PORTLAND STATE UNIVERSITY
CS494/594: INTERNETWORKING PROTOCOLS

PROGRAMMING PROJECT PART 1
DUE: February 12rd, 11:00 AM

1. Development Setup

For this programming assignment you will work in the CS Linux Lab (linuxlab.cs.pdx.edu) located in FAB 88-09 and 88-10. If you don't already have an account, go to http://www.cat.pdx.edu/students.html for instructions.

For this Project you must **use C/C++, UDP Sockets, POSIX Threads, Timers and Make**. Please refrain from any other languages or 3rd party libraries. We will use the GNU C/C++ Compiler (gcc/g++) and Make already installed in the development machines in the lab. Also please note **you are not allowed to use SO_RCVTIMEO for this project.**

Grading will be done in this setup so please make sure that your code works under this conditions.

2. Problem Description

As part of this class we will design and implement a simple File Transfer Protocol similar to RCP but much simpler. We will implement our protocol using UDP sockets and datagrams. **Our focus will be on implementing a reliable channel, flow control and retransmission** as part of our Simplified Remote Copy Protocol, so you should use TCP/IP to model your design.

You will implement both the Client and the Server applications for our Simplified Remote Copy Protocol. As we are using UDP to communicate between the Client and the Server applications, we will need to design a protocol that allows the user to: 1. Establish a connection between client and server, 2. Specify the absolute path location within the server filesystem where the desired file is located, 3. Transfer the file to the client and 4. Terminate the connection.

Your server must use the command line parameters to allow the user specify the port to listen for connections from other clients. This is an example of the command a user might use to start the server listening on port 1080:

```
./cs494rcp_server 1080
```

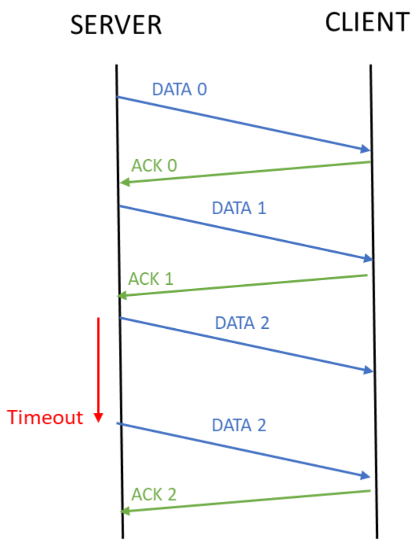
Your client program must use the command line parameters to allow the user to specify the IP address and port of the server to connect. It should also allow the user to specify the file path to the requested file. This is an example of the command a user might use to download a file located in the server computer at 120.45.67.126 and listening on port 1080:

```
./cs494rcp_client 120.45.67.126 1080 /Docs/Files/myfile.txt
```

The client should save a copy of the file, however to avoid name collisions when the server and client are run on the same computer, you must save the sent file with a different name. To do so the client must append the substring ".out" at the end of the filename. For the example above the resulting filename should be "myfile.txt.out".

3. Implementation Overview

For the First Part of the Programming Project we will implement the Stop-and-Wait ARQ flow control protocol. The Stop-and-Wait protocol sends on packet of data at a time. After a data packet is sent to the client, the server waits until an acknowledgment packet (ACK) is received by the server. If an ACK is not received by the server before a timeout in milliseconds expires, the server assumes that the packet was lost and resends the data packet. Only after the ACK has been received the server can continue sending the next packet. **Please note you are not allowed to use SO_RCVTIMEO for this project!** Using SO_RCVTIMEO will make the second part of the project much more complicated



Stop-Wait ARQ

These are a few key concepts you must address as part of your implementation:

1) You must implement a protocol that establishes handshake between the client and the server. We will implement a **three-way handshake** similar to TCP's handshake. As part of the handshake, the client must request a file to be sent by the server. To do this your protocol must implement a file request packet. A file request packet must specify the exact path of the file that the client wants to retrieve (Conceptually, this will be similar to a SYN packet in TCP). The server must acknowledge the request and notify the file size to the client (Similar to a SYN+ACK packet in TCP). If the file does not exist in the server the client must be notified and the connection terminated. Finally the client must acknowledge the reception of the file size and signal the server to start sending the file. (This is similar to the final ACK of the three-way handshake in TCP)
2) After the request is acknowledged, the server can start sending the data to the client, by sending only one packet at a time. **Before sending the next packet the server must wait for an acknowledgement from the client**. This method is not efficient but it will be the basis for more

efficient flow control mechanisms used in Part 2. It is recommended you keep your packets small, up to 1Kb maximum length, otherwise they can be dropped by intermediate routers.

3) Your server must wait for a timer to expire in case the packet is lost. **If no ACK has been received before the timer expires it means the packet was lost in transit and must be resent**.
4) After the file transfer completes your server must **notify the client using a Close packet before closing the connection**. The server must wait for the client to respond and only then close the connection. The client can terminate the connection as soon as he acknowledges the closing of the connection.

4. Packet Structure

As part of your project you will need to design and implement various packet types to handle the required requests and responses of your protocol. You are allowed to change or design your own packet as you see fit. However, we provide a possible design suitable for this project. Please note all the recommended sizes are specified in bytes

- Connection Packet (SYN)

Type[1]=S	PktLen[2]	Filename[PktLen]

- Connection Reply Packet (SYN+ACK)

Type[1]=R	FileSize[8]

- Connection Reply ACK Packet

Type[1]=W

- DATA Packet

Type[1]=D	SeqNum[4]	PktLen[2]	Data[PktLen]

- Data ACK Packet

Type[1]=A	SeqNum[4]

- Close Connection and Close Connection ACK Packet

Type[1]=C	SeqNum[4]

5. Testing Script

To aid in grading and testing your project, you must prepare a BASH script (test.sh) that starts the server on port 1080, then starts the client on port 1080 and send the file “test.jpg” provided as part of the project package. The script then must compare the input and output files for correctness. A good way to do this is by using md5sum.

The project package provides a sample BASH script you can use to write your own script.

6. Hand-In

For submission, you should provide only source code (*.c, *.cpp, *.h), a Makefile script that compiles both the Client and the Server code and a test script test.sh.

Please pack your files into a TAR file before submitting your solution. Do not include test.jpg or this PDF. We will use the D2L system for submission (https://d2l.pdx.edu/). Please remember that there is no late policy.