18-441/741: Computer Networks
Fall Semester, 2023

Project 3: Transport Layer

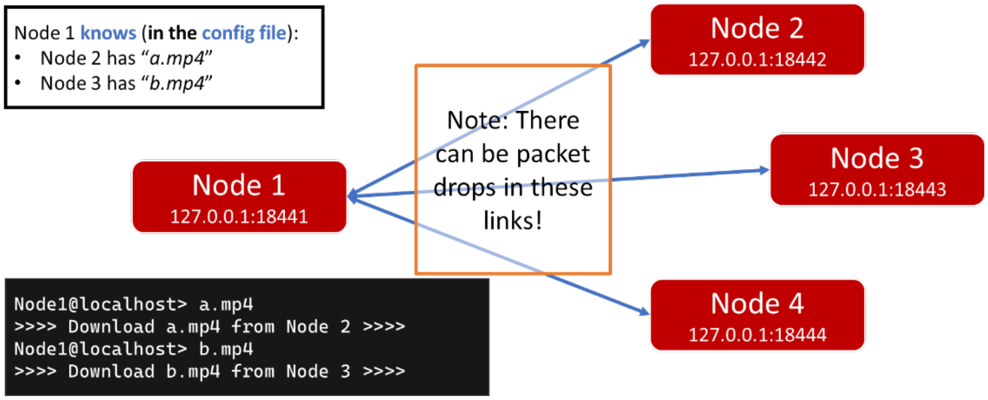
Deadline: 11:59pm ET, Nov 19th, 2023

Questions? Ask on Piazza

1. Introduction

In this project, we will implement a custom transport layer to transfer different types of files between multiple peer nodes. The main objective of this project would be to ensure complete and correct reception of multiple files across nodes in the absence and presence of a packet loss.

The transport protocol will be built on top of UDP. You should understand the concept of transport layer to successfully implement it. You should also be able to apply techniques from datalink layer and ARQ to the transport protocol as well.



Node 1 **knows** (**in the config file**):
- Node 2 has "*a.mp4*"
- Node 3 has "*b.mp4*"

```
Node1@localhost> a.mp4
>>>> Download a.mp4 from Node 2 >>>>
Node1@localhost> b.mp4
>>>> Download b.mp4 from Node 3 >>>>
```

Figure1. network configuration example

2. Project Specification

2.1 Objectives

Implement a reliable transport layer based on UDP, which can transfer content files between multiple peers.

2.2 Node Configuration Files

Your server should take an extra argument, which designates the configuration file for a particular node. For example, the following command should start a server which has a configuration file named node1.json. Note that there is no ‘-c’ anymore.

```
$ python3 tcpserver.py node1.json
```

We present an example configuration JSON file as follows.

```
{
  "hostname": "ece001.ece.local.cmu.edu",
  "port": 18741,
  "peers": 1,
  "content_info": ["abc.mp4", "def.mp4"],
  "peer_info": [
    {
      "hostname": "ece002.ece.local.cmu.edu",
      "port": 18742,
      "content_num": 2,
      "content_info": ["video.ogg", "audio.ogv"]
    }
  ]
}
```

In the above configuration file, different fields provide information about the server itself and its peers. For example, in the above file, the port for the “ece001.ece.local.cmu.edu” to use is 18741 with 2 content files whose names are given by ‘ –abc.mp4’ and ‘def.mp4’. The total number of peers to this node is 1 whose details are also given in the peer_info field. You can use standard JSON reading libraries in Python to read these configuration files.

2.3 File Transfer

For project3, there is only one command you need to implement.

Keyboard Input: FILE_NAME<newline>

FILE_NAME is the name of file that you need to download from near peers. We do not input FILE_NAME which is not present in near peers. The downloaded file should locate the same path as your python program.

We simplified project 3 for you to focus on the file transfer implementation. In project 3, you do not need to worry about addneighbor, map, rank or any other commands (e.g., broadcasting /advertising a list of contents from one node to **all** other nodes in the network). When a node get input of FILE_NAME , the node need to search for FILE_NAME in **only adjacent neighbor** nodes and we give you information about neighbor information in the configuration file. For example, in Figure1, node1 can download file from node2 or node6. But node2 can not download file from node3 or node4. Thus, when your program receive FILE_NAME command, you first find which node is having that file in the configuration and then initiate data request and download the file. Thus, project 3 is not an extension of project 2. We recommend you to start from the scratch.

2.4 Transport Layer Protocol

In this project, you will implement a custom transport protocol on top of UDP. The purpose of the transport protocol is to reliably transfer content amongst peer nodes. You cannot use TCP sockets to finish this project. Checks are implemented in Gradescope to ensure that TCP is not used to transfer files. If found, you will get a 0 in the project. Also, make sure your protocol can achieve the following requirements:

- **Multiplex:** You are allowed to use only one port for the exchange of files (include sending & receiving). All traffic (even simultaneous transfers) should go through this port.
- **Robustness:** Can reliably transfer the content, that is it can endure at least 5% packet loss
- **Simultaneous transfer:** Can simultaneously transfer multiple (at least five) content requests at the same time.

To download or transfer a file from a peer server node to the client node, again, we would enter the required FILE_NAME as an input in the client node and your client should be able to receive that file from that peer node.

A sample file transfer 'video.ogg' is shown below when there is a random packet loss. As we can see, this ARQ type protocol allows the client to receive full file even in the presence of a packet loss through retransmissions. This is just an example protocol; you are free to develop your own protocol which is even better than this.

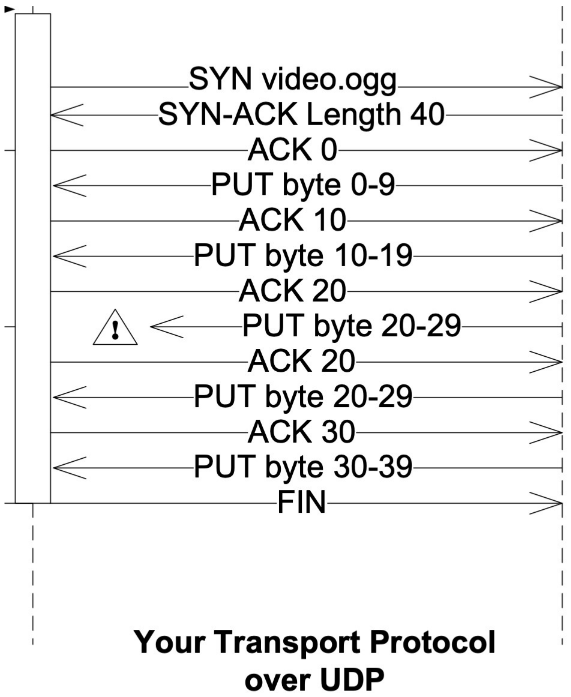
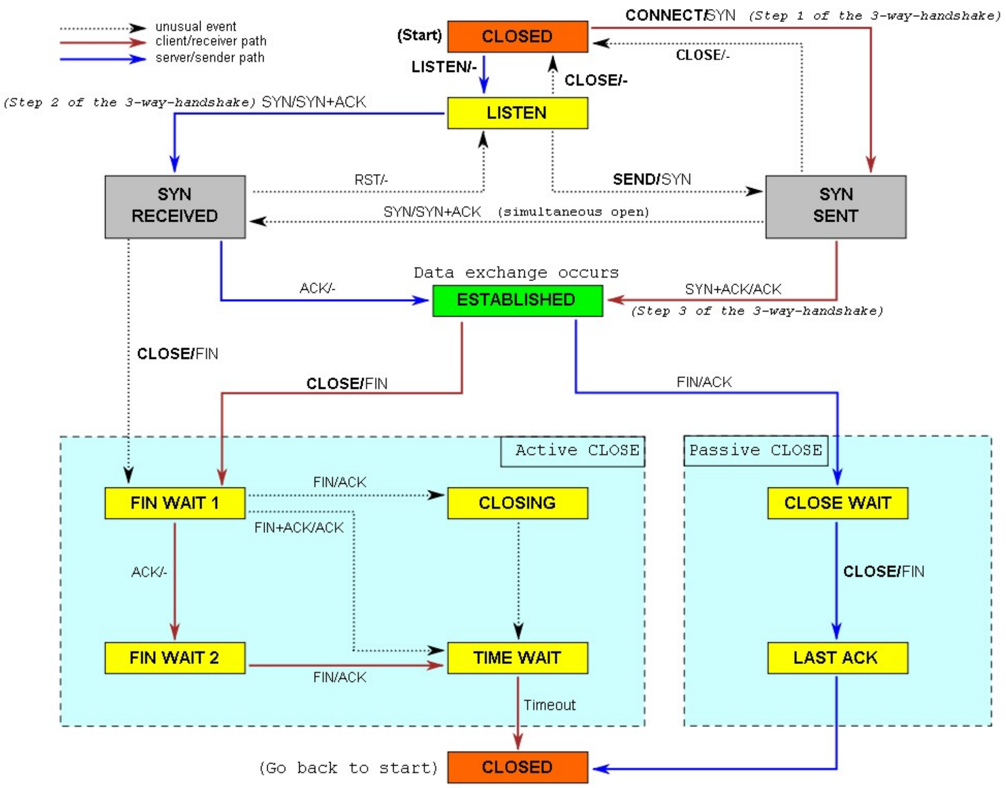
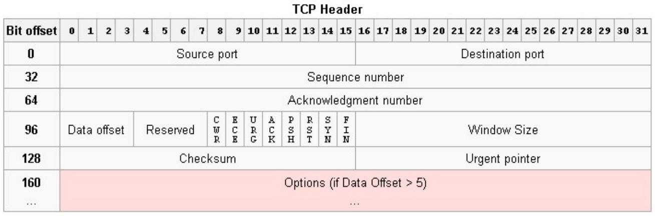


Figure2. example of reliable transfer protocol

To handle simultaneous connection among peers for the server by using one UDP port, you should think about how to manage this connection information. Here we attach an example of TCP (Figure3) for your reference. TCP use three way handshake and there are communication steps to establish connection, and associated states are maintained (e.g., LISTEN, SYN RECEIVED, SYN SENT, ESTABLISHED). You can refer to this when you design your own protocol and the proper packet format. Again, be advised that you should be able to utilize the same port for all traffic.



State communication diagram of a Node



TCP Header

Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Source port																Destination port															
32	Sequence number																															
64	Acknowledgment number																															
96	Data offset				Reserved				CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
128	Checksum																Urgent pointer															
160 ...	Options (if Data Offset > 5) ...																															

An example packet format

Figure3. TCP state diagram and packet header format

3. Testing

We provide some testing scripts that runs on Linux shells for you to test your code locally. The testing procedure is stated as follows. **Please read carefully through here to make sure that you are not doing anything wrong.**

Transmission Without Packet Loss

1. Prepare a UNIX-based environment. For MacOS and Linux users, you can simply start testing. For Windows users, it is recommended to use WSL2 (Please check Install WSL | Microsoft Learn). You can also test on ECE cluster machines where they use Ubuntu OS.

2. Download the testing scripts from CANVAS, and unzip it.

3. Go to the folder that contains the testing scripts. If you "ls", you will see "README.md". Follow the instructions inside.

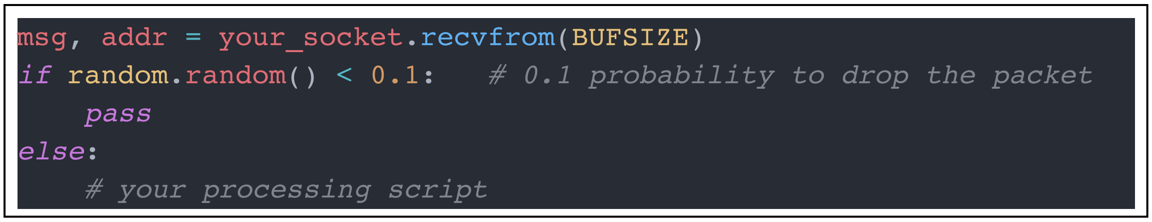
4. Hopefully you will be able to test locally for the three cases that are present on Gradescope.

Transmission With Packet Loss

Gradescope is not able to realize udp packet loss. Therefore, it only includes the 70 points that will be graded. The rest will be graded manually by running your code with a Linux packet drop script, where the packet drop rate will be set to be 5%. However, it would still be helpful to test your code locally with packet loss. It is suggested to achieve a robustness for 10% packet loss to ensure you're safe with the test. Below is the suggested procedure for you to test reliable transmission locally.

- Import `random` from python library.

- In your `tcpserver.py`, right after **each** `your_socket.recvfrom(BUFSIZE)`, add an "if" argument:



```
msg, addr = your_socket.recvfrom(BUFSIZE)
if random.random() < 0.1:    # 0.1 probability to drop the packet
    pass
else:
    # your processing script
```

- **Remember to remove all these codes before submission!**

4. Deliverable Items

The project is due at 11:59pm ET, Nov 20th, 2022.

4.1 Deliverable Items

The deliverable items are enumerated as follows.

1. A python script of a server node which could be run on multiple machines using different configuration JSON files. On each server node, the server can be started with the command **$ python3 tcpserver.py node1.json**

   Please ensure that the Python script you submit is named ‘tcpserver.py’ to ensure correct evaluation on Gradescope.

2. A brief design document regarding your server design in ‘design.pdf’ in the submission directory (2-3 pages). Tell us about the following,
   a. A summary of your overall system
   b. Final protocol design
      i. Packet Format ii. State
      transition diagram

5. Grading

Partial credits will be available based on the following grading scheme:

Note: Usage of TCP sockets to complete this project will lead to a score of 0. Appropriate checks have been ensured at Gradescope

Items	Points (100)
Logistics - Successful submission - Design documents	**20** 10 10
Transport System - Small file transfer	**80** 10
- Large file transfer - Simultaneous transfer on one port (at least 5 simultaneous transfers) - Robustness (5% packet loss) - Small file transfer - Medium file transfer	20 20 30 - 15 - 15

**Note that gradescope does not have test cases for Robustness (there were some technical issue simulating packet drops in gradescope). Thus, total score on gradescope is 70. For the remaining 30 points for robustness cases, we will manually grade them after the deadline.

Note that your local test script has robustness test cases, you can test it.