**Computer Networks: Project Report – Assignment 2**

10/12/2015

Thanmai Mallesha Bindi

**Design Header:**

Header Fields that have been used:

1. Sequence number (32 bit)
2. ACK number (32 bit)
3. ACK flag, FIN flag (8 bit)
4. length - length of the data in the packet (16 bit)
5. Data (1460 bits)

Reliable UDP Header Format

|  |  |  |
| --- | --- | --- |
| Source Port | | Destination Port |
| Sequence Number | | |
| Acknowledgement Number | | |
| ACK Flag | FIN Flag | Length |
| Data | | |

**Implementation:**

1. Sliding Window Algorithm:
   1. There is a Sender Window, operations it does are: (Queue Datastructure, Dequeue built-in type C++)
      1. Prepare & Store all the segments to be sent in Window buffer.
      2. Slide all the data in the window to Client.
      3. Waits for Ack and Pop’s out segment when it has been ACKed by Client.
      4. I have used FIFO logic to store the store & pop segments from window.
   2. There is a Receiver Window, operations it does are: (Queue Datastructure, Dequeue built-in type C++)
      1. Check if the packet is In Order or Out of Order. (Supports Out of Order packets)
      2. If In Order, send ACK.
      3. If Out of Order, store packet in window in its designated position.
      4. Check Window for inorder packets. Send Aggegrated ACK if order is found in the window.
2. Jacobson/Karels algorithm, Adaptive Retransmission:
   1. Timeout is calculated using Jacobson/Karels algorithm
   2. Initial Estimated Timeout is set to 500 micro seconds.
   3. Sample RTT is not calculated for retransmitted segments, which are marked.
3. Congestion Control:
   1. Congestion Control has 2 states:
      1. Slow Start
      2. Congestion Avoidance
   2. If Slow Start then Congestion Window/Sender Window will grow exponentially, Cwnd \*= 2;
   3. If Congestion Avoidance then Congestion Window will increase by 1MSS each cycle.
4. Simulation: Loss/Delay
   1. Loss is simulated with the help of rand() function at the client end. Client will randomly drop the packets based on the input by the user.
   2. Delay is simulated with the help of usleep() function at the client end. Client will randomly sleep for a while before processing the segments.

**Class Files/Header Files:**

**RudpSegment: This class maintains 1 MSS.**

unsigned char -> byte

Members:

1. Sequence Number (byte4)
2. Acknowledgement Number (byte4)
3. ACK Flag (byte)
4. FIN Flag (byte)
5. Length (byte2)
6. Data (byte\*)

This is the same Reliable UDP Segment in both Client & Server. It has prepare\_request & init\_request that will serialize data while parse\_request, parse\_response will deserialize data.

**Segment: This class helps in reading data from a file given the sequence number.**

Members:

1. Initial Sequence Number
2. File
3. Length of File
4. RudpSegment

It is implemented such as to help Adaptive Retransmit. With the help of initial sequence number and the requested sequence number I can figure out the pointer to read from. This has a method called getRudp, get\_segment which will return Rudp or the byte\* segment.

**Sliding: This class is used with deque to store the sent packets. (server sliding window)**

Members:

1. RudpSegment
2. Sent\_time (to calculate Sample RTT)
3. Sequence Number
4. Mark (false, if it has not been retransmitted; true, if it has been retransmitted)

**RecvBuffer: This class is used with deque to store the received packets. (client sliding window)**

Members:

1. Sequence Number received
2. Data

**Congestion: This class holds the congestion state variables.**

Members:

1. Cwnd
2. Ssthresh
3. dupAck
4. is\_slow (true, if slow start; false, if congestion avoidance)
5. estimated\_rtt
6. dev\_rtt
7. timeout\_interval

Based on the progress of the segments in the network, this will change the state & update accordingly. Slow Start updates with cwnd \*= 2, Congestion Avoidance updates with

cwnd += 1.

**ServerSock & ClientSock: This will help in opening a port with a UDP connection using sockets.**

**Congestion Control Logic:**

Loop(forever){

Send Data, Add Sent Segments to Window

Wait for an Event:

1. Timeout
2. ACK/ Read Event

}

send\_segments() function implements the congestion control logic which will switch between states to support Additive Increase/ Multiplicative Decrease.

**Out of Order Logic:**

Once a packet arrives I will verify if the packet is in order, if it is order I will not store it in my window and send an ACK for the next sequence number. If there is an out of order packet I will store it in my window at the position from the last byte read. There are 2 generic out of order cases that has been handled:

* + If the arrived packet was out of order and the packet arrived should go on the right
    1. [-1,X] ==> [-1,X,Y]
    2. [-1,X] ==> [-1,X,-1,Y]
    3. [] ==> [-1,-1,X]
  + If the arrived packet was out of order and the packet arrived should go on the left
    1. [-1,X,-1,Z] ==> [-1,X,Y,Z]

**How to Run the Code:**

Go to Server directory:

make all

./server <port\_number> <advertised\_window>

Go to Client directory:

make all

./client <localhost> <port\_number> <file\_name.txt> <advertised\_window> <mode> <percentage>

window\_size : input in terms of MSS

drop/delay/both : 0 None, 1 Drop, 2 Delay

percentage : Integer , (0-100) Ex: 33, 33 out of 100 packets will be dropped

Example for file transfer without drop:

1. Make all on server & client.

./client localhost port mb.txt 100 0 0

* 1. 100 – window, 0 – None , 0 – percentage

1. With drop (to test the slow start to congestion and vice versa state traversal):

make all on server & client.

./client localhost 9090 mb.txt 100 1 1

* 1. 100 – window, 1 – Drop, 1 – percentage of drop

./client localhost 9090 mb.txt 100 1 30

* 1. 100 – window, 1 – Drop, 30 – percentage of drop, this will verify timeout case where there is a lot of packet drop in it.
  2. With delay

./client localhost 9090 mb.txt 100 2 0

* + 1. 100 – window, 2 – delay input, 0 – no percentage. Delay is a random number of micro seconds that will be delayed.