Collection Tree Protocol

Technical Report SING-09-01

Omprakash Gnawali
University of Southern California
gnawali@usc.edu

Rodrigo Fonseca
UC Berkeley & Yahoo! Research
rfonseca@cs.berkeley.edu

Kyle Jamieson
University College London
k.jamieson@cs.ucl.ac.uk

David Moss
Rincon Labs
dmm@rincon.com

Philip Levis
Stanford University
pal@cs.stanford.edu

Abstract

This paper presents and evaluates two principles for designing robust, reliable, and efficient collection protocols. These principles allow a protocol to benefit from accurate and agile link estimators by handling the dynamism such estimators introduce to routing tables. The first is datapath validation: a protocol can use data traffic as active topology probes, quickly discovering and fixing routing loops. The second is adaptive beaconing: by extending the Trickle code propagation algorithm to routing control traffic, a protocol sends fewer beacons while simultaneously reducing its route repair latency. We study these mechanisms in an implementation called Collection Tree Protocol (CTP) and evaluate their contributions to its performance.

We evaluate CTP on 12 different testbeds ranging in size from 20–310 nodes and comprising 7 hardware platforms, on 6 different link layers, and on interference-free and interference-prone channels. In all cases, CTP delivers $>$ 90% of packets. Many experiments achieve 99.9%. Compared to standard beaconing, CTP sends 73% fewer beacons while reducing topology repair latency by 99.8%. Finally, when using low-power link layers, CTP has duty cycles of 3% while supporting aggregate loads of 30 packets/minute.

1 Introduction

Collection trees are a core building block for sensor network applications and protocols. In their simplest use, collection trees provide an unreliable, datagram routing layer that deployments use to gather data [21, 34, 38]. Additionally, tree collection protocols provide the topology underlying most point-to-point routing protocols, such as BVR [11], PathDCS [9], and S4 [22] as well as transport protocols such as IFRC [29], RCRT [25], Flush [14], and Koala [24].

But despite its key role in so many systems, historically collection has exhibited persistent problems. Research papers typically report excellent performance from controlled experiments [41]. Real deployments, however, report low delivery ratios of anywhere from 2-68% [16, 21, 34, 38].

This paper describes key principles for designing collection protocols that can simultaneously achieve four goals:

Reliability: a collection protocol should deliver at least 90% of end-to-end packets when a route exists, even under challenging network conditions. 99.9% delivery should be achievable without end-to-end mechanisms.

Robustness: it should be able to operate without tuning or configuration in a wide range of network conditions, topologies, workloads, and environments.

Efficiency: it should achieve this reliability and robustness while sending few packets.

Hardware Independence: because sensor networks use a wide range of platforms, the implementation should be robust, reliable, and efficient without assuming specific radio chip features.

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 as 500 milliseconds [32]. Being efficient requires using these links when possible, but avoiding them when they fail. The 4-bit link estimator, for example, is able to reduce delivery costs by up to 44% compared to other approaches, but achieves by changing its estimates as quickly as every 5 packets [10].

Such dynamism is inherently challenging. Rapid topology changes lead to routing loops and other problems that harm reliability and efficiency. Incorporating two mechanisms into a routing layer can make it robust, efficient, and reliable in the presence of rapid topology changes.

The first is adapting the Trickle [18] algorithm, 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.

The second is actively using its 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. Rather than drop such a packet, the routing layer tries to repair the topology and forwards the packet normally. Using data packets maintains inconsistency detection agility precisely when a consistent topology is needed, even when the control traffic rate is very low due to Trickle.

We ground and evaluate these principles in a concrete protocol implementation, which we call the Collection Tree Protocol (CTP). In addition to incorporating 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 above goals in a wide spectrum of environments with minimal adjustments, we evaluate CTP on 12 different testbeds ranging in size from 20–310 nodes and comprising 7 hardware platforms. While not deployments in the field, the testbeds comprise diverse environmental conditions beyond our control, provide reproducibility, and enough diversity that give us confidence that CTP achieves the above goals. Anecdotal reports from several deployments support this belief. In two testbeds that have Telos nodes, we evaluate CTP using three link layers: full power, low power listening [28] and low power probing [24]. 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.

Evaluating CTP's use of adaptive beaconing and datapath validation, we find:

- Across all testbeds, configurations, and CSMA layers, CTP's end-to-end delivery ratio ranges from 90.5% to 99.9%.
- CTP supports median duty cycles of 3% while sustaining aggregate packet loads of 30 packets per minute.
- Compared to MultihopLQI, a collection protocol used in recent sensornet deployments [38], 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.

This paper makes three research contributions. First, it describes two key mechanisms, adaptive beaconing and datapath feedback, which enable routing layers to remain efficient, robust, and reliable in highly dynamic topologies on many different sensor platforms. Second, it describes the design and implementation of CTP, a collection protocol that uses these two mechanisms. Third, by evaluating CTP on 12 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.

These two techniques have appeared before in the literature: the 6lowpan/IP stack by Hui at al. uses both Trickle-based beaconing and datapath validation [12]. Where this prior work presented them as small parts of a larger system evaluated on a single testbed, however, this paper deeply evaluates them across a wide range of link layers, platforms,

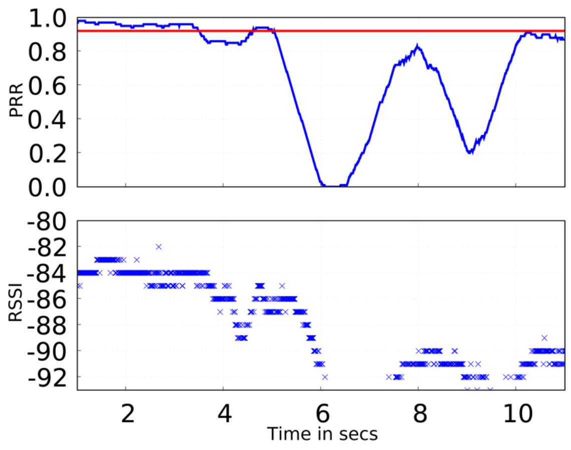


Figure 1. **(*Upper*) Packet reception rate (PRR) can vary on timescales orders of magnitude smaller than beacon rates. (*Lower*) RSSI and other physical-layer measurements are absent for dropped packets: using them can bias measurements.**

workloads, and environments, as well as examine the additional low-level systems issues that arise when incorporating them into a routing layer. Hui et al.'s prior work provides greater evidence that these principles are more general than our specific implementation of them in CTP.

In Section 2, we identify two main challenges in implementing robust and efficient wireless network protocols, namely link dynamics and transient loops. Based on these observations, we present adaptive beaconing and datapath validation in Section 3. Section 4 follows by giving detailed descriptions of their implementations in CTP, while Section 5 describes CTP's data plane. Section 6 presents a comprehensive experimental evaluation of CTP on the 12 testbeds, assessing how these design principles contribute to achieving reliability, robustness, and efficiency on different hardware platforms and varying challenging environmental and workload conditions. Section 7 presents prior work, and Section 8 concludes.

2 Motivation

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 today typically suffer from poor performance. Deployments observe delivery ratios of 2-68% [16, 21, 34, 38].

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 12 different testbeds and found two phenomena to be the dominant causes: link dynamics and transient loops.

2.1 Link Dynamics

Protocols today use periodic beacons to maintain their topology and estimate link qualities. The beaconing rate introduces a tradeoff 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 [41]. 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 [32].

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 2-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 4-bit estimator addresses link dynamics by actively using data packets to measure link quality [10]. 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 [26]. 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 6.3.

Some protocols prevent loops from forming altogether. DSDV, for example, uses destination-generated sequence numbers to synchronize routing topology changes and preclude loops [27]. The tradeoff 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 timescale 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 Overview

A collection protocol builds and maintains minimum-cost trees to nodes that advertise themselves as tree roots. Collection is address-free: when there are multiple base stations, it sends to the one with the minimum cost without knowing its address. In this paper, we assume all data packets are simple unicast frames.

Rapidly changing link qualities causes nodes to have stale topology information, which can lead to routing loops and packet drops. This section presents two mechanisms to be agile to link dynamics while also having a low overhead when the topology is stable. The first is datapath validation: using data packets to dynamically probe and validate the consistency of its routing topology. The second 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 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 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.

3.2 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 [2,41]. This interval poses a basic tradeoff. A small interval reduces how stale information can be and how long 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 tradeoff, achieving both fast recovery and low cost. It does so by extending the Trickle algorithm [18] 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 adapting 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 (τ_h). When Trickle hears a newer version number, it shrinks the timer interval to a small value (τ_l).

If all nodes have the same version number, their timer intervals increase exponentially, up to τ_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 τ_l, causing nodes to quickly learn of and receive new code.

Unlike algorithms in ad-hoc routing protocols such as DSDV [27], adaptive beaconing does assume the tree maintains a global sequence number or version number that might allow a simple application of Trickle. Instead, adaptive beaconing its routing cost gradient to control when to reset the timer interval. The routing layer resets the interval to τ_l on three events:

1. **It is asked to forward a data packet from a node whose ETX is not higher than its own.** The protocol interprets this as neighbors having a significantly out-of-date estimate and possibly a routing loop. It beacons to update its neighbors.
2. **Its routing cost decreases significantly.** The protocol advertises this event because it might provide lower-cost routes to nearby nodes. In this case, "significant" is an ETX of 1.5.
3. **It receives a packet with the P bit set.** The "Pull" bit advertises that a node wishes to hear beacons from its neighbors, e.g., because it has just joined the network and needs to seed its routing table. The pull bit provides a mechanism for nodes to actively request topology information from neighbors. Section 4.3 provides greater detail on the P bit.

In a network with very stable links, both the first and second events are rare. As long as nodes do not set the P bit, the beacon interval increases exponentially, up to τ_h. When the topology changes significantly, however, affected nodes reset their intervals to τ_l, and transmit to quickly reach consistency. While it could, we assume that adaptive beaconing does not use Trickle's suppression mechanism.[1]

3.3 Other Details

The two algorithms described above allow a collection to maintain an efficient yet agile routing topology. These two algorithms are insufficient by themselves, however. Numerous systems issues arise in packet forwarding that affect efficiency, reliability, and robustness, such as self-interference, link-layer duplicate suppression, retransmission policies, and queueing. We defer presenting these systems and implementation issues to Section 5, which discusses the data plane. The next section gives a detailed description of the implementation of these techniques in CTP's control plane.

[1]In the terminology of the Trickle algorithm, CTP sets $k = \infty$.

4 Control Plane

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

4.1 Route Computation and Selection

Figure 2 shows the CTP routing packet format nodes use to exchange topology information. The routing frame has two fields and two control bits. It advertises the node's current parent and routing cost. It also includes two control bits: the pull bit (P) and the congested bit (C). We discuss the meaning and use of the P bit below. The C bit is reserved for potential future use in congestion control and is not relevant for this paper.

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 sub-optimal 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 described above will prevent A from ever choosing C, as the resulting cost, 4.6, is not ≤ 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.

4.2 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

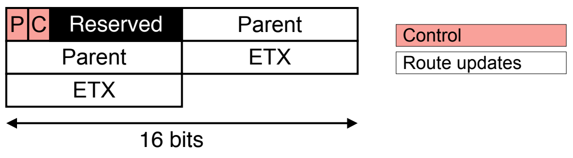


Figure 2. CTP routing frame format.

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 [18]. It maintains a beaconing interval which varies between 64ms and one hour. Whenever the timer expires, CTP doubles it, up to the maximum (1 hour). Whenever CTP detects an event which indicates the topology needs active maintenance, it resets the timer to the minimum (64ms). These values are independent of the underlying link layer. If a packet time is larger than 64ms, then the timer simply expires several times until it reaches a packet time.

4.3 Resetting the Beacon Interval

As discussed in Section 3.2 mentioned, 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:

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

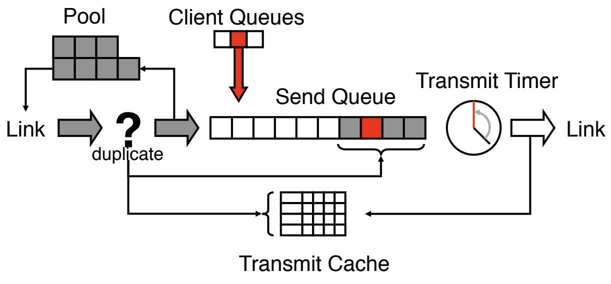


Figure 3. The CTP forwarding path.

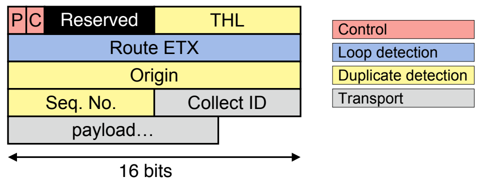


Figure 4. The CTP data frame format.

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

CTP forwards data packets in a possible loop normally: it does not drop them. However, it introduces a slight pause in forwarding, the length of the minimum beacon interval. 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 Data Plane

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 3 shows the CTP data path and how these four mechanisms interact.

Figure 4 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.

Testbed	Platform	Nodes	Physical size	Degree		PL	Cost	Cost	Churn
			m^2 or m^3	Min	Max			PL	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	Epic-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
Tutornet (26)	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 10 nodes.
[b] Blaze instrumentation does not provide cost and churn information.

Table 1. Testbed configuration and topology properties, from most to least dynamic. 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. There are two entries for Tutornet with TMotes: one is 802.15.4 channel 16 the other channel 26.

A one-byte application dispatch identifier called Collect ID, allows multiple clients 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.

5.1 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.

5.2 Hybrid Send Queue

CTP's lower level queue contains both route through- and locally-generated traffic (as in [40]), 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. Following this policy means that, technically, the send queue never rejects a packet. If it is full, this means the forwarding path is using all of its buffers and all clients have an outstanding packet.

When CTP receives a packet to forward, it first checks if the packet is a duplicate: Section 5.4 describes this process below. 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.

5.3 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 [19].

CTP prevents self interference by rate-limiting its transmissions. In the idealized scenario above where only the immediate children and parent are in the transmission range of a transmitter, if A waits at least 2 packet times between transmissions, then it will avoid self-interference, as C will have finished forwarding [40]. While real networks are more complex (the interference range can be greater than the transmit range), 2 packet times represents the minimum timing for a flow longer than 2 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 $2p$ but there is randomization to prevent edge conditions due to MAC backoff or synchronized transmissions.

5.4 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

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%	100.0%	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%	100.0%	Retransmit
NetEye	2.48GHz	CSMA	16s	99.9%	96.4%	Retransmit
Kansei	2.48GHz	CSMA	16s	99.9%	100.0%	Retransmit
Vinelab	2.48GHz	CSMA	16s	99.9%	99.9%	Retransmit
Quanto	2.425GHz	CSMA	16s	99.9%	100.0%	Retransmit
Twist (Tmote)	2.48GHz	CSMA	16s	99.3%	100.0%	Retransmit
Twist (Tmote)	2.48GHz	BoX-2s	5m	98.3%	92.9%	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

Table 2. Summary of experimental results across the testbeds. 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%. In Motelab, a small number of nodes (the 5th percentile) have poor delivery due to poor connectivity.

each hop. Over a small number of hops, this is not a significant issue, but in 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 match in the 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 a transmit cache. This cache contains the 3-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 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.

6 Evaluation

This section evaluates how the mechanisms described above, namely 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, using the 4-bit link estimator from [10], on 12 different testbeds, encompassing 7 platforms, 6 link layers, 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.

6.1 Testbeds

Table 1 summarizes the 12 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 USC. Neteye is at Wayne State University. Kansei is at Ohio State University. Vinelab is at UVA. Quanto is at UC Berkeley. Mirage is at Intel Research Berkeley. 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 16s. The interval is randomized to avoid collisions. To compute the link layer topology, we consider all links that delivered at least one packet. The minimum and maximum degree column in Table 1 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 Tutornet results 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 an uninterfering 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 6.11. All of the values in Table 1 for 802.15.4 testbeds, with the exception of Quanto and channel 16 Tutornet experiment (Mirage, Tutornet, Vinelab, Twist, Wymanpark, Kansei, Neteye, Motelab) use the non-interfering channel 26. Channel allocation concerns prevented us from doing the same in Quanto: it was measured with channel 15.

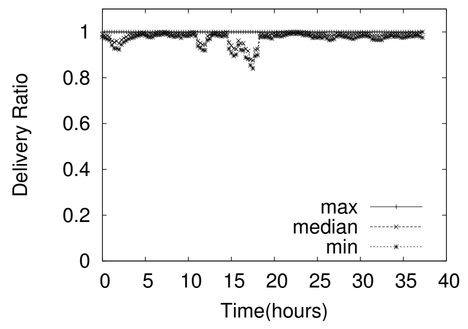
To roughly quantify link stability and quality, we ran CTP with an always-on link layer for 3 hours and computed three values: PL, the average path length (hops a packet takes to the collection root); the average cost (transmissions/delivery); and the node churn, or rate at which nodes change parents. 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.

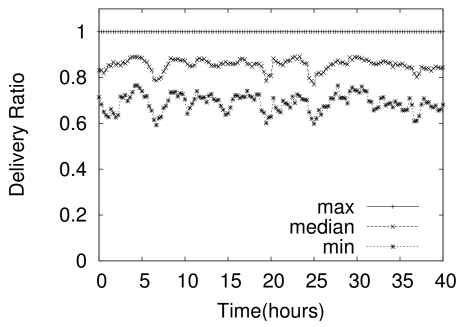
6.2 Experimental Methodology

We modified three variables across all of the experiments: the inter-packet 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 6 different MAC layers. On TelosB/TMote 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) [23], and low-power probing (LPP) [24]. On mica2dot nodes, we used the standard TinyOS 2.1.0 BMAC layer (B-MAC) for CC1000 radio [28]. On Blaze nodes, we used the standard TinyOS 2.1.0 BMAC layer (B-MAC) for CC1100. On eyesIFXv2 nodes, we use



(a) Delivery Ratio for CTP



(b) Delivery Ratio for MultiHopLQI.

Figure 5. 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.

both the TinyOS 2.1.0 CSMA layer (CSMA) as well as TinyOS 2.1.0 implementation of SpeckMAC [39]. 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 1 second, while "LPP-500ms" means low-power probing with a probing interval of 500ms.

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 highly used collection layer that is part of the TinyOS release. As MultihopLQI has been used in recent deployments, e.g., on a volcano in Ecuador [38], we consider it a reasonable comparison. Other notable collection layers, such as Hyper [31] and Dozer [5] 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.

6.3 Reliable, Robust, and Hardware-Independent

Before evaluating the effectiveness of each mechanism to the overall performance of CTP, we first look at high-level results from experiments across multiple testbeds, as well as

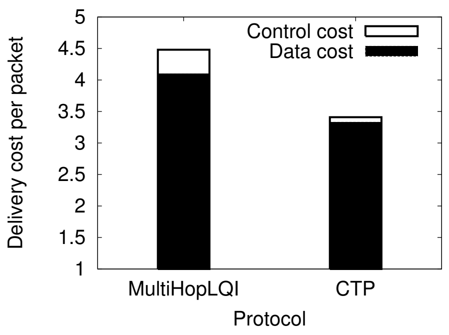


Figure 6. CTP's cost is 24% lower than MultiHopLQI and the portion of that is control is 73% lower.

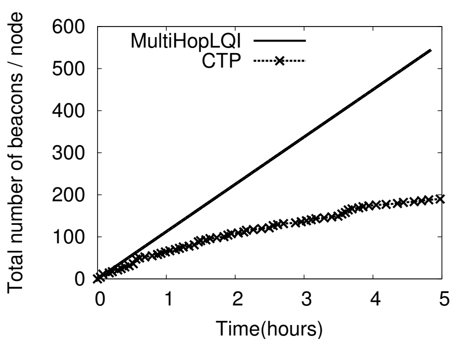


Figure 7. CTP's beaconing rate decreases and stabilizes over time. It is significantly smaller than MultihopLQI's over the long run.

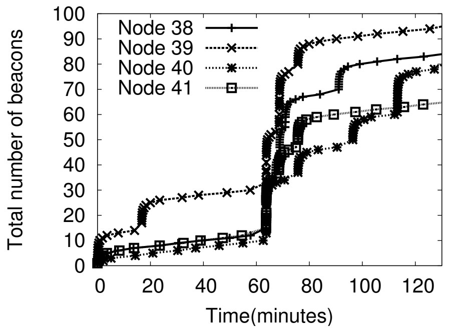


Figure 8. Number of beacons for selected nodes in the neighborhood of the new node. There is a big jump in control traffic shortly after four new nodes are introduced and it levels off.

a long duration experiment. Table 2 shows results from 21 experiments across the 12 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 (500ms), where it is 90.5%. The second lowest is Motelab using BoX-MAC (50ms), at 94.4%. In Motelab, packet losses 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 1 reflects this). Furthermore, CTP maintains this level of reliability across all configurations and settings: it meets the robustness and hardware independence goals. We therefore focus comparative evaluations on MultihopLQI.

To show the consistency of delivery ratio over time, in Figure 5(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 5(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.

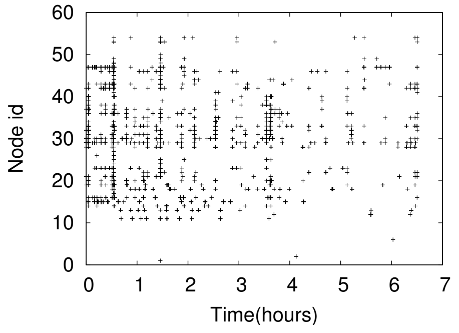
6.4 Efficiency

A protocol that requires a large number of transmissions is not well-suited for duty-cycled network. 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 the energy spent delivering a single packet to the sink. Figure 6 compares the delivery cost for CTP and MultiHopLQI. 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.

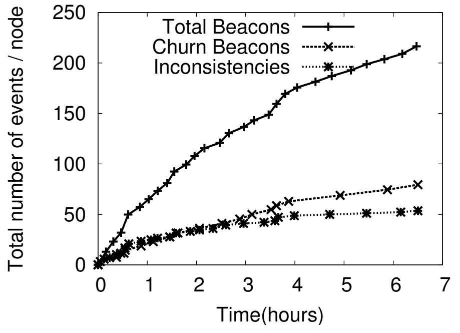
6.5 Adaptive Control Traffic

Figure 7 shows CTP's and MultihopLQI's control traffic from separate seven-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 30s. Using a Trickle timer allows CTP to send beacons as quickly as every 64ms and quickly respond to topology problems within a few packet times. By adapting its control rate and slowing down when the network is stable, however, CTP has a much lower control packet rate than MultiHopLQI. At the same time, it can respond to topology problems in 64ms, rather than 30s, a 99.8% reduction in response time.

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 8 shows that the control overhead for selected nodes in the



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



(b) Control overhead from route inconsistencies.

Figure 9. Route inconsistencies and repair

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.

6.6 Topology Inconsistencies

Next we look at how route inconsistencies are distributed over space and time, and their impact on control overhead. Figure 9(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 9(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.

When a node detects such an inconsistency, it resets its beacon timer. The top curve in Figure 9(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. The difference between these two curves provides an upper bound on the number of beacons sent due to inconsistencies. It is an upper bound because of the beacons that would have been sent normally, at the slowest beacon interval, and some occasional beacon caused by packets with the pull bit set. In 6.5 hrs, the nodes sent 12299 total beacons while they detected 3025 inconsistencies and triggered 4485 beacons due to parent change: CTP sent 2.6 beacons per inconsistency detected in order to repair and reestablish the path to the root.

6.7 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 8s. After 60 minutes, we removed the ten nodes that were forwarding the most packets in the network. CTP uses the 4-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.

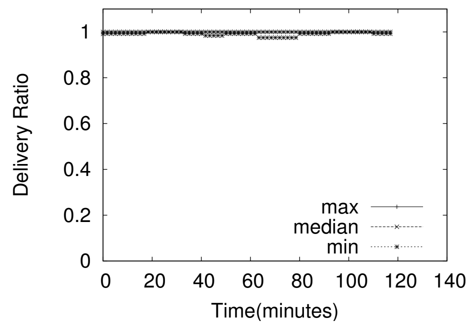
Figure 10(a) plots the minimum, median, and maximum delivery ratio across node over time. The figure shows only a tiny change in delivery ratio due to the disruption: the minimum delivery ratio across the network drops to 98%. 15 nodes dropped 1 or 2 packets each right after the disruption, and most nodes found new routes in under 1s. The 10-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 10(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%.

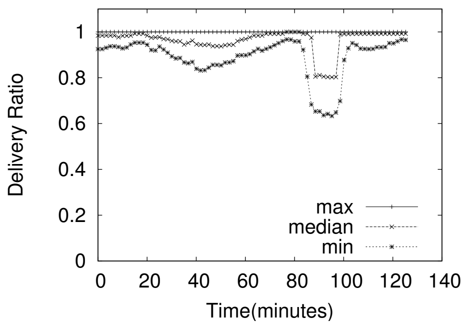
6.8 Agility

The prior experiment shows that CTP can quickly route around node failures when there is a constant stream of traffic. To delver deeper into how CTP adapts to sudden topology changes, we ran a slightly different experiment. We ran CTP on Tutornet with each node generating data packet every 8s 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 20th minute, we removed (erased the program running on the mote) node 26 from the network and 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 non-existent 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



(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.

Figure 10. Robustness of CTP and MultiHopLQI when the 10 most heavily forwarding nodes fail.

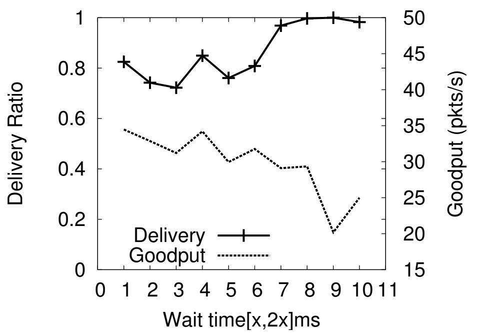


Figure 11. Effect of a per-hop rate-limiting transmit timer on goodput and delivery ratio on the CC2420 radio. Wait time between packets is [x,2x] ms.

every 8 minutes, CTP was able to quickly (in 325ms), select a new parent when its existing parent was removed from the network. CTP remains efficient even when the beacon interval decays tens of minutes, maintaining the ability to react to topology changes within a few packet transmission times.

6.9 Transmit Timer

CTP pauses briefly between packet transmissions to avoid self-interference, as described in Section 5.3. Here we show how we established this value for the CC2420 radio and quantify its benefit to CTP's reliability.

Figure 11 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 3-hops to the data sink. The transmission timers in Figure 11 range from $[1,2]$ to $[10,20]$ms. At values below $[7,14]$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 drops significantly as the timer introduces idleness. Therefore, CTP uses 7-14ms (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 [14,24,29,38], but CTP's transmit timers make its reliability more robust to the high load these protocols can generate.

6.10 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 1 because channel 16 suffers from 802.11 interference, while the results in Table 1 are on channel 26, which does not. The next section examines how CTP responds to external interference in greater detail.

6.11 External Interference

The first two results in the second set of rows in Table 2 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 3 summarizes the differences between the two settings.

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

Table 3. Results on how channel selection effects CTP's performance on Tutornet. Channel 16 overlaps with Wi-Fi; channel 26 does not.

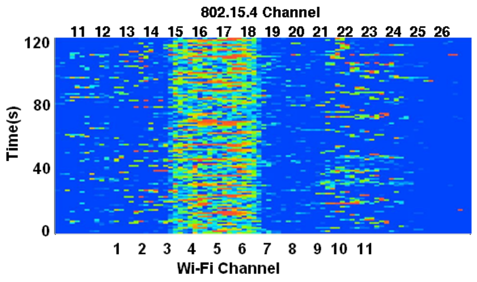


Figure 12. 802.11 activity captured using the Wi-Spy Spectrum Analyzer tool on Tutornet. Channel 1 and 11 are most heavily used by the building occupants.

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 12 shows Wi-Fi activity by Wi-Fi channel on the Tutornet testbed. RF interference from Wi-Fi causes link qualities to drop, 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 re-ran the channel 16 experiment with inter packet intervals of 22s and 30s. Table 2 shows the results. At 22s, CTP has an average delivery ratio of 97.9% and at 30s it has 99.4%.CTP achieves high delivery even with high external interference, but the business of the channel does lower the supportable data rate.

6.12 Link Layers

The first section of Table 2 contains six experiments on the Motelab testbed using the standard TinyOS CSMA layer,

Link Layer	Average Delivery	PL	Cost	Cost/PL	Duty Cycle Median	Duty Cycle 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%

Table 4. Detailed Motelab results on how link layer settings affect CTP's topology and performance.

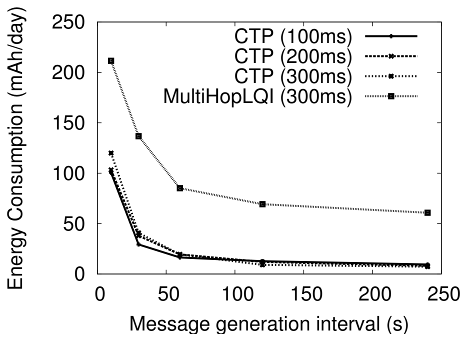


Figure 13. Energy consumption for CTP and MultiHopLQI for 100ms-300ms sleep intervals.

low-power listening with BoX-MAC, and low-power probing with LPP. Table 4 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 choose longer routes of more reliable links. This shift is especially pronounced once the interval is above 500ms. 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 [32].

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 both Twist testbeds. For the eyesIFXv2 platform, we used the SpeckMAC layer, with a check interval of 183ms. 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.

6.13 Energy Profile

For a more detailed examination of CTP's energy performance, we use the Blaze platform, developed by Rincon Research. Blaze has a TI MSP430 microcontroller and a CC1100 [1] radio.[2] We used a 20-node network that Rincon Research has deployed in a 33×33m space. One node in this testbed has a precision 1 Ω 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.

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

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

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 13 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 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.

6.14 Testbed Observations

The most salient differentiating dynamics property that we found 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 to 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 in Motelab are due to retransmission timeouts.

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 cause nodes to be unable to deliver 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.

7 Related Work

The mechanisms we describe in this paper draw on our experiences using collection layers such as MultihopLQI [2] and MintRoute [41], and the tradeoff they introduce between cost and responsiveness. CTP itself borrows from work on reliable sensornet transport protocols [14, 30, 33, 35] which seek to maximize throughput by timing transmissions on a path such that a pipelining effect occurs. These prior protocols motivated the CTP forwarding timer.

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. Mesh routing work has established that a good way of constructing routes in a wireless mesh network is to minimize either the expected number of transmissions (ETX [6]) or a bandwidth-aware function of the expected number of transmissions (ENT [7]) along a path [41], instead of simply the number of hops a packet must travel along any path. CTP draws on mesh routing work, using ETX as its routing metric.

Adaptive beaconing extends Trickle [18] 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 [42] in stable and static networks without sacrificing agility or new node discovery.

Other recent work in routing uses geographic location [15, 17], anchor beacons [11, 22], or DHT-like mechanisms [3] to achieve very highly-scalable routing. Like the former geographic routing proposals, CTP scales well with modestly increasing node density, but for a different reason: CTP uses Tickle timers to adapt beaconing rate, while geographic routing requires nodes to know their location. The later anchor beacon-based and DHT-based approaches can introduce stretch to routes, and are useful for collection on a larger scale than we consider in this paper.

A large body of sensor network protocol work examines how to mitigate the congestion that occurs when collection traffic concentrates in the vicinity of a sink. For example, CODA [36] and Fusion [13] propose different ways of mitigating congestion by using channel occupancy sampling [36] or a combination of backpressure and rate-limiting [13]. Ee and Bajcsy [8] and IFRC [29] attempt to allocate a fair rate to every sender in the network. CTP facilitates the use of such higher-layer protocols, but does not replace their functionality. CTP's transmit timers (described above in Section 6.9) prevent self-interference by a single transmitter, but do not coordinate transmitters, leaving this to higher layers.

Other congestion-control collection protocols take more unorthodox approaches. Siphon [37] uses a small number of virtual sinks equipped with high-bandwidth wireless radios to deliver data from congested regions of the network. Compared to Siphon, CTP does not rely on high-bandwidth external radios, and is thus suitable for the most general case of sensornet-only deployments. The Funneling MAC [4] is an example of a link-layer protocol that addresses congestion; compared to CTP, it uses a combination of TDMA and CSMA to improve fidelity when traffic becomes funneled near the collection sink. CTP, however, achieves high fidelity without the use of more complex TDMA medium access control. Finally, we note Dozer [5], a proprietary collection protocol running exclusively on Shockfish hardware, whose source code we could not obtain.

RAP [20] is a network architecture for real-time collection in wireless sensor networks. Unlike other collection protocols, RAP attempts to deliver data to a sink within a time constraint, or not at all: a different task as compared to collection. However, RAP uses similar mechanisms as CTP, such as MAC priorities and queuing mechanisms.

8 Conclusion

This paper describes two routing mechanisms, adaptive beaconing and datapath validation. These mechanisms allow a collection protocol to remain efficient, robust, and reliable in the presence of a highly 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 highly robust to topology changes and failures. It places a minimal expectations on the physical and link layer, allowing it to run on a wide range of platforms without any need for fine-tuning parameters. Minimizing control traffic, combined with efficient route selection, allows CTP to achieve duty cycles of $< 3\%$ while supporting loads of 25 packets/minute.

The efficacy of these two techniques suggests that the limitations of many wireless protocols today may be addressed by judiciously integrating the data and control planes. This indicates a way in which wireless protocols are fundamentally different than their wired siblings. While these techniques are presented in the context of collection, an open question worthy of future work is whether they are general to distance vector algorithms, and so may have broader applicability in ad-hoc networking.

9 References

[1] Texas Instruments, CC1100 Data Sheet. `http://focus.ti.com/lit/ds/symlink/cc1100.pdf`, 2003.

[2] The MultiHopLQI protocol. `http://www.tinyos.net/tinyos-2.x/tos/lib/net/lqi`, 2009.

[3] U. Acer, S. Kalyanaraman, and A. Abouzeid. Weak state routing for large scale dynamic networks. In *Proc. of the ACM MobiCom Conf.*, Sept. 2007.

[4] G.-S. Ahn, E. Miluzzo, A. Campbell, S. Hong, and F. Cuomo. Funneling MAC: A Localized, Sink-Oriented MAC for Boosting Fidelity in Sensor Networks. In *Proc. of the ACM SenSys Conf.*, pages 293–306, Boulder, CO, Nov. 2006.

[5] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 450–459, New York, NY, USA, 2007. ACM.

[6] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proc. of the ACM MobiCom Conf.*, San Diego, CA, Sept. 2003.

[7] R. Draves, J. Padhye, and B. Zill. Comparison of routing metrics for static multi-hop wireless networks. In *Proc. of the ACM SIGCOMM Conf.*, pages 133–144, Portland, OR, Aug. 2004.

[8] C. T. Ee and R. Bajcsy. Congestion control and fairness for many-to-one routing in sensor networks. In *Proc. of the ACM SenSys Conf.*, pages 148–161, Baltimore, MD, Nov. 2004.

[9] C. T. Ee, S. Ratnasamy, and S. Shenker. Practical Data-Centric Storage. In *Proc. of the USENIX NSDI Conf.*, San Jose, CA, May 2006.

[10] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four Bit Wireless Link Estimation. In *Hotnets-VI*, Atlanta, GA, Nov. 2007.

[11] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon Vector Routing: Scalable Point-to-Point Routing in Wireless Sensornets. In *Proc. of the USENIX NSDI Conf.*, Boston, MA, May 2005.

[12] J. W. Hui and D. E. Culler. IP is dead, long live IP for wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28, New York, NY, USA, 2008. ACM.

[13] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *Proc. of the ACM SenSys Conf.*, pages 134–147, Baltimore, MD, Nov. 2004.

[14] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: a reliable bulk transport protocol for multihop wireless networks. In *Proc. of the ACM SenSys Conf.*, pages 351–365. ACM, 2007.

[15] Y. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *Proc. of the USENIX NSDI Conf.*, Boston, MA, May 2005.

[16] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, pages 1–8, apr 2006.

[17] B. Leong, B. Liskov, and R. Morris. Geographic routing without planarization. In *Proc of the NSDI Conf.*, May 2006.

[18] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks. In *Proc. of the USENIX NSDI Conf.*, San Francisco, CA, Mar. 2004.

[19] J. Li, C. Blake, D. S. D. Couto, H. I. Lee, and R. Morris. Capacity of Ad Hoc wireless networks. In *Proc. of MobiCom*, pages 61–69. ACM, 2001.

[20] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He. RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks. In *Proc. of the IEEE RTAS Symposium*, San Jose, CA, September 2002.

[21] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications*, Sept. 2002.

[22] Y. Mao, F. Wang, L. Qiu, S. Lam, and J. Smith. S4: Small State and Small Stretch Routing Protocol for Large Wireless Sensor Networks. In *Proc. of the USENIX NSDI Conf.*, Cambridge, MA, Apr. 2007.

[23] D. Moss and P. Levis. BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Technical Report SING-08-00, Stanford Information Networks Group, 2008.

[24] R. Musaloiu-E., C.-J. Liang, and A. Terzis. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN 2008)*, 2008.

[25] J. Paek and R. Govindan. Rcrt: rate-controlled reliable transport for wireless sensor networks. In *Proc. of the ACM SenSys Conf.*, pages 305–319, New York, NY, USA, 2007. ACM.

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

[27] C. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *Computer Communication Review*, October 1994.

[28] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proc. of the ACM SenSys Conf.*, pages 95–107, Baltimore, MD, Nov. 2004.

[29] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. In *Proc. of the ACM SIGCOMM Conf.*, pages 63–74, Pisa, Italy, Aug. 2006.

[30] Y. Sankarasubramaniam, Özgür Akan, and I. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *Proc. of the ACM Mobihoc Conf.*, pages 177–189, Annapolis, MD, June 2003.

[31] T. Schoellhammer, B. Greenstein, and D. Estrin. Hyper: A routing protocol to support mobile users of sensor networks. Technical Report 2013, CENS, 2006.

[32] K. Srinivasan, M. Kazandjieva, S. Agarwal, and P. Levis. The beta-factor: Measuring wireless link burstiness. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[33] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proc. of the IEEE SNPA Workshop*, pages 102–112, Anchorage, AK, May 2003.

[34] G. Tolle, J. Polastre, R. Szewczyk, D. E. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proc. of the ACM SenSys Conf.*, pages 51–63, San Diego, CA, Nov. 2005.

[35] C.-Y. Wan, A. Campbell, and L. Krishnamurthy. PSFQ: a Reliable Transport Protocol for Wireless Sensor Networks. In *Proc. of the ACM WSNA Workshop*, pages 1–11, Atlanta, GA, 2002.

[36] C.-Y. Wan, S. Eisenman, and A. Campbell. CODA: Congestion Detection and Avoidance in Sensor Networks. In *Proc. ACM SenSys*, pages 266–279, Nov. 2003.

[37] C. Y. Wan, S. Eisenman, A. Campbell, and J. Crowcroft. Siphon: Overload Traffic Management using Multi-Radio Virtual Sinks. In *Proc. of the ACM SenSys Conf.*, pages 116–129, San Diego, CA, Nov. 2005.

[38] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and Yield in a Volcano Monitoring Sensor Network. In *USENIX Symposium on Operating Systems Design and Implementation*, Seattle, WA, Nov. 2006.

[39] K.-J. Wong and D. K. Arvind. Speckmac: low-power decentralised mac protocols for low data rate transmissions in specknets. In *REALMAN '06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 71–78, New York, NY, USA, 2006. ACM.

[40] A. Woo and D. E. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the seventh annual international conference on Mobile computing and networking*, Rome, Italy, July 2001.

[41] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. ACM SenSys*, pages 14–27, Los Angeles, CA, Nov. 2003.

[42] H. Zhang, A. Arora, and P. Sinha. Learn on the fly: Data-driven link estimation and routing in sensor network backbones. In *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.