Networks Assignment 2

Implementing a Reliable Data Transport Protocol

Made By: Mohamed Said Ibrahem.

ID: 57

Objectives.

To implement a reliable transfer service on top of the UDP/IP protocol. In other words, you need to implement a service that guarantees the arrival of datagrams in the correct order on top of the UDP/IP protocol, along with congestion control.

2.1 Specifications->(Steps):

- Select File from Server which you want to Get inside Client.
- The File splitted into chunks of fixed length.
- Chunk data is added to a UDP datagram packet.
- Move data by reliable data transfer TCP through UDP.

2.2 Packet Types:

- Two types of packets Data packets and Ack packets.
- Used the same example mentioned in the pdf.

3 Congestion Control:

- By following the FSM description of TCP congestion control We can implement it as it is so important.

4 Packet Loss Simulation:

- Done using the probability plp since packet loss is infrequent in a localhost or LAN environment.

5.1 Flow of data: Done.

5.2 Handling time-out: N/A.

5.3 Arguments for the client: Done. Client.in

5.4 Arguments for the server: Done. Server.in

6 Network system analysis: N/A.

7 Bonus: Error detection and checksumming : Done using crc16 method (Ready implementation Wikipedia).

Client Side.

check_index(): Check if there is an error in the sequence number.

recieveFileSelectiveR(): Selective Repeat receiver side.

recieveFileSelectiveR2(): Selective Repeat receiver side.
Receiver must be able to accept packets out of order.

Since receiver must release packets to higher layer in order, the receiver must be able to buffer some packets.

stopAndWait(): Stop and wait receiver side.

Rule 1) Send acknowledgement after receiving and consuming of data packet.

Rule 2) After consuming packet acknowledgement need to be sent (Flow Control)

sendACK(): Method to send an ACK to the Server.

read_input(): Read and parse I/P from input File.

fileNameSendAndWait(): Send the file name to the server and then waits until the server replies by different responses.

gbn(): Go Back N receiver side.

```
void check_index(int base, int next_seq);

void recieveFileSelectiveR(int sock, struct sockaddr_in
servAddr, FILE* fp);

void stopAndWait(int sock, struct sockaddr_in servAddr,
FILE* fp);

struct input_client read_input(char * file_path);

void sendACK(int next_seq, int sock, struct sockaddr_in
servAddr);

int fileNameSendAndWait(int sock, struct sockaddr_in
servAddr, char * fileNew);

void gbn(int sock, struct sockaddr_in servAddr, FILE*
fp);

void recieveFileSelectiveR2(int sock, struct
sockaddr_in servAddr, FILE* fp);
```

Server Side.

getFileSize(): Calculate the file size to use it to send it back to the Client as and ACK.

read_input(): Read and parse I/P from input File.

beginProcess(): Method to start the integration between other methods together so we can start and run the Server.

sendACK() : Method to send an ACK to the Client.

selectiveRepeat() : Choose the selectiveRepeat method as a sending method.

stopAndWait() : Choose the Stop And Wait method as a sending method.

gbn() : Choose the Go Back-N method as a sending method .

```
int getFileSize(char* filename);
struct input_server read_input(char * file_path);
void beginProcess(int fileSize, int sock, struct sockaddr_in
clientAddr, char * filename);
void sendACK(int next_seq, int sock, struct sockaddr_in addr);
void selectiveRepeat(char* file_path, int sock, struct sockaddr_in
clientAddr, int fileSize);
void stopAndWait(char* file_path, int sock, struct sockaddr_in
clientAddr, int fileSize);
void gbn(char* file_path, int sock, struct sockaddr_in clientAddr, int
fileSize);
void selectiveRepeat2(char* file_path, int sock, struct sockaddr_in
clientAddr, int fileSize);
```

<u>Server.in</u>

- 7777 -> Port.
- 50 -> Window size (for GBN).
- 7 -> Seed (for randomization SR).
- 0.00 -> Prob of loss (for loss simulation).

<u>Server Steps:</u>

- **<u>Waiting for File Name.</u>**
- **<u>Make sure the File exists.</u>**
- **<u>Send File size to Client as ACK.</u>**
- **<u>Wait for ACK of File size received from Client,</u>**
- **<u>Start sending the Pckts:</u>**

1. **<u>Read Data Size (500) byte from File.</u>**
2. **<u>Calculate Checksum with crc16.</u>**