The Collection Tree Protocol (CTP)

TEP:	123
Group:	Network Working Group
Type:	Documentary
Status:	Draft
TinyOS-Version:	2.x
Author:	Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo
Draft-Created:	3-Aug-2006
Draft-Version:	1.8
Draft-Modified:	2007-02-28
Draft-Discuss:	TinyOS Developer List <tinyos-devel at mail.millennium.berkeley.edu>

Abstract

This memo documents the Collection Tree Protocol (CTP), which provides best-effort anycast datagram communication to one of the collection roots in a network.

1. Introduction

2. Assumptions and Limitations

CTP is a tree-based collection protocol. Some number of nodes in a network advertise themselves as tree roots. Nodes form a set of routing trees to these roots. CTP is **address-free** in that a node does not send a packet to a particular root; instead, it implicitly chooses a root by choosing a next hop. Nodes generate routes to roots using a routing gradient.

The CTP protocol assumes that the data link layer provides four things:

1. Provides an efficient local broadcast address.
2. Provides synchronous acknowledgments for unicast packets.
3. Provides a protocol dispatch field to support multiple higher-level protocols.
4. Has single-hop source and destination fields.

CTP assumes that it has link quality estimates of some number of nearby neighbors. These provide an estimate of the number of transmissions it takes for the node to send a unicast packet whose acknowledgment is successfully received.

CTP has several mechanisms in order to improve delivery reliability, but it does not promise 100% reliable delivery. It is best effort, but a best effort that *tries very hard.*

CTP is designed for relatively low traffic rates. Bandwidth-limited systems might benefit from a different protocol, which can, for example, pack multiple small frames into a single data-link packet.

3. Collection and CTP

CTP uses expected transmissions (ETX) as its routing gradient. A root has an ETX of 0. The ETX of a node is the ETX of its parent plus the ETX of its link to its parent. This additive measure assumes that nodes use link-level retransmissions. Given a choice of valid routes, CTP SHOULD choose the one with the lowest ETX value. CTP represents ETX values as 16-bit fixed-point real numbers with a precision of hundredths. An ETX value of 451, for example, represents an ETX of 4.51, while an ETX value of 109 represents an ETX of 1.09.

Routing loops are a problem that can emerge in a CTP network. Routing loops generally occur when a node choose a new route that has a significantly higher ETX than its old one, perhaps in response to losing connectivity with a candidate parent. If the new route includes a node which was a descendant, then a loop occurs.

CTP addresses loops through two mechanisms. First, every CTP packet contains a node's current gradient value. If CTP receives a data frame with a gradient value lower than its own, then this indicates that there is an inconsistency in the tree. CTP tries to resolve the inconsistency by broadcasting a beacon frame, with the hope that the node which sent the data frame will hear it and adjust its routes accordingly. If a collection of nodes is separated from the rest of the network, then they will form a loop whose ETX increases forever. CTP's second mechanism is to not consider routes with an ETX higher than a reasonable constant. The value of this constant is implementation dependent.

Packet duplication is an additional problem that can occur in CTP. Packet duplication occurs when a node receives a data frame successfully and transmits an ACK, but the ACK is not received. The sender retransmits the packet, and the receiver receives it a second time. This can have disasterous effects over multiple hops, as the duplication is exponential. For example, if each hop on average produces one duplicate, then on the first hop there will be two packets, on the second there will be four, on the third there will be eight, etc.

Routing loops complicate duplicate suppression, as a routing loop may cause a node to legitimately receive a packet more than once. Therefore, if a node suppresses duplicates based solely on originating address and sequence number, packets in routing loops may be dropped. CTP data frames therefore have an additional time has lived (THL) field, which the routing layer increments on each hop. A link-level retransmission has the same THL value, while a looped version of the packet is unlikely to do so.

4. CTP Data Frame

The CTP data frame format is as follows:

```
                    1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|P|C| reserved  |      THL        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              ETX                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             origin              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     seqno     |   collect_id    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    data ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Field definitions are as follows:

- P: Routing pull. The P bit allows nodes to request routing information from other nodes. If a node with a valid route hears a packet with the P bit set, it SHOULD transmit a routing frame in the near future.
- C: Congestion notification. If a node drops a CTP data frame, it MUST set the C field on the next data frame it transmits.
- THL: Time Has Lived. When a node generates a CTP data frame, it MUST set THL to 0. When a node receives a CTP data frame, it MUST increment the THL. If a node receives a THL of 255, it increments it to 0.
- ETX: The ETX routing metric of the single-hop sender. When a node transmits a CTP data frame, it MUST put the ETX value of its route through the single-hop destination in the ETX field. If a node receives a packet with a lower gradient than its own, then it MUST schedule a routing frame in the near future.

- origin: The originating address of the packet. A node forwarding a data frame MUST NOT modify the origin field.
- seqno: Origin sequence number. The originating node sets this field, and a node forwarding a data frame MUST NOT modify it.
- collect_id: Higher-level protocol identifier. The origin sets this field, and a node forwarding a data frame MUST NOT modify it.
- data: the data payload, of zero or more bytes. A node forwarding a data frame MUST NOT modify the data payload.

Together, the origin, seqno and collect_id fields denote a unique ***origin packet.*** Together, the origin, seqno, collect_id, and THL denote a unique ***packet instance*** within the network. The distinction is important for duplicate suppression in the presence of routing loops. If a node suppresses origin packets, then if asked to forward the same packet twice due to a routing loop, it will drop the packet. However, if it suppresses packet instances, then it will route succesfully in the presence of transient loops unless the THL happens to wrap around to a forwarded packet instance.

A node MUST send CTP data frames as unicast messages with link-layer acknowledgments enabled.

5. CTP Routing Frame

The CTP routing frame format is as follows:

```
                     1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|P|C| reserved  |      parent     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     parent    |       ETX       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      ETX      |
+-+-+-+-+-+-+-+-+
```

The fields are as follows:

- P: Same as data frame.
- C: Congestion notification. If a node drops a CTP data frame, it MUST set the C field on the next routing frame it transmits.
- parent: The node's current parent.
- metric: The node's current routing metric value.

When a node hears a routing frame, it MUST update its routing table to reflect the address' new metric. If a node's ETX value changes significantly, then CTP SHOULD transmit a broadcast frame soon thereafter to notify other nodes, which might change their routes. The parent field acts as a surrogate for the single-hop destination field of a data packet: a parent can detect when a child's ETX is significantly below its own. When a parent hears a child advertise an ETX below its own, it MUST schedule a routing frame for transmission in the near future.

6. Implementation

An implementation of CTP can be found in the tos/lib/net/ctp directory of TinyOS 2.0. This section describes the structure of that implementation and is not in any way part of the specification of CTP.

This implementation has three major subcomponents:

1) A **link estimator**, which is responsible for estimating the single-hop ETX of communication with single-hop neighbors.

2) A **routing engine**, which uses link estimates as well as network-level information to decide which neighbor is the next routing hop.

3) A **forwarding engine**, which maintains a queue of packets to send. It decides when and if to send them. The name is a little misleading: the forwarding engine is responsible for forwarded traffic as well as traffic generated on the node.

6.1 Link Estimation

The implementation uses two mechanisms to estimate the quality of a link: periodic LEEP [1] packets and data packets. The implementation sends routing beacons as LEEP packets. These packets seed the neighbor table with bidirectional ETX values. The implementation adapts its beaconing rate based on network dynamics using an algorithm similar to the trickle dissemination protocol [2]. Beacons are sent on an exponentially increasing randomized timer. The implementation resets the timer to a small value when one or more of the following conditions are met:

1. The routing table is empty (this also sets the P bit)
2. The node's routing ETX increases by >= 1 trasmission
3. The node hears a packet with the P bit set

The implementation augments the LEEP link estimates with data transmissions. This is a direct measure of ETX. Whenever the data path transmits a packet, it tells the link estimator the destination and whether it was successfully acknowledged. The estimator produces an ETX estimate every 5 such transmissions, where 0 successes has an ETX of 6.

The estimator combines the beacon and data estimates by incorporating them into an exponentially weighted moving average. Beacon-based estimates seed the neighbor table. The expectation is that the low beacon rate in a stable network means that for a selected route, data estimates will outweigh beacon estimates. Additionally, as the rate at which CTP collects data estimates is proportional to the transmission rate, then it can quickly detect a broken link and switch to another candidate neighbor.

The component `tos/lib/net/le/LinkEstimatorP` implements the link estimator. It couples LEEP-based and data-based estimates.

6.2 Routing Engine

The implementation's routing engine is responsible for picking the next hop for a data transmission. It keeps track of the path ETX values of a subset of the nodes maintained by the link estimation table. The minimum cost route has the smallest sum the path ETX from that node and the link ETX of that node. The path ETX is therefore the sum of link ETX values along the entire route. The component `tos/lib/net/ctp/CtpRoutingEngineP` implements the routing engine.

6.3 Forwarding Engine

The component `tos/lib/net/ctp/CtpForwardingEngineP` implements the forwarding engine. It has five repsonsibilities:

1. Transmitting packets to the next hop, retransmitting when necessary, and passing acknowledgment based information to the link estimator
2. Deciding *when* to transmit packets to the next hop
3. Detecting routing inconsistencies and informing the routing engine
4. Maintaining a queue of packets to transmit, which are a mix of locally generated and forwarded packets
5. Detecting single-hop transmission duplicates caused by lost acknowledgments

The four key functions of the forwading engine are packet reception (`SubReceive.receive()`), packet forwarding (`forward()`), packet transmission (`sendTask()`) and deciding what to do after a packet transmission (`SubSend.sendDone()`).

The receive function decides whether or not the node should forward a packet. It checks for duplicates using a small cache of recently received packets. If it decides a packet is not a duplicate, it calls the forwading function.

The forwarding function formats the packet for forwarding. It checks the received packet to see if there is possibly a loop in the network. It checks if there is space in the transmission queue. If there is no space, it drops the packet and sets the C bit. If the transmission queue was empty, then it posts the send task.

The send task examines the packet at the head of the transmission queue, formats it for the next hop (requests the route from the routing layer, etc.), and submits it to the AM layer.

When the send completes, sendDone examines the packet to see the result. If the packet was acknowledged, it pulls the packet off the transmission queue. If the packet was locally generated, it signals sendDone() to the client above. If it was forwarded, it returns the packet to the forwarding message pool. If there are packets remaining in the queue (e.g., the packet was not acknowledged), it starts a randomized timer that reposts this task. This timer essentially rate limits CTP so that it does not stream packets as quickly as possible, in order to prevent self-collisions along the path.

7. Citations

Rodrigo Fonseca
473 Soda Hall
Berkeley, CA 94720-1776

phone - +1 510 642-8919
email - rfonseca@cs.berkeley.edu

Omprakash Gnawali
Ronald Tutor Hall (RTH) 418
3710 S. McClintock Avenue
Los Angeles, CA 90089

phone - +1 213 821-5627
email - gnawali@usc.edu

Kyle Jamieson
The Stata Center
32 Vassar St.
Cambridge, MA 02139

email - jamieson@csail.mit.edu

Philip Levis
358 Gates Hall
Computer Science Laboratory
Stanford University
Stanford, CA 94305

phone - +1 650 725 9046
email - pal@cs.stanford.edu

8. Citations

[1] TEP 124: Link Estimation Extension Protocol

[2] Philip Levis, Neil Patel, David Culler and Scott Shenker. "A Self-Regulating Algorithm for Code Maintenance and Propagation in Wireless Sensor Networks." In Proceedings of the First USENIX Conference on Networked Systems Design and Implementation (NSDI), 2004.