# CTP: An Efficient, Robust, and Reliable Collection Tree Protocol for Wireless Sensor Networks

OMPRAKASH GNAWALI, University of Houston
RODRIGO FONSECA, Brown University
KYLE JAMIESON, University College London
MARIA KAZANDJIEVA, Stanford University
DAVID MOSS, People Power Co.
PHILIP LEVIS, Stanford University

We describe CTP, a collection routing protocol for wireless sensor networks. CTP uses three techniques to provide efficient, robust, and reliable routing in highly dynamic network conditions. CTP's link estimator accurately estimates link qualities by using feedback from both the data and control planes, using information from multiple layers through narrow, platform-independent interfaces. Second, CTP uses the Trickle algorithm to time the control traffic, sending few beacons in stable topologies yet quickly adapting to changes. Finally, CTP actively probes the topology with data traffic, quickly discovering and fixing routing failures. Through experiments on 13 different testbeds, encompassing seven platforms, six link layers, and multiple densities and frequencies, and detailed observations of a long-running sensor network application that uses CTP, we study how these three techniques contribute to CTP's overall performance.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Wireless sensor network, wireless network protocol, link-quality estimation, adaptive beaconing, datapath validation, routing

**ACM Reference Format:**
Gnawali, O., Fonseca, R., Jamieson, K., Kazandjieva, M., Moss, D., and Levis, P. 2013. CTP: An efficient robust, and reliable collection tree protocol for wireless sensor networks. ACM Trans. Sensor Netw. 10, 1, Article 16 (November 2013), 49 pages.
DOI: http://dx.doi.org/10.1145/2529988



## 1. INTRODUCTION

Collection is a core building block for many sensor network applications. In its simplest use, collection provides an unreliable, datagram routing service that deployments use to gather data [Werner-Allen et al. 2006; Mainwaring et al. 2002; Tolle et al. 2005]. A natural and efficient way to implement collection is by using a tree topology rooted at one or more collection points. Such trees are themselves building blocks for other types of protocols. Tree-based collection protocols provide the topology

This article synthesizes and extends "Four Bit Wireless Link Estimation" published in *Proceedings of the 7th Workshop on Hot Topics in Networks (HotNets'07)* and "Collection Tree Protocol" published in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*.
O. Gnawali was partially supported by a generous gift from Cisco. K. Jamieson is supported by the European Research Council under Grant No. 279976.
Corresponding author's email: gnawali@cs.uh.edu.

DOI: http://dx.doi.org/10.1145/2529988

underlying most point-to-point routing protocols, such as BVR [Fonseca et al. 2005], PathDCS [Ee et al. 2006], and S4 [Mao et al. 2007], as well as transport protocols, such as IFRC [Rangwala et al. 2006], RCRT [Paek and Govindan 2007], Flush [Kim et al. 2007], and Koala [Musaloiu-E. et al. 2008].

This article describes the design and evaluation of a collection protocol that can simultaneously achieve four goals.

—*Reliability.* A collection protocol should achieve high data delivery reliability unless the quality of the underlying links makes that infeasible. In a network with high quality links, 99.9% delivery should be achievable without end-to-end mechanisms.
—*Robustness.* It should be robust against transient network failures, dynamic workloads, and topology changes. Despite these dynamics, it should operate without much tuning or configuration.
—*Efficiency.* It should achieve this reliability and robustness while sending few packets. Efficiency in communication oftentimes translates to energy efficiency or allows an opportunity to make the system more energy efficient.
—*Hardware Independence.* Since sensor networks use a wide range of platforms, it is desirable that a protocol be robust, reliable, and efficient without assuming specific radio chip features to the extent possible.

In this article, we explore three mechanisms that allow a routing protocol to achieve these goals in wireless networks.

First, achieving these goals depends on link estimation accuracy and agility. For example, recent experimental studies have shown that, at the packet level, wireless links in some environments have coherence times as small as 500 milliseconds [Srinivasan et al. 2008]. Being efficient requires using these links when possible, but avoiding them when they fail. The four-bit link estimator, which we describe later, combines information from the physical, link, and network layers to accurately estimate the link qualities, but it achieves this accuracy by changing its estimates as quickly as every five packets.

Second, such dynamism requires path selection and advertisement to also operate rapidly when the links change, while sending few beacons when the network is stable. We adapt the Trickle algorithm [Levis et al. 2004], originally designed for propagating code updates, to dynamically adapt the control traffic rate. This allows a protocol to react in tens of milliseconds to topology changes, while sending a few control packets per hour when the topology is stable.

Third, routing inconsistencies must be detected at the same time scale as data packet transmission. We use the datapath to validate the routing topology as well as detect loops. Each data packet contains the link-layer transmitter's estimate of its distance. A node detects a possible routing loop when it receives a packet to forward from a node with a smaller or equal distance to the destination. Rather than drop such a packet, the routing layer tries to repair the topology and forwards the packet normally. Using data packets maintains agility in inconsistency detection precisely when a consistent topology is needed, even when the control traffic rate is very low due to Trickle.

We ground and evaluate these three principles in a concrete protocol implementation, which we call the Collection Tree Protocol (CTP). In addition to incorporating agile link estimation, adaptive beaconing, and datapath validation, CTP includes many mechanisms and algorithms in its forwarding path to improve its performance. These include re-transmit timers, a hybrid queue for forwarded and local packets, per-client queueing, and a transmit cache for duplicate suppression.

To explore whether a routing layer with these principles can meet the stated goals in a wide spectrum of environments with minimal adjustments, we evaluate CTP on 13 different testbeds ranging in size from 20–310 nodes and comprising seven hardware

platforms. The testbeds comprise diverse environmental conditions beyond our control, still provide some reproducibility, and enough diversity that give us confidence that CTP achieves the stated goals. In two testbeds that have Telos nodes, we evaluate CTP using three link layers: full power, low-power listening [Polastre et al. 2004], and low-power probing [Musaloiu-E. et al. 2008]. In one Telos-based testbed where there is exceptionally high 802.11b interference, we evaluate CTP on an interference-prone and an interference-free channel. We conclude the evaluation with an analysis of CTP's performance in a large, long-term indoor deployment—Powernet [Kazandjieva et al. 2012].

Evaluating CTP's use of agile link estimation, adaptive beaconing, and datapath validation, we find the following.

—Across all testbeds, configurations, and link layers, CTP's end-to-end delivery ratio ranges from 90.5% to 99.9%.
—CTP achieves a median duty cycle of 3% across the nodes in a network in an experiment in which the network generates data at 30 packets/min and delivers them to the sink.
—Compared to MultiHopLQI, a collection protocol used in sensor network deployments [Werner-Allen et al. 2006], CTP drops 90% fewer packets while requiring 29% fewer transmissions.
—Compared to MultiHopLQI's fixed 30 second beacon interval, CTP's adaptive beaconing and datapath validation sends 73% fewer beacons while cutting loop recovery latency by 99.8%.
—Testbeds vary significantly in their density, connectivity, and link stability, and the dominant cause of CTP packet loss varies across them correspondingly.

Our work on CTP makes four research contributions. First, it describes three key mechanisms—agile link estimation, adaptive beaconing, and datapath feedback—which enable routing layers to remain efficient, robust, and reliable in highly dynamic topologies on many different sensor platforms. Anecdotal reports from several deployments by other researchers and our analysis of network performance from Powernet deployment validate the testbed results on efficiency, robustness, and reliability. Second, it describes the design and implementation of CTP, a collection protocol that uses these three mechanisms. Third, by evaluating CTP on 13 different testbeds, it provides a comparative study of their behavior and properties. The variation across testbeds suggests that protocols designed for and evaluated on only a single testbed are prone to failures when they encounter different network conditions. Fourth, we describe the experiences and lessons from a long-running large-scale deployment of CTP. We hope that our deployment experiences can inform future deployments of sensor networks.

This article extends prior descriptions of CTP in sensor network literature [Gnawali et al. 2009; Fonseca et al. 2007] by describing the following.

—Results from experiments on an additional testbed called Indriya, a 126-node sensor network testbed
—Results from experiments designed to understand the impact of beacon suppression threshold and network density on control overhead and the impact of hysteresis threshold on CTP's route selection and performance
—Results from evaluation of CTP with multiple roots
—Results and experiences from a large-scale, long-term deployment of CTP on Powernet, which is a sensor network deployed in a building to collect power measurements
—The impact CTP has made in sensor networking research and practice

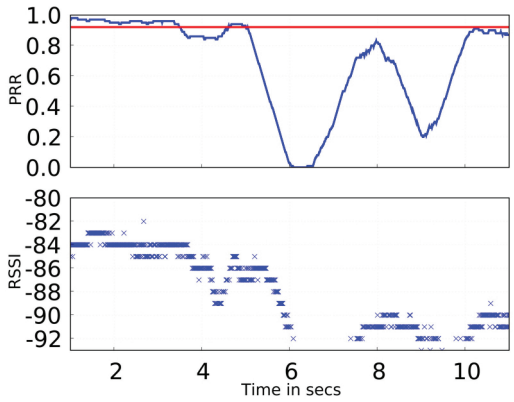



Fig. 1. Plot of packet reception rate (PRR) over sliding window of 100 packets (upper) and RSSI for every received packet (lower) during an experiment with interpacket interval of 10 ms. PRR can vary on time scales orders of magnitude smaller than beacon rates. RSSI and other physical-layer measurements are absent for dropped packets: using them can bias measurements.

## 2. CHALLENGES

Implementing robust and efficient wireless protocols is notoriously difficult, and protocols for collection are no exception. At first glance, collection protocols may appear very simple. They provide best-effort, unreliable packet delivery to one of the data sinks in the network. Having a robust, highly reliable, and efficient collection protocol benefits almost every sensor network application today, as well as the many transport, routing, overlay, and application protocols that sit on top of collection trees.

But despite providing a simple service that is fundamental to so many systems, and being in use for almost a decade, collection protocols can suffer from poor performance. Some deployments observe delivery ratios of 2–68% [Mainwaring et al. 2002; Langendoen et al. 2006; Tolle et al. 2005; Werner-Allen et al. 2006].

Furthermore, it is unclear why collection performs well in controlled situations yet poorly in practice, even at low data rates. To better understand the causes of these failures, we ran a series of experiments on 13 different testbeds and found two phenomena to be the dominant causes: link dynamics and transient loops.

### 2.1. Link Dynamics

Many protocols use periodic beacons to maintain their topology and estimate link qualities. The beaconing rate introduces a trade-off between agility and efficiency: a faster rate leads to a more agile network but higher cost, while a lower rate leads to a slower-to-adapt network and lower cost. Early protocol designs, such as MintRoute, assumed that intermediate links had stable, independent packet losses, and used this assumption to derive the necessary sampling window for an accurate estimate [Woo et al. 2003]. But in some environments, particularly in the 2.4 GHz frequency space, links can be highly dynamic. Experimental studies have found that many links are not stationary, but bursty on the time scale of a few hundred milliseconds [Srinivasan et al. 2008].

The upper plot in Figure 1, taken from the Intel Mirage testbed, shows an example of such behavior: the link transitions between very high and very low reception multiple times in a two-second window. Protocols today, however, settle for beacon rates on the order of tens of seconds, leading to typical rate mismatches of two to three orders of magnitude. This means that at low beacon rates, periodic control packets might observe a reception ratio of 50%, data packets observe periods of 0% and 100%. The periods of

0% cause many wasted retransmissions and packet drops. For a periodic beacon to be able to sample these link variations, the beacon rate would have to be in the order of few hundred milliseconds.

The four-bit estimator addresses the challege of accurate link estimation when the links are dynamic by actively using data packets to measure link quality. This allows it to adapt very quickly to link changes. Such agility, however poses a challenge in routing protocol design: how should a routing protocol be designed when the underlying link topology changes in the order of a few hundred milliseconds?

### 2.2. Transient Loops

Rapid link topology changes can have serious adverse effects on existing routing protocols, causing losses in the data plane or long periods of disconnection while the topology adjusts. In most variations of distributed distance vector algorithms, link topology changes result in transient loops which causes packet drops. This is the case even in path-vector protocols like BGP, designed to avoid loop formation [Pei et al. 2004]. The MultiHopLQI protocol, for example, discards packets when it detects a loop until a new next hop is found. This can take a few minutes, causing a significant outage. We experimentally examine this behavior of MultiHopLQI in Section 7.3.1.

Some protocols prevent loops from forming altogether. DSDV, for example, uses destination-generated sequence numbers to synchronize routing topology changes and prevent loops [Perkins and Bhagwat 1994]. The trade-off is that when a link goes down, the entire subtree whose root used that link is disconnected until an alternate path is found. This can only happen when the global sequence number for the collection root changes.

In both cases, the problem is that topology repairs happen at the time scale of control plane maintenance, which operates at a time scale orders of magnitude longer than the data plane. Since the data plane has no say in the routing decisions, it has to choose between dropping packets or stopping traffic until the topology repairs. This, in turn, creates a tension on the control plane between efficiency in stable topologies and delivery in dynamic ones.

## 3. DESIGN ELEMENTS

CTP uses three main techniques to achieve robustness, reliability, and energy-efficiency. First, it estimates link quality combining information from the physical, link, and network layers, updating the estimate as quickly as every five packet transmission. However, rapidly changing link qualities causes nodes to have stale topology information, which can lead to routing loops and packet drops. CTP uses the remaining two mechanisms to be agile to link dynamics while also having a low overhead when the topology is stable. The second mechanism is datapath validation: using data packets to dynamically probe and validate the consistency of its routing topology. The third mechanism is adaptive beaconing, which extends the Trickle code propagation algorithm so it can be applied to routing control traffic. Trickle's exponential timer allows nodes to send very few control beacons when the topology is consistent, yet quickly adapt when the datapath discovers a possible problem.

### 3.1. Agile Link Estimation

Predicting how well a particular link will deliver packets is fundamental when choosing reliable and efficient routes to deliver packets. As mentioned in Section 2, there are significant challenges in building a robust link estimator. The underlying links are highly dynamic, exhibiting a bursty behavior over short time scales. Traditional packet counting techniques for link estimator face a problem of rate mismatch in part due to these fast, correlated changes. On the other hand, strategies that sample the

quality over received packets suffer from an estimation bias. To address these issues, the four-bit link estimator (4B) used in CTP combines information from the physical, data link, and routing layers to provide accurate estimates despite these challenges. Link quality estimates in 4B come from two sources: low beaconing to bootstrap the topology and form a rough quality estimate and unicast data transmissions for fast, accurate updates, changing the estimate as quickly as every five link-layer unicast transmissions.

### 3.2. Datapath Validation

Every collection node maintains an estimate of the cost of its route to a collection point. We assume expected transmissions (ETX) as the cost metric, but any similar gradient metric can work just as well. A given node's cost is the cost of its next hop plus the cost of its link to the next hop: the cost of a route is the sum of the costs of its links. Collection points, or roots, advertise a cost of zero.

Each data packet contains the transmitter's local cost estimate. When a node receives a packet to forward, it compares the transmitter's cost with its own. Since cost must always decrease, if a transmitter's advertised cost is not greater than the receiver's, then the transmitter's topology information is stale and there may be a routing loop. Using the data path to validate the topology in this way allows a protocol to detect possible loops on the first data packet after they occur.

### 3.3. Adaptive Beaconing

We assume that the collection layer updates stale routing information by sending control beacons. As with data packets, beacons contain the transmitter's local cost estimate. Unlike data packets, however, control beacons are broadcasts. A single beacon updates many nearby nodes.

Collection protocols typically broadcast control beacons at a fixed interval [Tolle et al. 2007; Woo et al. 2003]. This interval poses a basic trade-off. A small interval reduces the length of time the topology information is allowed to be stale and the duration a loop can persist, but uses more bandwidth and energy. A large interval uses less bandwidth and energy but can let topological problems persist for a long time. Adaptive beaconing breaks this trade-off, achieving both fast recovery and low cost. It does so by extending the Trickle algorithm [Levis et al. 2004] to maintaining its routing topology.

Trickle is designed to reliably and efficiently propagate code in a wireless network. Trickle's basic mechanism is transmitting the version number of a node's code using a randomized timer. Trickle adds two mechanisms on top of this randomized transmission: suppression and adaptation of the timer interval. If a node hears another node advertise the same version number, it suppresses its own transmission. When a timer interval expires, Trickle doubles it, up to a maximum value ($\tau_h$). When Trickle hears a newer version number, it shrinks the timer interval to a small value ($\tau_l$). If all nodes have the same version number, their timer intervals increase exponentially, up to $\tau_h$. Furthermore, only a small subset of nodes transmit per interval, as a single transmission can suppress many nearby nodes. When there is new code, however, the interval shrinks to $\tau_l$, causing nodes to quickly learn of and receive new code.

Unlike algorithms in ad-hoc routing protocols, such as DSDV [Perkins and Bhagwat 1994], adaptive beaconing does not assume the tree maintains a global sequence number or version number that might allow a simple application of Trickle. Instead, adaptive beaconing uses its routing cost gradient to control when to reset the timer interval. The routing layer resets the interval to $\tau_l$ when any of these three events occur: (a) it detects an inconsistency in routing gradient between the nodes along a path, (b) discovers a significantly better path, or (c) receives a request from a neighbor to transmit beacons for faster topology discovery. In a network with very stable links, the first

two events are rare. When no new nodes are introduced into the network and high quality routing paths are present in the network, the third condition is also rare. Thus, in a typical network, the beacon interval increases exponentially, up to $\tau_h$. When the topology changes significantly, however, affected nodes reset their intervals to $\tau_l$, and transmit to quickly reach consistency.

## 4. LINK ESTIMATION

Accurate link quality estimates are a prerequisite for efficient routing in wireless networks. Many factors conspire to make accurate link quality estimation challenging, such as the prevalence of intermediate-quality links, the time-varying nature of a wireless channel, multipath inter-symbol interference, and the hardware variations. This section describes the four-bit (4B) link quality estimator used by CTP to discover the links to use for path selection. We identify various information that can be used to estimate the quality of a link. We then describe our design of the four-bit estimator that uses all those information to accurately estimate the link quality while remaining platform independent.

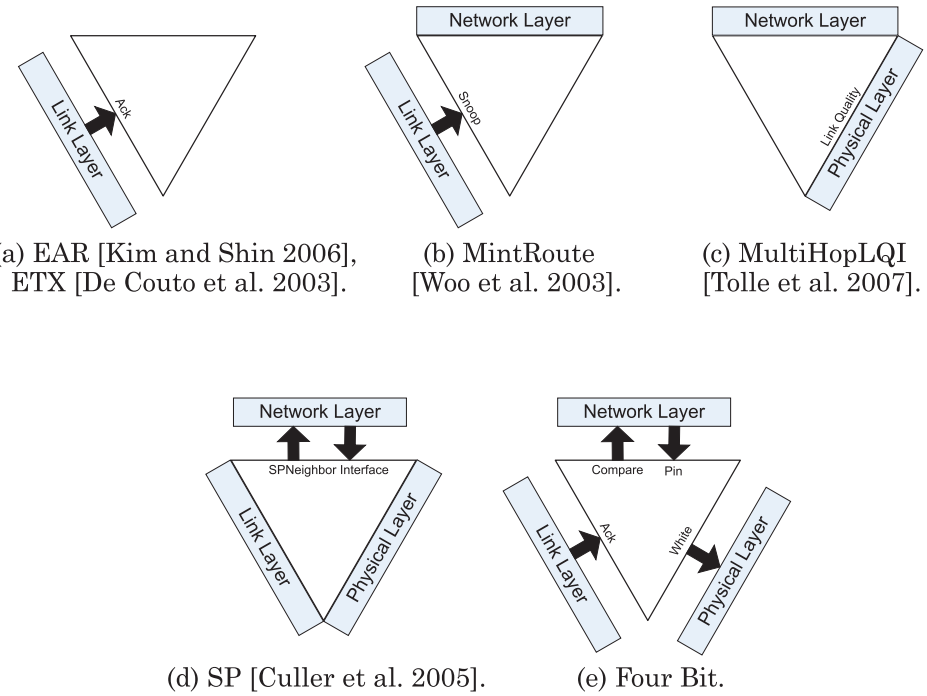
### 4.1. Information from the Physical, Link, and Network Layers

The physical, link, and network layers each have valuable information that can improve estimates, such as channel quality, packet delivery ratios, route utility, and acknowledgments. The complexity of this design space, combined with the rich information that certain chipsets or protocols can provide, has led many protocols to use *cross-layer* design, where each layer freely shares protocol-specific information in order to improve performance. In our design of the four-bit estimator (4B), we take a different approach. We distill the feedback provided by the physical, link, and network layers for accurate link estimation to narrow interfaces, thereby keeping layers decoupled and making the 4B estimator portable across platforms.

Next, we describe the pitfalls of not using information from all the layers of the protocol stack while estimating link qualities. Then we discuss the type of information available in the physical, link, and network layer that can help in making link quality estimation accurate and agile.

*4.1.1. Layer Limitations.* A link estimator should be accurate and efficient. It should provide good estimates of link qualities and be agile in detecting changes, all the while minimizing memory requirements and overhead traffic. Each of the physical, link, and network layers can provide valuable information for the link estimator, as demonstrated by previous work (c.f. Figure 2). We argue that a link estimator should use information from all three layers to best achieve these goals, not only because each layer can provide information that is unique or much more inexpensively obtained, but also because there are different link conditions that some layers can detect while others cannot. The physical layer's per-packet channel quality assessment cannot always detect channel temporal variations. While the link layer can accurately measure ETX, it cannot inexpensively decide which links to estimate. The network layer knows which links are most useful for routing, but estimating link qualities at the network layer is inefficient and slow to adapt.

As an example of how we can use information to help the link estimator, we take a closer look at two collection protocols, CTP-Beacons and MultiHopLQI. CTP-Beacons is a version of CTP that only uses periodic beacons to estimate link quality, just like MintRoute [Woo et al. 2003] or ETX [De Couto et al. 2003]. MultiHopLQI [Tolle et al. 2007] relies solely on the link quality indicator (LQI) provided by the CC2420 radio chip [TexasInstruments 2008] to estimate link quality. We ran collection protocol with these two estimators on an 85-node testbed with each node generating one data packet




(a) EAR [Kim and Shin 2006], ETX [De Couto et al. 2003].
(b) MintRoute [Woo et al. 2003].
(c) MultiHopLQI [Tolle et al. 2007].
(d) SP [Culler et al. 2005].
(e) Four Bit.

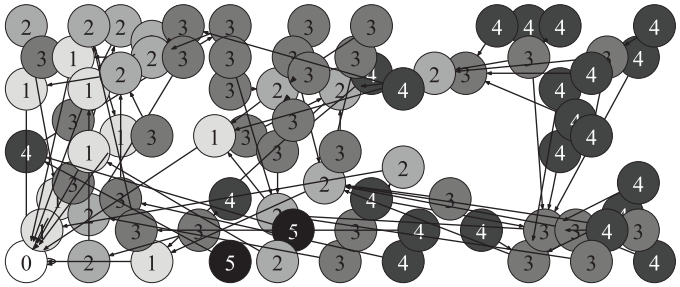
Fig. 2. A link estimator, represented by the triangle in the center of each figure, interacts with up to three layers. Attached boxes represent unified implementation. Outgoing arrows represent information the estimator requests on packets it receives. Incoming arrows represent information the layers actively provide.

every ten seconds. Figure 3 shows a typical routing tree formed by CTP-Beacons with a table size of ten (a), MultiHopLQI (b), and a version of CTP-Beacons with no restriction on the size of the link estimator tables (c). It also shows the average cost, in number of transmissions, for each delivered packet. Lower costs mean shorter paths with good quality.

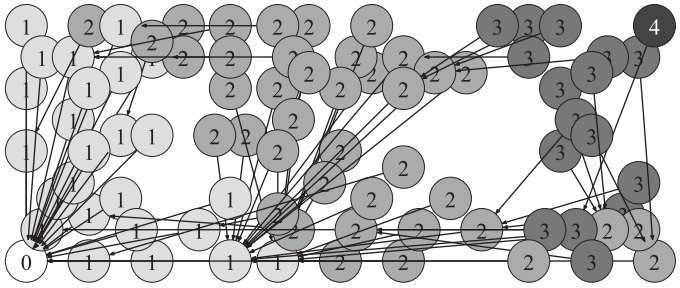
CTP-Beacons' cost is higher than MultiHopLQI's, even though the latter only uses physical-layer information. This is the symptom of two problems. First, because CTP-Beacons uses a bidirectional probe-based link estimator, its link table size limits a node's in-degree. The two nodes at the two ends of a link contribute link measurement for one direction each. If both the nodes do not have an entry for that link, it is not possible to compute bidirectional link quality estimate like ETX. Without link quality estimate, the routing protocol does not use that link for path selection. Hence, a small table limits the number of paths available in the network. Second, also because of the limited link table size, it may be that the best outgoing link is not even on the table to be selected for routing. In a dense network with a large number of links, it is possible for many links to be in the table of a node on one end of the link but not on the node at the other end because the nodes decide which links they want to estimate independently. Figure 3(c) shows that when the link table is unrestricted, CTP-Beacons can outperform MultiHopLQI.

In Section 7.3.2, we show how using information from the physical, link, and network layers we can mitigate these problems. The following sections elaborate on the benefits and limitations of each layer.

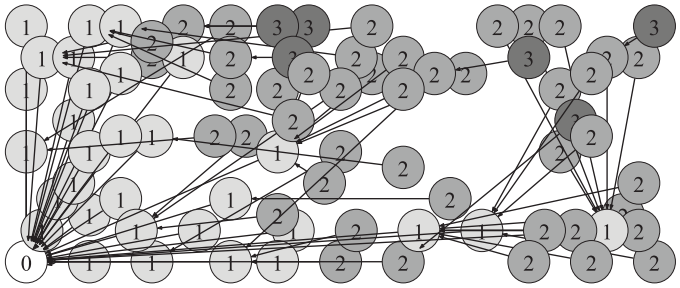
*4.1.2. Physical Layer.* The physical layer can provide immediate information on the quality of the decoding of a packet. Such physical layer information provides a fast



(a) CTP-Beacons (cost = 3.14).



(b) MultiHopLQI (cost = 2.28).



(c) CTP-Beacons unconstrained (cost = 1.86).

Fig. 3. Routing trees formed on 85 node topology by CTP with ten-node link table, MultiHopLQI, and CTP-Beacons with unrestricted link table. The average cost in transmissions per delivered message is in parenthesis. The root is the node in the bottom-left corner, and darker nodes (hop-count indicated inside each circle) mean longer paths to the root.

and inexpensive way to avoid borderline or marginal links. It can increase the agility of an estimator, as well as provide a good first order filter for inclusion in the link estimator table. We assume that the corruption on source address in the packet header not detected by packet CRC (which would cause attributing a measurement to the wrong link) is a rare event and thus a minor concern in link estimation. Figure 2 shows that MultiHopLQI (c) and SP[1] [Culler et al. 2005] (d) use physical layer information for link estimation.

As this physical layer information pertains to a single packet and it can only be measured for received packets, channel variations can cause it to be misleading. For example, many links on low-power wireless personal area networks are bimodal [Srinivasan et al. 2006], alternating between high (100% packet reception ratio, PRR) and low (0% PRR) quality. On such links, the receiver using only physical information

[1]When using information provided by the underlying radio.

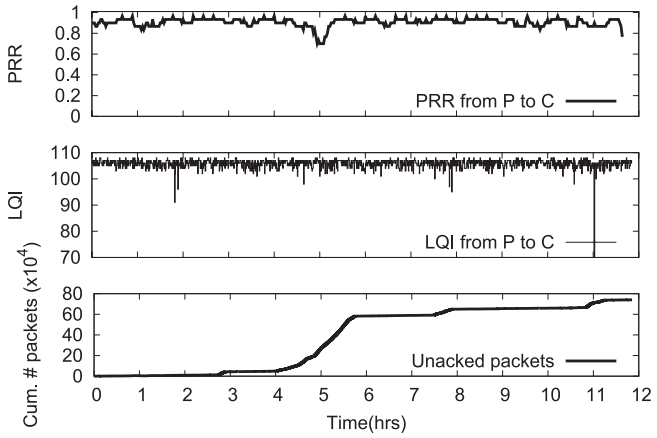



Fig. 4. Unaware of the reduced PRR, MultiHopLQI attempts to deliver packets on the same link between the forth and sixth hour causing increased number of retransmissions due to unacknowledged packets.

will see many packets with high channel quality and might assume the link is good, even if it is missing many packets.

Figure 4 shows a limitation of physical layer information that we observed when we ran MultiHopLQI collection protocol for twelve hours on a 94-node testbed. As Figure 2(c) shows, MultiHopLQI does not use link layer information. Although the protocol performed well overall, there were bursts of packet loss. As Figure 4 shows, for a period of time, the PRR between the nodes $C$ and $P$ dropped from an average of 0.9 to almost 0.6. This degradation in link quality was not accompanied by a drop in the decoding quality indicator (LQI). All of the packets $C$ received had high quality: it just was not receiving all the packets.

*4.1.3. Link Layer.* Link estimators, such as ETX [De Couto et al. 2003] and MintRoute [Woo et al. 2003], use periodic broadcast probes to measure incoming packet reception rates. These estimators calculate bidirectional link quality—the probability a packet will be delivered and its acknowledgment received—as the product of the qualities of the two directions of a link. While simple, this approach is slow to adapt, and assumes that periodic broadcasts and data traffic behave similarly.

By enabling link-layer acknowledgments and counting every acknowledged or unacknowledged packet, a link estimator can generate much more accurate estimates at a rate commensurate with the data traffic. These estimates are also inherently bidirectional. In Figure 2(a), EAR and ETX use feedback from the link layer for link estimation. Rather than inferring the ETX of a link by multiplying two control packet reception rates, with link-layer information on data traffic, an estimator can actually measure ETX. However, albeit accurate, relevant, and fast, sending data packets requires routing information, which in turn requires link quality estimates. This bootstrapping is best done at higher layers. Also, especially in dense networks, choosing the right set of links to estimate can be as important as the estimates themselves, which can get expensive if done solely at the link layer.

*4.1.4. Network Layer.* The physical layer can provide a rough measure of whether a link might be of high quality, enabling a link estimator to avoid spending effort on marginal or bad links. Once the estimator has gauged the quality of a link, the network layer can in turn decide which links are valuable for routing and which are not. This is important when space in the link table is limited. For example, geographic routing [Karp and Kung 2000] benefits from neighbors that are evenly spread in all directions, while

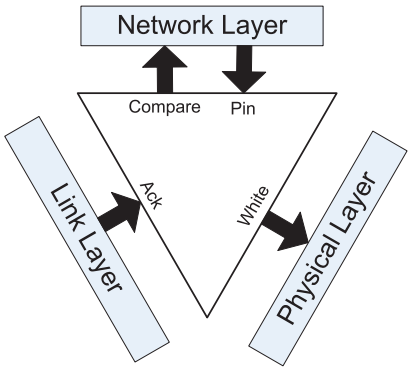



Fig. 5. The four-bit estimator interface. The link estimator, represented by the triangle in the center, uses four bits of information from the three layers. Outgoing arrows represent information the estimator requests on packets it receives. Incoming arrows represent information the layers actively provide.

the S4 routing protocol [Mao et al. 2007] benefits from links that minimize distance to beacons. One recent infamous wireless sensor network deployment [Langendoen et al. 2006] delivered only 2% of the data collected, in part due to disagreements between network and link layers on what links to use. For this reason, the MintRoute protocol (Figure 2(b)) integrates the link estimator into its routing layer. SP (Figure 2(d)) provides a rich interface for the network layer to inspect and alter the link estimator's neighbor table. The network layer can perform neighbor discovery and link quality estimation, but without access to information such as retransmissions, acknowledgments, or even packet decoding quality, this estimation becomes slow to adapt and expensive.

The following section describes how we can achieve cooperation between the link estimator and all three layers by using well-defined interfaces and only four bits of information. We then demonstrate in Section 7.3.2 that these interfaces allow significant performance gains through effective information exchange between the layers.

### 4.2. Estimator Interfaces

Figure 5 shows the interfaces each layer provides to a link estimator. Together, the three layers provide four bits of information: two bits for incoming packets and one bit each for transmitted unicast packets and link table entries.

The physical layer provides a single bit of information per received packet. If set, this *white bit* denotes that each symbol in the packet has a very low probability of decoding error. A set white bit implies that during the reception, the channel quality is high. The converse is not necessarily true (so a link layer may choose to not implement this interface): if the white bit is clear, then the channel quality may or may not have been high during the packet's reception.

The link layer provides one bit of information per transmitted packet: the *ack bit*. The link layer sets the ack bit on a transmit buffer when it receives a link-layer acknowledgment for that buffer. If the ack bit is clear, the packet may or may not have arrived successfully.

The network layer provides two bits of information, the *pin bit* and the *compare bit*. The pin bit applies to link table entries. When the network layer sets the pin bit on an entry, the link estimator cannot remove it from the table until the bit is cleared. The link estimator can ask a network layer for a compare bit on a packet. The compare bit indicates whether the route provided by the sender of the packet is better than the route provided by one or more of the entries in the link table. We describe how the 4B estimator uses the compare bit in Section 4.3.

The four bits represent what we believe to be the minimal information necessary for a link estimator to accurately estimate link qualities. Furthermore, we believe that the interfaces are simple enough that they can be implemented for most systems. For example, radios whose physical layers provide signal strength and noise can compute a signal-to-noise ratio for the white bit, using a threshold derived from the signal-to-noise ratio/bit error rate curve. Physical layers that report recovered bit errors or chip correlation can alternatively use this information. In the worst case, if radio hardware provides no such information, the white bit can be never set.

The interfaces introduce one constraint on the link layer: they require a link layer that has synchronous acknowledgments. While this might seem demanding, it is worthwhile to note that most commonly-used link layers, such as 802.11 and 802.15.4, have them. Novel or application-specific link layers must include acknowledgment to function in this model.

The compare bit requires that a network layer be able to tell whether the route through the transmitter of a packet is better than the routes through the current entries in the link table. As long as some subset of network layer packets, such as routing beacons, contain route quality information, the estimator can use the information to decide if it should insert or evict an entry in the table.

### 4.3. The Four-Bit (4B) Link Quality Estimator

We describe a hybrid estimator that combines the information provided by the three layers and link estimation beacons in order to provide accurate, responsive, and useful link estimates. The estimator maintains a small table (e.g., 10) of candidate links for which it maintains ETX values. It periodically broadcasts beacons that contain a subset of these links. Network layer protocols can also broadcast packets through the estimator, causing it to act as a layer 2.5 protocol that adds a header and footer between layers 2 and 3.

The estimator follows the basic table management algorithm outlined by Woo et al. [2003], with one exception: it does not assume a minimum transmission rate, since it can leverage outgoing data traffic to detect broken links. Link estimate broadcasts contain sequence numbers, which receivers use to calculate the beacon reception rate.

The estimator uses the white and compare bits to supplement the standard table replacement policy. When it receives a network layer routing packet from a node not in the estimation table yet, the estimator has to decide if it should insert an entry for this new node and possibly evict an existing entry to make room for the new node. It first checks if the white bit is set. If it is, suggesting a high-quality physical channel, it asks the network layer if the compare bit is set. Compare bit indicates if the new link can provide better routes than what is available through the current entries in the table. Thus, if both the white and compare bits are set, the estimator flushes a random unpinned entry from the table and replaces it with the sender of the current packet.

The estimator uses the ack bit to refine link estimates, combining broadcast and unicast ETX estimates into a hybrid value. We separately calculate the ETX value every $k_u$ or $k_b$ packets for unicast and broadcast packets, respectively. If $a$ out of $k_u$ packets are acknowledged by the receivers, the unicast ETX estimate is $\frac{k_u}{a}$. If $a = 0$, then the estimate is the number of failed deliveries since the last successful delivery. In our implementation, we use $k_u = 5$, which bounds the maximum ETX in one round of link ETX estimation to 5. The calculation for the broadcast estimate is analogous, but has an extra step. We use a windowed exponentially weighted moving average (EWMA) over the calculated *reception probabilities* and invert the consecutive samples of this average into ETX values. These two streams of ETX values coming from the

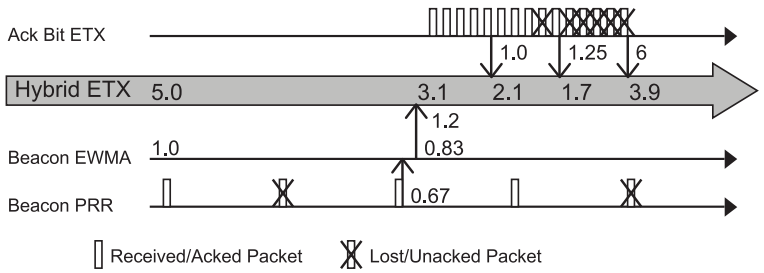



Fig. 6. The 4B estimator combines estimates of ETX separately for unicast and broadcast traffic with window sizes of $k_u = 5$ and $k_b = 2$, respectively. The latter are first themselves averaged before being combined. We show incoming packets as light boxes, marking dropped packets with an $\times$. The estimator calculates link estimates for each of the two estimators at the times indicated with vertical arrows.

two estimators are combined in a second EWMA, as shown in Figure 6. The result is a hybrid windowed-mean EWMA estimator. When there is heavy data traffic, unicast estimates dominate. When the network is quiet, broadcast estimates dominate.

Contrary to most pure broadcast-based estimators, the 4B estimator does not actively exchange and maintain bidirectional estimates using the beacons. Because the ack bit inherently allows the measurement of bidirectional characteristics of links, the 4B estimator can afford to only use the incoming beacon estimates as bootstrapping values for the link qualities, which are refined by the data-based estimates later. This is an important feature, as it decouples the in-degree of the nodes in the topology from the size of the link table. This decoupling allows the number of inbound links that are used for routing to exceed the table size, a feature that becomes critical in dense networks.

## 5. ROUTING

This section describes how CTP discovers, selects, and advertises routes.

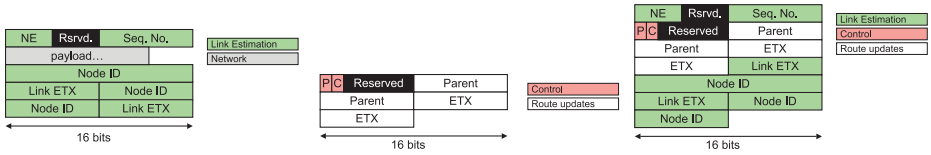
### 5.1. Implications of Agile Link Estimation

Because the wireless links are inherently dynamic, accurate link estimation requires agile link estimation. The 4B estimator uses a narrow, well-defined interface that allow a link estimator to use information from the physical, link, and network layers and shows significant improvements on cost and delivery ratio over the state of the art, while maintaining layered networking abstractions. Using the information provided by 4B presents some challenges, however. It reflects the underlying link dynamics, and its quality estimates can change in the time scale of data transmissions, as quickly as five packet times. This agility makes transient inconsistencies in the topology the norm rather than an exceptional condition.

CTP uses two mechanisms to address these inconsistencies. First, the routing algorithm has a variable timer to exchange path quality information with neighbors. We view the topology maintenance as a consistency problem and use the Trickle algorithm to control the dissemination of topology information. Exchanges are quick when changes in link quality are detected and slow down otherwise. Second, the forwarding is designed to work despite transient loops that will inevitably form. The forwarding logic detects gradient inconsistencies and loops as they are traversed by data packets and triggers topology adaptation while packets are in transit.

### 5.2. Route Computation and Selection

Figure 7 shows the CTP routing packet format nodes use to exchange topology information. The routing frame has two control bits and two fields. The two control bits are the pull bit (P) and the congested bit (C). We discuss the meaning and use of the P bit




(a) Link estimation frame. (b) Routing frame. (c) Complete routing frame.

Fig. 7. The CTP routing frame format.

later in this section. The C bit is reserved for potential future use in congestion control. Following these control bits are the fields describing the node's current parent and routing cost. A subset of the link estimation table entries follow these routing fields, as shown in Figure 7(c).

Changing routes too quickly can harm efficiency, as generating accurate link estimates requires time. To dampen the topology change rate, CTP employs hysteresis in path selection: it only switches routes if it believes the other route is significantly better than its current one, where "significantly" better is having an ETX at least 1.5 lower.

While hysteresis has the danger of allowing CTP to use suboptimal routes, noise in link estimates causes better routes to dominate a node's next hop selection. We use a simple example to illustrate how estimation noise causes the topology to gravitate towards and prefer more efficient routes despite hysteresis. Let a node A have two options, B and C, for its next hop, with identical costs of 3. The link to B has a reception ratio of 0.5 (ETX of 2.0), while the link to C has a reception ratio of 0.6 (ETX of 1.6).

If A chooses B as its next hop, its cost will be 5. The hysteresis just described will prevent A from ever choosing C, as the resulting cost, 4.6, is not $\leq 3.5$. However, the ETX values for the links to B and C are not static: they are discrete samplings of a random process. Correspondingly, even if the reception ratio on those links is completely stable, their link estimates will not be.

Assume, for simplicity's sake, that they follow a Gaussian distribution; the same logic holds for other distributions as long as their bounds are not smaller than the hysteresis threshold. Let $E_X$ be a sample from distribution X. As the average of the AB distribution is 2.0, but the average of the AC distribution is 1.6, the probability that $E_{AB} - E_{AC} > 1.5$ is much higher than the probability that $E_{AC} - E_{AB} > 1.5$. That is, the probability that AC will be at least 1.5 lower than AB is much higher than the probability that AB will be at least 1.5 lower than AC. Due to random sampling, at some point, AC will pass the hysteresis test and A will start using C as its next hop. Once it switches, it will take much longer for AB to pass the hysteresis test. While A will use B some of the time, C will dominate as the next hop.

### 5.3. Control Traffic Timing

When CTP's topology is stable, it relies on data packets to maintain, probe, and improve link estimates and routing state. Beacons, however, form a critical part of routing topology maintenance. First, since beacons are broadcasts, they are the basic neighbor discovery mechanism and provide the bootstrapping mechanism for neighbor tables. Second, there are times when nodes must advertise information, such as route cost changes, to all of their neighbors.

Because CTP separates link estimation from its control beacons, its estimator does not require or assume a fixed beaconing rate. This allows CTP to adjust its beaconing rate based on the expected importance of the beacon information to its neighbors. Minimizing broadcasts has the additional benefit that they are typically much more expensive to send with low-power link layers than unicast packets. When the routing topology is working well and the routing cost estimates are accurate, CTP can slow

its beaconing rate. However, when the routing topology changes significantly, or CTP detects a problem with the topology, it can quickly inform nearby nodes so they can react accordingly.

CTP sends routing packets using a variant of the Trickle algorithm [Levis et al. 2004]. It maintains a beaconing interval which varies between 64 ms and one hour. Whenever the timer expires, CTP doubles it, up to the maximum (one hour). Whenever CTP detects an event which indicates the topology needs active maintenance, it resets the timer to the minimum (64 ms). These values are independent of the underlying link layer. If a packet time is larger than 64 ms, then the timer simply expires several times until it reaches a packet time.

### 5.4. Resetting the Beacon Interval

As discussed in Section 3.3, three events cause CTP to reset its beaconing interval to the minimum length.

The simplest one is the P bit. CTP resets its beacon interval whenever it receives a packet with the P bit set. A node sets the P bit when it does not have a valid route. For example, when a node boots, its routing table is empty, so it beacons with the P bit set. Setting the P bit allows a node to "pull" advertisements from its neighbors, in order to quickly discover its local neighborhood. It also allows a node to recover from large topology changes which cause all of its routing table entries to be stale.

CTP also resets its beacon interval when its cost drops significantly. This behavior is not necessary for correctness: it is an efficiency optimization. The intuition is that the node may now be a much more desirable next hop for its neighbors. Resetting its beacon interval lets them know quickly.

The final and most important event is when CTP detects that there might be a routing topology inconsistency. CTP imposes an invariant on routes: the cost of each hop must monotonically decrease. Let p be a path consisting of $k$ links between node $n_0$ and the root, node $n_k$, such that node $n_i$ forwards its packets to node $n_{i+1}$. For the routing state to be consistent, the following constraint must be satisfied.

$$\forall i \in \{0, k-1\},\ ETX(n_i) > ETX(n_{i+1}),$$

where *ETX(x)* is the path ETX from node $x$ to the root.

CTP does not drop looping packets. Instead, it introduces a slight pause in forwarding, the length of the minimum beacon interval, and continues forwarding the packets. This ensures that it sends the resulting beacon before the data packet, such that the inconsistent node has a chance to resolve the problem. If there is a loop of length $L$, this means that the forwarded packet takes $L-1$ hops before reaching the node that triggered topology recovery. As that node has updated its routing table, it will pick a different next hop.

If the first beacon was lost, then the process will repeat. If it chooses another inconsistent next hop, it will trigger a second topology recovery. In highly dynamic networks, packets occasionally traverse multiple loops, incrementally repairing the topology, until finally the stale node picks a safe next hop and the packet escapes to the root. The cost of these rare events of a small number of transient loops is typically much less than the aggregate cost of general forwarding: improving routes through rare transient loops is worth the cost.

### 5.5. Beacon Suppression

In a dense network, it is often sufficient for a small subset of the nodes to transmit the beacons to allow all the nodes to discover a path to the root. If a node has received beacons from a few of its neighbors advertising the best path to the root, most likely it has already discovered the best possible path to the root. With each additional received

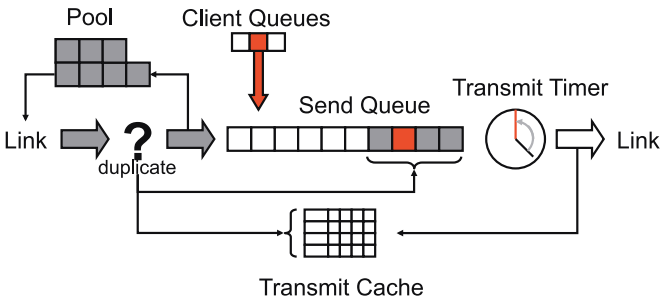



Fig. 8. The CTP forwarding path.

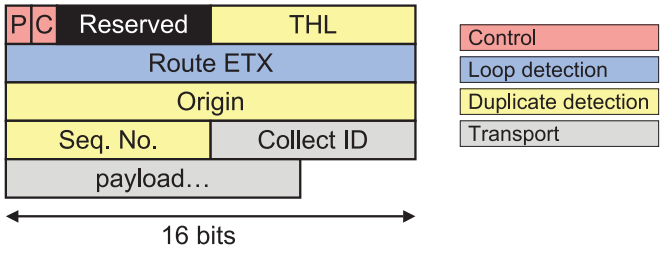



Fig. 9. The CTP data frame format.

advertisement, it becomes less likely that the node will discover a new and better path to the root. This decrease in marginal gain in path discovery from each additional beacon suggests that it is not necessary for all the nodes in the network to transmit beacons while still finding the best paths in the network.

CTP uses the packet beacon suppression technique, adapted from the Trickle algorithm, to limit the control overhead. Each node keeps a counter of beacons received after the last beacon transmission. If less than $k$ routing beacons were received during the current interval, the node transmits a beacon. Otherwise, the node does not transmit a beacon because there were already more than $k$ beacon transmissions in the neighborhood during the current beaconing interval and the best paths are likely already discovered. In both cases, the node resets the packet counter and makes the described beacon transmission decision after each period. The main challenge in beacon suppression is finding the value of the constant $k$. In Section 7.3.11, we present experiments that explore how the choice of $k$ impacts CTP's performance.

## 6. FORWARDING

This section describes CTP's data plane. Unlike the control plane, which is a set of consistency algorithms, the concerns of the data plane are much more systems and implementation oriented. In the previous section, we described the important role that the data plane plays in detecting inconsistencies in the topology and resetting the beacon interval to fix them. In this section, we describe four mechanisms in the data plane that deal with efficiency, robustness, and reliability: per-client queueing, a hybrid send queue, a transmit timer, and a packet summary cache. Figure 8 shows the CTP data path and how these four mechanisms interact.

Figure 9 shows a CTP data frame, which has an eight-byte header. The data frame shares two fields with the routing frame, the control field and the route ETX field. The 8-bit *time has lived*, or THL field, is the opposite of a TTL: it starts at zero at an end point and each hop increments it by one. A one-byte application dispatch identifier

called Collect ID, allows multiple clients, that is, applications or services in the upper layer, to share a single CTP layer.

CTP uses a very aggressive retransmission policy. By default, it will retransmit a packet up to 32 times. This policy stems from the fact that all packets have the same destination, and, thus, the same next hop. The outcome of transmitting the next packet in the queue will be the same as the current one. Instead of dropping, CTP combines a retransmit delay with proactive topology repair to increase the chances of delivering the current packet. In applications where receiving more recent packets is more important than receiving nearly all packets, the number of retransmissions can be adjusted without affecting the routing algorithm.

### 6.1. Per-Client Queueing

CTP maintains two levels of queues. The top level is a set of one-deep client queues. CTP allows each client to have a single outstanding packet. If a client needs additional queueing, it must implement it on top of this abstraction. These client queues do not actually store packets; they are simple guards that keep track of whether a client has an outstanding packet. When a client sends a packet, the client queue checks whether it is available. If so, the client queue marks itself busy and passes the packet down to the hybrid send queue. A one-deep queue per client provides isolation, as a single client cannot fill the send queue and starve others.

### 6.2. Hybrid Send Queue

CTP's lower-level queue contains both route through- and locally-generated traffic (as in Woo and Culler [2001]), maintained by a FIFO policy. This hybrid send queue is of length $C + F$, where C is the number of CTP clients and $F$ is the size of the buffer pool for forwarded packets. If it is full, this means the forwarding path is using all $F$ of its buffers and all $C$ clients have an outstanding packet.

When CTP receives a packet to forward, it first checks if the packet is a duplicate: Section 6.4 describes this process. If the packet is a duplicate, it returns it to the link layer without forwarding it. If the packet is not a duplicate, CTP checks if it has a free packet buffer in its memory pool. If it has a free buffer, it puts the received packet on the send queue and passes the free buffer to the link layer for the next packet reception.

### 6.3. Transmit Timer

Multihop wireless protocols encounter self-interference, where a node's transmissions collide with prior packets it has sent which other nodes are forwarding. For a route of nodes $A \rightarrow B \rightarrow C \rightarrow \ldots$, self-interference can easily occur at $B$ when $A$ transmits a new packet at the same time $C$ forwards the previous one [Li et al. 2001].

CTP tries to minimize self-interference by rate-limiting its transmissions. In the preceding idealized scenario, where only the immediate children and parent are in the transmission range of a transmitter, if $A$ waits at least two packet times between transmissions, then it will avoid self-interference, as $C$ will have finished forwarding [Woo and Culler 2001]. While real networks are more complex (the interference range can be greater than the transmit range), two packet times represents the minimum timing for a flow longer than two hops.

The transmission wait timer depends on the packet rate of the radio. If the expected packet time is $p$, then CTP waits in the range of $(1.5p, 2.5p)$, such that the average wait time is $2p$ and there is randomization to prevent edge conditions due to MAC backoff or synchronized transmissions. These wait times are helpful in minimizing occassional self-interference in a network servicing low to moderate data rates. On the other hand, these wait times could introdue inefficiency during bulk transfer or in a network servicing high data rates.

### 6.4. Transmit Cache

Link-layer acknowledgments are not perfect: they are subject both to false positives and false negatives. False negatives cause a node to retransmit a packet which is already in the next hop's forwarding queue. CTP needs to suppress these duplicates, as they can increase multiplicatively on each hop. Over a small number of hops, this is not a significant issue, but in the face of the many hops of transient routing loops, this leads to an exponential number of copies of a packet that can overflow all queues in the loop.

Since CTP forwards looping packets in order to actively repair its topology, CTP needs to distinguish link-layer duplicates from looping packets. It detects duplicates by examining three values: the origin address, the origin sequence number, and the THL. Looping packets will have matching address and sequence number, but will have a different THL (unless the loop was a multiple of 256 hops long), while link-layer duplicates have matching THL values.

When CTP receives a packet to forward, it scans its send queue for duplicates. It also scans the transmit cache. This cache contains the just-described three-tuples of the $N$ most recently forwarded packets. The cache is necessary for the case where duplicates are arriving more slowly than the rate at which the node drains its queue: in this case, the duplicate will no longer be in the send queue.

For maximal efficiency, the first intuition suggests the transmit cache should be as large as possible. We have found that, in practice and even under high load, having a cache size of four slots is enough to suppress most ($>$99%) duplicates on the testbeds that we used for experiments. A larger cache improves duplicate detection slightly but not significantly enough to justify its cost on memory-constrained platforms.

## 7. EVALUATION

This section evaluates how the mechanisms just described, namely four-bit link estimation, adaptive control traffic rate, datapath validation, and the data plane optimizations, combine to achieve the four goals from Section 1: reliability, robustness, efficiency, and hardware independence.

We evaluate our implementation of CTP on 13 different testbeds, encompassing seven platforms, six link layers and multiple densities and frequencies. Despite having anecdotal evidence of several successful real-world deployments of CTP, these results focus on publicly available testbeds, because they represent at least theoretically reproducible results. The hope is that different testbed environments we examine sufficiently capture a reasonable degree of variation in node density, radio technology, wireless environment, and deployment topology.

### 7.1. Testbeds

Table I summarizes the 13 testbeds we use. It lists the name, platform, number of nodes, physical span, and topology properties of each network. Motelab is at Harvard University. Twist is at TU Berlin. Wymanpark is at Johns Hopkins University. Tutornet is at the University of Southern California. Neteye is at Wayne State University. Kansei is at Ohio State University. Vinelab is at the University of Virginia. Quanto is at UC Berkeley. Mirage is at Intel Research Berkeley. Indriya is at the National University of Singapore. Finally, Blaze is at Rincon Research. Some testbeds (e.g., Mirage) are on a single floor, while others (e.g., Motelab) are on multiple floors. Unless otherwise noted, the results of the detailed experiments are from the Tutornet testbed.

To roughly quantify the link-layer topology of each testbed, we ran an experiment where each node broadcasts every 16 s. The interval is randomized to avoid collisions. To compute the link-layer topology, we consider all links that delivered at least one

Table I. Tested Configuration and Topology Properties, from Most to Least Dynamic

| Testbed | Platform | Nodes | Size $m^2$ or $m^3$ | Degree Min | Degree Max | PL | Cost | Cost PL | Churn node·hr |
|---|---|---|---|---|---|---|---|---|---|
| Tutornet (16) | Tmote | 91 | 50×25×10 | 10 | 60 | 3.12 | 5.91 | 1.90 | 31.37 |
| Wymanpark | Tmote | 47 | 80×10 | 4 | 30 | 3.23 | 4.62 | 1.43 | 8.47 |
| Motelab | Tmote | 131 | 40×20×15 | 9 | 63 | 3.05 | 5.53 | 1.81 | 4.24 |
| Kansei[a] | TelosB | 310 | 40×20 | 214 | 305 | 1.45 | – | – | 4.34 |
| Mirage | Mica2dot | 35 | 50×20 | 9 | 32 | 2.92 | 3.83 | 1.31 | 2.05 |
| NetEye | Tmote | 125 | 6×4 | 114 | 120 | 1.34 | 1.40 | 1.04 | 1.94 |
| Mirage | MicaZ | 86 | 50×20 | 20 | 65 | 1.70 | 1.85 | 1.09 | 1.92 |
| Quanto (15) | Quanto | 49 | 35×30 | 8 | 47 | 2.93 | 3.35 | 1.14 | 1.11 |
| Twist | Tmote | 100 | 30×13×17 | 38 | 81 | 1.69 | 2.01 | 1.19 | 1.01 |
| Twist | eyesIFXv2 | 102 | 30×13×17 | 22 | 100 | 2.58 | 2.64 | 1.02 | 0.69 |
| Vinelab | Tmote | 48 | 60×30 | 6 | 23 | 2.79 | 3.49 | 1.25 | 0.63 |
| Indriya | TelosB | 126 | 66×37×10 | 1 | 36 | 2.82 | 3.12 | 1.11 | 0.05 |
| Tutornet | Tmote | 91 | 50×25×10 | 14 | 72 | 2.02 | 2.07 | 1.02 | 0.04 |
| Blaze[b] | Blaze | 20 | 30×30 | 9 | 19 | 1.30 | – | – | – |

[a] Packet cost logging failed on ten nodes.

[b] Blaze instrumentation does not provide cost and churn information.

*Note*: Cost is transmissions per delivery and PL is path length, the average number of hops a data packet takes. Cost/PL is the average transmissions per link. All experiments are on 802.15.4 channel 26 except for the Quanto testbed (channel 15) and one of the Tutornet experiments (channel 16).

packet. The minimum and maximum degree column in Table I are the in-degree of the nodes with the smallest and largest number of links, respectively. We consider this very liberal definition of a link because it is what a routing layer or link estimator must deal with: a single packet can add a node as a candidate, albeit perhaps not for long.

As the differing delivery results on Tutornet in Table II indicate, the link stability and quality results should not be considered definitive for all experiments. For example, most 802.15.4 channels share the same frequency bands as 802.11: 802.15.4 on an interfering channel has more packet losses and higher link dynamics than on a non-interfering one. For example, Tutornet on channel 16 has the highest churn, while Tutornet on channel 26 has the lowest. We revisit the implications of this effect in Section 7.3.10.

To roughly quantify link stability and quality, we ran CTP with an always-on link layer for three hours and computed three values: PL, the average path length (hops a packet takes to the collection root); the average cost (transmissions/delivery); and the node churn, or rate at which nodes change parents. We also look at cost/PL, which indicates how any transmissions CTP makes on average per hop. Wide networks have a large PL. Networks with many intermediate links or sparse topologies have a high cost/PL ratio (sparsity means a node might not have a good link to use). Networks with more variable links or very high density have a high churn (density can increase churn because a node has more parents to try and choose from). As the major challenge adaptive beaconing and datapath validation seek to address is link dynamics, we order the testbeds from the highest (Tutornet on channel 16) to the lowest (Tutornet on channel 26) churn.

We use all available nodes in every experiment. In some testbeds, this means the set of nodes across experiments is almost but not completely identical, due to backchannel connectivity issues. However, we do not prune problem nodes. In the case of Motelab, this approach greatly affects the computed average performance, as some nodes are barely connected to the rest of the network.

Table II. Summary of Experimental Results across the Testbeds

| Testbed | Frequency | MAC | IPI | Avg Delivery | 5th% Delivery | Loss |
|---|---|---|---|---|---|---|
| Motelab | 2.48GHz | CSMA | 16s | 94.7% | 44.7% | Retransmit |
| Motelab | 2.48GHz | BoX-50ms | 5m | 94.4% | 26.9% | Retransmit |
| Motelab | 2.48GHz | BoX-500ms | 5m | 96.6% | 82.6% | Retransmit |
| Motelab | 2.48GHz | BoX-1000ms | 5m | 95.1% | 88.5% | Retransmit |
| Motelab | 2.48GHz | LPP-500ms | 5m | 90.5% | 47.8% | Retransmit |
| Tutornet (26) | 2.48GHz | CSMA | 16s | 99.9% | 99.9% | Queue |
| Tutornet (16) | 2.43GHz | CSMA | 16s | 95.2% | 92.9% | Queue |
| Tutornet (16) | 2.43GHz | CSMA | 22s | 97.9% | 95.4% | Queue |
| Tutornet (16) | 2.43GHz | CSMA | 30s | 99.4% | 98.1% | Queue |
| Wymanpark | 2.48GHz | CSMA | 16s | 99.9% | 99.9% | Retransmit |
| Indriya | 2.48GHz | CSMA | 8s | 99.9% | 99.8% | Retransmit |
| NetEye | 2.48GHz | CSMA | 16s | 99.9% | 96.4% | Retransmit |
| Kansei | 2.48GHz | CSMA | 16s | 99.9% | 99.6% | Retransmit |
| Vinelab | 2.48GHz | CSMA | 16s | 99.9% | 99.9% | Retransmit |
| Quanto | 2.425GHz | CSMA | 16s | 99.9% | 99.9% | Retransmit |
| Twist (Tmote) | 2.48GHz | CSMA | 16s | 99.3% | 84.9% | Retransmit |
| Twist (Tmote) | 2.48GHz | BoX-2s | 5m | 98.3% | 78.1% | Retransmit |
| Mirage (MicaZ) | 2.48GHz | CSMA | 16s | 99.9% | 99.8% | Queue |
| Mirage (Mica2dot) | 916.4MHz | B-MAC | 16s | 98.9% | 97.5% | Ack |
| Twist (eyesIFXv2) | 868.3MHz | CSMA | 16s | 99.9% | 99.9% | Retransmit |
| Twist (eyesIFXv2) | 868.3MHz | SpeckMAC-183ms | 30s | 94.8% | 44.7% | Queue |
| Blaze | 315MHz | B-MAC-300ms | 4m | 99.9% | – | Queue |

*Note*: The first section compares how different low-power link layers and settings affect delivery on Motelab. The second section compares how the 802.15.4 channel affects delivery on Tutornet. The third section shows results from other TelosB/TMote testbeds, and the fourth section shows results from testbeds with other platforms. In all settings, CTP achieves an average delivery ratio of over 90%. On Motelab, a small number of nodes (the 5th percentile) have poor delivery due to poor connectivity as described in Section 7.3.1.

### 7.2. Experimental Methodology

We modified three variables across all of the experiments: the interpacket interval (IPI) with which the application sends packets with CTP, the MAC layer used, and the node ID of the root node. Generally, to obtain longer routes, we picked roots that were in one corner of a testbed.

We used six different MAC layers. On TelosB/TMote [Polastre et al. 2005] nodes, we used the standard TinyOS 2.1.0 CSMA layer at full power (CSMA), the standard TinyOS 2.1.0 low-power-listening layer, BoX-MAC (BoX) [Moss and Levis 2008], and low-power probing (LPP) [Musaloiu-E. et al. 2008]. On mica2dot [Crossbow Technology 2006] nodes, we used the standard TinyOS 2.1.0 BMAC layer (B-MAC) for CC1000 radio [Polastre et al. 2004]. On Blaze nodes, we used the standard TinyOS 2.1.0 BMAC layer (B-MAC) for CC1100. On eyesIFXv2 nodes, we use both the TinyOS 2.1.0 CSMA layer (CSMA) as well as TinyOS 2.1.0 implementation of SpeckMAC [Wong and Arvind 2006]. In the cases where we use low-power link layers, we report the interval. For example, "BoX-1s" means BoX-MAC with a check interval of one second, while "LPP-500ms" means low-power probing with a probing interval of 500 ms.

Evaluating efficiency is difficult, as temporal dynamics prevent knowing what the optimal route would be for each packet. Therefore, we evaluate efficiency as a comparative measure. We compare CTP with the TinyOS 2.1 implementation of MultiHopLQI, a well-established, well-tested, and widely used collection layer that is part of the

TinyOS release. As MultiHopLQI has been used in recent deployments, for example, on a volcano in Ecuador [Werner-Allen et al. 2006], we consider it a reasonable comparison. Other notable collection layers, such as Hyper [Schoellhammer et al. 2006] and Dozer [Burri et al. 2007] are either implemented in TinyOS 1.x (Hyper) or are closed source and specific to a platform (Dozer, TinyNode). As TinyOS 2.x and 1.x have different packet scheduling and MAC layers, we found that comparing with 1.x protocols unfairly favors CTP.

### 7.3. Experiments and Results

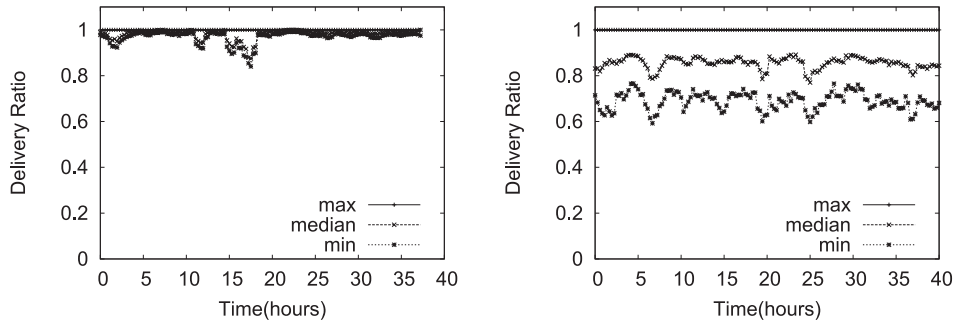
*7.3.1. Reliable, Robust, and Hardware-Independent.* Before evaluating the contribution of each mechanism to the overall performance of CTP, we first look at high-level results from experiments across multiple testbeds, as well as a long duration experiment. Table II shows results from 22 experiments across the 13 testbeds. In these experiments, we chose IPI values well below saturation, such that the delivery ratio is not limited by available throughput. The Loss column describes the dominant cause of packet loss: retransmit means CTP dropped a packet after 32 retransmissions, queue means it dropped a received packet due to a full forwarding queue, and ack means it heard link layer acknowledgments for packets that did not arrive at the next hop.

In all cases, CTP maintains an average delivery ratio above 90%: it meets the reliability goal. The lowest average delivery ratio is for Motelab using low-power probing (500 ms), where it is 90.5%. The second lowest is Motelab using BoX-MAC (50 ms), at 94.4%. In Motelab, retransmit is the dominant cause of failure: retransmission drops represent CTP sending a packet 32 times yet never delivering it. Examining the logs, this occurs because some Motelab nodes are only intermittently and sparsely connected (its comparatively small minimum degree of 9 in Table I reflects this). Furthermore, CTP maintains this level of reliability across all configurations and settings: it meets the robustness goal. CTP requires configuration of two constants across different radio chips—the threshold on physical measurement (LQI, RSSI) that determines if a channel is of high quality (white-bit) and rate-limiting interpacket delay in the forwarding engine. Because it requires these minimal reconfiguration across radio chips, we can port CTP to different platforms with little effort. Thus, CTP meets the hardware independence goal. We therefore focus comparative evaluations on MultiHopLQI.

To determine the consistency of delivery ratio over time, in Figure 10(a), we show the result from one experiment when we ran CTP for over 37 hours. The delivery ratio remains consistently high over the duration of the experiment. Figure 10(b) shows the result from a similar experiment with MultiHopLQI. Although MultiHopLQI's average delivery ratio was 85%, delivery is highly variable over time, occasionally dipping to 58% for some nodes. In the remainder of this section, we evaluate through detailed experiments how the different techniques we use in CTP contribute to its higher delivery ratio, while maintaining low control traffic rates and agility in response to changes in the topology.

*7.3.2. Four-Bit Link Estimation.* We first study the performance of the four-bit link estimator (4B) by replacing the estimator used in CTP with the estimator used in MultiHopLQI, and by removing some bits from the 4B estimator depending on the experiment. MultiHopLQI uses the Link Quality Indication (LQI), a feature of the CC2420 radio, and for that radio, it was the best performing collection implementation for TinyOS. We also compare the results to CTP-Beacons, which derives link estimates solely from the periodic beacons.

In our comparison, we run all three protocols on the Mirage testbed. Transmit power is set at 0 dBm unless otherwise specified. Each node sends a collection packet every 16 seconds to the sink, with some jitter to avoid packet synchronization with other




(a) Delivery ratio for CTP. (b) Delivery ratio for MultiHopLQI.

Fig. 10. CTP has a consistently higher delivery ratio than MultiHopLQI. In these plots we show for each time interval the minimum, median, and maximum delivery ratio across all nodes.

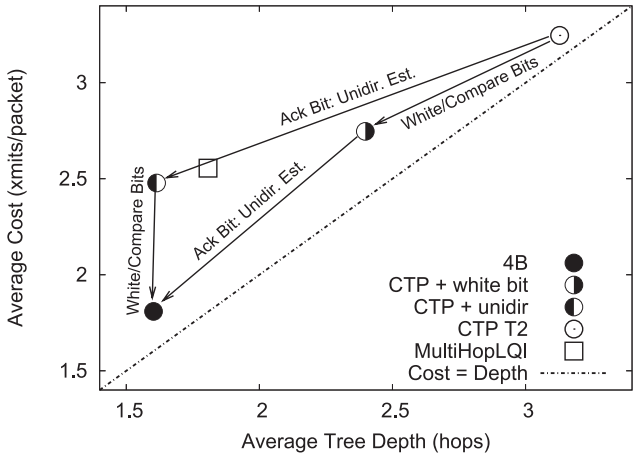



Fig. 11. Exploring the link estimation design space: adding the ack bit and/or the white and compare bits to CTP-Beacons (T2 in this figure) decreases cost and the average depth of a node in the routing tree.

nodes. This creates many concurrent flows in the network, converging at the sink, but not overloading the network, as our focus is on low data-rate sensor network systems. The experiments lasted between 40 and 69 minutes. We also ran these experiments on Tutornet, where we obtained similar results.

We first explore how the addition of each of the bits in Section 4.2 impacts cost and path length. We compare the CTP-Beacons with 4B, and two intermediary implementations. The uppermost-right point of Figure 11 shows the cost and path length of CTP-Beacons running on the Mirage testbed. Adding unidirectional link estimation to CTP-Beacons with the ack bit reduces the average path length by 93%, and reduces cost by 31%. Unidirectional estimates decouple in-degree from the link table size, hence the large decrease in depth.

Adding the white and compare bits to the resulting protocol decreases cost to 55% of the original CTP-Beacons, because of improved parent selection. Adding only the white and compare bits to CTP-Beacons provides reductions of 15% in cost and 23% in path length. The figure also shows MultiHopLQI's cost and depth in the same testbed for comparison. It is only when we use information from the three layers that 4B does better than MultiHopLQI. 4B has 29% lower cost and 11% shorter paths than MultiHopLQI. On experiments on Tutornet, 4B's cost and average depth were, respectively,

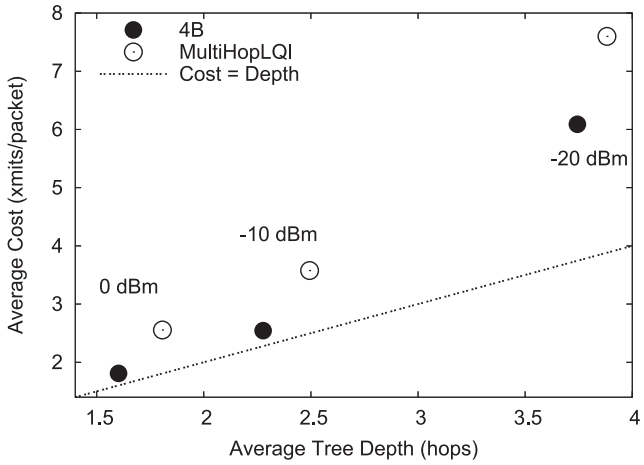



Fig. 12. Average node depth and cost for MultiHopLQI and 4B for decreasing transmit powers on the Mirage testbed. 4B reduces cost by 19–28%.

44% and 9.7% lower than MultiHopLQI's. Thus, we can be confident that these trends on lower average cost and depth with 4B compared to MultiHopLQI are consistent across the testbeds. The trees produced by 4B are very similar qualitatively and in average path length to the trees produced by CTP-Beacons with unrestricted link tables (Figure 3).

Figure 12 compares the cost and average path length of 4B and MultiHopLQI in the Mirage testbed as transmit power varies from −20 dBm to 0 dBm. In each protocol, we see that both average path length and cost increase with decreasing transmit power, as nodes need to route packets over more hops to get to the sink. 4B's improvement in cost over MultiHopLQI ranges from 29% to 11%, and the improvement in average depth from 11% to 3.5%. 4B's cost, for the 0 and -10 dBm cases, is at most 13% above the lower bound, while it is at most 43.4% above the lower bound for MultiHopLQI. In a network with many hops, both protocols become less efficient. The relative increase in cost (62% above average depth for 4B and 95% for MultiHopLQI) are indicative of retransmissions and/or losses. Even with similar tree depths, however, 4B is able to select better links in this situation.

Figure 13 looks at the per-node distribution of delivery ratios for the same experiments, and gives some insight on why the costs in Figure 12 grow faster than the average path length. For 0 and 10 dBm, 4B showed an average delivery ratio above 99.9%, with minimum 99.3%. For 0 dBm, MultiHopLQI's average delivery ratio over all nodes was 95.9%, with the worst node at 64%. As the transmit power decreases, the relative impact of RF noise on wireless performance increases, creating localized asymmetries in the network. As in the example of Section 4.1.1, MultiHopLQI's performance drops as some of this variation in link quality is not captured by the physical layer link quality indicator. The much smaller number of packet losses in 4B, even at −20dBm, indicates that most of the inefficiency seen in its cost is due to retransmissions, rather than loss. This suggests that the estimator is agile enough to notice packet losses and trigger the switch to a new route.

*7.3.3. Efficiency.* A protocol that requires a large number of transmissions is not well-suited for duty-cycled networks. We measure data delivery efficiency using the cost metric, which accounts for all the control and data transmissions in the network normalized by the packets received at the sink. This metric gives a rough measure of

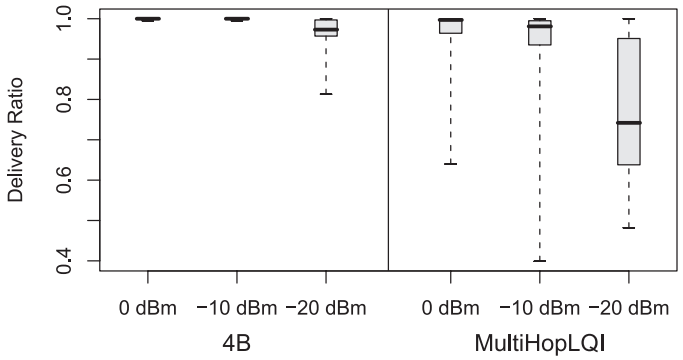



Fig. 13. Boxplots of per-node delivery distributions at decreasing transmit power, for both MultiHopLQI and 4B. Whiskers show the minimum and maximum values. Boxes show the 1st and 3rd quartiles. The line is the median. 4B maintains much higher and consistent delivery rates across the network.

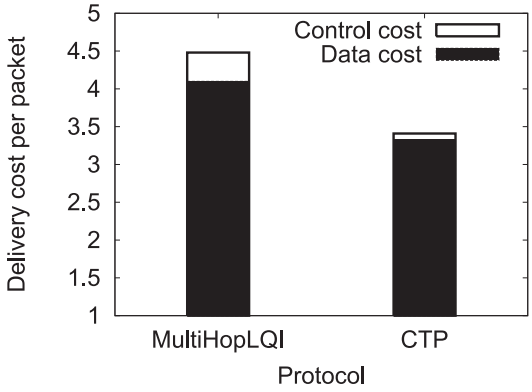



Fig. 14. CTP's cost is 24% lower than MultiHopLQI. Control cost is 73% lower.

the energy spent delivering a single packet to the sink. Figure 14 compares the average delivery cost for CTP and MultiHopLQI computed over 70,000 packets. CTP cost is 24% lower than that of MultiHopLQI. The figure also shows that control packets for CTP occupy a much smaller fraction of the cost than MultiHopLQI (2.2% vs. 8.4%). The decrease in data transmissions is a result of good route selection and agile route repair. The decrease in control transmissions is due to CTP's adaptive beaconing.

*7.3.4. Adaptive Control Traffic.* Figure 15 shows CTP's and MultiHopLQI's control traffic from separate five-hour experiments on Tutornet. CTP's control traffic rate is high at network startup, as CTP probes and discovers the topology, but decreases and stabilizes over time. MultiHopLQI sends beacons at a fixed interval of 30 s. Using a Trickle timer allows CTP to send beacons as quickly as every 64 ms and quickly respond to topology problems within a few packet times. Compared to 30 s, this is a 99.8% reduction in response time. By adapting its control rate and slowing down when the network is stable, however, CTP has a much lower control packet rate than MultiHopLQI. In this experiment, CTP's long-term average control packet rate was over 65% lower than that of MultiHopLQI.

Lower beacon rates are generally at odds with route agility. During one experiment, we introduced four new nodes in the network 60 minutes after the start. Figure 16

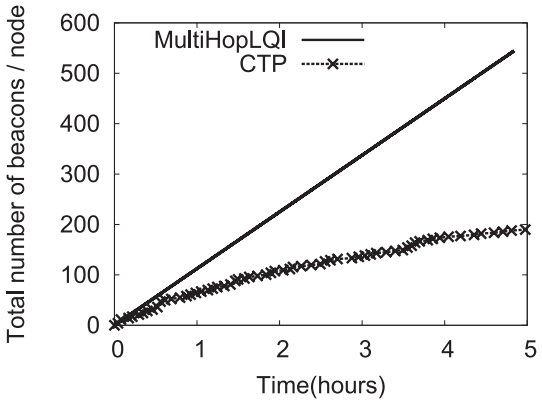



Fig. 15. CTP's beaconing rate decreases and stabilizes over time. It is significantly smaller than MultiHopLQI's over the long run.

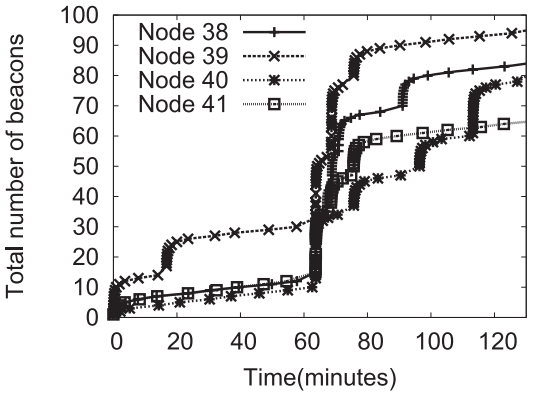
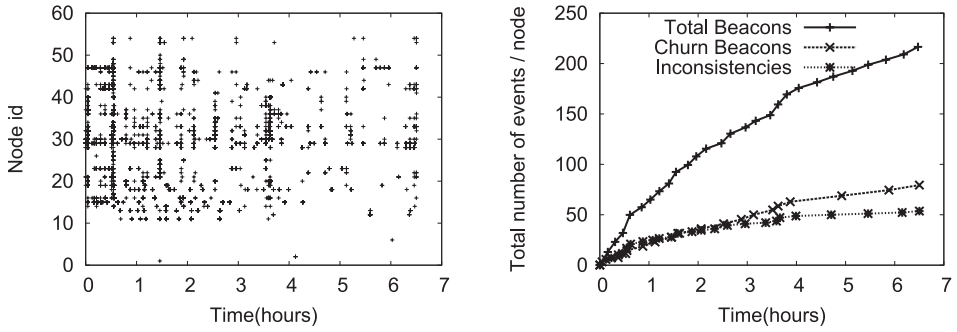



Fig. 16. Number of beacons for selected nodes in the neighborhood of the new node. There is a big jump in control traffic shortly after four new nodes are introduced and it levels off.

shows that the control overhead for selected nodes in the vicinity of the new nodes increases immediately after the nodes were introduced, as beacons are sent rapidly. The beacon rate decays shortly afterward. The increase in beaconing rate (in response to the pull bit) was localized to the neighborhood of the nodes introduced, and produced fast convergence. New nodes were able to send collection packets to the sink within four seconds after booting.

*7.3.5. Topology Inconsistencies.* Next we look at how route inconsistencies, as defined in Section 5.4, are distributed over space and time, and their impact on control overhead. Figure 17(a) shows inconsistencies detected by each node in an experiment over a 6.5-hour period. Inconsistencies are temporally correlated across nodes, and typically constrained to a subset of nodes. The lowest curve in Figure 17(b) shows the cumulative count of route inconsistencies in the same experiment, and how the rate decreases over time. In the beginning, most of the inconsistencies are due to discovery and initial rounds of path selection. Over time, link dynamics are the dominant cause of inconsistencies. We have also observed a similar trend in number of parent changes: frequent changes in the beginning as the nodes discover new links and neighbors and fewer changes once the network has selected high quality routes.




(a) Inconsistent routing states over time and across nodes. Each point is a detected route inconsistency.

(b) Control overhead from route inconsistencies.

Fig. 17. Inconsistent routing state and resulting control overhead.

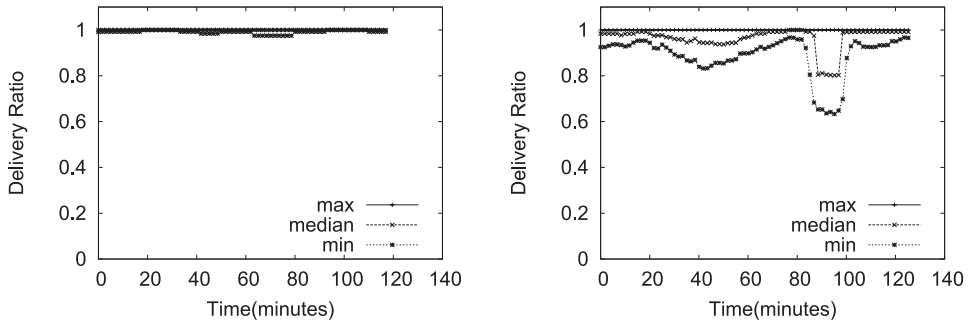
When a node detects such an inconsistency, it resets its beacon timer. The top curve in Figure 17(b) shows the total number or routing beacons sent (Total Beacons). The middle curve, Churn Beacons, is the subset of these beacons sent by the Trickle timer when a parent change resets the interval. In addition to these Churn beacons, Total beacons also include beacons sent in response to Trickle timer expiration, reception of packets with P-bit, and detection of inconsistency that did not result in parent change. Thus, the difference between the Total Beacons and Churn Beacons curve provides an upper bound on the number of beacons sent due to inconsistencies. In 6.5 hrs, the nodes sent 12,299 total beacons while they detected 3,025 inconsistencies and triggered 4,485 beacons due to parent change: CTP sent 2.6 beacons per inconsistency detected in order to repair and reestablish the path to the root.

*7.3.6. Robustness to Failure.* To measure how effectively the routes can adapt to node failures, we ran CTP for two hours with an application sending packets every eight seconds. After 60 minutes, we removed the ten nodes that were forwarding the most packets in the network. CTP uses the four-bit link estimator, which reflects changes in the topology in a few packet times. This resets the trickle timers and causes rapid route convergence around the failure.

Figure 18(a) plots the minimum, median, and maximum delivery ratio across nodes over time. The figure shows only a tiny change in delivery ratio due to the disruption: the minimum delivery ratio across the network drops to 98%. Fifteen nodes dropped one or two packets each right after the disruption, and most nodes found new routes in under 1 s. The ten-minute dip in the graph is an artifact of the sliding window we used to calculate average delivery ratio. The median delivery ratio remained at 100%.

Figure 18(b) shows the result of a similar experiment with MultiHopLQI. After 80 minutes, we removed ten nodes that were forwarding the most packets. The resulting disruption caused the delivery ratio of some nodes to drop as low as 60%, while the median delivery ratio dropped to 80%.

*7.3.7. Agility.* The prior experiment shows that CTP can quickly route around node failures when there is a constant stream of traffic. To delve deeper into how CTP adapts to sudden topology changes, we ran a slightly different experiment. We ran CTP on Tutornet with each node generating a data packet every 8 s for six minutes allowing CTP to settle on a good routing tree while it was delivering 100% of the packets. Then we stopped generating data traffic on all the nodes for 14 minutes. At the 20$^{th}$ minute, we removed (erased the program running on the mote) node 26 from the network and




(a) Nodes fail at 60 minutes and CTP does not observe any significant disruption.

(b) Nodes fail at 80 minutes and MultiHopLQI's median delivery drops to 80% for 10 minutes.

Fig. 18. Robustness of CTP and MultiHopLQI when the ten most heavily forwarding nodes fail.

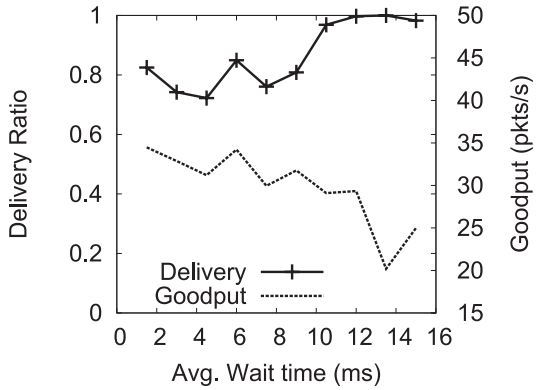



Fig. 19. Effect of a per-hop rate-limiting transmit timer on goodput and delivery ratio on the CC2420 radio.

shortly thereafter made node 53 (node 26's child in the routing tree) start sending data packets. As expected, packet transmissions from node 53 to nonexistent node 26 fails.

We found that after twelve packet transmissions (325 ms), CTP switched to node 40 as its new parent. Thus, although the beacon rate in the network had decreased to one beacon every eight minutes, CTP was able to quickly (in 325 ms) select a new parent when its existing parent was removed from the network. CTP remains efficient even when the beacon interval decays to tens of minutes, maintaining the ability to react to topology changes within a few packet transmission times.

*7.3.8. Transmit Timer.* CTP pauses briefly between packet transmissions to minimize self-interference, as described in Section 6.3. Here we show how we established this value for the CC2420 radio and quantify its benefit to CTP's reliability.

Figure 19 shows how the duration of a transmission wait timer affects a single node flow on channel 26 in the Tutornet testbed. In this experiment, a single node sends packets as fast as it can across three-hops to the data sink. The transmission timers in Figure 19 range from $[1, 2]$ ms (avg. 1.5 ms) to $[10, 20]$ ms (avg. 15 ms). At values below $[7, 14]$ ms (i.e., avg. 10.5 ms), delivery dips below 95%.

Although goodput increases slightly with smaller transmit timers, this benefit comes at a significant cost: the delivery ratio drops as low as 72%, which does not satisfy the reliability requirement. However, as the timer length increases past $[8, 16]$ ms, goodput

Table III. Result on How Channel Selection Effects CTP's Performance on Tutornet

| Chan. | Freq. | Delivery | PL | Cost | Cost / PL | Churn / node-hr |
|---|---|---|---|---|---|---|
| 16 | 2.43GHz | 95.2% | 3.12 | 5.91 | 1.894 | 31.37 |
| 26 | 2.48GHz | 99.9% | 2.02 | 2.07 | 1.025 | 0.04 |

*Note*: Channel 16 overlaps with Wi-Fi; channel 26 does not.

drops significantly as the timer introduces idleness. Therefore, CTP uses 7–14 ms (1.5–3 average packet times) as its wait timer for the CC2420 radio.

Similarly, it uses, 1.5–3 average packet times as its transmit timer on other radios. We have found that this setting works across the platforms and testbeds in these experiments, but factors, such as load, density, and link qualities, ultimately determine the maximum rate that a path can accommodate. Some MACs might introduce large delays between the packets, in which case, CTP transmit timers can be made smaller.

Although CTP's primary goal is not high throughput traffic, its use of transmit timers allows it to avoid collisions while under high load. Transmit timers are insufficient for end-to-end reliability: bottleneck links of low PRR can overflow transmit queues. Robust end-to-end reliability requires higher-layer congestion and flow control [Kim et al. 2007; Musaloiu-E. et al. 2008; Rangwala et al. 2006; Werner-Allen et al. 2006], but CTP's transmit timers make its reliability more robust to the high load these protocols can generate.

*7.3.9. Transmit Cache.* We evaluate the effect of the transmit cache by running two experiments on Tutornet. Both experiments use the CSMA link layer, send packets every 8s, and use 802.15.4 channel 16. The first experiment uses standard CTP; the second disables its transmit cache. Standard CTP has an average cost of 3.18 packets/delivery. With the transmit cache disabled, this jumps to 3.47 packets/delivery, a 9% increase. The transmit cache improves CTP's efficiency by 9%.

These cost values are over 50% higher than those reported in Table I because channel 16 suffers from 802.11 interference, while the results in Table I are on channel 26, which does not. The next section examines how CTP responds to external interference in greater detail.

*7.3.10. External Interference.* The first two results in the second set of rows in Table II are obtained from experiments on Tutornet with the same link layer, transmission rate, and root node, but differ significantly in their delivery ratio. This difference is due to the 802.15.4 channel they used. The experiment on channel 26 (2.48GHz) observed an average delivery ratio of 99.9%; the experiment on channel 16 (2.43GHz) observed an average delivery ratio of 95.2%. Table III summarizes the differences between the two settings.

Using channel 16, the average path length increased from 2.02 to 3.12 hops, and the cost increased from 2.07 to 5.91 transmissions per successfully delivered packet. The increase in cost is not only due to longer paths but also a larger number of transmissions per hop, which increased from 1.025 to 1.894.

Channel 16 overlaps with several 802.11b channels (2–6), while channel 26 is almost entirely outside the 802.11b band. Figure 20 shows Wi-Fi activity on different channels on the Tutornet testbed. RF interference from Wi-Fi causes link qualities to drop and increases tree depth because longer links are generally less reliable. It also causes a larger number of retransmissions, decreasing effective capacity.

To test this hypothesis, we reran the channel 16 experiment with interpacket intervals of 22 s and 30 s. Table II shows the results. At 22 s, CTP has an average delivery

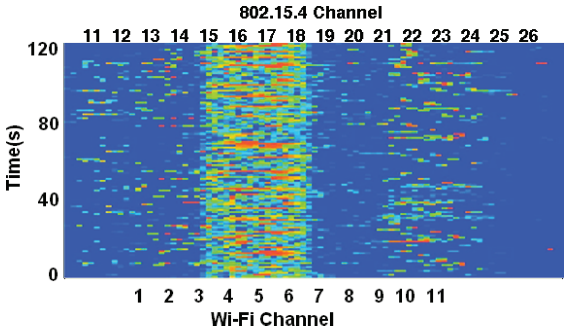



Fig. 20. 802.11 activity captured using the Wi-Spy Spectrum Analyzer tool on Tutornet. Channels 1 and 11 are most heavily used by the building occupants.

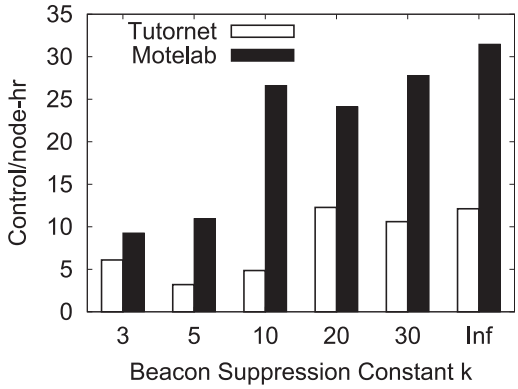



Fig. 21. Control overhead with different levels of beacon suppression. With low threshold, the nodes transmit fewer beacons.

ratio of 97.9% and at 30 s it has 99.4%. CTP achieves high delivery even with high external interference, but the external interference limits the data rate the network can support.

*7.3.11. Beacon Suppression.* To understand the impact of beacon suppression on CTP's performance (Section 5.5), we ran CTP with the suppression thresholds of 3 to infinity on Tutornet (54 nodes for these experiments) and Motelab, with each node sending one packet every eight seconds. In these Tutornet and Motelab experiments, CTP achieved delivery ratios of 99.9%–100% and 95.6%–97.1%, respectively. Although the delivery ratios are the same, and the delivery cost does not show persistent increasing or decreasing trend across the suppression thresholds, the control overhead was different for each threshold.

Figure 21 shows that CTP incurs lower control overhead with beacon suppression. On Tutornet, with no suppression ($k = \infty$), each node, on average, sent 12.1 beacons every hour. With, $k = 3$, each node sent 6.1 beacons/hr. On Motelab, each node sent 70.4% fewer beacons with $k = 3$ compared to the number of beacons sent without suppression. Figure 22 examines this reduction in control overhead in more detail. For each threshold, beacons are suppressed across the nodes to different extent. With $k = 3$, 100% of the beacons were suppressed on 25% of the nodes. With $k = 30$, no beacons were suppressed on 45% of the nodes. Overall, more beacons were suppressed with smaller threshold. Thus, with the information on Figure 22, we can conclude that beacon suppression causes the reduction in control overhead reported on Figure 21.

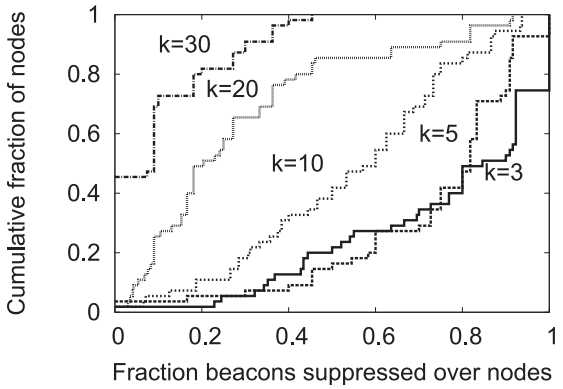



Fig. 22. The extent to which beacons are suppressed across the nodes on Tutornet. With smaller threshold, larger fraction of potential beacons are suppressed across the nodes.

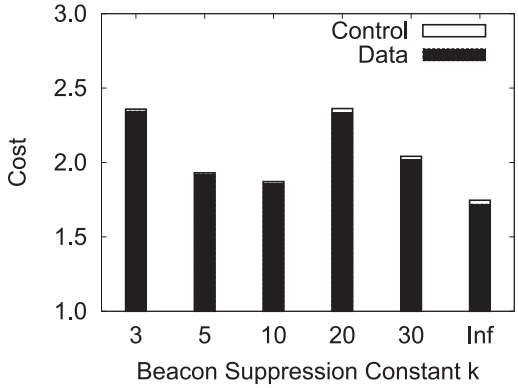



Fig. 23. The delivery cost with with different levels of beacon suppression. The total cost is largely independent of $k$.

Figure 23 puts the impact of beacon suppression on CTP's packet delivery cost in perspective. With no beacon suppression, control overhead constitutes 2.6% of total delivery cost. With $k = 3$, the control overhead is 1.3% of total delivery cost. Although this 50% reduction is dramatic, the impact of suppression on data delivery cost seems marginal because control overhead constitutes a tiny fraction of the total delivery cost. However, with LPL-based low-power MAC protocols, such as the BoX-MAC, even a marginal reduction in control overhead can have a large impact on duty-cycle. With a sleep interval of 1 s and assuming a 10ms packet transmission time, a broadcast packet transmission could require, in expectation, 50 times higher duty cycle than a data packet transmission. Thus in the extreme case, with $k = 3$, the total delivery cost could be lower by 35% compared to no suppression.

In the next set of experiments on Tutornet, we examine the effect of network density on beacon suppression. We do two sets of experiments with suppression thresholds of ten and infinity at different transmit power. At low transmission power, only the nearby nodes can receive the packets. At higher transmit power, the nodes farther away can receive the packets. Changing the transmission power changes the communication radius, and hence the density of the network as far as communication is concerned.

Figure 24 shows how CTP's control overhead changes with different density. At $-0.3$ dBm, CTP's per-node control overhead is lower by 33% compared to its overhead

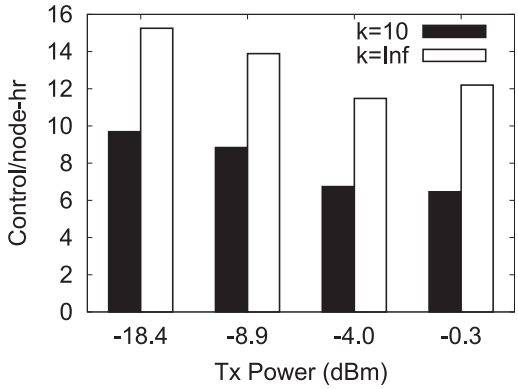



Fig. 24. Control overhead with different transmit power. At higher transmit power, that is, higher density, the per-node control overhead is lower.

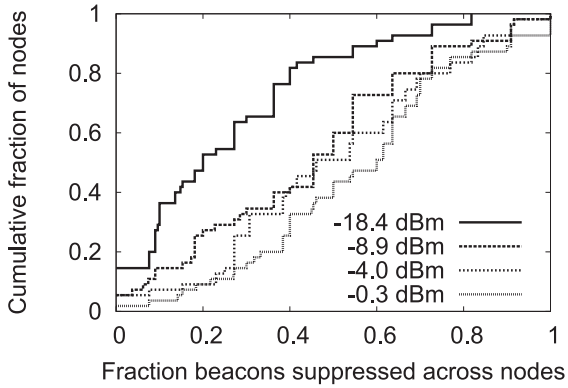



Fig. 25. Cumulative distribution of the control overhead with different transmit power. At higher transmit power, that is higher density, more beacons are suppressed. More suppression lowers the per-node control overhead.

at −18.4 dBm with a suppression threshold of ten. With no suppression, the difference is 20%. The experiment with no suppression quantifies the impact of density on control overhead: with a beacon reaching a larger fraction of nodes in the network, the chances of routing inconsistencies and discovery of newer and significantly better paths is lower. Both of these conditions reset the Trickle timer and trigger the transmission of routing beacons. Hence, there is a gradual decrease in control overhead as we increase the density. We see a similar trend in control overhead with beacon suppression. However, the control overhead decreases by 33% compared to 20% without suppression. Slicing this result differently, at −18.4 dBm, suppression lowers the control overhead by 36% compared to no suppression. At −0.3 dBm, the reduction is 47%. This suggests that beacon suppression is more effective at higher density. With higher transmit power, nodes receive beacons transmitted by a larger number of nodes in the neighborhood and hence suppress more beacons. With smaller transmit power, beacons are rarely suppressed as fewer beacons are received and the number rarely exceed the suppression threshold. Figure 25 shows how the fraction of suppressed beacons are distributed across the network. The suppression mechanism suppressed beacons across the network at all transmit power. The distribution is

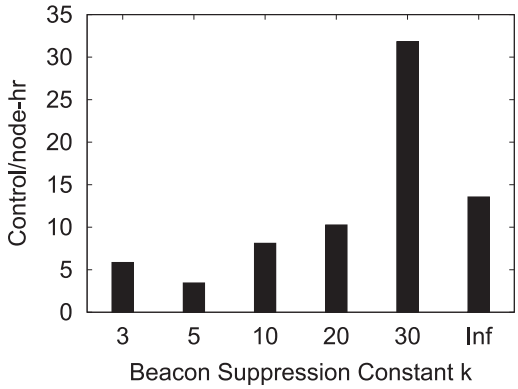
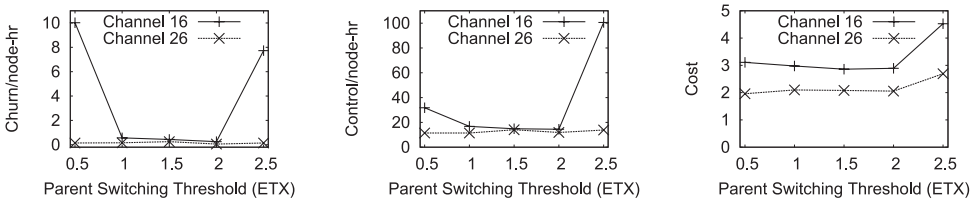



Fig. 26. Control overhead with WiFi interference. With smaller suppression threshold, the control overhead is generally lower.




(a) Churn with different path switching thresholds.

(b) Control overhead with different path switching thresholds.

(c) Delivery cost with different path switching thresholds.

Fig. 27. Impact of path hysteresis on CTP's performance.

slightly skewed for the lowest transmission power. At −18.4 dBm, 15% of the nodes had so few neighbors that none of their beacons were suppressed.

We conducted another set of experiments varying the suppression threshold but on channel 16 on Tutornet to understand the extent to which these observations hold in an environment with severe external interference. Figure 26 shows the results. We found that the trend—lower control overhead with beacon suppression—still holds. However, the level of interference during a specific experiment can make the trend noisier. During the experiment with $k = 30$, there was a ten-minute period when the control overhead and churn was significantly higher than rest of the time due to WiFi interference. Hence, in aggregate statistics, we observe higher control overhead with $k = 30$ compared to $k = \infty$.

*7.3.12. Path Hysteresis.* Excessive churn can lead to larger control overhead, which we want to minimize in low-power networks. CTP uses hysteresis in path selection to prevent excessive churn while still taking advantage of efficient paths that are discovered. CTP continues to use the current parent as its next hop until it discovers a path with the ETX metric that is at least 1.5 smaller than that of the current path. This means, the new path must have 1.5 fewer expected transmissions than the current path before the new path is used for data delivery.

We evaluated the impact of the hysteresis threshold on churn and efficiency on Tutornet with 54 nodes on both channel 16 and 26. With each path switching threshold, we ran CTP for an hour, each node sending one data packet every 8s. First, we discuss the results on channel 16. Figures 27(a) and 27(b) show that, with a threshold less than 1.0 or greater than 2.0, CTP experiences high churn and control overhead. With a low

Table IV. Results from experiments with Different Numbers of Roots

| Number of Roots | Average Delivery | PL | Cost | Control node-hr | Cost PL |
|---|---|---|---|---|---|
| 1 | 99.9% | 1.99 | 2.03 | 12.07 | 1.02 |
| 2 | 100.0% | 1.48 | 1.51 | 10.10 | 1.02 |
| 3 | 100.0% | 1.07 | 1.09 | 10.97 | 1.02 |

*Note*: The routing topology is shallower and the delivery cost smaller with multiple roots.

threshold, CTP selects a new parent even if it is marginally better, hence increasing the churn. Higher threshold also increases the churn. This is because a node advertises a new path when it is significantly better than the current one. When the neighbors discover this significantly better path, they reset their Trickle timer and send beacons increasing churn and control overhead across the network. In Figure 27(c), we show the delivery cost from the same experiment. The cost increases with a threshold greater than 2.0. Thus, with a low threshold, the control overhead and churn is too high and with a high overhead, the delivery cost is too high. On channel 26, although the threshold seems to matter less.

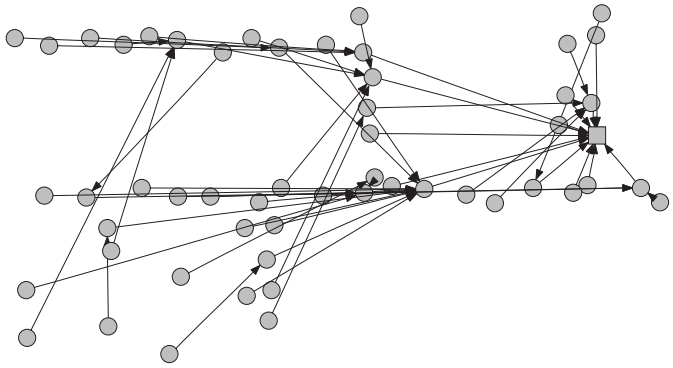
Thus, we conclude that a threshold between 1.0 and 2.0 enables CTP to achieve high efficiency while keeping the topology stable. Our CTP implementation uses a threshold of 1.5.

*7.3.13. Multiple Roots.* To understand how CTP's performance changes when there are multiple roots in the tree, we ran CTP on channel 26 with one, two, and three roots on a part of Tutornet with 54 nodes. In all these experiments, each node sent one data packet every 8 seconds.

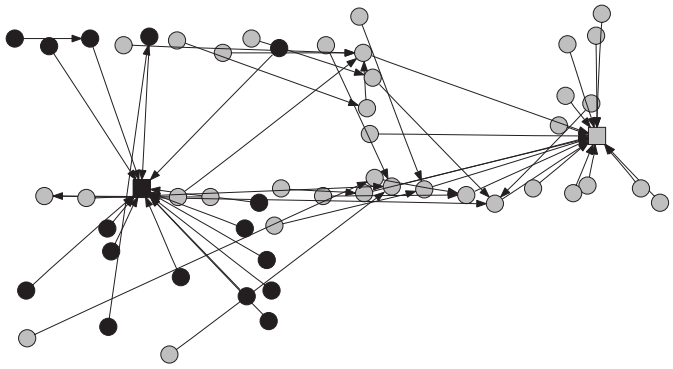
CTP is an address-free collection protocol. Nodes forward data packets to any root present in the network. With multiple roots, the nodes send data to the root reachable with the least expected number of transmissions. Table IV shows the results from our experiments with multiple roots. The delivery cost decreases by 26% with the addition of a root over a single-root network. The cost reduces by another 28% when we add an additional root in the network. This decrease in cost can be explained by the change in property of the routing tree: the average path length decreases by 26% and 28% when we add one and two additional roots in a single-root network. CTP formed two and three collection trees rooted at each root in the experiment with two and three roots. Each of these trees was shallower than the single collection tree in the single-root experiment. It turns out the control overhead remains relatively similar for one, two, or three roots. The Cost/PL metric, which indicates the quality of the link on the selected path, remains the same for one, two, and three roots.

In a collection network, deployment of multiple roots can alleviate congestion near the root. Instead of all the data traffic being directed to a single root, the packets are forwarded to one of the multiple roots in the network, hence splitting the traffic. In the experiment with a single root, the root received 6.79 packets/s. In the experiment with two roots, the first root received 4.74 packets/s and the second root received 2.1 packets/s. In the experiment with three roots, the traffic rates were 1.65 packets/s, 3.73 packets/s, and 1.42 packets/s for the three roots. The amount of data collected at different roots is not equal because the fraction of nodes that forward traffic to a given root depends on the position of the root in the network. Figure 28 shows the topologies.

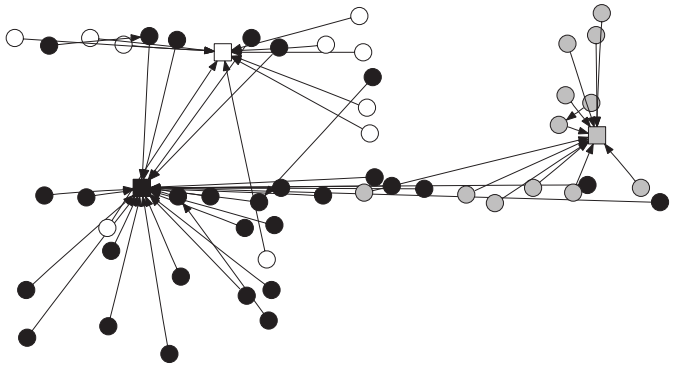
In these experiments, we found that it is rare that a node sent its data to different roots. In the experiment with three roots, 5 out of 54 nodes sent data to two roots. Figure 29 shows the number of packets received at each root.



(a) One root (cost = 2.03).



(b) Two roots (cost = 1.51).



(c) Three roots (cost = 1.09).

Fig. 28. Comparision of routing topology formed using one, two, and three roots in the network. The average cost in transmissions per delivered message is in parenthesis. The roots are drawn using square. Multiple roots make the trees shallower and lower the delivery cost.

*7.3.14. Link Layers.* The first section of Table II contains six experiments on the Motelab testbed using the standard TinyOS CSMA layer, low-power listening with BoX-MAC, and low-power probing with LPP. Table V has further details on these results.

Low-power listening observes longer paths (higher average PL) and higher end-to-end costs, but the per-link cost (cost/PL) decreases. Longer sleep intervals cause CTP to

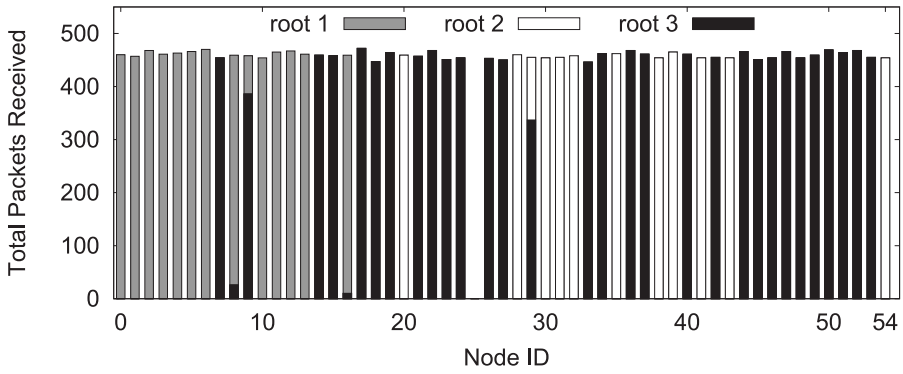



Fig. 29. Number of packets delivered from 54 nodes on Tutornet. With multiple roots in the network, CTP delivers packets to different roots.

Table V. Detailed Motelab Results on How Link-Layer Settings Affect CTP's Topology and Performance

| | Average | | | Cost | Duty Cycle | |
|---|---|---|---|---|---|---|
| Link Layer | Delivery | PL | Cost | PL | Median | Mean |
| CSMA | 94.7% | 3.05 | 5.53 | 1.81 | 100.0% | 100% |
| BoX-50ms | 94.4% | 3.28 | 6.48 | 1.98 | 24.8% | 24.9% |
| BoX-500ms | 97.1% | 3.38 | 6.61 | 1.96 | 4.0% | 4.6% |
| BoX-1s | 95.1% | 5.40 | 8.34 | 1.54 | 2.8% | 3.8% |
| LPP-500ms | 90.5% | 3.76 | 8.55 | 2.27 | 6.6% | 6.6% |

choose longer routes of more reliable links. This shift is especially pronounced once the interval is above 500 ms. The sleep interval the link layer uses affects the link qualities that CTP observes. If the signal-to-noise ratio is completely stable, link qualities are independent of how often a node checks the channel. There are temporal variations in the signal-to-noise ratio: this suggests that low-power link layers should consider the effects of link burstiness [Srinivasan et al. 2008].

Low-power probing generally under-performs low-power listening. For the same check interval, it has a lower delivery ratio and a higher duty cycle. This result should not be interpreted as a general comparison of the two, however. Each implementation can of course be improved, CTP is only one traffic pattern, and we only compared them on a single testbed with a single traffic rate. Nevertheless, CTP meets its reliability goal on both.

We also ran CTP with low-power link layers on the Twist testbed. For the eyesIFXv2 platform, we used the SpeckMAC layer, with a check interval of 183 ms. The lower delivery ratio with SpeckMAC compared to CSMA is due to queue overflows on the bottleneck links because of longer transmission times at the MAC layer.

*7.3.15. Energy Profile.* For a more detailed examination of CTP's energy performance, we use the Blaze platform, developed by Rincon Research. Blaze has a Texas Instruments MSP430 microcontroller and a Chipcon CC1100 [Instruments 2009] radio.[2] We used a 20-node network that Rincon Research has deployed in a 33m $\times$ 33m space. One node in this testbed has a precision 1 $\Omega$ resistor in series with the battery, connected to a high-precision 24-bit ADC using a data acquisition unit. We later converted this voltage to energy and extrapolated to per day energy consumption.

[2]Rincon Research maintains Blaze support code in the TinyOS 2.x "external contributions" repository.

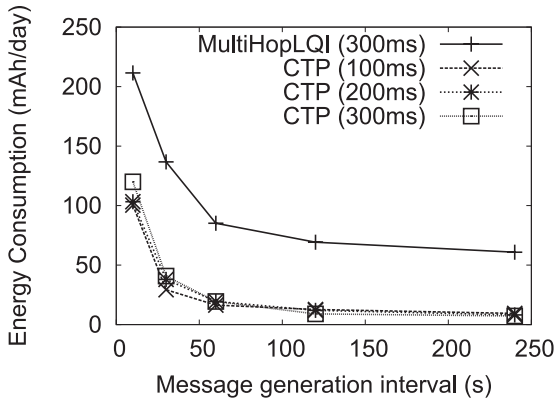



Fig. 30. Energy consumption for CTP and MultiHopLQI for 100ms–300ms sleep intervals.

We ran a total of 20 experiments for energy profiling, changing the MAC sleep interval and application data generation interval. For each experiment, we let the network warm up for about 15 minutes. We then use a dissemination protocol to request the nodes to start sending data at a given message interval. We collected energy data for 15 minutes.

The platform consumes 341.6 mAh/day in idle mode without duty cycling. The result from Figure 30 shows that the the full CTP stack can run for as little as 7.2 mAh/day, compared to 60.8 mAh/day for MultiHopLQI. During these experiments CTP consistently delivered 99.9% of the data packets to the sink.

This result suggests that CTP with a properly-designed low-power hardware platform can be used in long lasting deployments: even with a moderately-rapid (for a low power network) message interval of 240s, two AA batteries (5000 mAh) can supply sufficient energy to run a node for more than 400 days. This result is significant because the experiment takes into account the cost for running a full network stack consisting of dissemination and CTP protocols.

CTP's low energy profile is possible because it selects efficient paths, avoids unnecessary control traffic, and actively monitors the topology using the data plane. Reduction in control traffic is especially important in these networks because broadcast packets must be transmitted with long preambles. CTP's ability to taper off that overhead using exponentially increasing beacon interval allows it to achieve much lower energy consumption compared to the protocols that use periodic beacons.

*7.3.16. Testbed Observations.* The performance metric that varied the most across the testbeds is churn. On Motelab and Kansei, the churn is higher than on other testbeds. Analysis of CTP logs show that some sections of Motelab are very sparse and have only a few links with very low PRR. These low-quality links are bursty, such that nodes cycle through their list of alternative parents and actively probe the network in search of better parents. These small number of nodes account for most of the churn in the network—7% of the nodes accounted for 76% of parent changes on Motelab. This also explains why most packet losses on Motelab are due to retransmission timeouts. Thus, it is worth noting that CTP may not perform well on networks with insufficient connectivity.

On Tutornet and Kansei, churn is more uniform across nodes, but for different reasons. When operating on an interfering channel, Tutornet sees bursty links due to bursts of 802.11 interference, causing nodes to change parents often. On a non-interfering channel, CTP has very low churn on Tutornet. These bursts of interference

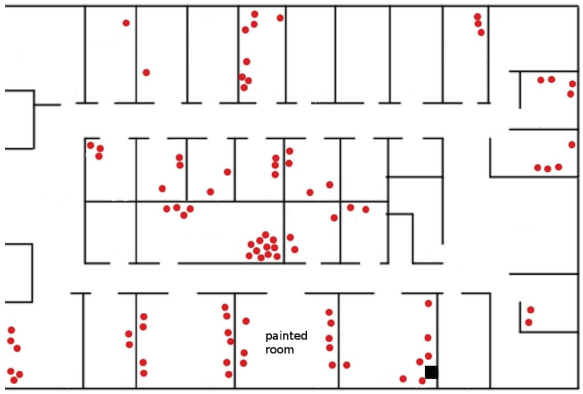



Fig. 31. The wireless power meter deployment spans one wing on one floor of an office building. The black square represents the sink and every dot is a power meter. Most meters are located under desks, near the floor.

prevent the nodes from delivering packets for periods of time, causing queue overflows to be the dominant cause of packet loss. In Kansei, the high churn is due to the sheer degree of the network: nodes actively probe their hundreds of candidate parents.

## 8. DEPLOYMENT EXPERIENCES

The previous section evaluated CTP's performance on a variety of testbeds. Next, we describe a large-scale, long-term deployment of CTP. The section presents experiences and lessons from using CTP in a real-world application that collects power measurements.

### 8.1. Powernet: A Sensor Network Deployment

Powernet [Kazandjieva et al. 2012] is a sensor network application that collects power and utilization data for the computing infrastructure of the computer science building at Stanford University. Plug-level power meters are connected to computing devices, such as PCs, LCD monitors, and network switches. In addition, building occupants and system administrators have volunteered to install software to monitor CPU utilization and network traffic. The meters report their data to the base station through a multihop network formed by CTP. All data is logged continuously to a central MySQL database, from once a second to once a minute, depending on the source.

The power meter used in Powernet is a modified version of the open-source ACme meter [Jiang et al. 2009]. Each ACme includes power measurement circuitry and an Epic core with microprocessor and radio chip [Dutta et al. 2008]. The meter software, built on TinyOS, includes sampling, routing, and dissemination capabilities. The top-level application reads power draw every second and sends a data packet after buffering ten samples. The motes use CTP as the underlying routing layer. The code includes Deluge [Hui and Culler 2004] for remote image upgrades. In addition to power data, the motes gather CTP statistics.

The 85-node wireless meter deployment is dense, covering a large fraction of the power outlets in one wing on one floor, shown in Figure 31. Powernet uses 802.15.4 channel 20, which overlaps with heavily used WiFi channel 6. We chose this so we could study the performance of CTP in a harsh wireless environment and also not interfere with research using quieter channels (e.g., 25 and 26).

Table VI. Information Collected by CTP Instrumentation

| Counters in Control Plane | Counters in Data Plane | In each packet |
|---|---|---|
| Transmissions | Transmissions | First link RSSI (source to parent) |
| Receptions | Receptions | First link RSSI (parent to source) |
| Parent changes | Acknowledgments received | First link ETX (source to parent) |
| Trickle resets | Queue drops | Number of hops |
| | Duplicates detected | Node ID of the first five hops |
| | Routing inconsistencies detected | |

*Note*: First link is the first link on the path from source to root.

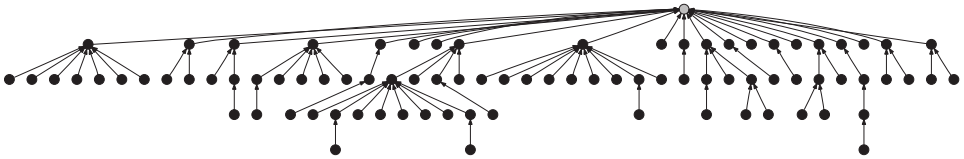


Fig. 32. Logical topology of the wireless network. The root of the tree is on top, and the number of nodes at each level is shown.

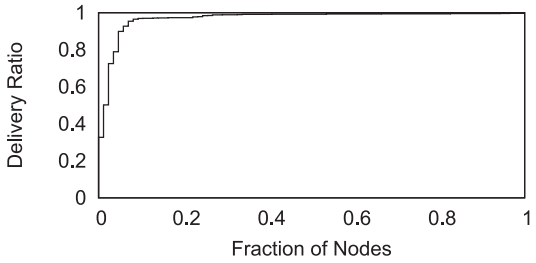



Fig. 33. Distribution of packet delivery ratio observed in Powernet.

### 8.2. Instrumentation

Because the wireless network does not have a wired back channel, we add instrumentation to CTP to report information about the state of the network and the protocol using CTP itself. Table VI lists all the information available from this instrumentation. The counters from the control and data plane are reported every five minutes. Additional information is included in each packet as shown in the table.

### 8.3. Overview of Results

Figure 31 shows the physical map of the wireless deployment, while Figure 32 presents a snapshot of the logical topology as constructed by CTP.

Overall, the backend collected 85.9% of the expected data. Figure 33 shows the distribution of delivery ratios across the nodes. Of the 14.1% of missing data, 8.2% is due to backend failures, such as whole-building power outages or server disk failures. This type of failure also affected data from the wireless meters and utilization sensors. Of the remaining 5.9%, we estimate that approximately 2.8% is due to users taking meters offline by unplugging them: the remaining 3.1% of data losses are due to CTP, assuming that delivery ratio for the 20 days selected for analysis is representative of the full deployment period.

Sifting through CTP's periodic reports, we find weekly and daily cycles of topology adaptation that correspond to human activity in the building. These periods of

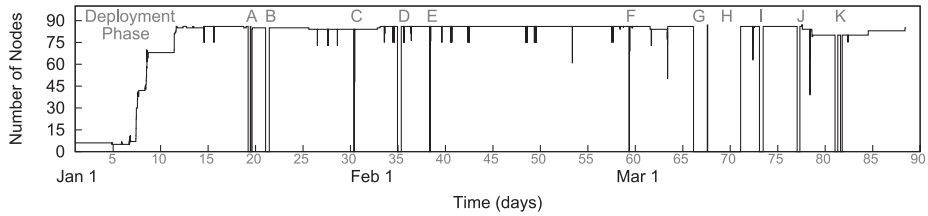



Fig. 34. Number of nodes from which packets were received at the base station during the deployment.

Table VII. List of Major System Outages during the Powernet Deployment

| Label | Date | Duration | Description |
|---|---|---|---|
| A | Jan 19 | 9 hrs | Building power outage |
| | | | MySQL recovery |
| B | Jan 21 | 10 hrs | Backend maintenance/backup |
| C | Jan 30 | 1 hr | Basestation maintenance |
| D | Feb 4 | 9 hrs | Basestation software failure |
| E | Feb 8 | 1 hr | Backend maintenance |
| F | Feb 28 | 0.5 hr | Backend maintenance |
| G | Mar 8 | 34 hrs | Backend disk failure |
| H | Mar 9 | 83 hrs | Backend disk replacement |
| I | Mar 14 | 9 hrs | Basestation buffering |
| J | Mar 18 | 7 hrs | Basestation buffering |
| K | Mar 22 | 4 hrs | Backend RAID1 rebuild |

adaptation see a significant increase in control traffic as well as increased path costs. In the middle of the night, the average cost (transmissions/delivery) of the network is just under 2, while during the day it can climb as high as 6. We find that CTP's datapath validation leads to a tiny fraction (1 in 20,000) of packets taking 10–100 times as many hops as normal, as they bounce through the topology repairing loops.

### 8.4. System Uptime

Figure 34 shows a 90-day trace of the number of connected wireless meters reported for each 15-minute period. Over the 90 days, the network experienced 11 network-wide outages in data logging, labeled (A–K). Table VII describes each outage, including whole-building power loss, backend downtime maintenance, disk failures, and gateway PC software failure. Overall, the backend was down for days, giving Powernet an uptime of 91%.

Small dips in the number of reporting nodes (e.g., the two dips at 15 days) represent logging delay due to MySQL buffering. These delays do not denote data loss.

While the high point of the plot remains stable (e.g., between points D and F), it does vary. For example, a week around K (days 77–84) shows eight nodes stopped reporting. This is not a network failure: the eight nodes were all in the same room (the labeled room in Figure 31). The eight-node outage occurred when the room was repainted and all computing equipment was unplugged and moved. Other, smaller dips represent users unplugging meters. Only a small fraction of the errors happened in the wireless meter network. The most likely cause is MAC problems that result in false ACKs. A tiny fraction of the observed data delivery outage were due to losses in the wireless routing, validating the earlier testbed results on CTP's robustness.

Table VIII. Summary of CTP Performance on Powernet

| | |
|---|---|
| Nodes | 85 |
| Path Length | 1.84 |
| Cost | 1.91 |
| Cost/PL | 1.04 |
| Churn/node-hr | 5.04 |
| Avg. Delivery | 0.969 |
| 5th % Delivery | 0.789 |
| Loss | Retransmit |

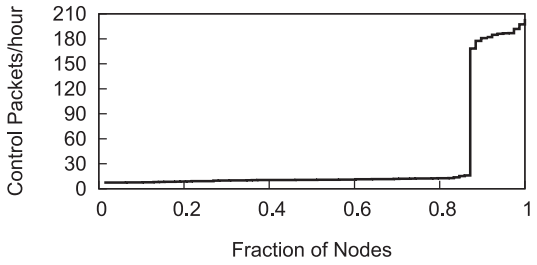



Fig. 35. Distribution of control packets sent by the nodes. There is a large difference in the number of control packet transmissions across the nodes.

### 8.5. CTP Performance

To isolate CTP's performance from network and node downtime, we analyze the network trace from a 20-day period in February (days 39–59 in Figure 34). CTP's behavior in this particular 20-day period is representative of the rest of Powernet's lifetime after deployment.

Table VIII shows high-level results. The Powernet network behaves differently than any of the studied testbeds. On one hand, its cost per path length of 1.04 indicates that intermediate link are rarely used (on average, out of 104 packets, only 4 were retransmission), making it similar to testbeds such as Mirage. On the other hand, its high average churn rate of 5.04 per hour makes it similar to harsher testbeds such as Motelab. This indicates that while Powernet has many high-quality links, those links come and go with reasonable frequency.

CTP's average delivery ratio was 96.9% and only five out of the 85 nodes reported delivery ratio below 90%. Two of these nodes were near many other wireless nodes, while another two were in the corner, possibly using longer links. The principal cause of packet loss is retransmission failure: CTP drops a packet after 30 retransmissions.

CTP's control traffic rate is bimodal, as shown in Figure 35. While 85% of nodes send a beacon every 15 minutes or less, 15% of the nodes send over ten times this many. As these high-traffic nodes are typically also forwarding many data packets, CTP's uneven control load can impose an even higher energy burden in low-power networks and harm network lifetime.

Figure 36 shows CTP's average cost (transmissions/delivery) over a 20-day period divided into data transmissions and control beacons. While the average cost is below 2, the middle of workdays can see the cost climb as high as 4.5, as the network adapts to topology changes. Figure 37 shows the same plot for a single work day. On this day, the cost rises as high as 6, and control beacons constitute 10.8% of the packets sent. The peak in Figure 37 is higher than those in Figure 36 due to longer averaging intervals.

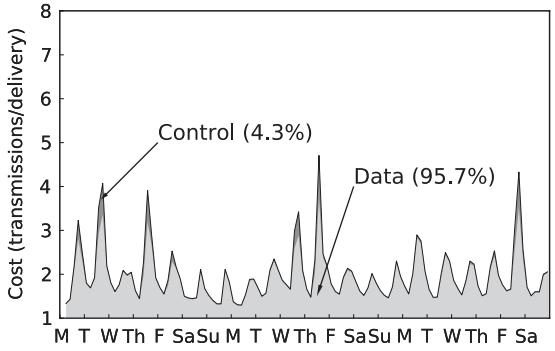



Fig. 36. Average packet delivery cost over 20 days. Weeknights and weekends show lower cost due to the availability of more efficient and stable paths. Cost of 1 is optimal.

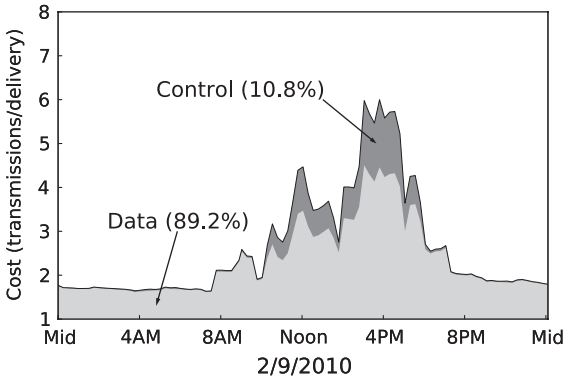



Fig. 37. CTP's packet delivery cost over one day; a value of 1 is optimal.

### 8.6. Daily and Weekly Cycles of Churn

Table VIII shows that CTP observes significant parent churn even during a stable, long-term deployment. This churn could be because CTP topologies are inherently unstable even in a stable environment, or because the underlying environment itself is unstable. Figure 38 shows a 14-day time series of one node's parent change rate with clear daily and weekly trend. During working hours, the node experiences much higher churn, up to 90 parent changes/hour. Furthermore, the peaks on the weekend are shorter and smaller than weekdays. In the absence of human activity, churn is fairly constant, at approximately six parent changes per hour.

### 8.7. Datapath Validation

CTP data packets contain a Time Has Lived (THL) field, which increments on each hop. Measuring THL at the gateway allows us to measure how many hops packet traverse in the network. Table IX shows a distribution of path lengths. Most packets fall in the range of 1–5 hops, one in 2,600 packets takes 6–20 hops, one in 20,000 packets takes 20–190 hops, and with one packet out of over 15,400,000 taking 190 hops.

The small percent of high THL packets stem from CTP's datapath validation algorithm. CTP uses data packets to validate and repair its topology. When a node detects the cost gradient is not decreasing, it sends beacons to repair the topology but forwards packets normally. This algorithm allows CTP to quickly detect potential loops in the network, but by the time the loop is repaired it is possible for the packets to go around

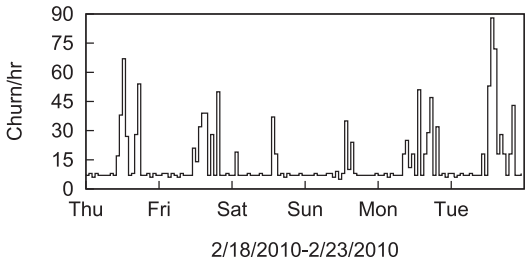



Fig. 38. Churn for one node over a six day period. Weekday afternoons and evenings show higher churn than weeknights and weekends.

Table IX. Distribution of Path Lengths Measured by the THL Field in CTP's Data Frame Header

| Hops | 1 | 2 | 3 | 4 | 5 | 6-20 | 20-190 |
|---|---|---|---|---|---|---|---|
| Fraction | 39% | 42% | 16% | 2.6% | 0.57% | 0.039% | 0.0051% |

the loop a few times. As a result, some packets report a very large number of hops. The longest reported THL was 190 hops. In that instance, the loop was repaired in 7.7 seconds.

### 8.8. Duplicate Suppression

We find that overall 1.7% of the packets received at the base station were duplicates. Packets from eight nodes had a duplication rate above 3.7%. During our 90-day deployment, due to misconfiguration, we deployed two nodes with ID 185 in two different areas of the network. The two nodes continue to report readings to the base station, but there are twice as many packets logged at the server. These packets elude CTP duplicate suppression due to two reasons. First, these two nodes often do not share a path. Second, the packet signature used for duplicate detection includes node ID, sequence number, and number of hops but the latter two are rarely the same between packets of the two nodes.

The Powernet deployment has been continuously operational for more than two years. The success of the deployment and the collected metadata show that CTP can achieve the four goals set forth in the introduction—reliability, robustness, efficiency, and hardware independence. Our experience with Powernet also demostrates the challenges in reliable data delivery in an end-to-end sensor network system. Although the wireless network does not drop the packets, the server might due to power outage.

## 9. RELATED WORK

CTP's lineage covers a vast body of work in sensor network routing in the last decade. The most common application for sensor networks is reliable data collection. The community designed, implemented, and deployed a large number of collection protocols, such as MintRoute [Woo et al. 2003] and MultihopLQI [Tolle et al. 2007], among many others.

The mechanisms we describe in this article draw on our experiences using such collection layers and the trade-off they introduce between cost and responsiveness. CTP's forwarding timer borrows from work on reliable sensornet transport protocols [Kim et al. 2007; Stann and Heidemann 2003; Sankarasubramaniam et al. 2003; Wan et al. 2002] which seek to maximize throughput by enabling pipelining through transmission timing.

Trickle-based beaconing and datapath validation [Hui and Culler 2008] has also been used in the 6LoWPAN/IP stack by Hui and Culler. Where their paper presented

the two techniques as small parts of a larger system evaluated on a single testbed, this article deeply evaluates them across a wide range of link layers, platforms, workloads, and environments, as well as examines the additional low-level systems issues that arise when incorporating them into a routing layer. The use of both techniques in two heavily tested, robust network layers provides greater evidence that these principles are more general than our specific implementation of them in CTP.

At a high level, adaptive beaconing and datapath validation combine elements of proactive and reactive routing paradigms, proactively maintaining (at low cost) a rough approximation of the best routing gradient, and making an effort to improve the paths data traffic traverses. CTP draws on mesh routing work, using ETX as its routing metric: this work established that minimizing either the expected number of transmissions (ETX [De Couto et al. 2003]) or a bandwidth-aware function of the expected number of transmissions (ENT [Draves et al. 2004]) along a path [Woo et al. 2003] constructs good routes. While ETX does not effectively capture throughput—a limitation in IP meshes—its measurement is perfectly suited to low-power sensor networks, which seek to minimize transmissions. Where modern WiFi protocols, such as ROMA [Dhananjay et al. 2009], still struggle with the discrepancy between periodic beacon measurements and actual link behavior, CTP avoids this problem by using the four-bit link estimator.

Adaptive beaconing extends Trickle [Levis et al. 2004] to time its routing beacons. Using Trickle enables quick discovery of new nodes and recovery from failures, while at the same time enabling long beacon intervals when the network is stable. This approximates beaconless routing [Zhang et al. 2006] in stable and static networks without sacrificing agility or new node discovery.

A large body of sensor network protocol work examines how to mitigate congestion when traffic concentrates around one node [Wan et al. 2003, 2005; Hull et al. 2004; Ee and Bajcsy 2004; Rangwala et al. 2006; Ahn et al. 2006]. CTP's transmit timers (Section 7.3.8) minimize self-interference by a single transmitter but do not coordinate transmitters. CTP provides an underlying routing topology and leaves internode congestion to higher or lower layers. Finally, we note Dozer [Burri et al. 2007], a proprietary collection protocol running exclusively on Shockfish hardware, whose source code we could not obtain.

Like IP routing layers, CTP does not provide end-to-end reliability. In contrast, RAP [Lu et al. 2002] attempts to deliver data to a sink within a time constraint, or not at all, a different set of requirements than is typical to packet routing. However, RAP uses similar mechanisms as CTP, such as MAC priorities and queuing mechanisms. RBC [Zhang et al. 2007] attempts to delivers bursty sensornet data reliably, using similar mechanisms as CTP as well as block acknowledgments.

## 10. IMPACT

CTP was released with TinyOS five years ago. CTP has been referenced in hundreds of research publications since its release and has found its use in different places and forms. In this section, we categorize and describe a few ways CTP has had impact in sensor networking research and practice, and beyond.

### 10.1. Data Collection System

The first and foremost value of CTP is its intended use, as a ready-to-use data collection protocol. Prior to CTP, almost all the researchers wrote their own data collection protocols. While this is a research opportunity to a networking researcher, to a sensor networking application researcher, this is an unnecessary engineering task. By providing a robust, reliable, and efficient collection protocol, CTP has enabled many researchers to deploy and experiment application ideas quickly and without having to

write low level networking code. Deployment of sensors in medical environments [Ko et al. 2010; Chipara et al. 2010] or power measurement in a building [Kazandjieva et al. 2012] are examples of deployments that collected data without inventing new routing protocols.

### 10.2. Enabler of Networking Research

One of CTP's biggest impacts is lowering the barrier to testbed-based wireless sensor networking research. By providing a robust code base and supporting it through bug fixes and community interaction on TinyOS-help mailing list, the networking researchers can experiment with new ideas in networking with relative ease. Most often these ideas were implemented as an add-on module that integrates with CTP. For example, work on making sensor networking routing energy-aware [Challen et al. 2010] or exploiting bursty links in sensor networks [Alizai et al. 2009] integrated their ideas to CTP code for proof-of-concept implementation and testing. Some researchers ran CTP on top of multiple radio interfaces [Hu et al. 2011], some integrated CTP with topology control [Hackmann et al. 2008], while others studied the benefit of combining CTP's beacons with beacons from other protocols [Hansen et al. 2011].

These research projects that integrate with CTP in different forms also highlight the importance of making the source code freely available to the research community.

### 10.3. Benchmark Protocol

Prior to CTP, sensor network research community had built a large number of routing protocols, such as MintRoute [Woo et al. 2003] and MultihopLQI [Tolle et al. 2007]. These protocols worked well in either specific deployments or platforms. CTP emerged as a benchmark collection protocol for two reasons. First, it was largely platform independent. A protocol that is tied to a particular platform is unlikely to be adopted as a benchmark by sensor network community because it uses a diverse set of platforms for research. In addition, CTP is robust, reliable, and efficient. Thus, availability on a large number of platforms and reasonable performance helped establish CTP as a protocol to beat when proposing new collection protocols or mechanisms.

Having a benchmark protocol allowed the community to experiment with many approaches to sensor network routing and determine if they improve on the state-of-the-art. For example, backpressure routing [Moeller et al. 2010], whirlpool routing [Lee et al. 2010], receiver initiated protocol [Unterschütz et al. 2012], and low-power wireless bus [Ferrari et al. 2012] all beat CTP for specific applications, scenarios, or constraints. Having an established benchmark also allows us to determine if more general-purpose protocols, such as RPL [Ko et al. 2011], are efficient enough in low power networks.

### 10.4. Representative Test Protocol and Software

The application-specific nature of wireless sensor networks and the need to make systems energy efficient has led to a large body of work in new protocols, operating systems, programming and debugging systems, and platforms. To test the usefulness or feasibility of these ideas, the community routinely uses CTP as a nontrivial use case of protocol or application.

Researchers have used CTP to demonstrate that a proposed MAC protocol provides enough features to implement a complex routing protocol [Dutta et al. 2010]. Software testing systems analyze CTP code to show that the tools can uncover subtle bugs in the system [Li and Regehr 2010; Zhou et al. 2010]. Operating system researchers have tested their fault tolerant [Chen et al. 2009], tracing [Sauter et al. 2011], or virtual machine [Brouwers et al. 2009] systems using CTP code. Researchers have also used CTP to test simulators [Alizai et al. 2011], testbed nodes [Lim et al. 2012], testbed

programming infrastructure [Donzelli et al. 2012], or protocol evaluation methodology [Puccinelli et al. 2011]. Thus CTP serves as a typical example of sensor network code base of moderate complexity to be used for testing different ideas in systems and networking research.

### 10.5. Other Platforms, Languages, and Protocols

Although the original CTP implementation was done in TinyOS, the community has ported it to other languages and operating systems. MANTIS OS [Bhatti et al. 2005] 1.0 release included an implementation of CTP. Contiki OS [Dunkels et al. 2004] also included a collection tree protocol based on CTP [ContikiCollect 2010]. We are aware of an implementation of CTP in Java. With these implementations, CTP is now used even on platforms that do not support TinyOS.

Finally, CTP has inspired and informed the design of many network protocols. Most of those protocols are research systems, as discussed in the previous sections. In addition, the experiences from CTP has also informed the design of RPL (Routing over Low Power and Lossy Networks) [Winter et al. 2012], which is an IPv6 routing protocol standardized by the IETF for use on low-power and lossy networks. In particular, RPL incorporates Trickle timer, adaptive beaconing, and datapath validation in its specification.

## 11. CONCLUSIONS

Our experiences with the collection protocols from the early days of sensor network research, to a large extent, guided the development of the key techniques used in CTP. The lack of agility in the traditional ETX estimator led us to develop the four-bit link estimator. The difficulty in determining the optimal routing beacon frequency for a given scenario led us to develop adaptive beaconing. The speed at which the energy gets depleted and hence the need to quickly detect loops led us to develop datapath validation. These three techniques allow a collection protocol to remain efficient, robust, and reliable in the presence of dynamic link topology. Our implementation of these mechanisms, CTP, offers 90–99.9% packet delivery in highly dynamic environments while sending up to 73% fewer control packets than existing approaches. It is robust to topology changes and failures. It places minimal expectations on the physical and link layer, allowing it to run on a wide range of platforms. As a result, CTP continues to enable prototype deployments in commercial and research environments. It also continues to serve the role of a benchmark collection protocol in sensor network routing research.

Our experiences from CTP deployments and further research identified that datapath validation and adaptive beaconing are applicable more widely to other types of low-power and lossy networks. The IPv6 routing protocol called RPL (Routing over Low Power and Lossy Networks) [Winter et al. 2012], recently standardized by the IETF, adopts these two techniques. The results presented in this article partly informed the standardization of those two techniques in RPL.

With multiple generation of network protocols and wireless sensor devices in the last decade, there has been a steady increase in the performance of data collection applications in sensor networks. Just a decade ago, the ability to collect some data from tens of inexpensive wireless sensor devices was considered a breakthrough. With the availability of protocols like CTP, delivery ratio close to 100% in networks of hundreds of wireless sensors is now routinely achieved. Such results set the expectation for performance of the future IPv6 networks of low-power wireless devices.

## ACKNOWLEDGMENTS

The authors wish to thank numerous users of CTP and the wider TinyOS community for helping to improve CTP.

## REFERENCES

AHN, G.-S., MILUZZO, E., CAMPBELL, A., HONG, S., AND CUOMO, F. 2006. Funneling MAC: A localized, sink-oriented MAC for boosting fidelity in sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys'07)*. ACM, New York, NY, 293–306.

ALIZAI, M. H., LANDSIEDEL, O., LINK, J. A. B., GÖTZ, S., AND WEHRLE, K. 2009. Bursty traffic over bursty links. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*. ACM, New York, NY, 71–84.

ALIZAI, M. H., WIRTZ, H., KIRCHEN, B., VAEGS, T., GNAWALI, O., AND WEHRLE, K. 2011. TinyWifi: Making network protocol evaluation portable across multiple phy-link layers . In *Proceedings of the 6th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH'11)*.

BHATTI, S., CARLSON, J., DAI, H., DENG, J., ROSE, J., SHETH, A., SHUCKER, B., GRUENWALD, C., TORGERSON, A., AND HAN, R. 2005. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Netw. Appl. 10,* 4, 563–579.

BROUWERS, N., LANGENDOEN, K., AND CORKE, P. 2009. Darjeeling, a feature-rich VM for the resource poor . In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*. ACM, New York, NY, 169–182.

BURRI, N., VON RICKENBACH, P., AND WATTENHOFER, R. 2007. Dozer: Ultra-low power data gathering in sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN'07)*. ACM, 450–459.

CHALLEN, G. W., WATERMAN, J., AND WELSH, M. 2010. IDEA: Integrated distributed energy awareness for wireless sensor networks. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*. ACM, New York, NY, 35–48.

CHEN, Y., GNAWALI, O., KAZANDJIEVA, M., LEVIS, P., AND REGEHR, J. 2009. Surviving sensor network software faults. In *Proceedings of 22nd ACM Symposium on Operating Systems Principles (SOSP'09)*.

CHIPARA, O., LU, C., BAILEY, T. C., AND ROMAN, G.-C. 2010. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*. ACM, New York, NY, 155–168.

CONTIKICOLLECT. 2010. Contiki Collect Memo. `http://comments.gmane.org/gmane.os.contiki.devel/5790`.

CROSSBOW TECHNOLOGY. 2006. Mica2dot Datasheet. `http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2DOT_Datasheet.pdf`.

CULLER, D., DUTTA, P., EE, C. T., FONSECA, R., HUI, J., LEVIS, P., POLASTRE, J., SHENKER, S., STOICA, I., TOLLE, G., AND ZHAO, J. 2005. Towards a sensor network architecture: Lowering the waistline. In *Proceedings of the 10th Conference on Hot Topics in Operating Systems (HOTOS'05)*. USENIX Association, Berkeley, CA, 24–24.

DE COUTO, D. S. J., AGUAYO, D., BICKET, J., AND MORRIS, R. 2003. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom'03)*. ACM, New York, NY, 134–146.

DHANANJAY, A., ZHANG, H., LI, J., AND SUBRAMANIAN, L. 2009. Practical, distributed channel assignment and routing in dual-radio mesh networks. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'09)*.

DONZELLI, C., HANDZISKI, V., AND WOLISZ, A. 2012. Demo abstract: Testbed-independent experiment specification and execution using the COTEFE platform. In *Proceedings of the 9th European Conference on Wireless Sensor Networks (EWSN'12)*.

DRAVES, R., PADHYE, J., AND ZILL, B. 2004. Comparison of routing metrics for static multi-hop wireless networks. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'04)*. 133–144.

DUNKELS, A., GRONVALL, B., AND VOIGT, T. 2004. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*. IEEE Computer Society, Los Alamitos, CA, 455–462.

DUTTA, P., DAWSON-HAGGERTY, S., CHEN, Y., LIANG, C.-J. M., AND TERZIS, A. 2010. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*. ACM, New York, NY, 1–14.

DUTTA, P., TANEJA, J., JEONG, J., JIANG, X., AND CULLER, D. 2008. A building block approach to sensornet systems. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*. ACM, New York, NY, 267–280.

EE, C. T. AND BAJCSY, R. 2004. Congestion control and fairness for many-to-one routing in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM, New York, NY, 148–161.

EE, C. T., RATNASAMY, S., AND SHENKER, S. 2006. Practical data-centric storage. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation (NSDI'06)*. USENIX Association, Berkeley, CA, 325–338.

FERRARI, F., ZIMMERLING, M., MOTTOLA, L., AND THIELE, L. 2012. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys'12)*. ACM, New York, NY, 1–14.

FONSECA, R., GNAWALI, O., JAMIESON, K., AND LEVIS, P. 2007. Four-bit wireless link estimation. In *Proceedings of the 6th workshop on Hot Topics in Networks (HotNets'07)*.

FONSECA, R., RATNASAMY, S., ZHAO, J., EE, C. T., CULLER, D., SHENKER, S., AND STOICA, I. 2005. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation (NSDI'05)*. USENIX Association, Berkeley, CA, 329–342.

GNAWALI, O., FONSECA, R., JAMIESON, K., MOSS, D., AND LEVIS, P. 2009. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*. ACM, New York, NY, 1–14.

HACKMANN, G., CHIPARA, O., AND LU, C. 2008. Robust topology control for indoor wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*. ACM, New York, NY, 57–70.

HANSEN, M., JURDAK, R., AND KUSY, B. 2011. Unified broadcast in sensor networks. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN'11)*. 306–317.

HU, W., KUSY, B., RICHTER, C., BRUENIG, M., AND HUYNH, C. 2011. Demo abstract: Radio-diversity collection tree protocol. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN'11)*. ACM, New York, NY, 111–112.

HUI, J. W. AND CULLER, D. 2004. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM, New York, NY, 81–94.

HUI, J. W. AND CULLER, D. E. 2008. IP is dead, long live IP for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*. ACM, New York, NY, 15–28.

HULL, B., JAMIESON, K., AND BALAKRISHNAN, H. 2004. Mitigating congestion in wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM, New York, NY, 134–147.

JIANG, X., DAWSON-HAGGERTY, S., DUTTA, P., AND CULLER, D. 2009. Design and implementation of a high-fidelity AC metering network. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'09)*. IEEE Computer Society, Los Alamitos, CA, 253–264.

KARP, B. AND KUNG, H. T. 2000. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*. ACM, New York, NY, 243–254.

KAZANDJIEVA, M., HELLER, B., GNAWALI, O., LEVIS, P., AND KOZYRAKIS, C. 2012. Green enterprise computing data: Assumptions and realities. In *Proceedings of the 3rd International Green Computing Conference (IGCC'12)*.

KIM, K.-H. AND SHIN, K. G. 2006. On accurate measurement of link quality in multi-hop wireless mesh networks. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MobiCom'06)*. ACM, New York, NY, 38–49.

KIM, S., FONSECA, R., DUTTA, P., TAVAKOLI, A., CULLER, D., LEVIS, P., SHENKER, S., AND STOICA, I. 2007. Flush: A reliable bulk transport protocol for multihop wireless networks. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys'07)*. ACM, New York, NY, 351–365.

KO, J., DAWSON-HAGGERTY, S., GNAWALI, O., CULLER, D., AND TERZIS, A. 2011. Evaluating the Performance of RPL and 6LoWPAN in TinyOS. In *Proceedings of Extending the Internet to Low power and Lossy Networks (IP+SN'11)*.

KO, J., LIM, J. H., CHEN, Y., MUSVALOIU-E, R., TERZIS, A., MASSON, G. M., GAO, T., DESTLER, W., SELAVO, L., AND DUTTON, R. P. 2010. MEDiSN: Medical emergency detection in sensor networks. *ACM Trans. Embed. Compu. Syst. 10,* 1, 11:1–11:29.

LANGENDOEN, K., BAGGIO, A., AND VISSER, O. 2006. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'06)*. 1–8.

LEE, J. W., KUSY, B., AZIM, T., SHIHADA, B., AND LEVIS, P. 2010. Whirlpool routing for mobility. In *Proceedings of the 11th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'10)*. ACM, New York, NY, 131–140.

Levis, P., Patel, N., Culler, D., and Shenker, S. 2004. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (NSDI'04)*. USENIX Association, Berkeley, CA, 2–2.

Li, J., Blake, C., De Couto, D. S., Lee, H. I., and Morris, R. 2001. Capacity of ad hoc wireless networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01)*. ACM, New York, NY, 61–69.

Li, P. and Regehr, J. 2010. T-check: Bug finding for sensor networks. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'10)*. ACM, New York, NY, 174–185.

Lim, R., Walser, C., Ferrari, F., Zimmerling, M., and Beutel, J. 2012. Distributed and synchronized measurements with FlockLab. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys'12)*. ACM, New York, NY, 373–374.

Lu, C., Blum, B. M., Abdelzaher, T. F., Stankovic, J. A., and He, T. 2002. RAP: A real-time communication architecture for large-scale wireless sensor networks. In *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*.

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international Workshop on Wireless Sensor Networks and Applications (WSNA'02)*. ACM, New York, NY, 88–97.

Mao, Y., Wang, F., Qiu, L., Lam, S., and Smith, J. 2007. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI'07)*. 101–114.

Moeller, S., Sridharan, A., Krishnamachari, B., and Gnawali, O. 2010. Routing without routes: The backpressure collection protocol. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'10)*. ACM, New York, NY, 279–290.

Moss, D. and Levis, P. 2008. BoX-MACs: Exploiting physical and link layer boundaries in low-power networking. Tech. rep. SING-08-00 Stanford Information Networks Group.

Musaloiu-E., R., Liang, C.-J. M., and Terzis, A. 2008. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN'08)*. IEEE Computer Society, 421–432.

Paek, J. and Govindan, R. 2007. RCRT: Rate-controlled reliable transport for wireless sensor networks. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys'07)*. ACM, New York, NY, 305–319.

Pei, D., Zhao, X., Massey, D., and Zhang, L. 2004. A study of BGP path vector route looping behavior. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. IEEE Computer Society, 720–729.

Perkins, C. E. and Bhagwat, P. 1994. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM'94)*. ACM, New York, NY, 234–244.

Polastre, J., Hill, J., and Culler, D. 2004. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM, New York, NY, 95–107.

Polastre, J., Szewczyk, R., and Culler, D. 2005. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*. IEEE Press, Piscataway, NJ, 48.

Puccinelli, D., Gnawali, O., Yoon, S., Santini, S., Colesanti, U., Giordano, S., and Guibas, L. 2011. The impact of network topology on collection performance. In *Proceedings of the 8th European Conference on Wireless Sensor Networks (EWSN'11)*. 17–32.

Rangwala, S., Gummadi, R., Govindan, R., and Psounis, K. 2006. Interference-aware fair rate control in wireless sensor networks. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'06)*. ACM, New York, NY, 63–74.

Sankarasubramaniam, Y., Özgür A., and Akyildiz, I. 2003. ESRT: Event-to-sink reliable transport in wireless sensor networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*. 177–189.

Sauter, R., Saukh, O., Frietsch, O., and Marron, P. 2011. TinyLTS: Efficient network-wide logging and tracing system for TinyOS. In *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM'11)*. 2033–2041.

Schoellhammer, T., Greenstein, B., and Estrin, D. 2006. Hyper: A routing protocol to support mobile users of sensor networks. Tech. rep. 2013, Center for Embedded Network Sensing, University of California.

Srinivasan, K., Dutta, P., Tavakoli, A., and Levis, P. 2006. Some implications of low power wireless to IP networking. In *Proceedings of the The 5th Workshop on Hot Topics in Networks (HotNets-V)*.

Srinivasan, K., Kazandjieva, M. A., Agarwal, S., and Levis, P. 2008. The beta-factor: Measuring wireless link burstiness. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*. ACM, New York, NY, 29–42.

Stann, F. and Heidemann, J. 2003. RMST: Reliable data transport in sensor networks. In *Proceedings of the the IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*. 102–112.

Texas Instruments. 2008. CC2420 Data Sheet. `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`.

Instruments. Texas 2009. CC1100 Data Sheet. `http://focus.ti.com/lit/ds/symlink/cc1100.pdf`.

Tolle, G., Levis, P., Buonadonna, P., Woo, A., and Polastre, J. 2007. The MultiHopLQI protocol. `http://www.tinyos.net/tinyos-2.x/tos/lib/net/lqi`.

Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., and Hong, W. 2005. A macroscope in the Redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*. ACM, New York, NY, 51–63.

Unterschütz, S., Renner, C., and Turau, V. 2012. Opportunistic, receiver-initiated data-collection protocol. In *Proceedings of the 9th European Conference on Wireless Sensor Networks (EWSN'12)*. Lecture Notes in Computer Science, vol. 7158, Springer-Verlag, Berlin, 1–16.

Wan, C.-Y., Campbell, A., and Krishnamurthy, L. 2002. PSFQ: A reliable transport protocol for wireless sensor networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*. 1–11.

Wan, C.-Y., Eisenman, S., and Campbell, A. 2003. CODA: Congestion detection and avoidance in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. ACM, New York, NY, 266–279.

Wan, C. Y., Eisenman, S., Campbell, A., and Crowcroft, J. 2005. Siphon: Overload traffic management using multi-radio virtual sinks. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*. ACM, New York, NY, 116–129.

Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., and Welsh, M. 2006. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*. USENIX Association, Berkeley, CA, 381–396.

Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J., and Alexander, R. 2012. RPL: IPv6 routing protocol for low-power and lossy networks. RFC 6550.`http://tools.ietf.org/html/rfcb550.`

Wong, K.-J. and Arvind, D. K. 2006. SpeckMAC: Low-power decentralised MAC protocols for low data rate transmissions in specknets. In *Proceedings of the 2nd International Workshop on Multi-Hop Ad Hoc Networks: From Theory to Reality (REALMAN'06)*. ACM, New York, NY, 71–78.

Woo, A. and Culler, D. E. 2001. A Transmission control scheme for media access in sensor networks. In *MobiCom'01 Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01)*. ACM, New York, NY, 221–235.

Woo, A., Tong, T., and Culler, D. 2003. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. ACM, New York, NY, 14–27.

Zhang, H., Arora, A., Choi, Y. R., and Gouda, M. 2007. Reliable bursty convergecast in wireless sensor networks. *Comput. Commu. 30,* 13 (Dec.), 2560–2576.

Zhang, H., Arora, A., and Sinha, P. 2006. Learn on the fly: Data-driven link estimation and routing in sensor network backbones. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM'06)*. Barcelona, Spain.

Zhou, Y., Chen, X., Lyu, M., and Liu, J. 2010. Sentomist: Unveiling transient sensor network bugs via symptom mining. In *Proceedings of the 30th IEEE Conference on Distributed Computing Systems (ICDCS'10)*. 784–794.

Received October 2012; revised January 2013; accepted February 2013