Hyper: A Routing Protocol To Support Mobile Users of Sensor Networks

ABSTRACT

Wireless sensor networks for environmental monitoring promise to be a rich source of ecological, biological, and meteorological data. However, current systems largely return data to a central location for offline analysis, and do not support access by mobile users in the instrumented environment. In many environmental monitoring applications, it is critical to support users in the field so that they can correlate manual observations with the sensor network data, engage in system topology adjustments and calibration tasks, and perform system management. However, it is critical that such mobile users do not interfere with the regular data collection functions of deployed systems.

One of the critical systems functions needed to support mobile users of wireless sensor networks is routing. In this paper we identify key mobility usage scenarios and present Hyper, a routing layer that enables efficient and reliable data collection for both static and mobile users.

1 INTRODUCTION

Wireless sensor networks are an emerging technology with broad applicability in a variety of science, engineering, civil and military applications. For the most part, systems research to date has focused on mechanisms to create robust self-configuring and energy efficient systems. These systems have been designed to be taskable [10, 14] but the usage model generally has been of a human sitting in a remote location to both task behavior and analyze the data.

As we have gained experience with prototype wireless sensor networks in environmental monitoring applications, we have recognized the need for these systems to also serve interactive users in the field. This interactive capability arises at all stages of the system life cycle, from design debugging, to deployment testing, and ongoing system health maintenance, as well as for data visualization and analysis in the field in order to support the collection of and correlation with manual observations. At the same time it is critical to minimize the impact of such interactive use with the ongoing activities with which the system is tasked.

The two system requirements that emerge from this interactive, in-the-field, usage scenario are the need for a data tasking and routing architecture that supports mobile and transient queriers, as well as a host of data analysis and visualization tools to assist the scientist. This paper describes the design and performance of the system support, and we leave the discussion of data tools to future work. We will use examples from our experiences deploying and using WSNs for environmental monitoring at Sensor Mountain, however we are confident that these techniques have broad applicability.

The system, which we call Hyper, has the following features key to supporting mobility: fast neighborhood assessment and efficienct tree convergence to offer a mobile user low latency access to a network; multiple collection tree support for concurrent use by several mobile users and a fixed collection microserver; a scheme for opportunistic use of high-bandwidth backchannels such as by 802.11; a link transmission policy that infers disconnection and reduces needless radio use; and a storage system for delay tolerant transmission (when routes temporarily are bad) and support of lonely mote clouds (when routes intentionally do not exist for long periods until mobile users connect to them).

MobiHoc 2006, May 22–25, 2006, Florence, Italy.
Copyright 2005 ACM X-XXXXX-XX-X/XX/XX . . . $5.00.

2 DESIGN: ENABLING MOBILITY

Low-power networks typically trade responsiveness for efficiency. The three sensor networks at Sensor Mountain totalling about 50 motes and four microservers, communicate at a very low duty cycle using the Mica2's low-power, onboard CC1000 radio [2]. Since links typically span only 30m to 80m, a multihop collection tree is formed to transport data to a microserver beyond this range. When deploying a new node, or fixing an old one, a technician must be patient. Motes operate at a low duty cycle, so it may take a significant amount of time (approximately five minutes) in order for a mote to evaluate its neighborhood, establish link quality estimates and join the network.

Each microserver is equipped with two radios, the CC1000 for communicating with the motes, and a standard 802.11 radio for high-bandwidth transmissions to other microservers and to the Internet at large. As a result of the latter, some of the reserve is blanketed in 802.11. However due to dense foliage and other obstacles there are many locations where it is not available.

A technician tasked with deploying a new sensor node must verify that the new node has end-to-end connectivity to a collection sink. From the field, this can be determined using an 802.11 backchannel connection from the technician's laptop directly to the sink or, when such connectivity is not available, by walkie talkie. In the worst case when there is no backchannel connection of any sort (either via 802.11 or walkie talkie), then the researcher can manually infer connectivity by snooping the mote radio channel for control and data packets to verify that at least the mote's communication with its one hop neighbors seems correct. This is time consuming and cumbersome.

This situation is equally challenging to the scientist that goes to the field to make observations. Scientists want to view data while in the field in order to verify the accuracy of that data, experiment with stimulation and response, and to annotate the data with their own observations. Using existing techniques, a user must rely on 802.11 to the network's collection sink for these purposes. Current routing implementations do not support multiple collection sinks, so establishing even a temporary connection directly to a sensor node will break any previous route that node may have established [2, 19].

Furthermore, to make such observations, a scientist might bring new sensors to the field. Again though, to activate a new sensor node even for a quick ad hoc measurement, the application scientist will need to wait minutes for it to join the network, and again can only verify end-to-end connectivity via 802.11 or voice radio. In addition, it may be neccessary to couch the ad hoc measurements in the context of the surrounding sensors.

For both scientific exploration and basic maintenence an ideal routing scheme for Sensor Mountain would allow new nodes to join a network quickly (so that a user doesn't have to stand and wait), would support multiple independent routing trees simultaneously for networks with concurrent fixed and mobile users, and opportunistically would use 802.11 to connect application scientists to the backend for retrieval of current and historical data. In addition it would allow application scientists to go into the field, get quick access to sensor data services, deploy new sensors and get results from them quickly, and even perform calibration experiments in realtime. And it would do all this while burdening the network as little as possible.

In the remainder of this section we discuss the mobility usage scenario we intend to support, which is based upon our own experience deploying and maintaining sensor networks and feedback from application scientists. We then describe a new routing protocol, Hyper, that addresses the challenges of supporting the mobile user efficiently, including fast neighborhood assessment and tree construction, multiple tree support, quality aware retransmission, opportunistic use of high-bandwidth backchannels, and disconnection tolerance.

2.1 Mobility Usage Scenario

From our experience deploying real systems at Sensor Mountain and biologists' experience using them, we've observed the following dominant scenario for mobile users:

- **Users remain in each location for several minutes.** Technicians go to a location to add a new node, add new sensors, or change batteries. Users go to a location to verify data, annotate data, or to directly monitor stimulus/response. These tasks are all short-lived relative to the duty-cycled, latency-insensitive data gathering style of many existing systems. The chief concern in supporting this sort of behavior is in providing a connection to the mote network as quickly as possible.

- **Even when moving, users rarely switch transmission domains.** Radio links are on the order of 50m. After working with a node a user typically moves to a neighboring node.

- **Connectivity is not necessary during periods of mobility.** In-field users interact with nodes while seated near those nodes. The only reason for movement is to relocate near other nodes. Disconnections caused by large movements are not problematic or undesirable because sensor data services aren't typically needed during transition. Data that is stuck in-network due to the user moving somewhere else can either be dropped in-network, or be stored until a connection is re-established, depending on the user's tolerance for delay. This aspect of the usage scenario is very different from cellular connectivity or mobile IP, where the technical challenge is in maintaining a connection that provides timely delivery through a series of hand-offs and and where disruptions in the connection are always undesirable.

- **A network often has a dedicated and stationary microserver that collects data from it.** This microserver stores the data it receives into a database and provides access to that database via 802.11 and the Internet. A mobile user with 802.11 connectivity may access this database for current and historical sensor information.

- **Disconnected operation of sensor nodes is a common case.** Some deployment sites are so remote that it is not cost effective or even possible to construct a multiple-hop routing tree to a permanent microserver-class collection sink. For these *lonely cloud* deployments, sensor nodes must store the information they collect until a user acting as a *data mule* can get close enough to them to establish network connectivity and offload that data.

2.2 Routing Trees for Mobile Users

A mobile user is particularly latency sensitive; thus establishing a connection quickly is important. In the case of connection via 802.11, Hyper relies upon standard techniques such DHCP, and thus its latency performace is governed by their parameters. To connect by mote radio, Hyper provides three services designed to set up an efficient and unobtrusive connection with low latency: fast neighborhood evaluation, fast tree formation, and routing support for multiple sinks.

2.2.1 Fast Neighborhood Assessment

Two techniques for link assessment are employed widely in sensor network deployments: direct measurement over a window of time of the fraction of transmission attempts that are successful; and indirect inference of this probability of success using a radio's link quality indicator (*lqi*) field [1]. Both techniques rely on transmission and reception of radio packets. The former technique requires four to eight packets, depending on the desired accuracy, to generate a good estimate of the probability of successful transmission; the latter technique requires a single packet.

The problem is that to save energy, deployed nodes generate data packets or special link estimation beacons at a period on the order of minutes. Thus, a user that arrives at a site and wants access to sensor data services must wait several minutes to establish a connection.

The fix to this problem is fairly straightforward, but is not available in any sensor network routing protocols that we know of: generate one or several link assessment packets immediately after activating a new node.

Our routing protocol uses an expected number of transmissions (*etx* [6]) as its cost metric. For a link, $\overleftrightarrow{ab}$, *etx* factors the probability of successful transmission in each direction ($\overrightarrow{ab}$ and $\overrightarrow{ba}$) into the cost: $etx_{\overleftrightarrow{ab}} = \frac{1}{p_{\overleftrightarrow{ab}}} = \frac{1}{(p_{\overrightarrow{ab}})(p_{\overrightarrow{ba}})}$, where $p_{\overrightarrow{ab}}$ is the probability that a packet sent by a will be correctly received by b and $p_{\overleftrightarrow{ab}}$ is the probability that a packet will be both correctly received and acknowledged in a single try. Hence, to establish an *etx* estimate, not only must a new node generate link assessment packets, but its neighbors must also generate them.

When a mobile sink arrives at a new location, it initiates neighborhood evaluation by sending out a series of link beacon packets, each of which causes one-hop neighbors to respond with a packet. The mobile sink and its one hop neighbors can estimate the quality of the links that connect them after only a few iterations of this "call and respond."

The CC1000 radio on the mica2 nodes deployed at Sensor Mountain does not provide an *lqi* field. Thus for these deployments the probability of success, $p_{\overrightarrow{ab}}$ is measured directly using counts of the number of packets transmitted and the number of packets correctly received: $p_{\overrightarrow{ab}} = \frac{rxCount_{\overrightarrow{ab}}}{txCount_{\overrightarrow{ab}}}$. $txCount_{\overrightarrow{ab}}$ is inferred by the receiver using a sequence number embedded in each packet; this number increments before each transmission attempt.

Hyper's fast link assessment protocol works as follows:

- Periodically, each node transmits a Beacon Packet (BP) containing an array of the measured ingress link qualities of its neighbors. Nodes exchange BPs to establish bidirectional link quality.

- The mobile user sends out a Fast Beacon Packet (FastBP), which is identical (except for the content of its *type* field) to a BP; the difference is that any node that hears a FastBP is expected to respond with a BP immediately. Initially the sink's FastBP is empty since it has no ingress quality estimates.

- The mobile user sends out a total of *N* FastBPs and waits for *S* seconds between each FastBP in order to receive responses. Neighbors each respond with *N* BPs. *N* and *S* are parameters that should be tuned; increasing *N* improves accuracy; increasing *S* provides extra time to alleviate contention in dense neighborhoods. BPs and FastBPs serve a dual role; they may be used for estimating ingress quality and for advertising it.

- Once neighborhood connectivity is assessed, the link layer alerts the routing software that the fast link convergence stage has completed.

When a mote sends out a FastBP, it forces all nodes who heard it to respond. It takes about five seconds for this fast neighborhood evaluation and link quality estimation to converge. We anticipate that fast neighborhood discovery will be used infrequently (compared to the lifetime of the network) so the brief flurry of messages should not impact network lifetime significantly. The protocol is designed to operate over a MAC with carrier sensing and backoff to reduce contention. At high neighborhood densities, however, contention will result in packet loss and thus associated link cost measurements will be artificially increased. Our protocol does not yet address this problem; the neighborhood sizes in our deployments tend to be small enough to avoid contention. However, one simple solution, which we leave for future work, would be for nodes to randomize the send time of their responses over a range of packet times equal to the neighborhood size.

2.2.2 Neighborhood Evaluation and Tree Construction

Once neighborhood connectivity is established and evaluated, Hyper builds a collection tree. The time it takes for tree convergence to occur is proportional to the depth of tree desired, but for most trees it takes only about a second.

The tree routing algorithm forms a tree with minimum path costs. Path cost is the sum of the individual link costs that make up a path. Thus, an efficient tree can only be constructed if a good estimate of link cost can be established.

For efficiency, the tree formation algorithm defers evaluating a routing control packet for a time proportional to the cost advertised within it. In this manner, useful control packets advertising good routes will be considered before those that advertise bad routes. The consequence is that radio use will be decreased; control packets advertising poorer routes will not propagate.

The sink initiates tree formation by transmitting a tree update message. Tree update messages contain the transmitter's address (to be used as the parent in the routing tree), the address of the root, the path cost, an epoch number to distinguish new updates from old ones), a TTL used to limit the depth of the tree, and a time indicating when to expect the next update.

A node that receives a tree update message waits to evaluate it for an amount of time proportional to the cost of the link over which the update came. It sets a "wait timer" for this purpose. If before the wait time expires another update arrives (from a different node, but pertaining to the same sink) then the two are compared in terms of the path qualities they offer; the update message containing the better path is retained. Once the wait timer expires, a node broadcasts its own update message containing its best path cost to the sink.

Good paths propagate quickly, while poor paths propagate slowly, or not at all. In the event that no update packets are lost, and each node's link estimates are accurate then the tree that is built will be a minimum spanning tree, and each node will only send out one update message.

Using cost-dependent delays is an optimization made popular by SRM [7]. A tree with the same path qualities could be formed without this optimization, but would be less energy-efficient, as more messages would be transmitted. Without the optimization, a node would receive updates from its neighbors and process them in an order that has little relation to the qualities of the paths they advertise. In the worst case—when paths are processed from worst to best—kN messages will be sent by a network of size N with neigborhood size k. Using the control delay technique, this is reduced to N.

A route update message contains a field indicating the current epoch duration before which the sink will initiate the process of reforming its tree. The duration may be varied in accordance with application requirements and environmental factors; the field propagates this information to the network. Mote routing code uses this field to determine if it has completely missed control messages from one or more routing epochs. By default, if no routing update is received during three such epochs, the tree is considered dead, and is removed from the node's routing table.

Anything that prevents update messages from being successfully received throughout the network (such as aggressively high-rate data traffic or temporarily poor connectivity) may result in tree destruction. Hyper provides a locking mechanism as an added precaution. When the application knows more about the network state than what can be inferred by the timeout mechanism, it may prevent route teardown by requesting a lock on a particular route's state. This lock is functionally equivalent to setting the tree timeout to infinity.

The control traffic overhead of maintaining a tree is one packet per node per update period. The longer the update period the smaller the impact of control overhead on network lifetime. However, a longer update period also increases the amount of time, on average, it will take to fix a route when a link truly does break (such as when a node fails, moves, or when there are significant and sudden environmental changes). To ensure that data is not lost, packets are queued in a combination of volatile and non-volatile memory (discussed in Section 2.3.2).

The network layer supports unicast communication from a mote to a microserver and vice versa, and tree-oriented communication between all motes and a microserver as these are the most common communication patterns in sensor networks [8]. It does not support efficiently one-to-one communication from one mote to another or many-to-one communication from a subset of motes to the microserver. To achieve the former, messages may be forwarded via the microserver. To accomplish the latter, a microserver may send packets to all motes in the network; on the motes, logic operating above routing may filter packets by destination group so that only a desired subset responds.

In Hyper, route formation and maintenence is initiated by the root, path costs are strictly increasing with depth, and each node keeps track only of its parent for each tree to which it belongs. As a result, loops cannot form. This is in contrast to MintRoute, the previous state of the art in sensor network collection tree routing, and other distance vector protocols that must implement sophisticated loop detection and repair algorithms. Unlike other loop free protocols such as the Centroute [18], Hyper avoids loops without incurring the overhead of adding complete source routes to each packet at the expense of maintaining routing and neighbor state at each node.

2.2.3 Multi-Tree Support

There are potentially multiple simultaneous, independent users that want access to the network; Hyper thus supports multiple concurrent sinks. However the number of sinks is expected to be relatively small compared to the number of data source nodes.

In the network, each tree is maintained independently. Therefore, multiple trees is supported by a node by expanding the management state for a single tree into an array of state. It is expected that each node will connect only to a few routing trees, thus limiting the state recorded in a mote's RAM.

this increase in state does not consume too significant a portion of the mote's limited RAM.

Each state structure contains the ID of the root node, the next hop along the path to the root (i.e. the parent), and the route cost. The structure also maintains an indication of whether the route is active so as to hide trees from the application that are in the process of forming, the time when the route will next be advertised, and a time after which time tree should be dismantled if no control messages are received in the interim.

Since each tree is managed independently, the update messages for more than one tree are never aggregated into a single packet. Although doing so would reduce routing control overhead, it would complicate tree formation. First, combining messages would interfere with the route construction protocol's reliance on delays. Hyper's timing property that reduces the number of control messages each node sends would no longer hold. For this reason, combining control messages would either lead to building inefficient trees (and hence, spending more energy on collecting data) or potentially could cause nodes to send many more control messages, which again wastes energy. Second, it might lead to routing loops and other persistent incorrect routing state.

Transport over an alternate routing structure such as an any-to-any multicast tree might reduce redundant transmissions when multiple users request the same data. However, we determined that forming this sort of routing structure, determining which data requests are shared, and scheduling multicast transmissions was too complicated and error prone for resource-constrained mote networks with severely limited debugging visibility.

Therefore, Hyper employs a different sort of optimization. When sinks request the same data and when there is 802.11 connectivity, Hyper determines whether a data set is already available from a sink with 802.11 connectivity before it tries to form a tree. A longer discussion of this optimization is provided in Section 2.3.1.

Even when 802.11 is not available, the cost of redundant transmission of data is not that great. First, The number of concurrent sinks in the network is low (usually no more than two) so data will not be duplicated more than a few times. Second, it is rarely the case that two independent mobile users will require data sets with significant overlap. Third, users without 802.11 connectivity to each other are usually geographically separated. Thus even when they request the same data, there would be little opportunity to share a transport path.

2.2.4 *Ad Hoc Sensor Deployment and Node Rebirth*

Sensor nodes are activated by researchers and technicians alike. Application scientists may want to temporarily augment a sensor deployment with their own sensors—for evaluation of these sensors, investigations into cross-modality correlation, or for collection from sophisticated sensors that are too sensitive or require too much energy to deploy permanently. Technicians change batteries in existing nodes.

Both types of users must verify that their new sensors are communicating properly and both are sensitive to long delays in this process. Rather than forcing users to wait until the start of the next route construction epochs of each sink in the network, Hyper provides a *fast join* mechanism that allows a new sensor node to graft onto existing routing trees immediately.

A new node first invokes the fast link convergence protocol to determine to which of its neighbors it is best connected. A "Route Graft" message is then sent to this neighbor. In reply, the neighbor sends a "route reply," which contains the IDs of all roots that it knows about, and the timeout for each of the roots. The new node adds this information to its routing state, marking the replying neighbor as its parent for all trees. The new node then sends a message to each of its sinks to indicate that it is alive. Each root may then initiate a full tree reformation for improved efficiency if it deems it necessary.

The goal of the fast join protocol is to get a node connected to the network as fast as possible. The protocol does not necessarily form the most efficient routes possible from a new node to its sinks. Routes will, however, be improved at the beginnings of subsequent tree building epochs.

2.3 Efficient Data Collection

Data collection must be efficient, whether to a static node or mobile one. We employ several mechanisms in conjunction with Hyper's tree routing protocol to provide more efficient operation for both; of these, one mechanism addresses challenges specific to mobile users. These features include opportunistic use of an 802.11 backchannel, a protocol for quickly inferring disconnection and reacting to it, and mechanistic support for mobile interaction with a cloud of intentionally disconnected motes.

2.3.1 *Opportunistic Use of a Backchannel*

When a user wants access to sensor data services, 802.11 and mote radio connectivity are assessed. Sensor Mountain has significant 802.11 coverage, but there are some regions where it does not reach.[1]

There are several reasons to use Wifi when available. First, 802.11 connectivity may simplify the process of data acquisition for the mobile node. Rather than contend with mote link assessment and route establishment, a mobile node may connect to an established collection sink and collect sensor data by proxy. In doing so, the mobile node makes use of existing IP services such as ssh. Second, historical data may be available only by 802.11. An established sink may log the data it receives to a database to be later retrieved. Third, an established sink may already be collecting the data a mobile node needs. This notion was first suggested in Section 2.2.3. Finally, forming a collection tree can put undue strain on a network, particular one that is bandwidth limited. It will also consume energy, both in tree setup and during data collection. Using a backchannel to access sensor network services reduces the number of routing control messages that motes must process. Since each new collection tree would generate its own control traffic, using a microserver that already maintains a tree for background collection as a proxy will save energy and reduce network strain.

It usually makes sense for a mobile node to use an 802.11 backchannel to an established sink. There are several reasons, however, to use a mote collection tree directly instead of 802.11. In terms of the energy consumption of the network, it might be more efficient to form a new tree than to use an existing one. This is the case when the data source is much closer to the mobile user than to the fixed collection sink and the data rate is significant.

Cosider a network with two microserver-class devices: an established collection sink (S) and a mobile node (MN). Let m_i, $i = 1, ..., k$ be the set of sensors that produce data that MN needs, and let $cost(m_i, X)$ be the energy expended in routing a data packet directly from mote m_i to microserver X. When S is not already col-

[1]The ubiquotous presence of Wifi on traditional computing platforms suggests that it will be available to many future sensor network deployments as well.

lecting data that *MN* needs, it is more efficient to form a collection tree rooted at *MN* than to use a tree established to *S* when: $t(\sum_{i=1}^{k} cost(m_i, MN)) + RouteCtrl_{MN} < t(\sum_{i=1}^{k} cost(m_i, S))$, where t is the number of data packets that need to be collected each route formation epoch and $RouteCtrl_{MN}$ is the cost to maintain a routing tree rooted at *MN* for one epoch. As $t \rightarrow \infty$, the relative contribution of $RouteCtrl_{MN}$ becomes negligible. Cost correlates to network distance, suggesting that it will more often be reasonable for *MN* to establish a collection sink to close-by nodes than to ones far away.

Even so, sometimes it is reasonable to forsake efficiency for simplicity. This is particularly the case for short-lived data requests. Even if forming a tree is more efficient than using an efficient one the savings may be slight and will be outweighed by the benefits of using an unobtrusive and robust backchannel.

2.3.2 *Disconnection Assessment*

To provide reliable data delivery we couple persistent storage to a stop-and-wait radio link acknowledgement scheme. The former is used to queue packets while a node is disconnected from the network. The latter is used to overcome radio transmission errors. This section discusses how Hyper uses this mechanism, and in particular, how it uses its retransmission policy to differentiate disconnection from short-lived bad luck.

Transient disconnections happen suddenly and without warning. They occur alike on links that have been characterized as good and bad although bad links experience them more often.[5] Longer lasting link failures can be caused by environmental factors (e.g. bad weather), node death, and mobility.

Broken links lead to broken routes. Broken routes are resolved by probing the link to detect if it has recovered, and alternatively by constructing a new route that doesn't use the bad link. A faulty link is probed to see if the packet crosses the link successfully. Success causes the route to be activated again. If a route update arrives, regardless of what neighbor it was received from, then the route is considered to be activated again.

Hyper will retransmit a packet some number of times. If it is never acknowledged, disconnection (or extremely poor link quality) will be inferred, and the packet will be queued for later transmission when conditions improve. Acknowledgments, retransmissions, and flash usage all increase the energy consumption of the node. Since both the energy required to store a packet in flash and to transmit it over the radio are great with respect to other mote operations, reducing unnecessary transmissions and storage transactions will save energy.

When a packet destined for a sink fails to traverse a link, it gets queued for later transmission. Until the route is fixed, all packets that must cross the link to reach their destination are similarly queued. The queue is made up of a volatile queue (in RAM) and a non-volatile queue (in flash). The volatile queue is small relative to the size of the non-volatile queue, and is helpful during small periods of disconnection to avoid costly writes to flash.

The scheme we employ uses *etx* as an indicator of how hard a radio should try to deliver a packet before giving up. Hyper adjusts the number of transmission attempts it will make, $k_{\overleftrightarrow{ab}}$, in proportion to the number of transmission attempts it should take, $etx_{\overleftrightarrow{ab}}$. Previous work suggessts that *etx* is a good metric to use for link cost. (In Section 3.1, we verify this empirically and show that *etx* accurately estimates the number of transmissions that will be required to communicate a packet.) If a packet is not successfully transmitted after $k_{\overleftrightarrow{ab}}$ transmissions, disconnection is assumed. This has two consequences. First, less energy will be wasted retransmitting packets over a broken link. Second, energy will not be wasted by prematurely queuing a packet that would have had a good chance of being transmitted successfully.

To understand why a variable retransmission scheme improves performance, consider two extreme cases: a fixed retransmission scheme that makes exactly one transmission attempt before assuming disconnection ($k_{\overleftrightarrow{xy}} = 1, \forall \overleftrightarrow{xy}$), and one that makes an infinite number of attempts ($k_{\overleftrightarrow{xy}} = \infty, \forall \overleftrightarrow{xy}$). Since the probability of successful transmission, $p_{\overleftrightarrow{xy}}$, even over good links, is never 1, the former case will incorrectly assume disconnection and incur the cost of queueing a packet with probability $1 - p_{\overleftrightarrow{xy}}$. Clearly, in the latter case, packet transmission attempts will continue even when there is a disconnection, thus wasting energy by unnecessarily using the radio. Thus, a retransmission policy that sets $k_{\overleftrightarrow{xy}}$ too low leads to wasted energy due to excessive use of storage. On the other end of the spectrum a retransmission policy that is too aggressive will lead to wasted energy in the form of fruitless transmissions during periods of disconnection. The goal then is to employ a policy that identifies and uses a midpoint in this tradeoff space. Enough effort should be expended in order to communicate the packet (and avoid using storage), but periods of disconnection should be detected in order to avoid wasteful transmissions. In essence, a good policy will correctly identify those periods of disconnection.

2.3.3 *Supporting Lonely Clouds of Motes*

Support for collections of motes that are intentionally disconnected from IP-capable infrastructure is critical because they provide data for locations that lack even ad hoc server support. Lonely motes experience long-term disconnections, during which data is stored to flash, followed by brief periods of connection when data is communicated in bulk.

When a mobile user interacts directly with the network and then moves, disconnections are inevitably introduced. While the mobile user is transitioning from one location to the next data can be stored at each node to ensure reliability. When the mobile user re-establishes contact with the network, the data can then be sent towards the sink. Local storage is important to survive both transient or accidental as well as long-term disconnections.

3 EVALUATION

In this section we present empirical evidence that Hyper's link estimator accurately captures the state of the wireless channel, that the fast link convergence protocol provides a reasonable estimate of link quality in only a few seconds, and that high quality trees can be built in at most a few seconds. We show analytically that Hyper is able to infer disconnections, while avoiding the use of persistent storage when there is not a disconnection.

3.1 *etx* as a Link Metric

Efficient routing trees cannot be assembled without accurate estimates of underlying link qualities. The *etx* link metric, which was first demonstrated to be effective for 802.11[6], has already been applied to several sensor network routing protocols including MintRoute and CentRoute. Since several sensor network radios do not provide *etx* values directly, we confirm that the beaconing technique introduced in Section 2.2.1 generates accurate results.

We evaluated *etx* as a link metric in an office setting, an outdoor urban setting, as well as in dense foliage and found that in practice it works well. In each scenario, we deployed two nodes, one as a sender and the other as a receiver and adjusted the distance between these nodes to create links of varying quality.

The experiment consisted of two steps. The first was to use Hyper's periodic beaconing procedure to estimate *etx*. Once the *etx* stabilized for a particular link configuration, the second step was to measure how many transmissions were required of the sender to successfully deliver a packet to the receiver. Success was identified by reception of an acknowledgment packet. We limited the number of transmission attempts for a single packet to at most twenty. Over all scenarios we collected more than 500,000 such samples.

Figure 1 summarizes our results. For each of several ranges of *etx* values, [1,2), [2,3), [3,4), and [4,5), we constuct a CDF describing the number of transmission attempts that were actually required to transmit a packet. Results from all deployment scenarios are combined.

Let $p_{\overleftrightarrow{ab}}$ be the probability that a packet is successfully transmitted and acknowledged over link $\overleftrightarrow{ab}$ on the first try. The probability of success on the i^{th} try then is $p_{\overleftrightarrow{ab}} \cdot (1 - p_{\overleftrightarrow{ab}})^{i-1}$ and the probability that a message is communicated successfully in N or fewer transmissions, $P(N)_{\overleftrightarrow{ab}}$ is $\sum_{i=1}^{N} p_{\overleftrightarrow{ab}} \cdot (1-p_{\overleftrightarrow{ab}})^{i-1} = 1-(1-p_{\overleftrightarrow{ab}})^{N}$. We use $P(N)$ and the relation $p_{\overleftrightarrow{ab}} = \frac{1}{etx_{\overleftrightarrow{ab}}}$ to construct a theoretical distribution of transmissions required to deliver a packet for *etx* values of 2, 3, and 4.

The empirically derived CDFs match very closely to the ones for the theoretical binonial distributions. We draw two conclusions from these results. First, since the shapes of the distributions are similar, it appears that the probability that a transmission will succeed in *i* transmissions is geometric for real links. Second, the fact that the empirical distribution of transmission attempts for a measured *etx* matches the theoretical distribution for a given *etx*, provides evidence that the measured *etx* is accurate.

3.2 Fast Link Convergence

The goal of fast link convergence is to get an estimate of the link quality between a new node and all of its neighbors in a short period of time. To do this several beacon requests are sent. Any node that hears a beacon request sends a beacon response in reply. In the absence of collisions, we would like to get an estimate of link quality to $\pm 10\%$.

The problem is that the faster the call and respond protocol operates, the more likely it is that temporarily excessive contention will aversely skew link quality estimates. This problem is exacerbated in sufficiently dense networks where the replies from a node's multitude of neighbors will inevitably collide.

To quantify the relationships at a particular beaconing rate between neighborhood density and the accuracy of link cost measurements, we ran experiments that involved a varying number of active nodes in a single-hop communication domain. For these experiments, the *etx* for each link was known to be close to 1. At the onset of each experiment, a newly activated node sent a fast beacon packet (FastBP). All nodes that received this packet responded with a beacon packet (BP). For each experiment, this call and response was repeated ten times; the period between calls was fixed at 0.5 seconds.

In figure 2 the number of messages sent by the neighbors of a newly activated node is plotted for various densities. The average number of BP responses sent by each of these nodes is very close to the number of FastBPs sent by the new node.

Over the range of densities the number of FastBP packets received by the neighbors only deviates slightly from the maximum. This indicates that in the space of 0.5 seconds neighbor responses aren't colliding with the next request.

In Figure 2, we also plot the number of responses that the new node heard from each neighbor for various densities. As the number of neighbors increases the deviation from the maximum also increases, growing beyond 20% for nine neighbors. This suggests that the number of collisions increases as the density increases. Clearly, as the neighbor size increases beyond nine, this problem only grows worse. This suggests that the MAC layer only does a reasonable job of avoiding collisions among a group of synchronized senders for neighborhoods of size up to around five to seven. In that range, the number of responses heard by the new node differs from the ideal by about 10%. The major limitation in our current implementation of fast link convergence is its reliance on the MAC to do effective collision avoidance.

Fast link convergence need not provide perfect link estimates. It is a way to get some idea of the neighborhood size and link quality in a small amount of time. It needs to be fast enough so that in a short amount of time the technicians can determine if configuration adjustments are needed.

3.3 Tree Building

We evaluated Hyper's routing protocol in terms of the quality and depth of trees produced and the cost and speed of their construction; we found that for a fairly large network, Hyper was able to build low cost trees in about one second.

To build and maintain a tree, each mote participating in the protocol generates one route update message for every update generated by the root. This level of overhead is the same as for the popular MintRoute protocol. Likewise, the size of each control message, the overhead per data message, and the amount of state maintained for MintRoute and Hyper are comparable. Figure 3 lists their respective RAM use, transport headers and control overhead. Hyper incurs this extra overhead to provide greater functionality, including support for multiple trees, link and route locking, and runtime control of the route update rate.

To test the protocol's responsiveness we ran several experiments over a 27-node Mica2 testbed with a routing diameter of about six hops. Hyper delays the processing of route update messages for a time proportional to the cost of the route being advertised. We varied this proportionality constant, *c*, and measured its effects on latency of tree construction and quality of the routes constructed in terms of path *etx* and the depth of the trees produced. For each value of *c* hundreds of route updates were sent, causing thousands of measurements to be made from the set of nodes.

We instrumented the network to transmit paths and associated costs over a wired serial backchannel in order to measure the time it takes to form a routing tree from a root to every node in the network. Each node transmitted this information over the backchannel immediately before transmitting a route update packet over its radio. The serial packets were timestamped upon arrival at a PC. The latency in forming a path from a node to its root was computed as the difference in time between when the node and root transmitted their respective update messages.[2]

Figure 4, figure 5 and figure 6 depict CDFs generated using the aforementioned testbed. Curves in each graph are associated with the various values of *c* we tested, namely 100ms, 50ms, 20ms, and 10ms. The unpredictable contribution of various MAC, radio transmission, and processing delays could interfere with our delay-sensitive tree formation algorithm. To measure the impact of nondeterministic delay, we use the CDFs from our 100ms tests as a basis to compare with experiments for the other three values, as this largest *c* should be least impacted. [3]

We find that to form low cost trees in a short period of time a value

[2]Differences in transmission time over serial and processing time on the PC were not incorporated into these measurements. However, since serial transmission is fairly deterministic and PC processors are fast, these differences should be at most on the order of several milliseconds.

[3]To confirm that the 100ms experiments were not aversely impacted by the aforementioned nondeterminism, we compared the results from these experiments with results for *c* = 1000ms and found only negligible differences in the path qualities produced.

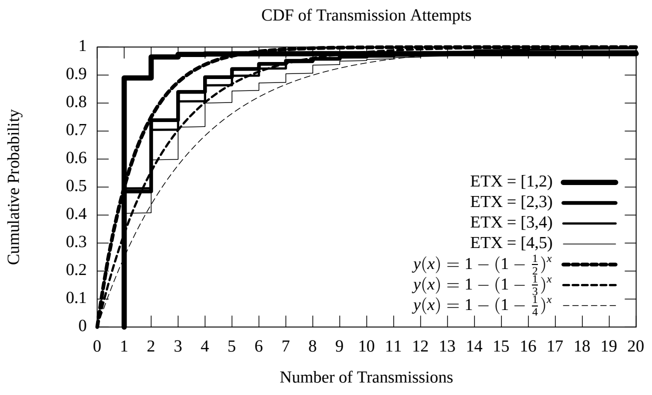

Figure 1—The number of transmissions attempts actually required to deliver a packet were recorded for several *etx* ranges. This figure presents these results as CDFs. The empirically derived curves closely match the theoretical geometric distribution.

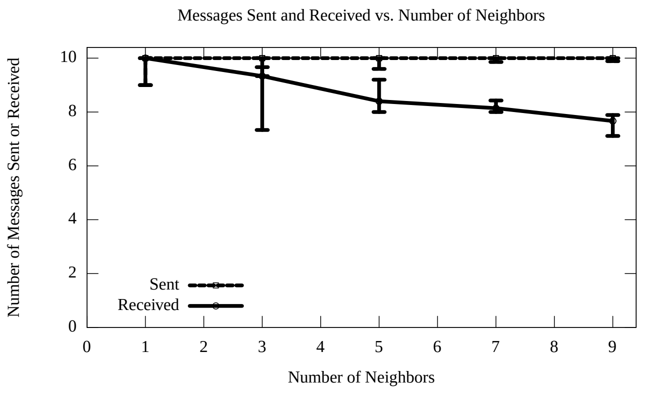

Figure 2—The average number of beacon messages sent by neighbors of a newly activated node and received by the new node during the processing of fast link convergence. As the density increases collisions diminish the number of messages received, thus aversely affecting neighborhood's *etx* measurements.

	Hyper	**MintRoute**
State per Neighbor	**22 B**	**19 B**
Routing Header for Data Packets	**10 B**	**7 B**
Routing Control Message Size	**21 B**	**15 B**
Total RAM Allocation	**2,113 B**	**1,560 B**

Figure 3—Hyper and MintRoute use comparable amounts of memory for managing neigbors and for general protocol operation, and incur similar numbers of bytes of overhead when sending control packets and application data.

of *c* between 50ms and 100ms works well. Figure 4 shows CDFs of path costs for each of the four values of *c*. The further a path cost CDF is to the left side the more desirable it is, indicating that a higher percentage of paths in the topology have a lower cost. The 50ms CDF is almost identical to the 100ms CDF, which indicates that although the 50ms tree was built more quickly than the 100ms, the path cost was not impacted. Increasing *c* above 50ms provides little or no benefit in terms of reducing the path cost, as once *c* is large enough to overcome the effects of nondeterministic timing delays in other software and hardware layers, the protocol will operate precisely as intended. When *c* is sufficiently low, on the other hand, nondeterministic delays induce incorrect protocol behavior.

It takes about 15ms to transmit a route update packet over the CC1000's 19.2 kbps radio. It is no surprise, therefore, that the quality of path construction is reduced when *c* approaches this transmission time. In Figure 4, the curves for 20ms and 10ms values of *c* show this increase in path cost. There are two reasons for this degradation of quality. First, a decrease in control update delays increases the number of nodes on average that transmit during each unit of time; this may increase contention for the channel and may result in the loss of packets that would have advertised better routes. Furthermore, the accumulation of MAC and processing delays may become significant, particularly as a route update travels over several hops. Under the right conditions, an advertisement for a good route might be delayed longer than one for a bad route and a node, therefore, might bind to a bad route before hearing about the good one. Since Hyper only sends one route update packet per node, this poor decision would be sustained until the next route update epoch.

Even with an adequately large *c*, it is always the case that poor paths might form when control packets are lost. The impact of these poor (albeit rare) paths can be minimized by microsever intervention. Since route update epochs can be changed at runtime, a root that deems a newly constructed path to be sufficiently bad may generate another routing update immediately. On the flip side, if a routing tree seems to be especially stable then the microserver can increase the route update period, reducing the control overhead.

Figure 5 shows that the latency to build a tree is directly affected by *c* when *c* is significantly larger than other sources of delay in the system. However, when *c* is on the same order as the other sources of delay, its influence on tree formation latency is decreased. The latency CDFs of Figure 5 for 20ms and 10ms illustrate this. For clarity of presentation, This figure is cut off on the right hand side. A very small percentage of paths can sometimes take a very long time to converge. This may happen, for instance, when due to packet loss, a node only hear a route update from a very poor neighbor with a correspondingly high *etx*. The delay can be on the order of tens of seconds.

Lastly, we look at the distribution of path length (measured in hops) for each wait constant. Although the hop count of a path and the *etx* of path are not neccessarily related (both short paths with high cost and long paths with low cost can exist) the number of hops does enforce a lower bound on the *etx* for a path. It is impossible for the *etx* of a path to be less than the hop count.

As previously noted, larger values of *c* generated higher quality paths. This can also be seen in the hop CDFs. In figure 6 the CDFs for the hop count from the root to each node is plotted. The increased number of hops that 10ms exhibits has several effects. First, the latency in forming a path is increased because the update must cross more hops. This increased latency is more than made up for by the decreased wait constant of 10ms. However, there is a hidden latency cost as well. The latency experienced by a message going from a node to the sink is also increased by the longer paths. Lastly, the increased number of hops translates to increased number of transmis-

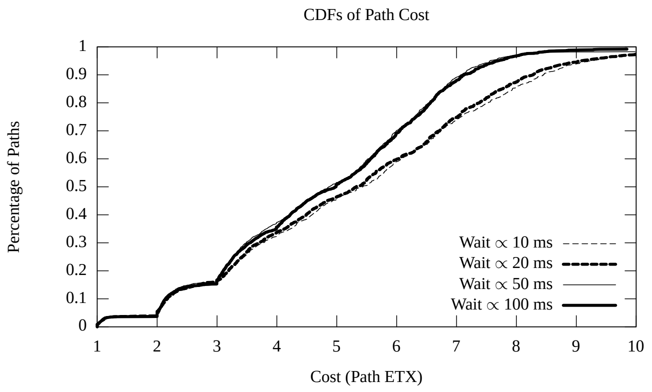

Figure 4—Path cost CDFs for wait constants of 100ms, 50ms, 20ms and 10ms. Hyper can support a wait constant of 50ms, which results in trees being built in about a second. Below 50ms, nondeterministic delays induced by other software and hardware layers hamper Hyper's operation.

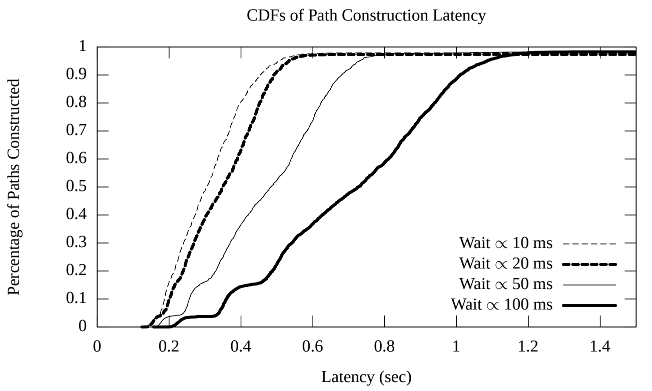

Figure 5—Latency CDFs for wait constants of 10ms, 20ms, 50ms and 10ms. Since an increase in the wait constant directly corresponds in an increase in tree construction latency, smaller wait constants are preferred. However, below 20ms, timing interference from other software layers and radio contention reduce the efficiency of the trees Hyper produces.

sions required to communicate data (the dominant cost in long-lived deployments).

The cost of collecting sensor data is the dominant energy cost over the lifetime of a deployment. Therefore, the more sensor data that is required over the lifetime of a deployment the more careful the root should be in building and maintaining its tree. Hyper not only creates low cost routing trees, but in very little time as well. Because Hyper can build trees in under a second, it is very suitable for a "delay intolerant" user.

3.4 Inferring Link Disconnection

Except during fast link convergence, the constrained energy budget of a deployed mote dictates that *etx* be estimated as seldomly as possible, usually on the order of minutes. In this section we evaluate our *etx*-based retransmission policy in terms of the energy savings it provides when a link's quality temporarily and rapidly degrades and when the link suddenly goes down.

For a given *etx*, a link problem can be inferred when more than *etx* transmissions have been made without success in an attempt to deliver a data packet. Once it is determined that the link is bad, the packet and successive packets may be queued in RAM and flash until either the link recovers or a new route is found. This section demonstrates how dynamic adjustment of the retranmission policy in accordance with *etx* will prevent unnecessary and costly stores to persistent memory and unnecessary radio transmissions.

The general form of retransmission policies that we consider is:

- Try to send a packet over a link *k* times. Before each attempt, set a timer. If an acknowledgement is not received before this timer expires, retransmit the packet.
- If no acknowledgment is received after *k* tries then store it for later transmission. Store all subsequent packets as well until conditions improve.
- After the link is positively reassessed using the periodic beaconing mechanism, or a new route is constructed, remove the head-of-line packet from storage and attempt to transmit it. Continue transmitting packets from storage until there are none left or until another link problem is encountered.

Figure 7 presents the probability of storing a packet that several retransmission policies incur by incorrectly assessing a previously good link as broken. We experiment with three fixed policies, where nodes all make 1, 3, and 10 transmission attempts respectively before queueing a packet in storage for later delivery, and two dynamic policies, where nodes make *etx* and $2 \cdot etx$ transmission attempts. All of these policies are trivial to implement on a mote. We measure the probability of a packet being stored when it didn't need to be.

Data is binned by *etx* into the ranges [1,2), [2,3), [3,4), and [4,5). Using the derived CDFs first presented in Figure 1, we calculate the probability of using storage for each retransmission policy. Figure 7 shows that for values of *etx* close to 1 all retransmission policies perform similarly because the probability of failure is small. For the static retransmission policies, as the value of *etx* increases the storage probability also increases. However, the dynamic retransmission policies (those dependent on the *etx* value) show that sending a number of times proportional to the *etx* uses very little storage across the range of *etx* values.

When a disconnection does occur it is best to store data directly to flash without making any transmission attempts. In the case that a disconnection occurs suddenly, each retransmission policy will generate some number of useless transmissions before determining the link is bad. The more transmission attempts a node makes when in a disconnected state, the more wasteful it is. In this respect, a fixed transmission policy that makes many transmission attempts before giving up will perform well when the link is in a connected state because it will seldomly use storage when it doesn't have to, but will perform horribly when disconnected. Conversely, a fixed policy that transmits only a few times before resorting to storage will perform well when in a disconnected state, but poorly when in a connected state. Because *etx* provides an estimate of just how much effort should be required to communicate a packet, a retransmission policy that is proportional to the *etx* will be efficient in both states.

4 Related Work

Several sensor network projects have focused on support mobility. However, the focus has been on autonmous mobile sensors and sensing a phenomenon in motion, not on a mobile sensor network user, such as a technician or application scientist. Common themes include coverage and exploration [9] [4], pursuer/evader games and target tracking [11] [15] [3], and data muling [17].

Several tree collection protocols have been designed and implemented specifically for sensor networks. Of these, MintRoute [19] is the most widely used. MintRoute provides ad hoc tree routing for wireless sensor networks. Like several other protocols, MintRoute uses *etx* in order to abstract the variations in channel quality [6]. Hyper and MintRoute both use distance vector routing, in so far as they both advertise global state locally. In MintRoute, nodes independently and periodically beacon their routing state in order to establish a route to the sink. Hyper differs from MintRoute in that route updates are only triggered by the sink. As a result loops cannot form in Hyper. Therefore, unlike MintRoute, Hyper does not need to employ techniques like poison reverse or split horizon. Furthermore, Hyper also allows motes to participate in multiple independant routing trees simultaneously, which MintRoute does not support.

A second sensor network routing protocol, Centroute, uses a combination of source routing (which prevents loops), and centralized path computation (on a micorserver) in an effort to relieve motes of the burder of making routing decisions. In contrast to Hyper, each mote can only interact with a single microserver at a time.

Hyper builds high quality trees quickly and uses adaptive timers to reduce control overhead. This is similar to [7] in how the decision to send control messages is made. In [7] the delay before sending requests or repair packets is a function of the distance between the node that triggered the request or repair and the potential responder. Thus, nearby nodes respond before those that are far away. We use a similar idea. Control messages are forwarded after delaying an amount of time proportional to the quality of the link overwhich the control message arrived. If a control message advertising a shorter path to the same sink arrives before that delay expires then the previous message is discarded. In this manner, good paths propagate quickly through the network, while poor paths propagate slowly, or not at all.

To ensure reliability, persistent storage is used to queue packets. This is particularly important for *lonely clouds*. Unlike traditional IP networks where an end-to-end path is the common case, in sensor networks link quality can change frequently. In addition, sensor nodes can be deployed in remote regions where there is neither access to an 802.11 backchannel nor to the Internet in general. Motes may act as long-lived data loggers for extended periods of time, with a "data mule" periodically visiting in order to offload sensor data. Storing data persistently in the event of disconnection is critical for efficient reliability and is discussed in detail in [12].

5 Future Work

Hyper provides a robust and efficient routing substrate that supports the mobility requirements of field researchers and technicians.

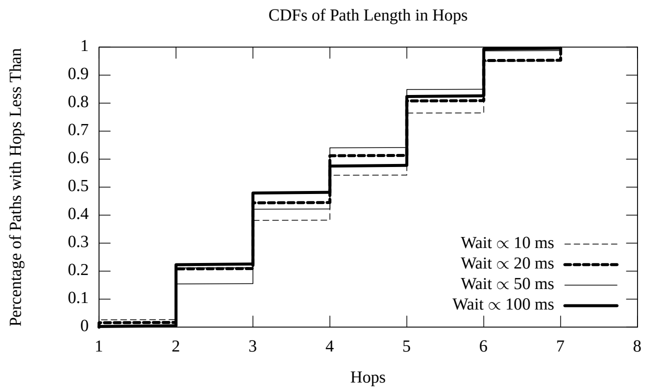

Figure 6—Decreasing the wait constant leads to the construction of deeper trees. Tree depth is the lower bound on path cost.

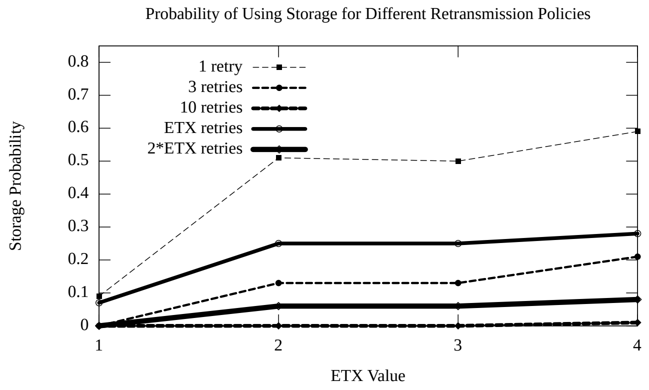

Figure 7—Empirically derived storage costs for different retransmission policies. Using *etx* number of retries has low probability of using storage over the range of *etx* values.

However, there are several steps left before we can integrate this work into systems that habitat scientists already use.

First, although we have done extensive indoor and outdoor testing of Hyper itself, we haven't yet tested the operation of Hyper in our target environments when integrated with real applications running above it. In the near term, we will be deploying Hyper as part of a sensor data collection application at Sensor Mountain in an experimental testbed in the woods near our other deployed systems. From this deployment we hope to learn how well Hyper responds to varying user requests in the face of considerable RF fluctuation due to severe weather changes.

Second, we will augment our fast convergence protocol to figure neighborhood density into its timing. The idea is to decrease the beaconing rate to a level the underlying MAC can tolerate. Third, Hyper currently expects underlying MAC transmission and backoff timings similar to what the CC1000 driver and link code in TinyOS [13] provide. To generalize Hyper to work over most CSMA MACs, we will add more accurate timestamping to account for transmission, processing and other MAC-level delays. These delays, then, will be factored into the SRM-style delays Hyper uses when processing route control messages. By carefully accounting for these delays, Hyper will be able to support protocols with very different timing profiles, such as low power listening (LPL [16]) for communication on low duty cycle devices.

6 Conclusion

To the best of our knowledge, Hyper is the first sensor network routing protocol that explicitly supports a mobile user, allowing direct access to sensor network services while minimizing route construction latency and building high quality routes.

Like several distance vector protocols, Hyper ensures that each node sends at most one routing update per tree building epoch. Unlike these protocols, since construction is initiated by the root, the routing code does not need to include techniques (such as split-horizon or poisonous reverse) to construct loop-free trees. Furthermore, it uses *etx* to produce trees that are inexpensive to use.

With Hyper, a sensor node may participate in multiple trees simultaneously, with the potential to produce different data, at different rates for each collection sink. Most importantly, support for multiple trees enables the use of the network by mobile researchers and technicians, even when the network is already forwarding data to a stationary collection sink.

To support mobile users further, a new node can join an existing network within seconds using our fast link convergence protocol; once a node evaluates its neighborhood, it may form a tree. Trees can be built very quickly, often in less than a second and in only a few seconds for very deep trees. Furthermore, newly activated sensor nodes may be grafted to existing trees quickly.

Finally, Hyper is disruption tolerant. When a link's quality degrades significantly or disappears altogether, hyper employs RAM and flash storage to queue packets for later delivery. This has two consequences. First, it saves energy during temporary network disruptions, because packets aren't transmitted until conditions improve. Second, it provides support for motes that have been deployed intentionally or otherwise in environments with no radio connectivity to a collection sink. Such *lonely motes* may queue packets for hours or months until a mobile user comes within range to collect them.

References

[1] Chipcon cc2420 radio. http://www.chipcon.com".

[2] Crossbow technology inc. http://www.xbow.com.

[3] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y-R. Choi, T. Herman, S. S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, pages 605–634, 2004.

[4] Maxim Batalin and Gaurav S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Telecommunication Systems Journal, Special Issue on Wireless Sensor Networks*, 26(2):181–196, 2004.

[5] Alberto Cerpa, Jennifer L. Wong, Miodrag Potkonjak, and Deborah Estrin. Statistical model of lossy links in wireless sensor networks. In *IPSN '05: Proceedings of the Fourth ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2005.

[6] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Mobicom 2003*. ACM, 2003.

[7] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. on Networking (TON)*, 5(6):784–803, 1997. ISSN 1063-6692.

[8] Ramesh Govindan, Eddie Kohler, Deborah Estrin, Fang Bian, Krishna Chintalapudi, Om Gnawali, Ramakrisnha Gummadi, Sumit Rangwala, and Thanos Stathopoulos. Tenet: an architecture for tiered embedded networks. In *CENS Technical Report No. 53*.

[9] Andrew Howard, Maja J. Matarić, and Gaurav S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots Special Issue on Intelligent Embedded Systems*, 13 (2):113–126, 2002.

[10] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-197-6.

[11] Boyoon Jung and Gaurav S. Sukhatme. Tracking targets using multiple robots: The effect of environment occlusion. *Autonomous Robots*, 13 (3):191–205, Nov 2002.

[12] Rahul Kapur and Deborah Estrin. Reliability and storage in sensor networks. In *CENS Technical Report No. 59*.

[13] Philip Levis, Sam Madden, David Gay, Joe Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in tinyos. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[14] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, pages 131–146, Boston, MA, USA, December 2002.

[15] Luis O. Mejias, Srikanth Saripalli, Gaurav S. Sukhatme, and Pascual Cervera. Detection and tracking of external features in an urban environment using an autonomous helicopter. In *IEEE International Conference on Robotics and Automation*, pages 3983–3988, Apr 2005.

[16] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-879-2.

[17] R. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a three-tier architecture for sparse sensor networks, 2003. URL citeseer.ist.psu.edu/article/shah03data.html.

[18] Thanos Stathopoulos, Lewis Girod, John Heidemann, and Deborah Estrin. Mote herding for tiered wireless sensor networks. In *CENS Technical Report No. 58*, December 7 2005.

[19] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Sensys 2003*, 2003.