**Assignment 2** **To the Cloud and Back**

Due Date - October 20, 2020, 10PM CST

**Objectives**

1. Learn to create robust network protocols
2. Learn about reliable communication
3. How a premitive proxy might work. You can use this to get around sensorship or firewalls.

**Overview**

$ anonserver -p <PORT> -s <LOG FILE LOCATION> -p <web page to download>

1. PORT - The port server listens on.

2. Log file location - Where you will keep a record of actions.

3. p - Which webpage to download and serve.

For example:

$ anonserver -p 30000 -l /tmp/logfile -p www.nytimes.com

Functional requirements

1. The server must open a UDP socket on the specified port number
2. The server should gracefully process incorrect port number and exit with a non-zero error code
3. The server should send a FIN after done sending the packet
4. The server should download the file specified by -p and save it from the memory

***Client Specifications***

$ anonclient -s <SERVER-IP> -p <PORT> -l LOGFILE

The client takes three arguments:

1. Server IP - The IP address of the server.

2. PORT - The port the server listens on.

2. Log file location - Where you will keep a record of packets you received.

For example:

$ anonclient -s 192.168.2.1 -p 6543 -l LOGFILE

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Sequence Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Acknowledgment Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Not Used |A|S|F|

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Where:

1. Sequence Number (32 bits): The sequence number of the first data octet in this packet (except when SYN is present). If SYN is present the sequence number is the initial sequence number (12345).
2. Acknowledgement Number (32 bits): **If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.**
3. The acknowledgement number is given in the unit of bytes.
4. Not Used (29 bits): Must be zero.
5. A (ACK, 1 bit): Indicates that there the value of Acknowledgment Number field is valid
6. S (SYN, 1 bit): Synchronize sequence numbers
7. F (FIN, 1 bit): Finish, No more data from sender

Here is what a sample interaction looks like:

Server Client

| |

| seq=12345, ack=0, SYN |

| <---------------------------------- |

| seq=100, ack=12346, SYN, ACK |

| ----------------------------------> |

| seq=12346, ack=101, ACK |

| <---------------------------------- |####handshake complete, start getting data

|seq=102,ack=12347,ACK,512Byte payload |

| ----------------------------------> |

| seq=12347, ack=614, ACK |

| <---------------------------------- |

|seq=614,ack=12348,ACK,512Byte payload |

| ----------------------------------> |

| seq=12348, ack=1126, ACK |

| <---------------------------------- |

|seq=1126, ack=12349,ACK,512Byte paylod|

| ----------------------------------> |

**Client requirements**

1. The client must open a UDP socket and initiate 3-way handshake to the specified hostname/ip and port
2. Send UDP packet src-ip=DEFAULT, src-port=DEFAULT, dst-ip=HOSTNAME-OR-IP, dst-port=PORT with SYN flag set, Connection ID initialized to 0, Sequence Number set to 12345, and Acknowledgement Number set to 0
3. Expect response from server with SYN | ACK flags.
4. Send UDP packet with ACK flag, you should now start receiving data
5. The client should gracefully process incorrect hostname and port number and exit with a non-zero exit code
6. After file is successfully transferred, expect a FIN message from the server
7. On a FIN message, send an FIN|ACK, and gracefully shutdown the connection and write the content to an output file

**Server Requirements**

1. The server must open a UDP socket on the specified port number
2. The server should gracefully process incorrect port number and exit with a non-zero error code (you can assume that the folder is always correct). In addition to exit, the server must print out on standard error (std::cerr) an error message that starts with ERROR: string.
3. The server should be able to accept and process multiple connection from clients at the same time
4. Keep track of the last acknowldged byte - if data is lost(ack does not come back), retransmit after 0.5 seconds

**Additional requirements**

1. Code must compile/run on Google Cloud Ubuntu VM (18.04) - we will test your code only on the VM.
2. For each packet received, log both at server and receiver in the following format:

"RECV" <Sequence Number> <Acknowledgement Number> ["ACK"] ["SYN"] ["FIN"]

"SEND" <Sequence Number> <Acknowledgement Number> ["ACK"] ["SYN"] ["FIN"]

1. If packet is dropped and the server retransmits the data, log the following:

"RETRAN" <Sequence Number> <Acknowledgement Number> ["ACK"] ["SYN"] ["FIN"]