COMPUTER NETWORKING

Assignment 3

Implementing a Reliable Data Transport Protocol

Overall Organization

For each strategy (GBN/Stop & Wait) there are 2 classes:

- Server
- Client

For GBN:

- At first, the server expects that a client will request a file from it, so this request needs to be backed up by a timeout in case the datagram is lost.
- Then the client requests the large file from the server side, and once it is received, the server sends an ACK to indicate that the request reaches him.
- The server starts to read the file and then splits the content of it into small packets (500 bytes) to be sent.
- The cwnd = 1 at first, so the server sends one packet only, and then starts to receive the ack of this packet from the client and check the FSM of the congestion control to decide what to do depending on the current phase (slow start/congestion avoidance fast recovery) and also on the happened event (new ack/duplicate ack/timeout).
- In case of Timeout or 3 Duplicate ACKs, the server re-transmits the packets starting from the last unacked packet.
- Sending packets depending on the given probability of loss. The packet will not be sent unless its probability to be sent is greater than the probability of loss.
- When there are no more packets to be sent, the server sends a FIN packet by letting the seqno of this packet to be the maximum value of uint32_t (UINT32_MAX). And closes its socket.
- When the client receives this packet, it closes its socket too.

For Stop And Wait:

Approximately the same as the GBN except some differences:

1. The window size always equals = 1.
2. The sequence numbers have only 2 values (0, 1).
3. The ACK number and the sequence number of a packet are the same.
4. Only one duplicate ACK is enough to indicate loss.

Major functions

Server side

- **get_packets(**file**)**
  It takes the requested file and splits its content into chunks, then puts them in the form of packets to be sent.
- **send_packet(**packet pck**)**
  It prepare the given packet to be sent to the client side by converting it into a char* buffer.
- **receive_ack(**int sockfd**)**
  It receives the buffer containing the sent ACK from the client side, and then converts it to be in the form of ACK structure.
- **FSM (**pair <ack_packet, int> p**)**
  It applies the finite state machine of the congestion control. it takes the received ack and its no. bytes to check the happens of timeout/duplicate acks/ new ack, and follows the FSM to take the corresponding action.
- **greater_than_prop(**prob**)**
  It checks if the packet will be sent or not, if its probability is greater than the given, it will be sent.
- **make_fin_pck()**
  It returns a packet with a specific sequence number to be the final packet to be sent from the server.

Client side

- **request_file(**file_name**)**
  It requests the file from the server by sending it a datagram containing the file name, then receives the ack from the server to indicate that the file name reaches the server.
- **receive_pck()**
  It receives the buffer from the client side and converts it to the packet structure.
- **check_fin(**pck**)**
  It checks if the received packet is the final one or not to break and close the socket.
- **correct_ack(**pck**)**
  It checks if the received packet is matched with the expected ack number or not.
- **new_pck_actions(**pck, ack**)**
  The actions must be taken when a new correct packet is received, such as writing the datagram on the file, sending the ack and updating the ack number.
- **lost_pck_actions(**pck, ack**)**
  The actions must be taken when a new correct packet is received, such as sending a duplicate ack and increasing no. duplicate acks by 1.
- **set_seconds(**int sec**)**
  It sets a period of time for timeout on receiving packets from the server. (for the ack of the first packet)
- **send_ack(**ack_packet ack**)**
  It sends an ack packet to the server side after converting the ACK structure to a char* buffer.
- **store_file(**content**)**
  It stores the content of the requested file into another file on the client side.

Network system analysis

First, try the probability of loss to be **1%**.

Iter/Strategy	GBN	Stop And Wait
1	9 sec	10 sec, 9903 mcec
2	4 sec	10 sec, 9912 msec
3	5 sec	10 sec, 9928 msec
4	3 sec	10 sec, 9901 msec
5	3 sec	10 sec
Average	**4.8 sec**	**10 sec, 9929 msec**

Then, try the probability of loss to be **5%**.

Iter/Strategy	GBN	Stop And Wait
1	7 sec	46 sec, 46440 msec
2	6 sec	46 sec, 46520 msec
3	6 sec	47 sec
4	7 sec	46 sec, 46329 msec
5	5 sec	47 sec, 46677 sec
Average	**6.2 sec**	**46.5 sec**

Then, try the probability of loss to be **10%**.

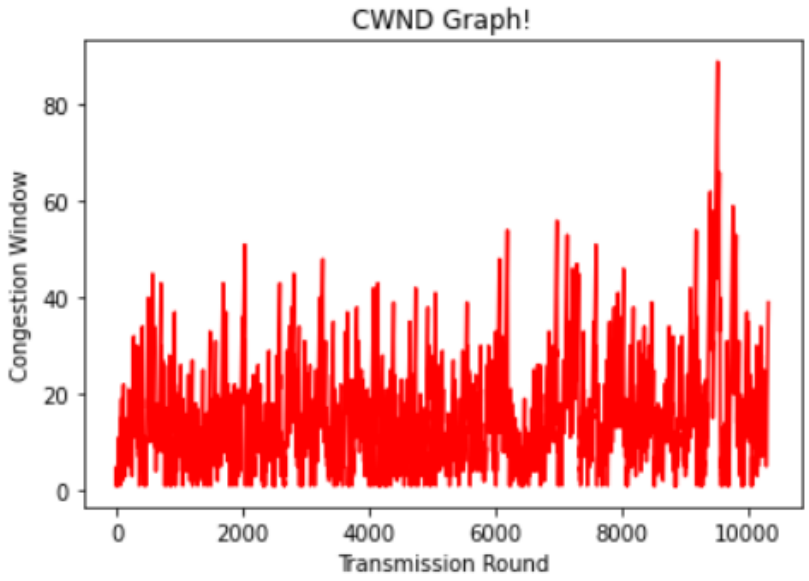
Iter/Strategy	GBN	Stop And Wait
1	18 sec	102 sec
2	17 sec	101 sec
3	18 sec	102 sec
4	18 sec	102 sec
5	18 sec	101 sec
Average	**17.8 sec**	**101.6 sec**

Then, try the probability of loss to be **30%**.

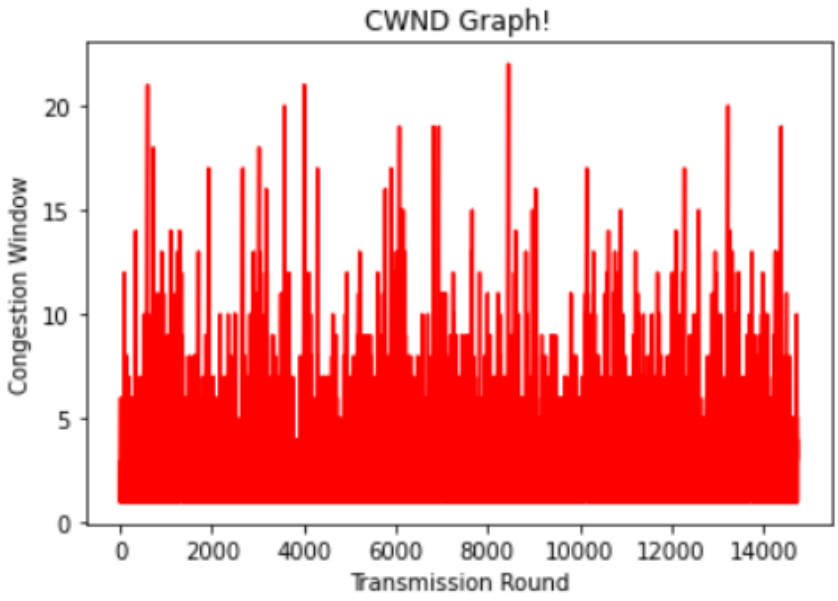
Iter/Strategy	GBN	Stop And Wait
1	290 sec	407 sec
2	289 sec	399 sec
3	282 sec	388 sec
4	282 sec	387 sec
5	281 sec	380 sec 380623 msec
Average	**27.5 sec**	**392.2 sec**

For congestion control

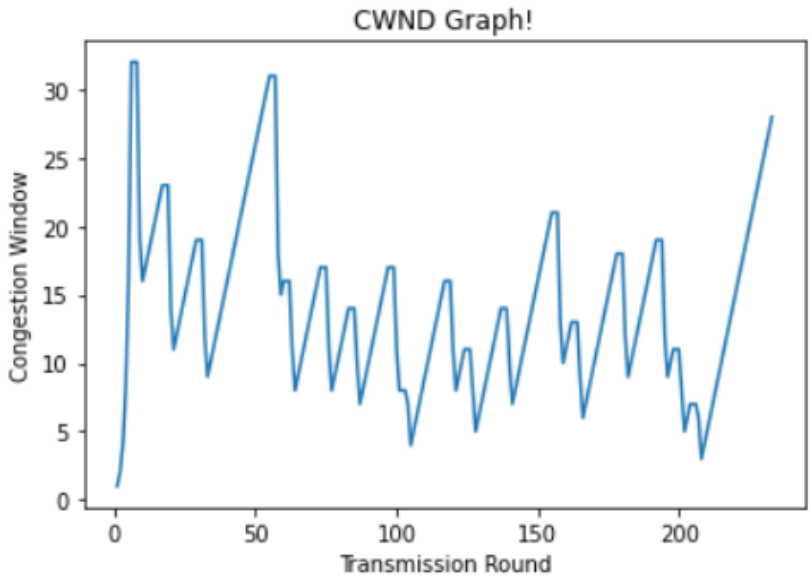
Graph obtained for probability **10%** of loss for a 4 Mbytes file. (Many duplicate Acks and many Timeouts).



Graph obtained for probability **30%** of loss for a 4Mbytes file. (Many Timeouts).



Graph obtained for probability **10%** of loss for a 90k bytes file. (Many duplicate Acks).



Graph obtained for probability **30%** of loss for a 90k bytes file. (Many Timeouts).