Assignment 4: TCP-like UDP

COL 334/672, Diwali'24

October 16, 2024
Deadline: October 30, 2024

Goal: In this assignment, you will implement a simple file transfer protocol using UDP sockets. Since UDP (User Datagram Protocol) is unreliable and does not provide congestion control, you will design and implement mechanisms to ensure reliability and congestion control at the application layer.

Overview: You will be tasked with creating a client-server application where the client downloads a pre-specified file from the server using UDP. The client and server should implement:

- Reliability: Ensure that all packets are delivered to the client in the correct order and without loss.
- Congestion Control: Implement a congestion control algorithm to prevent overwhelming the network. You will implement TCP Reno and CUBIC (if you attempt the bonus part).

General Instructions

- Any instance of cheating will receive strict penalty.
- You are allowed to do this assignment in a pair.
- You will need Mininet and Ryu controller for the analysis part in this section. You can re-use the installation that you used in Assignment 3.
- You should submit a single zipped folder containing all the code as well as the report. Name it `<entry_no_1>_<entry_no_2>.zip`, i.e., the names of your and your partner's entry number. *Please note: Only one submission per group is required.*
- **Piazza protocols**: We will follow the following Piazza protocols like last time:
  - **Piazza protocol 0**: You are encouraged to ask questions on Piazza.
  - **Piazza protocol 1**: Each *new* question should be asked in a separate thread with a **clear** subject line. Using a descriptive subject line is helpful to everyone.
  - **Piazza protocol 2**: *Please ask questions in advance.* Not all questions that are asked within 2 days of the deadline may be answered. No questions will be answered after the deadline.

1 Part 1: Reliability (40%)

Since UDP does not guarantee reliability, you will implement the following mechanisms to ensure reliable data transmission:

- **Acknowledgments (ACKs)**: The client uses ACKs to intimate the server about the successful delivery of a message. You should implement cumulative ACKs. Feel free to use delayed ACKs if you find that improves the efficiency.
- **Retransmissions**: If the server does not receive an ACK within a specified timeout period or detects 3 duplicate ACKs, it should retransmit the packet.

- **Fast Recovery**: The server should implement a fast recovery mechanism that initiates retransmission upon receiving 3 duplicate ACKs.
- **Packet Numbering**: Each packet sent by the server should include a sequence number. The client will use these sequence numbers to reconstruct the file in the correct order.
- **Timeout Mechanism**: The server should implement a timeout mechanism to trigger retransmissions of unacknowledged packets. The timeout value should be estimated using the method discussed in class. If you are using another method, please mention that in your report.

You can implement reliability only for server-client communication. For the initial client-server communication, you can simply keep re-trying until the request for file download is successfully delivered. You can assume the server is handling only one client.

1.1 Packet Format

While TCP uses a 32-bit integer to represent packet sequence number, in this assignment, we will assume that sequence numbers can be arbitrarily large. For us, the first byte will start with a sequence number of zero and the rest increment from there. Thus, a data packet contain the following fields: sequence number of the first byte, number of data bytes, the data. Moreover, we will serialize information using json to keep debugging and serializing/deserializing simple. You are free to use your own identifier for these fields.

Acknowledgement packets simply contain the next expected sequence number, indicating the sender that all bytes before this sequence number have been received.

1.2 Analysis

You should empirically measure the performance improvements due to fast re-transmissions by conducting the following analysis:

- **Setup**: Run a simple two-node topology in Mininet, provided along with this assignment. It consists of two hosts (h1 and h2) connected via a switch (s1). Use a Ryu controller application running a simple learning switch to set up the forwarding tables on the switch. Use your program that implements reliability but no congestion control to transfer a file between the client and server. Use a a fixed rate of 50 Mbps to send the packets.
- **Experiments**: Conduct two sets of experiments: one measuring the improvement due to fast recovery under varying loss conditions, and another measuring the improvement with varying delay. For the loss experiments, set a fixed delay of 20 ms on the h1-s1. link and vary the loss rate from 0% to 5%, in increments of 0.5%. For the delay experiments, set a fixed packet loss rate of 1% on the h1-s1 link and vary the delay from 0 to 200 ms, in increments of 20 ms. In both cases, set the loss and delay for the h2-s2 link to 0. **Data Collection**: For each experiment, log the time taken to download the file. Repeat each experiment 5 times to account for noise.
- **Plotting Results**: For the loss experiments, create a line plot comparing the file transmission time with and without fast recovery as the loss varies. Similarly, generate a plot for the delay experiments, comparing the transmission times as delay changes. Explain the observed trends in the results in the report.

1.3 What to submit

Submit the client file and server file, naming them `p1_client.py` and `p1_server.py`. We should be able to run the code as follows:

```
Running server: python3 p1_server.py <SERVER_IP> <SERVER_PORT> <FAST_RECOVERY_BOOL>
Running client: python3 p1_client.py <SERVER_IP> <SERVER_PORT>
```

Here, `SERVER_IP` and `SERVER_PORT` represent the IP address and port that the server is listening on. The `FAST_RECOVERY_BOOL` indicates whether fast recovery is enabled, with a value of 1 meaning it is on and 0 meaning it is off.

In addition, add the plot and your explanation in the report.

2 Part 2: Congestion Control (60%)

You will implement a congestion control algorithm based on a sliding window approach, i.e., the client should send multiple packets without waiting for an ACK for each one. However, the number of packets (window size) should be controlled to avoid overwhelming the network. You should implement TCP Reno-like congestion control. In particular, you should implement the following mechanisms:

- **Slow-start**: Exponentially increase the congestion window (cwnd) size (double every RTT) until it reaches a threshold or detects packet loss.
- **Congestion avoidance**: When cwnd exceeds the slow start threshold, TCP Reno increases the window size linearly to avoid congestion.
- **Fast recovery**: After detecting 3 duplicate ACKs, halve the cwnd and perform linear growth (1 MSS every RTT) to quickly recover from minor congestion.
- **Timeout behavior**: If a timeout occurs, reset cwnd to 1 and initiate slow start, assuming severe congestion.

Please refer to TCP Reno for any missing details. For instance, you should dynamically update the slow start threshold as done in TCP Reno.

2.1 Implementation details

- You should implement the congestion control mechanisms on top of the code that implements reliability in Part 1.
- Assume the same format of the packet as specified in Part 1.
- You should start with an initial window of 1 MSS. Assume an MSS is 1400 bytes.
- Use an additive increase factor of 1 MSS and multiplicate decrease factor of 0.5.
- No need to implement flow control.

2.2 Analysis

1. Efficiency: In this set of experiments, you should measure the impact of varying delay and packet loss on TCP throughput. Use a similar 2-node topology as in Part 1. In the first set of experiments, you should vary the link delay as done in Part 1 and log the average throughput. You should plot the average throughput as a function of link delay. Repeat the experiments with loss using similar setup as described in Part 1. Plot the average throughput as a function of packet loss. Related literature suggests that throughput $\propto \frac{1}{RTT \times \sqrt{p}}$, where p is the packet loss rate. Do you observe similar results? Explain your observation in both cases.

2. Fairness: In this experiment, we will evaluate the fairness of the developed congestion control algorithm. Particularly, we will conduct experiment with a dumbbell topology consisting of two pairs of clients and servers, sharing a bottleneck link as shown in Figure 1. You should vary the link delay of Switch2-Server2 link from 0 ms to 50 ms at 5 ms intervals. Run the client and server program on both client-server pairs and log the observed throughput for both. Plot the Jain's fairness index as a function of link latency. Explain your observations.

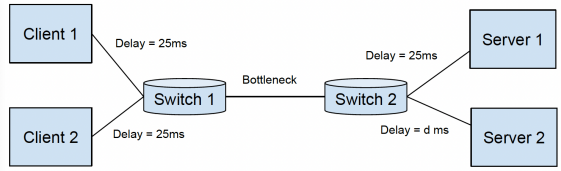

Figure 1: Dumbbell topology for Experiment 2 in Part 2

2.3 What to submit

Submit the client and server file, naming them `p2_client.py` and `p2_server.py`. We should be able to run the code as follows:

```
Running server: python3 p2_server.py <SERVER_IP> <SERVER_PORT>
Running client: python3 p2_client.py <SERVER_IP> <SERVER_PORT>
```

In addition, add the plot and your explanation in the report.

3 Part 3: Bonus (20%)

Implement TCP CUBIC congestion control algorithm as discussed in class. More specifically, use the following cubic function for window increase:

$$W(t) = C(t - K)^3 + W_{max} \tag{1}$$

Here, $K = \sqrt[3]{\frac{W_{max}\beta}{C}}$. Use $\beta = 0.5$ and $C = 0.4$

3.1 Analysis

1. Efficiency: Repeat the efficiency experiments as with TCP Reno. Compare the throughput of TCP CUBIC with what was observed for TCP Reno and explain your observations.

2. Fairness: Use a dumbbell topology with two client-server pairs. One of the server (S1) runs TCP Reno CCA while the other server (S2) runs TCP CUBIC CCA. Start both the instances of client-server pairs. Log the throughput observed over time for each CCA and plot it in a graph with y-axis representing the CCA throughput and x-axis representing the time. You should run this experiment with two set of delay parameters: 1). short delay path, use a delay value of 2 ms for each link, and 2). long delay path, use a delay value of 25 ms for each link. Compare the fairness observed in both set of experiments and explain your observations.