roject2: TCP Packet-Level Simulator

M2608.001200 Introduction to Data Communication Networks (2022 Fall)

Instructors: Prof. Kyunghan Lee
TAs: Gibum Park, Jongseok Park, Sanghyun Han
Due date: Nov. 30, 2022, 11:59PM

Object

In this project, we will design our own congestion control with various parameters in a given network environment.

Introduction

The goal of this project is to **1) implement a TCP packet-level simulator by Python**, **2) observe its performance in a given network environment**, and **3) design congestion control for improving the performance**. Implementing a generalized simulator will be challenging, but what you will do in the project is to implement a simulator for simple network topology which consists of a sender, a base station and a receiver. You will then be able to observe the performance of naïve congestion control and design congestion control which is better than the naïve.

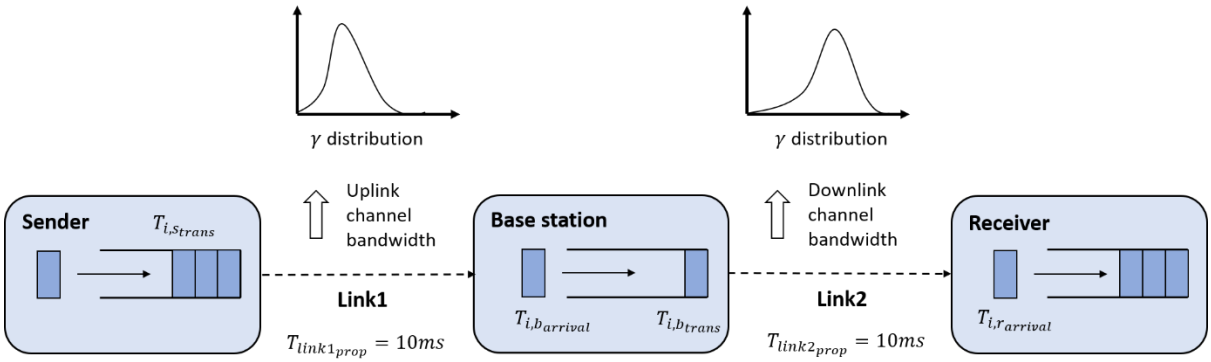


Figure 1: A simple network topology.

Project Specification

1) Network topology

Figure 1 shows a simple network topology that you need to implement. There are a sender, a base station, and a receiver. The downlink link1 and the link2 are wireless channels. We assume that packet loss does not occur in the wireless channel. At first, the sender creates packet list and pushes the list into the sender queue. The size of a packet is 1450 bytes. Then, the sender sends packets to the base station with the link1. $T_{i,s_{trans}}$ is the transmission time of the ith packet from the sender (i.e., the time when the whole packet is transmitted). $T_{link1_{prop}}$ and $T_{link2_{prop}}$ is the propagation time for the link1 and the link2. We assume that both of them are 10ms. After that, the base station receives the packets and enqueue them into its queue. $T_{i,b_{arrival}}$ is the arrival time of the ith packet at the base station (i.e., $T_{i,b_{arrival}} = T_{i,s_{trans}} + T_{link1_{prop}}$). Then, the base station sends packets with the link2. $T_{i,b_{trans}}$ is the transmission time of the ith packet

from the base station. Finally, the receiver receives packets. $T_{i,r_{arrival}}$ is the arrival time at the receiver.

2) Link bandwidth distribution

We assume that the channel bandwidth of the link1 and the link2 follow *γ (gamma) distribution* (https://en.wikipedia.org/wiki/Gamma_distribution). The distribution varies depending on the parameter α and β as in Figure 2. You can generate the distribution in Python by using *scipy* function (*scipy.stats.gamma* function in Python). The link1 and the link2 follow different distributions. Given the distributions of the links, the base station can experience congestion or not. We assume that ACK of the ith packet arrives at the sender after 20ms from the time of arrival packet at the receiver (i.e., $T_{i,r_{arrival}}$ + 20ms) and assume that ACKs are not lost. We assume that a time step is 1ms and the channel bandwidth varies every time step given the distribution. (i.e., $bandwidth_{t,link1} = x\text{Mbps}$)

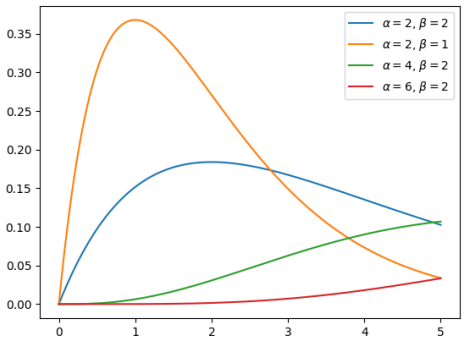


Figure 2: *γ* distribution with various parameter settings.

3) Assumptions

This project has following assumptions, and those values are provided **by default in program.**

- A size of packet: **1450bytes**; # of packets: **2000**
- A time step: **1ms**; simulation time: **5s (5000ms)**
- The uplink channel bandwidth at time step t: ~**Mbps (*γ* distribution with α = 3, β = 1**)
- The downlink channel bandwidth at time step t: ~**Mbps (*γ* distribution with α = 3.5, β = 2**)
- A packet loss does not occur in the link1 and the link2.
- $T_{link1_{prop}}$ and $T_{link2_{prop}}$: **10ms**
- The arrival time at sender of ith packet's ACK: $\boldsymbol{T_{i,r_{arrival}}}$ **+ 20ms.**
- ACK packets are not lost. **Thus, there is no packet retransmission timeout (RTO)**

4) Evaluation scenario

Case 1) Base station with unlimited queue & without congestion control

- The sender sends packets without its queue and sliding window. It can send packets as much as the bandwidth of link1 at time t.
- The base station also can send packets in its queue as much as the bandwidth of link2 at time t. The base station queue size is unlimited so that there is no packet losses.

Case 2) Base station with unlimited queue & with naïve congestion control

- The sender sends packets with a sliding window (inflight). If the window size is larger than the congestion window (cwnd), the sender cannot send the packets although the packets can be transmitted in link1 bandwidth. The sender should send the packets in the sender queue only when inflight < cwnd at the next available time step. When the ACK of the first packet in the sliding window arrives to the sender (i.e., $T_{cwnd\_idx,r_{arrival}} + 20ms$), **the window moves to the next packet and increases the window size (cwnd) of 2**.
- The naïve congestion control's cwnd is managed by AIMD (additive increase multiplicative decrease). ACK generation in receiver use Go-back-N method (cumulative ack). If the sequence of ACK doesn't match the packet sequence, retransmit the packets from sequence of the ACK again, and **reduces the cwnd divided by 4.**
- For simplicity, **we do not implement packet timeout**. The retransmission is occurred only when ACK.sequence != packet.sequence.
- We note that naïve congestion control doesn't have **delayed ACK but can be implemented.**

Case 3) Base station with limited queue & with naïve congestion control

- Case 3 is expanded from case 2. In this case, the base station has a limited queue size. Therefore, unlike version 2, when the queue size is larger than its queue capacity, a loss occurs. Then the ACK will not arrive at the sender. For loss handling, the sender retransmit the packets with Go-back-N method without timeout.

5) Performance measurement

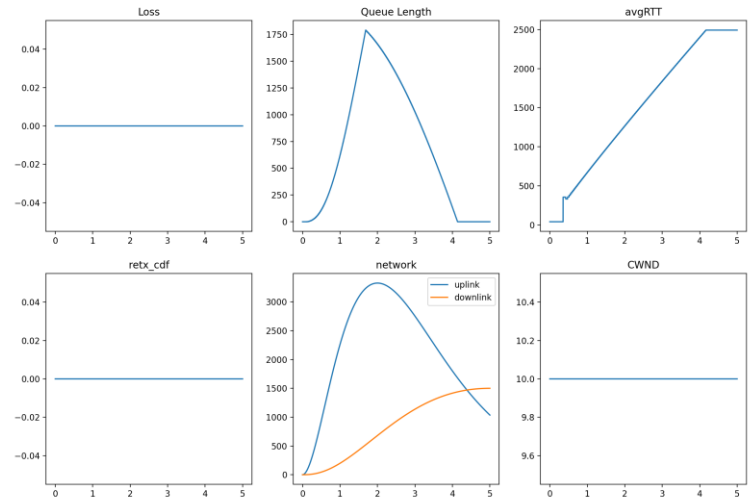
In this project you can measure the performance of case 1, 2 and 3. A plotting method is provided in include/utils.py. You can measure the following performances in the figure

- the packet losses in base station at (0,0)
- queue length of base station at (0,1)
- average RTT at (0,2)
- number of retransmitted packets (cdf) at (1,0)
- uplink and downlink bandwidth at (1,1)
- sender's cwnd size at (1,2)

Also, the total # of acked packets (transmitted success) is provided in command line, and the simulation done time is provided if the all the packets are transmitted

Case 1) Performance measurement

- simulation time: 5s (5000ms)
  # of packets: 2000
  Link1: $\alpha = 3,\ \beta = 1$
  Link2: $\alpha = 3.5,\ \beta = 2$
  command: **python3 main.py**

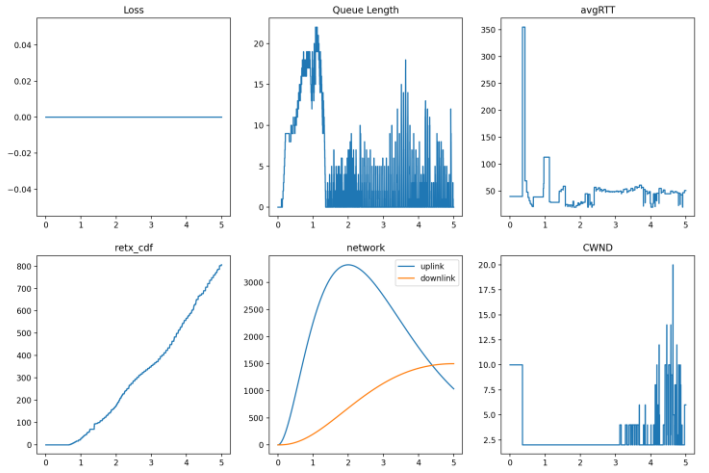



```
simulation without congestion control
simulation done at time:  4.168
# of transmitted packets:  2000
```

Figure 3: performance measurement for case 1

Case 2) Performance measurement

- simulation time: 5s (5000ms)
  # of packets: 2000
  Link1: $\alpha = 3,\ \beta = 1$
  Link2: $\alpha = 3.5,\ \beta = 2$
  command: **python3 main.py**

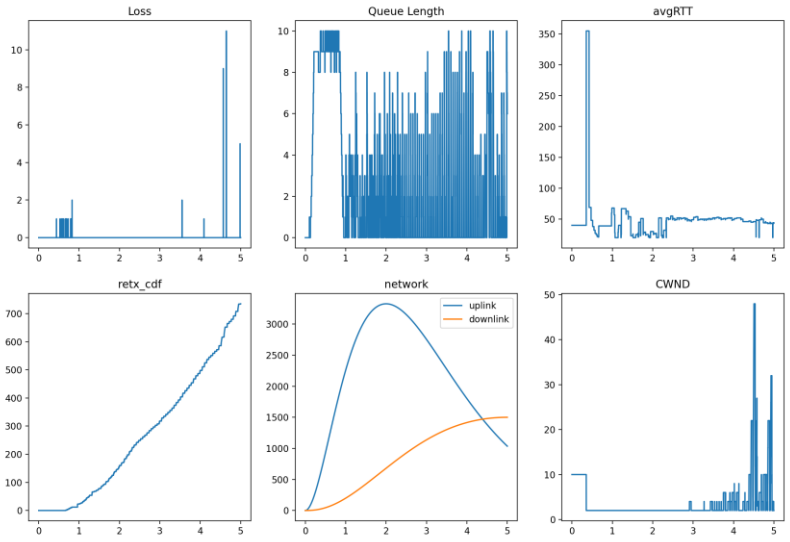



```
simulation with congestion control
# of transmitted packets:  368
```

Figure 3: performance measurement for case 2.

Case 3) Performance measurement

- simulation time: 5s (5000ms)
  # of packets: 2000
  Link1: $\alpha = 3,\ \beta = 1$
  Link2: $\alpha = 3.5,\ \beta = 2$
  queue length of base station: 10
  command: **python3 main.py**




```
simulation with congestion control
# of transmitted packets:  334
```

Figure 5: performance measurement for case 3

Implementation Guidelines

The skeleton code is provided in the assignment. **You need to implement the part commented out as IMPLEMENT in the code**. The code is composed as follows.

- **main.py**

plot_gamma() function implements the gamma function and plots the graph by reading the time_slice.

simulation() function implements simulators for case 1, 2, and 3. It needs to set the threshold value for all the cases (unlimited or limited size) before executing the simulator.

- **includes/simulator.py**

Simulator() class defines simulator for measure congestion control performance. It needs to implement the uplink and downlink channel with provided bandwidth before executing the simulator

- **includes/model.py**

Packet() class defines packet object to transmit from sender to receiver

- self.start_time indicates start time of the packet. It is recorded on the sender.

- self.bs_arrival indicates arrival time of the packet from sender to BS. It is recorded on the sender.

- self.recv_arrival indicates arrival time of the packet from BS to receiver. It is recorded on the base station (BS).

- self.ack_arrival indicates arrival time of the acknowledgment from receiver to sender, recorded on the receiver.

BaseStation() class defines base station object for simulation. All the components and member functions are provided in the skeleton code. You may use the base station's method function for the code blocks you need to implement.

Client() class defines sender and receiver object for simulation

- Client.send() function sends packets in terms of channel condition and cwnd size. In case 1 (without congestion control), the function only considers channel condition. In case 2 and 3, the function needs to consider both channel condition and cwnd size. As you can see the skeleton code, the implementation code for case 1 is provided.

- **Client().recv()** function implement the generation of cumulative ACK packets. The function should be implemented with naïve Go-Back-N based ACK generation. In the Go-Back-N method, you only need to send cumulative ACK (only 1 ACK). You can add any additional logic to the function.

Client().congestion_control() function implement the congestion control for sender. The function is implemented with naïve AIMD, and naïve measurement of average RTT. You may add any additional logic to the function (i.e. BDP) or change the value of any initialized parameter of AIMD, RTT, etc.

- **includes/utils.py (don't need to implement it)**

this file defines methods that are needed for plotting graph, packet generation, etc.

- **Design example**

Looking at figure 4, you can find that there exist lots of retransmission when there are no loss packets in the base station. It means that TCP sender has a problem. (hint. **Ignore duplicated ACK**). By ignoring the duplicated ACK, we can eliminate or reduce the retransmission retransmission, see figure 6

In other side, we can implement the congestion control more aggressively with parameter initializing and parameter control. As we can see the figure 7, aggressive control logic sends more packets in a given time (5 second)

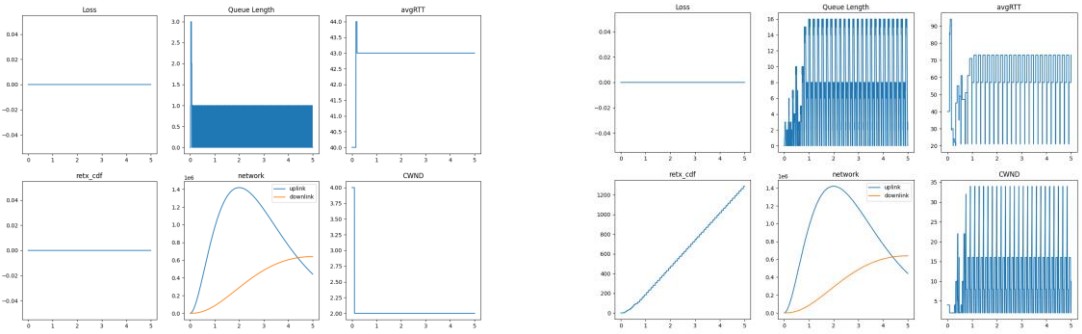


Figure 6: congestion control without retx by ignore duplicated_ACK (left), naïve congestion control for Case 2 (right).

```
simulation with congestion control
# of transmitted packets:  334
```

```
simulation with congestion control
# of transmitted packets:  582
```

Figure 7: aggressive congestion control with queue_size = 10 (left), naïve congestion control for Case 2 (queue_size = 10) (right).

Grading

The grading policy is followed

- Reproduce plot (20%) – gamma function (5%), Case 1 (5%), Case2 (5%), Case 3 (5%)
- Ignore duplicated ACK with Case 3 (20%) – less than 5% of the retransmission packets get full grade
- Aggressive congestion control with Case 3 (20%) – more than 600 get full grade
- Report (40%) – justify of your congestion control logic (30%), explain implemented code (10%)

Submission

- Submit your codes and report explaining your output and codes(*simulator_version1, simulator_version2 and simulator_version3*) to eTL project section.
- Include your codes to a folder “project2_<student id>” and compress the folder.
- Make the compressed zip file name as “project_<student id>”.
- You should submit only one compressed zip file.