1 Introduction

So far, you have implemented a simple web server following HTTP over TCP in the programming assignment 1, i.e., the client requests a file and the server provides the requested file if exists, and a simple reliable Ping application over UDP in your programming assignment 2 given a certain percent of packet loss rate, i.e., one side needs to retransmit a ping request message if the Ping server has decided to drop it.

In this programming assignment, you are required to design your own **Reliable HTTP Server** over **UDP**, which means you need to implement a reliable HTTP server but over UDP socket. More specifically, the server and client need to talk to each other to establish a connection before the client starts to request a file. In addition, if the content of the requested files is larger than what a normal UDP packet can accommodate, multiple UDP packets have to be sent over the network. These packets may get lost, duplicated, reordered or corrupted along the way, and neither the sender nor the receiver has the capability to recover them. Further, the server should be able to indicate to the client the end of transmitting the content of the requested file, so the client does not need to wait any further for more packets.

Therefore, you need to add some **error control** capabilities, as well the **start** and **finish** indicators, to UDP, leading to a “new” protocol that we call CSC361 “**Reliable Datagram Protocol (RDP)**”. The goal of this assignment is to emulate HTTP-TCP but over UDP through links with a certain level of packet loss rate.

2 Reference Design

In order to help you finish the assignment on time successfully, we provide the following design for your reference. You are also encouraged to design your own schemes.

RDP Header

	Possbile value	Meaning
Packet type	GET	Indicate whether it is an HTTP request packet
	SYN	Synchronization packet
	ACK	Acknowledgment packet
	FIN	Finish packet
	DATA	Data packet
Sequence number		Byte sequence number
Acknowledgment number		Byte acknowledgment number
Payload		RDP payload length in bytes

SYN: Synchronization packet, sent by the client/server to establish a connection.

DATA: Data packet, sent by the server and carries the content of the requested file.

FIN: Finish packet, sent by the client/server to finish the data transfer.

ACK: Acknowledgment packet, sent by the client/server to acknowledge the reception of DATA, SYN or FIN packets.

GET: Get packet, sent by the client to the server after the establishment of connection to request a file.

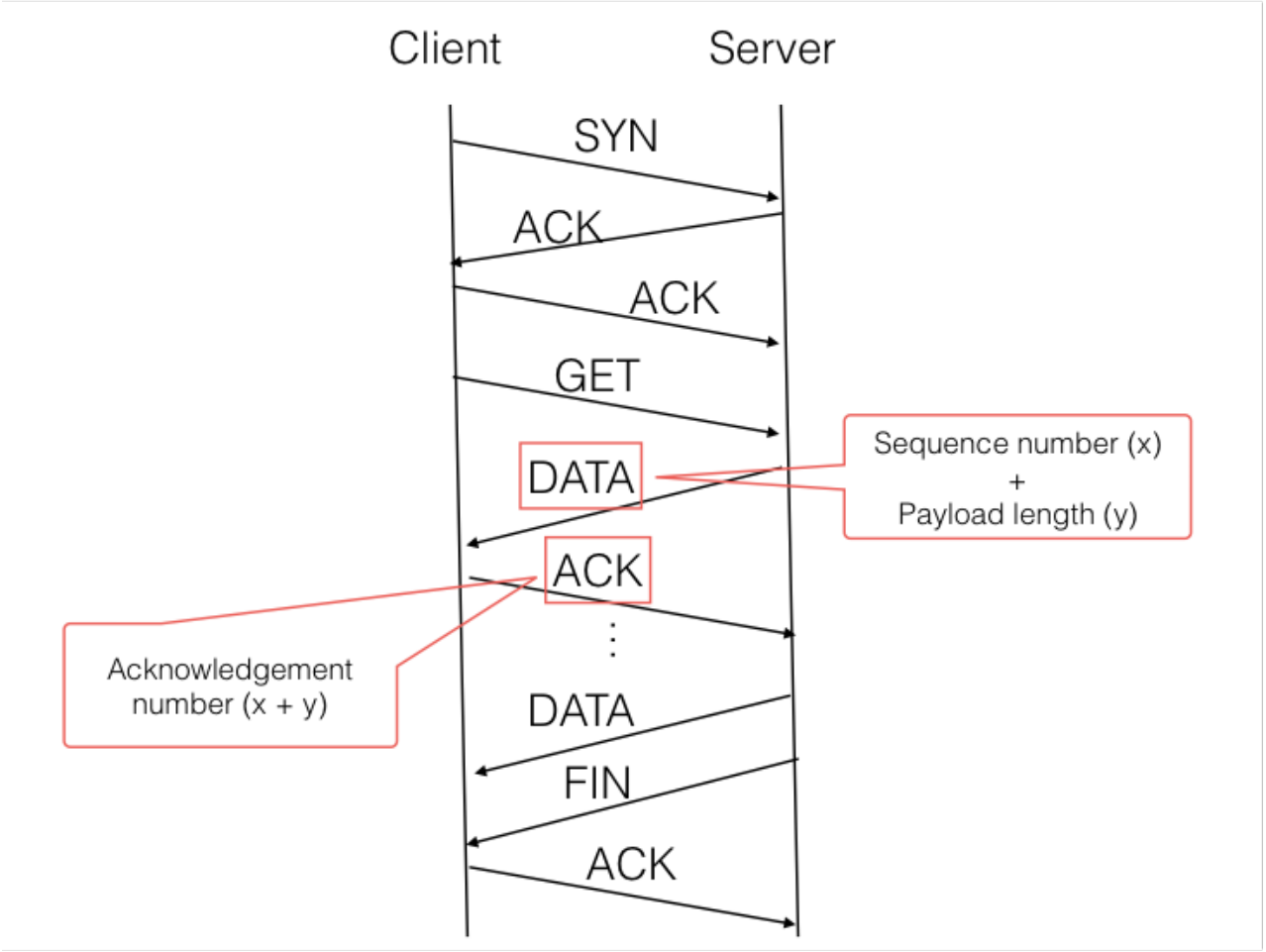
Sequence or Acknowledgment number: integer, byte sequence or acknowledgment number (i.e., x+1 represents the byte immediately after the byte represented by x in the data stream). Although the initial sequence number is chosen by the client/server randomly, in this assignment, you can set to 0 for simplicity.

Payload: integer, byte payload length (i.e., x represents x data bytes). For packets such as DATA, it indicates the length of the **data** payload in the packet.

Note that you may be able to define more types of packets and add more fields in your extended design.

Interaction Example

A very simple way is to use the stop-and-wait strategy to track the delivery of each datagram (UDP packet).



Socket timeout

Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client or server cannot wait indefinitely for a reply. A simple implementation to tackle this issue is to get the client/server wait up to one second for a reply; if no reply is received within one second, your client/server program should assume that the packet was lost during transmission across the network. You will need to look up the Python documentation to find out how to set the timeout value on a datagram socket.

3 Run the code

Network Topology and Packet Loss Rate

Open a terminal in Mininet and type the following command

```
sudo mn --link tc,loss=5 -x
```

Then each link is setting 5% packet loss. More details can be found in Mininet Walkthrough.

RDP Server

Pick host h1 (or h2), and run

```
python RDP_Server.py <IP addr of RDP_Server> <Port of RDP_Server>
```

The buffer size at the server side need to be set to 1,024 bytes, i.e., the maximal RDP packet size (including RDP packet header and data payload) is 1,024 bytes.

RDP Client

Pick host h2 (or h1), and run

```
python RDP_Client.py <IP addr of RDP_Server> <Port of RDP_Server> <Reques
ted file name> <Received file name>
```

Note that the argument `<Requested file name>` is the name of the file you would like to request; the argument `<Received file name>` is the name of the file you create to store the received content from the server. However, this argument is optional as long you know the file name you have created to store the content sent by the server.

The buffer size at the client side can be set larger than that of the server, e.g., 2,048 bytes.

Requested file == Received file?

The received file should be identical to the requested file existing at the Server folder in content. For a quick check, you can use md5sum to know that two files of the same size are actually different. If your received file is identical to the requested file, then your job is done!

4 Marking Scheme

Basic part

1. Documentation (1 point)
2. Without setting any packet loss in the link, code works perfectly (5 points)
3. With setting 5% packet loss rate in each link, code works perfectly when requesting a small

size file (2)

4. With setting 5% packet loss rate in each link, code works perfectly when requesting a file with a large size file (2)

Bonus part

In this programming assignment, you may get an extra 2 points if you implement one of the following additional features such as

1. Flow control: A fast server can send packets much faster than how much the client buffer can handle and may potentially overflow the client's buffer and cause packet loss, even if the packets have arrived at the receiver. Note that the buffer size at the client side should be set smaller than that at the server side. Hint: add more fields in the RDP header such as **Window size**.

2. Error control mechanisms other than stop-and-wait strategy: For performance concerns, using the stop-and-wait strategy is not efficient. The design that the server can send multiple datagrams and the client uses accumulative acknowledgement is more preferred.

5 What to Hand in

Server/client code and documentation (in pdf format) about how to implement your RDP. If you have additional features implemented, you also need to document it. All these need to be submitted in Connex.