COMP3331 Computer Networks and Applications

Assignment Report – Simple Transport Protocol (STP)

-by Aryaman Sakthivel (z54557585)

Objective:

The objective this assignment is to implement a reliable transport protocol over the UDP. This reliable transport protocol is referred to as Simple Transport Protocol (STP). Features implemented in this protocol include timeouts, sliding windows, unreliable channel simulation, buffers, states, etc.

There are distinct STP endpoints, "sender" and "receiver", respectively. Data packets flow in the "forward" direction from the sender to the receiver, while acknowledgments flow in the "reverse" direction from the receiver back to the sender. STP also implements a sliding window protocol wherein multiple segments can be sent by the sender in a pipelined manner.

Languages used:

The programming language used to implement this protocol is Python.

Code organization:

The protocol is implemented in two python scripts named, “sender.py” and “receiver.py”. Both scripts begin with the python3 shebang and use utf-8 encoding.

They include necessary imports such as: **sys, os, socket, random, pickle, time, threading**.
The sender script includes classes such as: **STPPacket, FLP, RLP** and **Sender.** While the receiver script uses classes such as: **STPPacket** and **Receiver.**

The sender script creates a text file named **Sender_log.txt** which logs the details of each Segment sent or received by the sender. The receiver script also creates a text file **Receiver_log.txt** which logs the details of each Segment received or sent by the receiver, it also creates another text file (whose name is specified in the argument) where it appends the data received in the packets sent by the sender.

STP protocol implementation:

Sender Design:

The STP in the sender side involves 4 classes (mentioned above), 1 main thread and several functions.

The STPPacket class defined the structure of the packets, these include the data, sequence number and the flags (ACK, SYN,FIN).
The FLP and RLP classes are used to calculate the forward loss probability and reverse loss probability respectively, based on the flp and rlp values provided in the arguments.

The Sender class initializes the sender and contains all the necessary functions. It creates a socket, and includes functions to get data from the file and split it into packets, receive packets from the receiver, create packets with specified flags, send or retransmit segments over the UDP as well as update the sender log.

The sender has 5 states: CLOSED, SYN_SENT, ESTABLISHED, CLOSING, FIN_WAIT.

Sender Operation:

The main function stores the necessary arguments and creates an ISN. The trackers are initialized including a progress bar, and the socket is binded to the sender_port. The sender log is created and length of the file (to be transferred) is calculated. A timer is started, and main loop begins.

The sender sends an SYN packet and on the receival of an ACK from the receiver a 2-way handshake is established, and the sender is set to the ESTABLISHED state where its ready to send the data packets. Log entries are made for each packet.

Sliding window is initialized and the data is split and stored into a packet. The sender continues to send the packets (increasing seq_num based on the size of the data) until the window reaches max_win size or the progress bar is complete calculating the flp for each packet sent, dropped packets are retransmitted. Log entries are made for each packet.

Then the sender receives all the acks until the number of unacknowledged packets are 0, if an ack is dropped the packet is retransmitted from the buffer. Log entries are made for each ack.

This continues until the entire file has been sent, once that happens the sender sends a FIN segment and waits for the ACK, it resends FIN if it isn't received. Once, it receives an ACK, it moves to the closed state and closes the connection.

Finally, it appends all the tracking data to the sender log and displays it.

Receiver Design:

The STP in the receiver involves 2 classes, 1 main thread, 1 timer thread and several functions.

The STPPacket class is similar to that of the sender. The Receiver class is also similar to the one in the sender however it appends data to a new file given in the arguments.

The receiver has 4 states: CLOSED, LISTEN, ESTABLISHED, TIME_WAIT.

Receiver Operation:

The main function stores the necessary arguments. The trackers are initialized including a progress bar, and the socket is binded to the receiver port. Buffers are created to handle out of order packets. Receiver log is created and the main loop begins.

The receiver waits in the LISTEN state for an SYN and once it receives it, it sends an ACK and the 2-way handshake is established.

It then moves to the ESTABLISHED state and continues to receive packets sent form the sender and sends back and ACK for them. Entries are made in the log for each packet sent or received. Duplicate SYNs are also handled, if and out of order packet arrives it is appended to the buffer.
The data from the packets is appended to the new file (specified in the arguments), the buffer and sequence_buffer assist to ensure data is appended correctly.

Once the receiver receives a FIN packet it returns an ACK and moves into TIME_WAIT state where it waits for 2*MSLs then moves to the closed state and closes the connection.

Log entries are made into the receiver log for each packet sent or received, and all tracking data is appended to the receiver log. The contents of the final file as well as the receiver log are displayed.

Design Trade-offs:

- The program utilizes UDP making it simple and efficient, however it isn't as reliable as the TCP.
- The STP is a simplified version of the TCP, therefore it lacks features such as delayed acks, congestion control and extensive error handling.
- The sliding window implemented in the sender isn't comparable to the TCP's congestion control algorithms.
- Does not include timeout estimation or double timeout interval.

Code Borrowing and References:

- Sample Codes provided were used as a reference for using the Sockets and parsing ports and arguments.
  Sender.py - https://webcms3.cse.unsw.edu.au/COMP3331/24T1/resources/96898
  Receiver.py - https://webcms3.cse.unsw.edu.au/COMP3331/24T1/resources/96899.

- Code skeleton provided by brianlam38 was used as a reference for creating classes and few functions.
  https://github.com/brianlam38/Networks?tab=readme-ov-file#simple-transport-protocol