

# CS344A Assignment 2: Transport

Due: 11:59 PM, May 4, 2006

## Basics

In this assignment, you'll gain experience and a better understanding of some of the challenges and complexities in low-power wireless networking by designing and implementing a reliable network transport protocol. The protocol supports communication between a single node pair at any time: it does not support multiple concurrent flows.

The protocol has three basic components: route discovery/maintenance, flow control, and reliability. The first establishes routes between node pairs and updates them during a flow if, for example, a node along the route disappears. The second controls the rate at which packets are transmitted in order to maximize throughput. The third ensures that all of the data arrives at the destination.

While the protocol needs to optimize its transmission bandwidth, it also needs to do so without sacrificing energy efficiency.

## Protocol Requirements

Your protocol must be able to reliably deliver a medium-sized (6K) data item. The destination is a fixed node in the network. The source can be any node in the network. This is essentially reliable transport up a collection tree. The source initiates and terminates the protocol. If the destination is already receiving a flow when a node tries to start a new one, the attempt to start a new flow should fail.

Your protocol must be able to adjust to nodes failing while a transfer is in progress. It does not have to consider the source or destination failing. While nodes along a selected route may fail, the protocol does not have to consider the case where the source and destination are completely disconnected. The protocol must be able to handle asymmetric links.

The source and destination must be able to store the entire data item in memory (your protocol cannot use more than 3K of RAM). There should be an end-to-end check that the data item has arrived uncorrupted.

## Interface

Your protocol should provide two interfaces:

```
interface ReliableTransportSend {
    command void* getData();
    command uint16_t getLen();
    command uint16_t getMaxLen();
    command error_t setLen(uint16_t len);
    command error_t send();
    event void sendDone(error_t err);
}

interface ReliableTransportReceive {
    event void received(void* data, uint16_t len);
}
```

You must use these interfaces exactly as they are, with these names.

If the protocol cannot establish a flow because it has no route to the fixed root, then send should return FAIL. If the protocol cannot establish a flow because the current node is already part of a flow, it send should return EBUSY. If the protocol cannot establish a flow because some other part of the network already has a flow, then sendDone should pass EBUSY.

These are the steps to sending a flow:

1. Call setLen() to set the length of the data item. If a flow is in progress, this should return EBUSY. If it is too large, it should return EINVAL.
2. If setLen() was successful, use getData() to fill in the data.
3. Call send() to start the transfer.
4. Handle sendDone().

The protocol should be encapsulated in a component named ReliableTransportC, which has this signature:

```
generic configuration ReliableTransportC(uint16_t maxLen) {
  provides interface ReliableTransportSend;
  provides interface ReliableTransportReceive;
}
```

The maxLen parameter determines the size of the largest data item that the protocol can send: calls to ReliableTransportSend.getMaxLen() should return this value. Your component must accept values of 6144 (6K) and smaller. You may not increase the size of message\_t.

ReliableTransportC should automatically wire all Init/StdControl/SplitControls as needed. The TinyOS programming manual has a section on how to do this and why it is useful.

## Evaluation

If  $R$  is the time the radio is powered and  $T$  is the time between a successful call to send() and a sendDone() with a successful transfer, then one way to measure the performance of your protocol is  $RT$ , or the product of overall transmit time with the time the radio is awake.

We will evaluate your protocol by wiring application driver components to your ReliableTransportC and measuring how long it takes to send data items of different sizes. We will also test whether your protocol works on real nodes and that its performance is not tremendously different.

The group whose protocol performs best under our test cases will receive a small prize. However, don't let this completely dominate your approach. From a research standpoint, a clean and simple protocol with a description of several optimization possibilities that you did not have time to get to is better than a brittle and tweaked protocol that takes advantage of some knowledge of TOSSIM's peculiarities.

## Resources

Good papers that might get you thinking about this assignment include several papers on the reading list under Routing and Transport, including Woo et al., Hull et al., and Kim et al. The first deals with selecting routes, the second deals with congestion control, and the third deals with throughput and end-to-end acknowledgments.

## Handing In

All of your code should be in a single directory which can work properly if placed in tos/lib. It should be reasonably documented (e.g., explain, at a high level, what each function does if it's not very simple). The directory must have a README that describes your algorithm and how it works. If you have any TOSSIM test scripts, include those as well, with a description of their expected behavior in comments. Send a tarball of this directory to Nelson.