# *Assignment 6: Sliding Window ARQ with Congestion Control- **A modular approach - TCP Reno (fast retransmission and recovery)***

## *(Over UDP)*

**Suyash Damle (15CS10057)**
**Arunansh Kaushik (15CS30004)**

## ★ *Objective:*

The objective of this assignment is to implement a congestion control algorithm over the sliding window ARQ protocol. We are implementing on a UDP based socket. We will create library file ( **<protocol_stack>.h** file) for implementation of the entire protocol stack.

## ★ Compilation and Execution steps:

In this for connection between server and client so that multiple clients can be connected to a single server, also we are using TCP Sockets so bytes ordering is also done by itself.

**To compile and run TCP file transfer:**

- **Server**

  **>** g++ server.cpp -lpthread -o server.out

  **>** ./server.out **<***port where server will be running***> <***drop probability***>**

- **Client**

  **>** g++ client.cpp -lpthread -o client.out

  **>** ./client.out **<***hostname of server system***> <***port of server***>**

  **>** enter the file path that is to be sent

- *The file to be sent must be present in the same folder as the client, or the complete address must be specified with respect to the folder in which client.out is present.*
- *The server side uses the same file name as that given by the client to store the file. Hence, relative addressing, if used from the client side must be used with caution, as the server program would expect the same address to exist on the server side.*
- *The obtained file is run over an md5 hash-creating function using the **system()** function and the functionality provided by linux itself is used directly.*
- *The submitted zip includes the main protocol_stack : **protocol_stack.h** and 2 codes for the ftp sending and receiving and both use the header file.*

We checked the above implementation by generating random text file as below:

- **To generate a random text file of specified size:**
  **>** base64 /dev/urandom | head -c "*file_size_in_bytes*" **>** output_file.txt

**Eg**: base64 /dev/urandom | head -c 10000000 > file.txt
generates a file of size 10MB of random text.
(Such text files were used to test and debug the codes)

## ★ Functions and Threads created and their task:

Functions Created in library ***"protocol_stack.h"*** and their use:

❖ *void mysig(int sig) :* Used for setting alarm signal

❖ *(**Thread**) void * buffer_controller(void *data) :* Receiving packets and identifying whether they are DATA or ACK/md5 data packets and processing them accordingly like storing data of data of DATA packets in receiver buffer.

❖ *char * create_packet(int start_byte,int number_bytes,int finish_at=-1) :* Return string containing data from buffer from start_byte to finish_byte or according to number_bytes.

❖ *(**Thread**) void * congestion_control(void *data) :* Runs in an independent thread & send data packets according to sender window and data available also considering timeout and triple duplicate acknowledgement.

❖ *void update_window(int y,int z) :* Update sender window size & buffer pointers according to acknowledgements received.

❖ *int parse_packets(char buf[]) :* Parse the acknowledgement packets.

❖ *void appSend(int sender_socket_fd, char *data_to_send) :* Update sender buffer by data received from application to be sent.

❖ *int establish_connection(char *hostname, int portno) :* Record information about other end of connection and return sender_socket_fd.

❖ *int ftp_send_handshake(const char* file_loc,long int file_size) :* Exchange/Send file information to receiver.

❖ *int ftp_send_md5_checksum(char* file_loc) :* Calculate and receive md5 data from receiver end of file sent.

❖ *void sendACK(int byte_idx,int socket_fd) :* Send acknowledgement for byte number given in argument.

❖ *void appRecv(char *data,int socket_fd,int len):* Send date received in sender to application and update buffer pointers.

❖ *int ftp_receive_handshake(int sockfd, char*filename, long int *filesize) :* Exchange/Receive file information to sender.

❖ *int establish_server(int portno) :* Record information about other end of connection and return sockfd for use of receiver side.

❖ *int ftp_receive_md5_checksum(int sockfd,char * filename) :* Calculate and send md5 data to sender end of file received.

## ★ Observations / Other specifics of Implementation :

- Perhaps because of the code-strategy, or something related to the coding of the image files, the trial to transfer image files did not work. The client - side does not read the image file properly using the standard **getc()** function used, even when the file is opened in **binary mode** (“rb”) and the server-side keeps waiting for packets.

- In this assignment because we are sending and receiving data byte wise so we needed to a one bit record for every byte received.
- A total of **4 type of threads** run during the example of the ftp application - one is the congestion control thread- running on the sending side ( started by the appSend() function ), one is the buffer controller thread - running on *both* sides independently. The main thread on both sides handle the packet parsing, receiving, etc.