. The radio collision model is simplified in the sense that collisions can occur only at nodes - packets collide only if they are destined for the same node. The flood level for the multicast queries is set at three i.e the queries are flooded for three levels and are then forwarded to the base along the routing tree.

Parameter	Range in simulation	Range in Mica motes
Time	1	1s
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 IV

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 various custody selection policies are modeled using the approaches mentioned in the last 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 IV.

B. Evaluation

We compare the different metrics by considering a number of parameters. We perform all experiments for both unicast and multicast queries. However, since all the policies have very similar behaviors with unicast queries, we mostly present the results for the multicast queries.

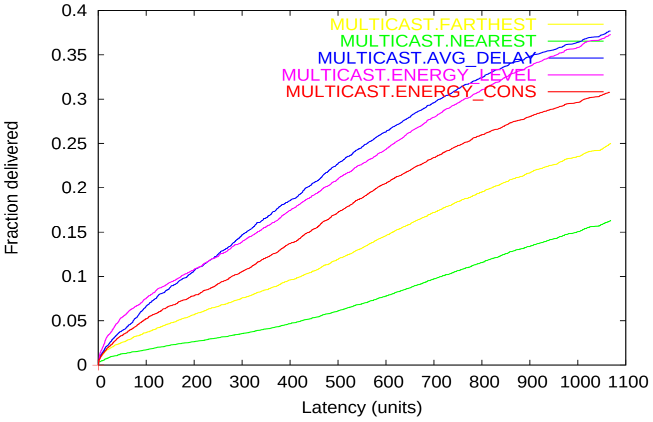

Fig. 3. Cumulative delivery time

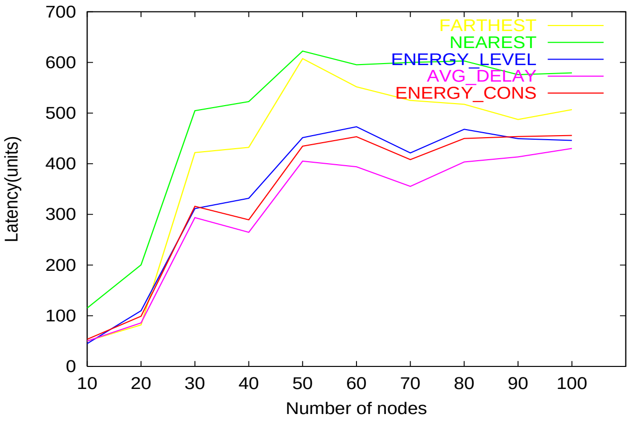

Fig. 4. Average latency for message delivery

1) Delivery Time: The first set of graphs compare the different custody transfer policies with respect to the delivery time of messages. Figure 3 shows cumulative delivery time of messages for a network of 100 nodes. 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 simulation interval. Not surprisingly the *NEAREST* and the *FARTHEST* policies have the worst average latency, also illustrated in figure 4.

Figure 5 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 sink and suggests that the data generation rates should be lower for reliable delivery.

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. Figure 6 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 figure 5, it sacrifices the delivery ratio.

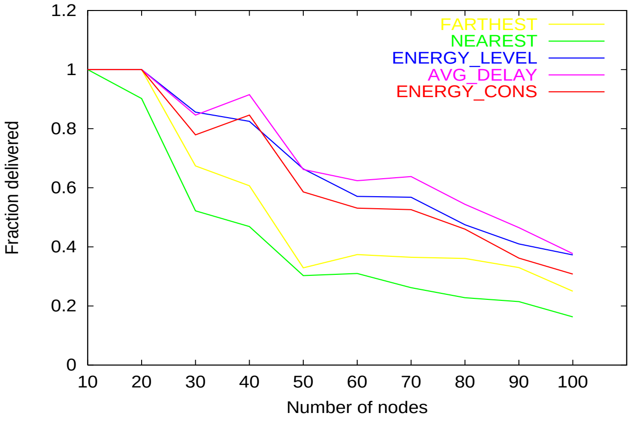

Fig. 5. Fraction of messages delivered

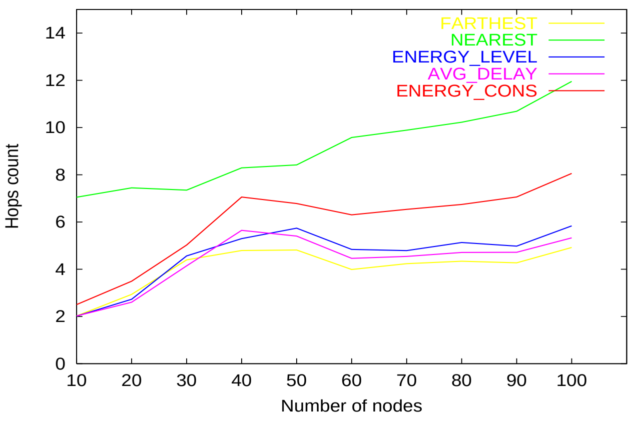

Fig. 6. Custody hops per message

Figure 7 plots the average number of network (not custody!) hops needed for each successful message. Here we see that the *NEAREST* policy uses the maximum number of network hops for delivering a message.

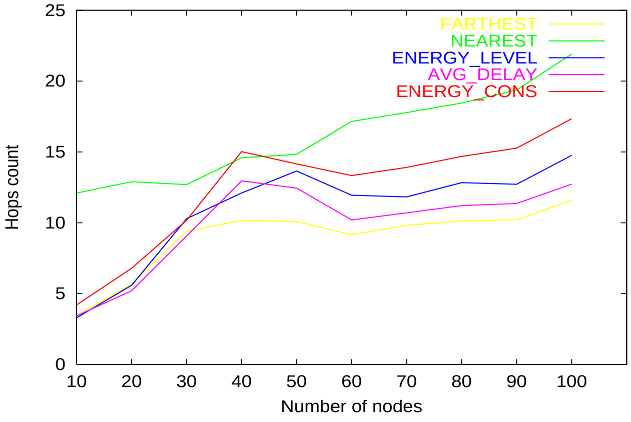

Fig. 7. Networks hops per message

3) Energy: In this experiment we compare the energy efficiency of different policies. Figure 8 shows the average node energy level over time for all the policies, while figure 9 shows the standard deviation in the 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 non-volatile storage), it has far worse delivery performance. However the *ENERGY_LEVEL* policy is able to equalize the energy level differences among the nodes.

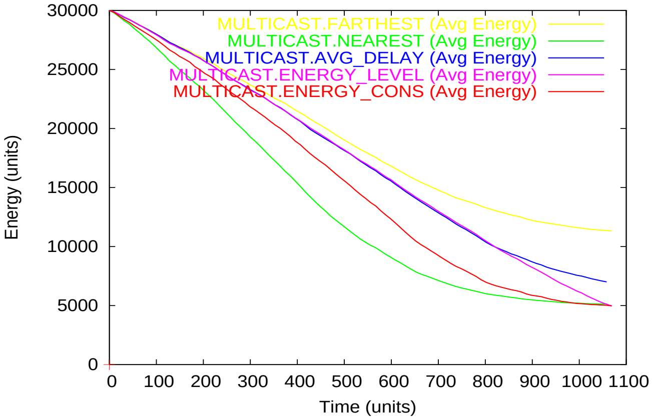

Fig. 8. Average Node energy level over time(nodes=100)

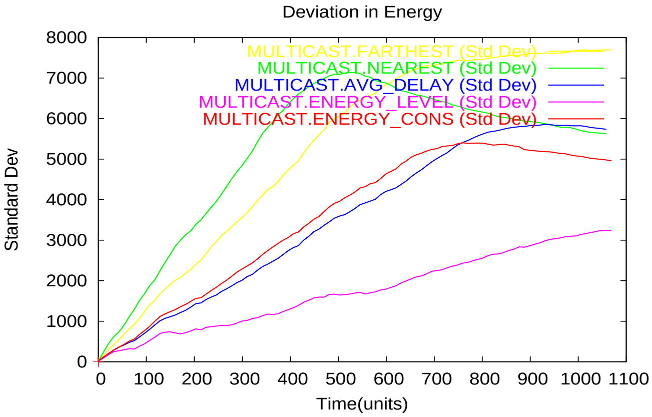

Fig. 9. Standard deviation of energy level with time(nodes=100)

Finally, figures 10 and 11 compare the average energy spent per successful message and per custody transfer respectively. Here we see that there is little to choose from between the policies, except for the ones that use either short or long custody hops.

C. Results

We can conclude from the above results that the selection policies based on energy are able to minimize the energy usage variation among the nodes while those based on delay are able to minimize latency of message delivery. This implies that variation in the link qualities can be exploited even while using purely local information and estimates. However, since no policy was able to satisfactorily optimize both energy and delay, we need to devise better metrics that can combine energy and delay.

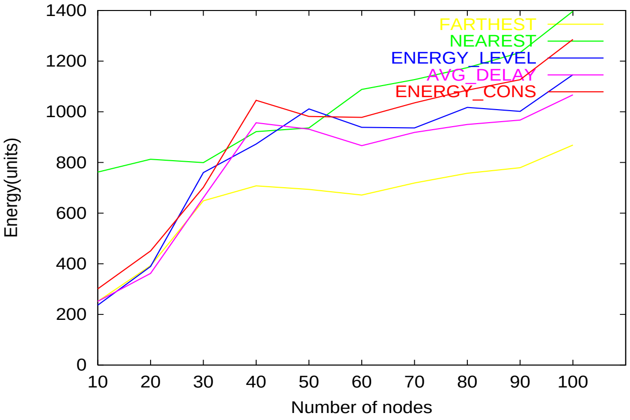

Fig. 10. Energy per message

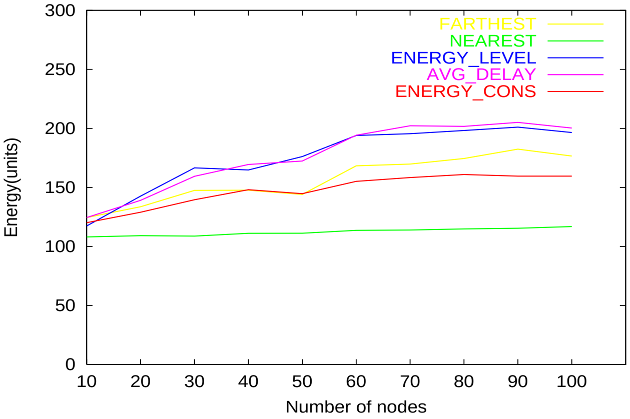

Fig. 11. Energy per custody transfer

D. Simulation shortcomings

Our aim for the above evaluation was to study custody transfer mechanisms at a high level. The simulation results are optimistic, because they do not model the query mechanisms for selecting custody hops at packet level. However, modelling the delay and overhead implied by the queries might not alter the relative differences among the mechanisms. The radio collision model in the simulation is very simplistic as well. In the future we wish to modify our simulator to rectify these issues and to obtain more realistic experimental results.

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.

We separately simulated custody transfer and evaluated the performance of various selection criteria based on energy and delay. The key conclusion of the evaluation was that it is possible to optimize for a particular objective like energy or delay while using purely local information. This means that the different selection policies were able to successfully exploit the network asymmetries in connectivity and resources. However, we realize the need for better selection policies that can optimize for both energy and delay.

In the future, we intend to perform a comparative evaluation of our architecture, relying on the custody transfer mechanism, with other reliable transfer protocols that have been proposed (II). We plan to perform extensive testing of the DTNLite implementation for TinyOS on Mica motes as well as using TOSSIM, in order to measure the overhead and performance of the custody transfer mechanism. Finally, there are plenty of other issues in reliable custody transfer, such as routing, duplicate detection, interaction with application-specific compression and aggregation of data that need to be explored in more detail.

ACKNOWLEDGMENT

We would like to thank Professor David Culler, the class instructor, and Kevin Fall, for their guidance and insightful advice. We also thank Sukun Kim for letting us use his LRX data transfer protocol.

REFERENCES

[1] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, “PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks,” *First Workshop on Sensor Networks and Applications (WSNA), September 28, 2002, Atlanta, GA*.

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

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

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

[5] K. F. et al., “DTN Implementation,” *http://www.dtnrg.org*, 2003.

[6] P.Juang, H.Oki, Y.Wang, M.Martonosi, L.S.Peh, and D.Rubenstein, “Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet,” *Proceedings of ASPLOS-X, San Jose, October 2002*.

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