HW4

HW4: `ciTCP`

Goal

The goal of this homework is to design, implement, and test a reliable transport over an unreliable network (actually a reliable transport on top of a unreliable transport, UDP).

The packets should be delivered and in order to the application even though some packets may be lost in the network.

CS Edition

Develop a ciTCP client/server that serves a single file to its clients. The client should reliably receive the file and present it completely and in order. Note that our server and client are acting as both an application and a transport layer library.

```
# Server (sender) side:
$ python3 citcp-server.py PORT-NUMBER FILE

# Client (receiver) side:
$ python3 citcp-client.py IP-OR-HOSTNAME PORT-NUMBER FILE-NAME
```

Development Roadmap

Target Date	Points	#	Milestone	Submission
Nov 4	10	1	High level design of client and server, must be detailed and approved to continue.	Commit tag
Nov 13	20	2	Connection establishment and teardown.	Commit tag
Nov 20	20	3	Small data transmission, assume no packet loss. Acknowledge packets received.	Commit tag

Target Date	Points	#	Milestone	Submission
Nov 27	30	4	Large data transmission, divide file into multiple packets and transmit based on congestion window. Assume no packet loss.	Commit tag
Dec 4	20	5	Assume packet loss, add timers, retransmissions, duplicate ACK handling.	Pull request

Principles of Reliable Transport

Our implementation must have the following principles of reliable transport:

- Connection establishment
- Connection tear down
- Acknowledgement and sequence numbers
- Timeouts
- TCP Tahoe congestion control (slow start, congestion avoidance, and fast recovery)
- Receive and sending windows

High Level Design

Draw/describe in great detail how you plan to implement the client and server.

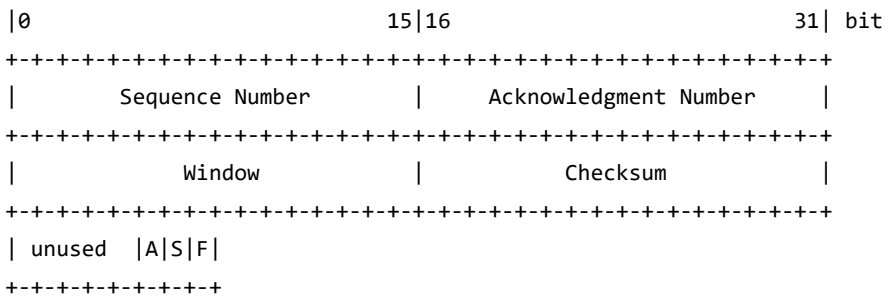
Include: necessary state machine diagrams with explanations, protocol communication diagrams with examples, data structure for the information needed, etc.

Some things to consider:

- How is the initial sequence number chosen?
- Does the timeout value have to change?
- How are the segment sizes decided?

`ciTCP` Header

Use the following header:



- Sequence Number: 16 bits (2 bytes)
- Acknowledgement Number: 16 bits (2 bytes)
- Window: Size of the window in bytes. 16 bits (2 bytes)
- Checksum: 16 bits (2 bytes)
- Flags: Last 8 bits (5 unused)
  - Unused bits (must be 0!)
  - A (`ACK` nowledge) - If set, acknowledgment number present.
  - S (`SYN` chronise) - If set, sequence number present.
  - F (`FIN` ish) - Used in connection termination.

Client / Serer

Server

The server should behave as a RDT sender and reliably transport a file to the client. After the file has been transferred, the connection should be terminated.

You are welcome to write the server as a single threaded server. Meaning, you don't have to handle multiple clients connecting at once.

Client

The client should behave as a RDT receiver and reliably write (and in order) the file transmitted from the server.

Connection Establishment/Teardown

Connection Establishment

Use a three-way handshake to establish the server's initial sequence number. The receiver does not send data (only control messages), so a sequence number from the receiver should not be sent.

Connection Teardown

Implement the initiator/responder teardown sequence from the TCP connection state diagram. Ignore the simultaneous close sequence. Your single threaded server should then be ready for another

connection, make sure you cleanup any data structures that might be reused with the next connection.

Windowed Congestion Control

Coming eventually.

Testing Unreliability

Coming eventually.

Submission

- Work must be submitted through your bitbucket account.
- Tag your commit in bitbucket as milestoneX where X is the milestone number.

Grading

Each milestone will be graded after the Target Date.
If you fall behind: try this.