Cs3331 Networks Assignment 1 Report

Mathew Vo z5019176 (standard version of assignment)

1. A brief discussion of how you have implemented the STP protocol. Provide a list of features that you have successfully implemented.

In my STP protocol, I have managed to implement most of the standard (not extended features). These included:

- Multithreaded Sender – one thread to send data and another to receive acks
    - My sender transfer process utilizes two threads, called a sender and receiver
    - Sender threads makes packets and sends them, but sleeps if it has reached its window limit
    - Receiver receives ACKS, and removes elements from the window corresponding to the ACK. It will notify the Sender if it receives an ACK so that Sender can resume making packets
- Sender window
    - Sender Window was implemented utilizing a Queue. By adding Packets to the queue in my Sender thread, and removing Packets when they're acked in the ReceivingSender thread, I can maintain my Queue to be a direct representation of the window
- Fast retransmit
    - When my Sender's Receiver thread receives 3 duplicate ACKS (which correspond to the lowest byte unacked), the Sender will automatically resend that corresponding packet
- Single timeout
    - The timeout is configured to correspond with the lowest byte Packet which is unacked.

- The timer turns on and is restarted very similarly to figure 3.5.4 of the prescribed text:

```
loop (forever) {
    switch(event)

        event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length(data)
            break;

        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged segments)
                    start timer
                }
            break;
```

- Simulated Packet Loss
  - Packet loss is simulated through the pld(), which returns a Boolean. Pld() is called whenever a packet is to be sent in the Sender, and thus drops packets if it returns false.
- Receiver Buffer
  - The receiver's buffer was initialized through a Queue as well, as LIFO is very useful since the lower sequence number data packets will always arrive first.
  - A packet is added to the buffer if it doesn't match the Receiver's ackbase field, which represents the next expected sequence number. (or the previous ack number of the ACK that was most recently sent out)
- Receiver cumulative ACKS
  - Cumulative ACKS were implemented quite simply by sending one ACK to ACK many corresponding data packets.
  - In the Receiver, when a data packet arrives that fills in a gap in the receiver's byte stream, the packet is read, as well as the entire buffer of packets. Then, an ACK is made to ACK only the last packet in the buffer.
  - In the Sender, the Sender's receiver would remove multiple ACKed packets in the Sender's window in response to a cumulative ACK.

In terms of implementation of the stpPacket, it was necessary to be able to convert it into a byte array, as well as convert a byte array back into an object. This is as DatagramPacket()'s data parameter is a byte array. Thus, two methods had to be designed, getBytes() and getObject(). These utilized bit shifting and knowledge of the header to be able to deconstruct the object into a byte

array, and then correspondingly reconstruct it back into an object. This is fundamental for the Sender and Receiver to be able to read fields in the header, as well as access the data.

1. A detailed diagram of your STP header and a quick explanation of all fields (similar to the diagrams that we have used in the lectures to understand TCP/UDP headers).

<table>
<tr><td colspan="3">int seqNumber;</td></tr>
<tr><td colspan="3">int AckNumber;</td></tr>
<tr><td>Boolean SYN;</td><td>Boolean ACK;</td><td>Boolean FIN;</td></tr>
<tr><td colspan="3">int length;</td></tr>
<tr><td colspan="3">Byte[] data;</td></tr>
</table>

In my STP Header, there are 7 fields

- Int seqNumber – 4 byte field, representing the sequence number of the packet. The sequence number is always the same as the sequence number of the last packet that was sent plus its packet data length, or is the value of the last ackNumber from the last ACK that was received
    - It is always important to note that seqNumber only counts in terms of length of data
- Int ackNumber – Similar to seqNumber, is a 4 byte field which represents the ACK number of the packet. The ack number is always the same as the sequence number of the last packet that was sent plus its packet data length.
    - ACK numbers are only read if the packet's ACK field is true
    - However, if the length of data is 0, then the ackNumber will be an increment of one of the corresponding sequenceNumber
- Boolean SYN – used to determine if the packet is for connection establishment
    - Is set to true if the packet is for connection establishment
- Boolean ACK – used to determine if the packet is an ACK packet
- Boolean FIN – is used to determine if the packet is for connection teardown
- Int length – used by both sender and receiver to determine the length of the data in the packet.
    - Critical for use to calculate the seqNumbers and ackNumbers
- Byte[] data – where text data is stored
    - Byte[] format allows for easy conversion into a byte buffer

These were deemed the only fields necessary for the scope of the assignment.

For the standard version of the assignment, answer the following questions:

(a)

With pdrop being 0.1, and an average of 10 packets in the window, the timeout has to be able to account for RTT as well as the creation and sending off the 10 packets. However, due to the low pdrop, it is rare that a packet would be dropped, and thus it is safe to allow for a low timeout value. Assuming to create one packet requires a time of A ms, a round trip time of B ms, a receiver processing time of C and that the Sender window would ideally like to continue making packets all the time. Also, since the window is so large, fast retransmit would be very powerful. Thus, a timeout of around B + 5A + C would be ideal as lost packets to timeout and be retransmitted very quickly, but still be high enough such that it would be very unlikely for packets to be sent as duplicates (for a timeout to occur before an ACK arrives back). Thus, as Australian ping is around 20ms, I'd estimate a timeout value of 50ms would be sufficient.

After performing both experiments, it is clear to see that 50ms would be sufficient for Experiment 1 (pdrop of 0.1). This is as the receiver receives 0 duplicate segments, and hence the timeout value to allow for an ACK to arrive back before a timeout occurs. The data sample drops only 2 packets. For the first packet, fast retransmit is utilized and quickly resends a packet. Even in this scenario, the fast retransmit is still sent, and the corresponding ACK is received before the timer even runs out. Also due to the low pdrop, it is less likely for retransmitted packets to be dropped as well, making fast-retransmit more efficient too. 2 packets are dropped in this sequence, one with sequence number 1351 and one with sequence number 1501.

In the second experiment, there is one duplicate segment received by the Receiver, which occurs when the Sender sends a packet through timeout where an existing ACK from the Receiver may have already been in transit. Thus, due to the higher pdrop value, more packets are dropped, resulting in more lost packets and thus resulting in more duplicate ACKS from the receiver. Thus, for this experiment, a higher timeout would have been desirable. In the results, packets with sequence number 101, 251, 351, 401, 1101, 1251, 1301 were dropped, and data packet with sequence number 1101 was retransmitted twice.

(b)

Timeout Value (ms)	Original Segments Sent	Retransmitted Segments Sent	Duplicate ACKS received	Total Packets Sent	Time to finish since program execution (ms)
50	40	2	11	42	109
200	40	2	8	42	107
12.5 (13)	40	4	10	44	133

Analyzing the data, all timeout values all yielded relatively similar results. Time to complete, as well as packets sent were highest for the timeout of 13ms. This is to be expected as the timeout is exceptionally low, and excess packets would have been retransmitted due to the timer as well as fast retransmit. Surprisingly, time to complete for a timeout value of 50ms and 200ms were extremely consistant and similar; however this is simply one data sample and may have been subject to computer latency/lag. In this case, the data is consistant because of the low pdrop value, as well as the safety of the computing environment (no packet loss from the Receiver). Also, all retransmitted segments were due to the fast retransmit (by analyzing the Sender_log.txt file), and thus the timeout value had very little impact on program time to finish. Therefore, a timeout value of 50 was extremely reliable, and if more packets were lost, the lowest timeout value in comparison to 200 would have been beneficial in resending packets that would not have transmitted through fast retransmit. However, a timeout of 13 was far too slow and resulted in more packets having to be resent.

Appendix

Experiment 1: (test1.txt) pdrop = 0.1, MWS = 500 bytes, MSS = 50 bytes, seed = 300, timeout = 50ms

Type	time (since program execution)	type of packet	seqNum	data.length()	ackNum
rcv	1587	D	1	50	1
rcv	1588	D	51	50	1
rcv	1588	D	101	50	1
rcv	1589	D	151	50	1
rcv	1589	D	201	50	1
rcv	1590	D	251	50	1
rcv	1591	D	301	50	1
rcv	1592	D	351	50	1

rcv	1592	D	401	50	1
rcv	1593	D	451	50	1
rcv	1593	D	501	50	1
rcv	1593	D	551	50	1
rcv	1594	D	601	50	1
rcv	1595	D	651	50	1
rcv	1595	D	701	50	1
rcv	1596	D	751	50	1
rcv	1596	D	801	50	1
rcv	1597	D	851	50	1
rcv	1597	D	901	50	1
rcv	1597	D	951	50	1
rcv	1598	D	1001	50	1
rcv	1598	D	1051	50	1
rcv	1599	D	1101	50	1
rcv	1599	D	1151	50	1
rcv	1599	D	1201	50	1
rcv	1600	D	1251	50	1
rcv	1600	D	1301	50	1
rcv	1600	D	1401	50	1
rcv	1601	D	1451	50	1
rcv	1602	D	1551	43	1
rcv	1603	D	1351	50	1
rcv	1692	D	1501	50	1
rcv	1693	F	1594	0	1

1. Amount of Data Received: 1593
2. Number of Original Data segments received: 32
3. Number of duplicate segments received: 0

Experiment 2: (test1.txt) pdrop = 0.3, MWS = 500 bytes, MSS = 50 bytes, seed = 300, timeout = 50ms

rcv	1837	D	1	50	1
rcv	1838	D	51	50	1
rcv	1839	D	151	50	1
rcv	1840	D	201	50	1
rcv	1840	D	301	50	1
rcv	1841	D	451	50	1
rcv	1842	D	501	50	1
rcv	1843	D	101	50	1
rcv	1843	D	551	50	1
rcv	1844	D	601	50	1
rcv	1845	D	651	50	1
rcv	1845	D	701	50	1
rcv	1845	D	251	50	1
rcv	1846	D	751	50	1
rcv	1846	D	801	50	1
rcv	2100	D	351	50	1
rcv	2101	D	851	50	1
rcv	2151	D	401	50	1
rcv	2153	D	901	50	1

rcv	2153	D	951	50	1
rcv	2154	D	1001	50	1
rcv	2155	D	1051	50	1
rcv	2156	D	1151	50	1
rcv	2157	D	1201	50	1
rcv	2158	D	1351	50	1
rcv	2158	D	1401	50	1
rcv	2158	D	1451	50	1
rcv	2158	D	1501	50	1
rcv	2159	D	1101	50	1
rcv	2161	D	1101	50	1
rcv	2162	D	1551	43	1
rcv	2213	D	1251	50	1
rcv	2266	D	1301	50	1
rcv	2269	F	1594	0	1

Amount of Data Received: 1593

Number of Original Data segments received: 32

Number of duplicate segments received: 1