Computer Networks HW2 Report

B03902062 資工三 董文捷

1. **Flow of my program**

   (a) Preparation

   - Use `./agent <AGENT_SENDER_PORT> <AGENT_RECEIVER_PORT> <drop rate>` to execute agent
   - Agent will bind a sockfd and start to wait for `SYN` from receiver
   - Use `./receiver <AGENT_IP> <AGENT_RECEIVER_PORT> <destination file>` to execute receiver
   - Receiver sends `SYN` to agent, and receive `SYNACK` from agent.
   - Agent will bind another sockfd and start to wait for `SYN` from sender
   - Use `./sender <AGENT_IP> <AGENT_SENDER_PORT> <source file>` to execute sender
   - Sender sends `SYN` to agent, and receive `SYNACK` from agent.

   ⟶ Both agent, receiver, sender are ready for data transmission.

   (b) Data transmission

   - **Sender**
     To deal with the need for data retransimission, a segment should be kept until it is `ACKed` in order. Instead of declare a large array, I use `malloc` to allocate memory dynamically for a new segment, and store it into an array of pointer of segment. An array is used to record status of each segment. There are five defined status, `EMPTY`, `WAITING`, `ACKED`, `OUT_OF_ORDER`, and `TIMEOUT`. For each batch of segments, sender checks whether the status of a segment is `EMPTY` to see if it is already made. After a segment is sent, its status becomes `WAITING`. Since sender does not know how many `ACK` it will receive, to avoid being blocked by `recvfrom`, `select` is used to monitor `sockfd`, examining if any `ACK` segment is ready. When the corresponding `ACK` is received, the status of a segment temporarily becomes `ACKED`. If some `ACK` are not received in the timeout interval, segments with status `WAITING` become `TIMEOUT`. Sender will check if there is a gap between $base$ and $base + cwnd - 1$, those `ACKED` segments after the gap are actually `OUT_OF_ORDER` and should be resent. `ACKED` segments before the gap is received

in order by the receiver, use `free` to release allocated memory for the segment.

- **Agent**
  In the agent part, sender and receiver have their respective `PORT` and `sockfd`. Since agent can not predict the transmission time and order of sender and receiver, `select` is used to monitor both `sockfd`. When the sender agent get a data, it uses `rand` to decide whether to drop the segment or not. The receiver agent simply forwards every `ACK` segment.
- **Receiver**
  Similar to the sender part, receiver uses `malloc` to allocate memory for the newly received segment and store it into a buffer array of pointer of segment. When buffer is flushed, `free`.

(c) Finish

- When the input file ends, sender waits until all segments are `ACKed`, then it sends `FIN` to agent.
- Receiver receives `FIN` from agent, it sends `FINACK` to agent and `flush` to ensure no data is remained in the buffer.
- Agent forward `FINACK`
- Sender receive `FINACK`

2. **structure of a segment**
   `struct segment` is defined in `protocol.h`, a segment contains `struct header` and `data`. `struct header` contains `source_port`, `dest_port`, `Seq`, `ACK`, `SYN`, `FIN`, and `data_length`.

3. **Multipath transmission**
   To increase throughput, I use `pthread` in sender and agent to send and receive data simultaneously on multiple path. Different `PORT` are assigned as arguments in `pthread_create`. `pthread_mutex_t` is used to avoid race conditions on modification of global variables.

4. **Test**
   The program is tested to send a 1MB test data on the same computer (`Ubuntu 16.04 LTS`) with IP `127.0.0.1`, and to the workstation with IP got by `ifconfig`, then I use `diff` to check the correctness of the output. Something noteworthy is that in the case to transmit data to the workstation, a 50ms timeout interval may result in a premature timeout, a 100ms timeout interval seems to be more appropriate to avoid unecessary resource waste.