ECS 152A: Computer Networks

Spring 2022

Project 2

(150 points)

Due Date: Wednesday, May 25th, 2022 (by 11:59 PM – before midnight)

Team: The project is to be done in a team of ***at most 4 students***. You *cannot* discuss your code/data with other classmates (*except* your project partners)

All submissions (including your code) will be checked for ***plagiarism*** against other submissions as well as the public Internet. Plagiarized submissions will be entitled to ***zero*** points.

This project must be completed in Python 3+.

You are asked to implement a sender that provides reliable data delivery service and congestion control over the unreliable ***UDP*** transport using socket API. You will implement the following variations of this service:

1. Stop and Wait Service
2. Static Sliding Window Service
3. Dynamic Sliding Window Service (TCP Tahoe)
4. Custom Congestion Control Service ***(Extra Credit)***

You need to implement a sender for each of the above services separately.

Your Sender:

The sender should run on ***localhost*** on a port other than the one used by the receiver. Sender for each service needs to transfer the ***message.txt*** file of size ~2.5MB to the receiver. You should use 5 seconds as the fixed timeout unless suggested otherwise. Set packet size to 1000 bytes. Each sent packet should contain only 1 header field in the following format:

"***x***|" where "***x***" represents the non-negative integer sequence number of the packet and "|" represents the delimiter between the sequence number and file payload.

Receiver (*receiver.py*):

You are provided with the receiver code. Everytime you re-test the sender, make sure you restart the receiver as well. You cannot make any modifications to the receiver code. Receiver also runs on ***localhost***.

The receiver implements a *cumulative acknowledgment*. For example:

If the receiver receives packets with sequence number 1, 2, 4, 5 (packet 3 was lost) then it acknowledges:

- packet 1 with sequence # 1
- packet 2 with sequence # 2
- packet 4 with sequence # 2
- packet 5 with sequence # 2

When the sender eventually retransmits packet 3 to the receiver, the receiver acknowledges the receipt of packet 3 with sequence # 5, i.e. every packet from 1 to 5 has been received.

The cumulative acknowledgment means that the receiver receives a packet, extracts the sequence number, and sends back the acknowledgment cumulatively where the acknowledgment *n* being sent back means that every packet from *n, n-1. n-2,..., to 1* has been received.

Command line inputs:

- Receiver should take exactly 1 command line input = port number on which you want your receiver to run.
- Sender should take exactly 1 command line input = port number on which your receiver is running.

Terminal Output:

Your sender code should print the following information for each transmitted packet *and* received acknowledgement:

```
<blank line>
Current Window: [1, 2, 3]
Sequence Number of Packet Sent: 1
Acknowledgment Number Received: 1
<blank line>
```

Once the entire message.txt file has been transmitted, print the following:

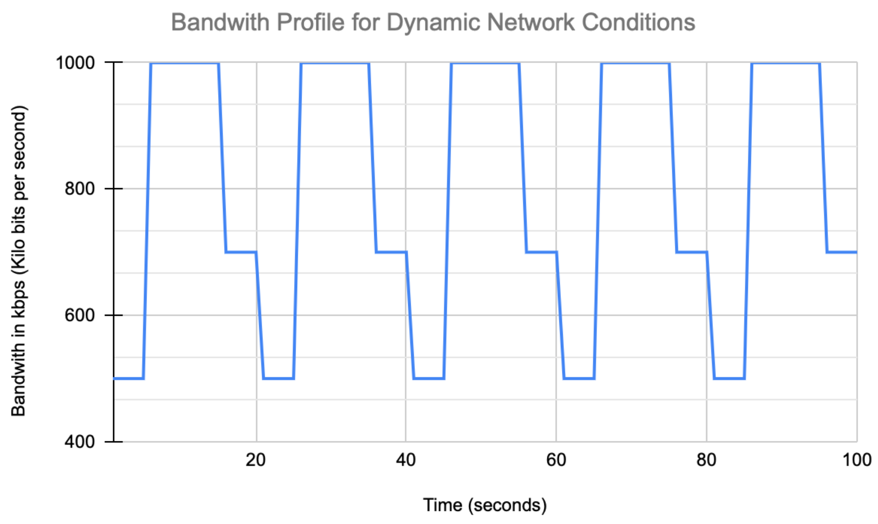
Average Delay = <Delay_{avg}>
Average Throughput = <Throughput_{avg}>
Performance = <$\log_{10}(\text{Throughput}_{avg}) - \log_{10}(\text{Delay}_{avg})$>

Here, Delay_{avg}, Throughput_{avg}, and Performance are as defined in the *metrics* section below.

Traffic Controller (*train.sh*):

You need to run and test all the services in presence of the dynamic network conditions as enforced by "*Linux Traffic Controller (TC)*" implemented in the bash file ***train.sh***. You need to explicitly impose these dynamic network conditions before running/testing your sender implementations. This can be achieved by running ***bash train.sh*** on shell with sudo access.

The ***train.sh*** provides the following bandwidth profile.



TC is a Linux-based utility that also requires root permissions to "shape" your network interface. Thus, we ask that you set up a Linux Virtual Machine (VM) to avoid any issues with your regular operating system. Following are the steps that you can follow to install a VM:

1. Download and Install **Oracle's VM VirtualBox**: https://download.virtualbox.org/virtualbox/6.1.34/VirtualBox-6.1.34-150636-Win.exe
2. Download **Ubuntu 20.04.4 LTS (Focal Fossa)** (ubuntu-20.04.4-desktop-amd64.iso): https://releases.ubuntu.com/20.04/ubuntu-20.04.4-desktop-amd64.iso
3. Create a "New" VM Instance:
    a. OS: Linux Debian (64-bit)
    b. RAM: 20364 MB (~20GB)
    c. Virtual Hard disk (49.66GB) with type: VDI and dynamically allocated storage
    d. Provide path of (2) to load ISO file and Install the OS.

Note: This is a configuration in which your submissions will be tested. However, you can use any other RAM and Hard Disk Sizes. Ubuntu 20.04.4 installation shall require at least 8 GBs, so make sure you configure your VM with a size larger than that.

Metrics:

For each service you implement, you need to compute the following metrics:

1. Per-Packet Delay

Delay for a packet is defined as the time elapsed from the point of start of transmission of a packet until the point of receipt of its acknowledgement. So,

$$\textbf{Delay} = \mathbf{T}_{\textbf{acknowledgement arrival}} - \mathbf{T}_{\textbf{packet dispatch}}$$

Note: If a duplicate acknowledgement is received then DO NOT report/compute the delay for the previously transmitted packet(s) and do not use it in your average per-packet delay calculation. This means that you need to compute packet delays only when you receive non-duplicate acknowledgements. If it is a non-consecutive acknowledgement (e.g., there is a jump in sequence numbers of the acknowledgement) then use it to compute the delay of all outstanding unacknowledged packets. Consider the following example where 3 packets are being transmitted:

- Packet 1 is transmitted at t_1
- Acknowledgement # Received: 1 @ time T_1
- Packet 2 is transmitted at t_2
- Acknowledgement # Received: 1 @ time T_{1Dup1}
- Packet 3 is transmitted at t_3
- Acknowledgement # Received: 1 @ time T_{1Dup2}
- Packet 2 is retransmitted at $t_{2Retrans1}$
- Acknowledgement # Received: 3 @ time T_3

Here,

- Delay for Packet 1 = $T_1 - t_1$
- Delay for Packet 2 = $T_3 - t_2$
- Delay for Packet 3 = $T_3 - t_3$

2. **Per-Packet Throughput**

Per-packet throughput should be computed for each packet in the same manner as per-packet delay is explained above. Use the same assumptions as mentioned above.

3. **Performance**

We define Performance as the following:

$$\textbf{Performance} = \log_{10}(\text{Throughput}_{avg}) - \log_{10}(\text{Delay}_{avg})$$

where,
Throughput_{avg} = Average per-packet throughput observed across all packets transmitted
Delay_{avg} = Average per-packet delay observed across all packets transmitted

Units: The Delay should be in ***milliseconds*** and the Throughput should be in ***bits per second***.

Simulation:

You may test 1 sender and 1 receiver scenario while building your sender. But, you are required to simultaneously run two pairs of the same sender and receiver to report the above metrics. So, you shall first run two receivers (e.g. on separate terminals) with different port numbers and then run two senders (one for each receiver) at the same time. Finally include the output of any one sender in your report. Your submission will be tested in this scenario of two sender-receiver pairs competing with each other.

Part 1: Stop and Wait Service (*15 points*)

Implement an application-layer reliable "Stop and Wait" data delivery service over UDP sockets using sequence numbers, acknowledgments, timeouts (fixed), and retransmissions to ensure successful delivery of the provided text file (***message.txt***).

Part 2: Static Sliding Window Service (2*5 points*)

Implement an application-layer reliable "Static Sliding Window" data delivery service over UDP sockets using sequence numbers, acknowledgments, timeouts (fixed), and retransmissions to ensure successful delivery of the provided text file (***message.txt***). Implement a fixed-size (5-packet) sliding window.

Part 3: Dynamic Sliding Window Service (*40 points*)

In this part, implement a "Dynamic Sliding Window" service at the sender. Assume that the receive window (rwnd) at the receiver is sufficiently large. Thus, the sliding window size is determined by **cwnd** at the sender. Implement slow start and congestion avoidance as defined in TCP Tahoe. Start with the *initial congestion window* of 1 packet and *ssthresh* of 16 packets. You should use 5 seconds as the initial timeout value and then use a dynamic timeout based on the procedure described in Section 3.5.3 of the textbook to transfer the ***message.txt*** file.

Part 4: Custom Congestion Control Service (Optional) (*Extra Credit*)

Implement your custom UDP-based congestion control protocol that ensures reliable delivery of ***message.txt***. You are allowed to change the sender any way you want (e.g., use custom window sizing/sliding strategy, timeout adjustments, re-transmissions, etc), but you cannot change the receiver.

Extra Credit Details:

1. If you outperform *TCP Tahoe* in terms of the *performance metric* on the ***train.sh*** bandwidth profile, you will be awarded 50% extra credit for this project.
2. If you outperform TCP Tahoe on the secret ***test.sh*** bandwidth profile, you will be awarded an additional 50% extra credit for this project. Thus, you can effectively double your grade for this project by simply outperforming *TCP Tahoe*!
3. If your proposed congestion control outperforms *TCP Tahoe* and all other student submissions on ***test.sh*** bandwidth profile, you will be awarded a direct A + in the course.

Project Report Details (*70 points*)

Specify the following at the beginning of the report:

1. Full Name of student 1 (Student ID) (Discussion Group)
2. Full Name of student 2 (Student ID) (Discussion Group)
3. Full Name of student 3 (Student ID) (Discussion Group)
4. Full Name of student 4 (Student ID) (Discussion Group)
5. Name of the code files submitted

Question I: (*10 points*)

1. Report Ping Times with and without TC. Justify the values using *train.sh*.
2. Report packets lost during the entire file transfer. Match/Justify the values using *train.sh.*

Question II: All parts (*60 points* for first 3 parts: *20 points* per part)

For each Service implemented from Part 1 through 3, include the following in your report:

1. Compute and Report $\text{Delay}_{\text{avg}}$, $\text{Throughput}_{\text{avg}}$, and Performance for your sender implementation. Include a screenshot of any one sender terminal window as it looks when the file transfer is completed. **(*8 points*)**
2. Plot of per-packet delays. **(*6 points*)**
3. Plot of per-packet throughputs. **(*6 points*)**

Question II: Part 4 (Optional)

In addition to the information provided above, describe your idea/approach using suitable text description and figures. You need to describe the key ideas behind your custom congestion control in reasonable detail to be eligible for extra credit.

Required Files:

Download and us the following files as directed in the above description:

1. **message.txt**
2. **train.sh**
3. **receiver.py**

Testing Environment:

All submissions will be tested on Python 3+ and inside a VM specified in the description. Thes testing will occur for a two sender two receiver scenario.

Late Submission Policy:

No late submissions are allowed. However, if you barely miss the deadline, you can get partial points upto 24 hours. The percentage of points you will lose is given by the equation below. This will give you partial points up to 24 hours after the due date and penalizes you less if you narrowly miss the deadline.

$$Total\ Marks\ you\ get\ =\ (Actual\ Marks\ you\ would\ get\ if\ NOT\ late)\ \times\ \left[1\ -\ \frac{hours\ late}{24}\right]$$

Late Submissions (later than 24 hours from the due date) will result in zero points, *unless you have our prior permission or documented accommodation*.

Best of luck

Submission Page

Submission Checklist: Submit the following documents on Canvas:

1. Replace placeholder text below with actual file names before printing. Name the files using the name of the first team member listed below. Include this page as a scanned PDF after signing the plagiarism statement below. *Your submission will be rejected without inclusion of this signed statement.*

2. **Stop and Wait Service Sender Program (.py) (15 *points*)**
   part1_[name1]_[student_id1].py

3. **Static Sliding Window Service Sender Program (.py) (25 *points*)**
   part2_[name1]_[student_id1].py

4. **Dynamic Sliding Window Service Output (.py) (*40 points*)**
   part3_[name1]_[student_id1].py

5. **Custom Congestion Control Service Sender Program (.py) (optional)**
   part4_[name1]_[student_id1].py

6. **Project 2 Report (.pdf) (*70 points*)**
   report_[name1]_[student_id1].py

I certify that all submitted work is my own work. I have completed all of the assignments on my own without assistance from others except as indicated by appropriate citation. I have read and understand the university policy on plagiarism and academic dishonesty. I further understand that official sanctions will be imposed if there is any evidence of academic dishonesty in this work. I certify that the above statements are true.

Team Member 1:

Full Name (Printed)	Signature	Date

Team Member 2:

Full Name (Printed)	Signature	Date

Team Member 3:

Full Name (Printed)	Signature	Date

Team Member 4:

Full Name (Printed)	Signature	Date