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 |
| 4. | PTP header including seq, ack and flags |
| 5. | Cumulative ACKs |
| 6. | Fast Retransmission |
| 7. | Sliding Window, sender window |
| 8.  9.  10. | Selective Repeat  Reliable data transfer, buffer for out of order packets  PL Module to simulate packet loss |

Note: non-blocking or multithreading in Sender.py NOT achieved due to lack of knowledge in concurrency theory.

# 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.

Because I’m not able to use multithreads or non-blocking mechanism. I had to set a very small number for socket timeout. On the event when receiving is blocked, I continue to the next packet. I have tried to use locks and events, but the behaviours of my programs become very unpredictable. I decided to leave this feature for future improvements.

# 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.

# 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 10ms and doubled the value until there is no duplicate segments received by the receiver, then I gradually reduce the timeout value for optimisation. The file I’m testing on is the 32KB.txt. As shown, when the timeout value is low, the number of duplicates segment vary significantly. The ideal timeout will be as small as possible because we don’t want to keep resending a packet to the Receiver.

|  |  |
| --- | --- |
| Timeout value (ms) | No. of duplicates received by Receiver |
| 10 | 749 |
| 20 | 348 |
| 40 | 23 |
| 80 | 5 |
| 90 | 1 |
| 100 | 0 |

From the table above, 100 ms is my selected timeout value. At this timeout, the number of duplicate segments received is 0. With increment in pdrop, the number of packets dropped significantly increased.

|  |  |
| --- | --- |
| Pdrop | Sender statistics @ 100ms |
| 0.1 | Amount of Data Transferred: 32768  Number of Data Segments Sent: 1238  Number of Packets Dropped: 126 |
| 0.3 | Amount of Data Transferred: 32768  Number of Data Segments Sent: 1080  Number of Packets Dropped: 459 |

1. Tcurrent in part a is 100ms.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Pdrop | Sender statistics |
| Case i | 100 ms | 0.1 | Amount of Data received: 262144  Number of Data segments Received: 9939  Number of duplicate segments received: 0 |
| Case ii | 400 ms | 0.1 | Amount of Data Transferred: 32768  Number of Data Segments Sent: 1080  Number of Packets Dropped: 459 |
| Case iii | 25 ms | 0.1 |  |

# Appendix