CSE 160 Computer Networks

University of California, Merced

Project 3 – Reliable Transport and Congestion Control

Introduction

Your assignment is to design and implement a protocol for reliable transport, that is, a simplified version of TCP. Your code will complement and make use of the code you wrote for previous assignments to do flooding, and routing.

Objectives and Goals

Your job is to design and implement a transport protocol with the following features. For transport packets, use the PROTOCOL_TCP protocol. The packet payload should be a struct you created which includes ports, flags, advertise windows, etc.

Connection setup/teardown: A connection is identified by a four-tuple, the combination of a source and destination address plus a source and destination port value.

Unlike the real TCP, packets with the SYN, ACK, or FIN flag never carry payload data. You must support multiple, concurrent connections. We suggest that you define your own transport connection structure (i.e. socket_t), with your node having an array of such structures, one per connection in use. The structure will encode all state associated with a connection, including sequence numbers, buffered data (both sent awaiting acknowledgment and possible retransmission, and received awaiting processing by the application), and connection state (such as established, the SYN has been sent but not acknowledged, etc.).

Reliability and Sliding Window: Each payload packet should be transmitted reliably by using sequence/acknowledgment number field and timeouts and retransmissions.

1. The sequence number advances in terms of bytes of data that are sent.
2. The acknowledgement number operates as in TCP to give the next expected in-order sequence number, and an acknowledgement should be sent every time that a data packet is received. That is, the acknowledgement number does not increase when a packet has been lost until that packet has been retransmitted and received. (Note that since an ACK never carries data, we put the acknowledgment number in the Transport header sequence number field.)
3. Note that packets carry both a sequence number in the packet header (which is unique among all packets sent by this node), and a sequence number in every transport header

(which is the place within the byte stream for this data or ACK packet). These two roles for sequence number are semantically distinct, and in fact, the IP header has its own unique identifier field separate from the sequence number in TCP's header.

4. We recommend that you first implement a "stop and wait" style scheme, where only a single packet can be outstanding at a time. Once that is working, implement a fixed-size sliding window.

Flow control: Your protocol should use the advertised window field along with the sequence and acknowledgement numbers to implement flow control so that a fast sender will not overwhelm a slow receiver. The advertised window field tells the other end how much buffer space is available to hold data that has not yet been consumed by the application. Note that this will not be much of an issue in this assignment (as the in-node protocol may process data immediately, as it arrives, rather than having to buffer it until a separate application is ready to receive it) but it must work and will be needed by the next assignment. You can drop data that is to the next expected byte. In fact, it would be a VERY good idea to implement a dummy application that just reads data on the receiver (this will be very useful for project 4).

Requirements

Create the following commands in command in run.py:

`cmdTestServer([address],[port])`

Initiates the server at node **[address]** and binds it to **[port]**. After this it listens for connections. Once a client attempts a connection the server will try and accept it. If the connection is accepted a new socket is made for that connection and your server continues to listen. So the following steps should be taken (pseudo-code):

```
cmd issued
     global fd = socket();
     socket address =  NODE_ID, [port] //Only source info.
     bind(fd, socket address);
     startTimer(Attempt_Connection_Time);


timer fired:
     int newFd = accept();
     if newFd not NULL_SOCKET
          add to list of accepted sockets
     for all sockets added
          read data and print
```

Note: Some small details are missing such as checks to see if commands are successful. Be sure to take this into consideration.

cmdTestClient([dest],[srcPort],[destPort],[transfer])

Initiates the client and binds it to **[srcPort]**, attempts to make a connection to **[dest]** at port **[destPort]**. After a connection has been established, send **[transfer]** bytes to the server. The data that should be sent is a 16 bit unsigned integer increasing from 0 to **[transfer]**. Ensure that the data gets their reliable and are printed in order after all the data is received. So the following steps should be taken (pseudo-code):

```
cmd issued:
     global fd = socket();
     socket address =  NODE_ID, [srcPort] //Only source info.
     bind(fd, socket address);
     server address = [dest], [destPort]// Only dest info.
     if you are able to connect(fd, server address)
          startTimer(CLIENT_WRITE_TIMER)
          global variable amount of data equal to [transfer]

timer fired:
     if all data in buffer has been written or the buffer empty
          create new data for the buffer
          // data is from 0 to [transfer]
     subtract the amount of data you were able to write(fd,
     buffer, buffer len)
```

Note: There are some fine details missing such as sending only exactly the number of bytes written and checks to see if commands are successful. Be sure to take this into consideration.

cmdClientClose([client address],[dest],[srcPort],[destPort])

Terminates the connection gracefully on the socket that is associated with **[srcPort]**, [**destPort]**, and [**dest]**. So the following steps should be taken (pseudo-code):

```
     find fd associated with [client address], [srcPort],
     [destPort], [dest]
     close(fd)
```

Be sure to test thoroughly in a multihop network with variable noise. See noise models to learn more about this. The topology tested will be a few hop network using the routing implemented in project 2.

Use the debug channel “Project3TGen” for general debugging that will help clearly show what is happening for major events such as:

```
“Debug(1): Syn Packet Arrived from Node 2 for Port 80”
```

`"Debug(1): Syn Ack Packet Sent to Node 2 for Port 80"`

or printing out the entire data set that has been received when the connection closes.

Do not change out the packet size (28) or buffer size (128 bytes).

Hints

1. There is an interface provided for you for this assignment. Please use it and apply functionality requested by it.
2. You might find it easier to make two ways to index your storage structure, one interiorly and the other on the socket layer. This can be done by using a data structure, such as a hashmaps, to point to the same piece of data. The exterior method of indexing will use the file descriptor to point to files while the interior method will be using the socket address. When reading and writing you will use your file descriptor when receiving and sending syn/acks/data/fin.
3. Section 5.2.3 Figure 5.7 of your textbook Computer Networks: A System Approach 6th Edition has a state diagram that can be used to understand your state transitions.
4. You **DO NOT** need to implement **congestion control** for this project (You could do it for **extra credit**, see below).
5. The header information should include the Source Port, Destination Port, Sequence, Acknowledgement, Flags, Advertised Window, Data.

Mid-Review

What is expected on the due date is the following:

- Setup and teardown
- Start of data transfer, if possible, stop and wait.

Deliverables

What is expected on the due date is the following:

- Source code of the TinyOS implementation with working TCP and flow control
- A single page document describing the design process and your design decisions.
- A document with short answers to the discussion questions.

A physical copy of the document and related questions is required on the due date. All documents and a copy of the source code should be submitted to class web page. Please create a

compressed tarball such as Student-Name-proj3.tar.gz. You must do this before class on the day that it is due.

Additionally, you will need to demonstrate that the code you submitted works in the next lab, and be able to describe how it works. Also discuss the design process in the code.

Discussion Questions

1. Your transport protocol implementation picks an initial sequence number when establishing a new connection. This might be 1, or it could be a random value. Which is better, and why?
2. Your transport protocol implementation picks the size of a buffer for received data that is used as part of flow control. How large should this buffer be, and why?
3. Our connection setup protocol is vulnerable to the following attack. The attacker sends a large number of connection request (SYN) packets to a particular node, but never sends any data. (This is called a SYN flood.) What happens to your implementation if it were attacked in this way? How might you have designed the initial handshake protocol (or the protocol implementation) differently to be more robust to this attack?
4. What happens in your implementation when a sender transfers data but never closes a connection? (This is called a FIN attack.) How might we design the protocol differently to better handle this case?

Please be concise with your answers.

Grading Guidelines

Each part of the project is graded on a 5 point (0-4) scale, multiplied by the weight of the project. The weighted grades from all parts of the project are added together to produce the final grade.

The five point scale is based on how well you show your understanding of the problem, and in case of code how well your implementation works:

0 – nothing turned in

1 – show minimal understanding of the problem / most things don't work

2 – show some understanding of the problem / some things work

3 – show a pretty good understanding of the problem / most things work

4 – show excellent understand of the problem / everything works

The weights for project 3 are:

80% - Transport Layer
10% - Mid-review grade
5% - Write-up design decisions
5% - Discussion questions

Your submission will be graded on correctness, completeness, and your ability to explain its implementation.

Congestion Control (Optional Extra Credit)

Goal

Design and implement a method to utilize links efficiently when multiple nodes are congesting the network. Feel free to use any combination of support at the sender, receiver, or intermediate forwarding nodes -- you aren't constrained to only TCP-style dynamic windows, although that is obviously a valid design point. You should still be able to successfully transfer files to (and through) other nodes that do not implement congestion control (e.g., when there isn't congestion!).

A correct implementation should try to do something similar to slow start + AIMD as a minimum, in order to find the path equilibrium point (slow start) and stay around the equilibrium point once it is reached (AIMD).

In addition, you will need to design some test cases to illustrate the behavior of your design.