COMP3331 Computer Networks and Applications Assignment Report

Implementation of reliable data transfer over UDP

Student ID: z3278107

Student Name: Liam Chen

Tutorial: T18B

# Summary of features

|  |  |
| --- | --- |
| Features achieved in this submission | |
| 1. | Three-Way handshake |
| 2. | Four-segment connection termination |
| 3. | Single timer for timeout operation, has bug, found in part b but not able to debug |
| 4. | PTP header including seq, ack and flags |
| 5. | Cumulative ACKs |
| 6. | Fast Retransmission |
| 7. | Sliding Window, sender window |
| 8.  9.  10.  11. | Selective Repeat, buffer for out of order packets  Reliable data transfer, file can be transferred reliably  PL Module to simulate packet loss  One receiving thread, main thread is used for sending data (bugs detected, however, due to lack of experience, I was unable to fully debug it) |

Although most of the features have been implemented, due to lack of experience in concurrency, some aspects may not be perfect.

# Description of Design

The Sender and Receiver are written in Python3. The Receiver program implements section 4.6 of the specification. It is able to set up PTP connection using three-way handshake, keep listening to the port and receiving file until FIN is received from the Sender. It buffers out any out of order packets and use cumulative acks to let the Sender know which packet to send next. It maintains the order or packet when writing data into file. Upon the reception of FIN, it closes PTP connection.

The Sender establishes connection using three-way handshake. Once connection is established, it open and reads a file (assume file exists in the directory) and resize line length based on the maximum segment size given by the user. It then starts sending packet to the PL module and start a timer on packet. During the sending phase, while the program hasn’t reached the end of file, if the maximum window size is not full, the sender keeps sending data. If the oldest packet is acknowledged by the Receiver, the sender window shifts by one. In the event when cumulative ack is received, the window could shift multiple steps. Every packet is sent to the PL module before it is sent to the Receiver, this includes the retransmission. The Sender is able to handle timeout and triple duplicate acks

* On timeout event, the Sender retransmit the oldest unacknowledged packet on expiry.
* On a triple duplicate event, the Sender resends the triple duplicate acknowledged packet.

For every packet sent to the PL module, the sender is also trying to receive an ack from the Receiver. Once all files are sent and the sender window becomes 0, the Sender sends a FIN to Receiver to close the connection. Multithread is used to handle sending and receiving. However, the correctness is not guaranteed due to lack of experience and knowledge.

# PTP header

The PTP segment is created using function: create\_segment(seq, ack, flags, payload).

|  |  |  |  |
| --- | --- | --- | --- |
| UDP Header | | | |
| Sequence Number | | | |
| Acknowledge Number | | | |
| FIN flag | SYN flag | ACK flag | Data flag |
| Payload | | | |

The sequence number and acknowledge number are similar to TCP seq and ack. The flags section is a simplification of the TCP flag section. Because we only have four statuses in PTP, I used four bits to represent FIN, SYN, ACK and DATA. For instance, if the flags are ‘1000’, it means it is a FIN. This is to minimise the size of the segment. A typical segment looks like ‘151’| ‘50’| ‘0001’ | ‘Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say’. Noted that the delimiter only works for text file without |. This can be improved by more complicated delimiter. However, this report doesn’t cover this feature.

# Experiments

1. Use the following parameter setting: pdrop = 0.1, MWS = 500 bytes, MSS = 50 bytes, seed = 300.

Various timeout values have been experimented in this section. To identify a suitable value for timeout, I started with 20 ms and increment the timeout value by 20 ms. The number of duplicates received by the receiver is very small. However, I do notice the difference when I’m running the programs on my own machine and when I ran the programs on the CSE machine. When they are running on CSE machines, the number of duplicates is very small. I decided to use 60 ms.

|  |  |
| --- | --- |
| Timeout value (ms) | No. of duplicates received by Receiver |
| 20 | 65 |
| 40 | 20 |
| 60 | 6 |
| 80 | 4 |

Then I ran the tests for different probability.

|  |  |
| --- | --- |
| Pdrop | Sender statistics @ 60ms |
| 0.1 | Amount of Data Transferred: 32768  Number of Data Segments Sent: 656  Number of Packets Dropped: 62  Number of Retransmitted Segments: 62  Number of Duplicate Acknowledgements received: 356 |
| 0.3 | Amount of Data Transferred: 32768  Number of Data Segments Sent: 656  Number of Packets Dropped: 281  Number of Retransmitted Segments: 281  Number of Duplicate Acknowledgements received: 444 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case | timeout | Pdrop | Sender statistics | Time |
| i | 60 ms | 0.1 | Amount of Data Transferred: 262144  Number of Data Segments Sent: 5243  Number of Packets Dropped: 581  Number of Retransmitted Segments: 581  Number of Duplicate Acknowledgements received: 3074 | 43.763s |
| ii | 240 ms | 0.1 | Amount of Data Transferred: 262144  Number of Data Segments Sent: 5243  Number of Packets Dropped: 580  Number of Retransmitted Segments: 580  Number of Duplicate Acknowledgements received: 3074 | 147.93s |
| iii | 15 ms | 0.1 | Amount of Data Transferred: 16850  Number of Data Segments Sent: 354  Number of Packets Dropped: 36  Number of Retransmitted Segments: 42  Number of Duplicate Acknowledgements received: 212 | 1.111s |

Tcurrent in part a is 60ms. I noticed that when the timeout value becomes small, after a threshold, my program becomes unstable. The reliability is not guaranteed anymore. I think there are bugs in my multithreading part. I tried to debug but wasn’t able to solve the problem. Compare two of the three results. I can see that the 147.93s / 43.763s = 3.38. The time it takes for the programs to transfer files is proportional to the timeout value. By examining the file, with larger timeout value, reliability is guaranteed.

# Table Description automatically generatedAppendix I Part B

Table

Description automatically generatedTable

Description automatically generated

# Appendix II Part a

Table

Description automatically generatedTable

Description automatically generated0.1 60ms

0.3 60

Table

Description automatically generatedTable

Description automatically generated