Introduction to Computer Networks

Assignment 3: Pipelined Reliable Data Transfer over UDP

1. Goal

- Develop a pipelined reliable data transfer protocol using UDP socket programming.
- Use Mininet to emulate various network environment.

2. Development environments

- TA will evaluate your results on Mininet. (because Mininet can be run on Linux, you need to install Linux or virtual machine).
- You use Python (version 3.4+), C or C++ on Linux.
- You have to describe your development environment information in detail in the report.
  If not, TA cannot evaluate your program and you will get zero points.
- We will give you the preset VM image which has Mininet and skeleton codes.
  - "execute_mn.py" is python code that containing preset topology which has two hosts and bottleneck link and execute receiver program and sender program automatically.
    - You can manipulate bottleneck link's bandwidth, one way delay, loss by changing parameters of constructor of 'Assignment3Topo' in "execute_mn.py".
    - "execute_mn.py" receives 4 argument, Window size, Source Filename, Destination Filename. So, you need to execute it as follows. The basic execution and output of "execute_mn.py" example is as shown in Figure 1.

sudo python execute_mn.py <Window size> <source Filename> < Destination Filename >

◆ There are three mode. You can choose either one of them by uncomment it.

```
"If you want to test with ping and iperf, uncomment this"
"""
net.pingFull([receiver,sender])
net.iperf([receiver,sender],seconds=5)
"""

popens = {}
"If your code is python, uncomment this"
"""
popens[receiver] = receiver.popen('python','receiver.py')
popens[sender] = sender.popen('python','sender.py',recvAddr, str(windowSize),
srcFilename, dstFilename)
"""

"If your code is c++ or c, uncomment this"
"""
popens[receiver] = receiver.popen('receiver.out')
popens[sender] = sender.popen('sender.out',recvAddr, str(windowSize),
srcFilename, dstFilename)
"""
```

- First mode is for testing connectivity between two host using ping and iperf.

  "Ping" is used to test reachability of a host on an IP network. The result of Ping shows packet loss and RTT.

  "Iperf" is used to test the performance of network between two hosts. The result of iperf shows bandwidth between two hosts using TCP.

- Second mode is for running file transfer with python code.
- Third mode is for running file transfer with c or c++ compiled program.

■ The skeleton codes for sender.py (or sender.cpp) and receiver.py (or receiver.cpp) contain the function(method) about logging and receiving execution argument. Please use them.

■ Every file treated by all the programs are saved in root directory(where the execute_mn.py is).

■ For more details about Mininet, please read PPT file.

3. Functionalities to implement

● Sender

■ When starting a sender program, receives the IP address of a receiver, and window size, source File name, Destination Filename by execution argument.
(The destination port number is fixed with 10080.)

■ After running the program, sender start to send file immediately.

- The files must be in the same directory where the sender program is located.
- Send the Destination Filename and the file itself to the receiver.
- Divide the file into packets that are 1400 bytes or less in size.
- Transmit packets as much as the given window size

- By receiving ACKs, send the next packet in available or detect and recover dropped packets.
  - Calculate the average RTT and configure the timeout value. (The initial timeout value is set to 1 second.)
  - Use "3 duplicated ACKs" for Fast Retransmission. If 3 duplicated ACKs are detected, retransmit the lost packet. If the lost packet is successfully retransmitted, continue to send the next in-order packets.
  - Use a **single** timer to check "timeout" of packets.
  - At the beginning of the program, timeout size is set by initial timeout from the argument. As the program runs, the timeout value must be adaptively change according to the packet RTT(round trip time)
- Per-file transfer, write a log file
  - E.g. " fileAAA_sending_log.txt"
  - Write packet & ACK event information with the logging time as shown in Figure 2.
  - If the file transfer is finished, write the goodput and the last calculated average RTT in the log file :
    * goodput = #packets / sec (the number of unique packets / the total transfer time).

- Receiver
  - Bind a socket with the 10080 port number.
  - When the receiver receives a packet successfully, send a cumulative ACK.
    - Store received out-of-order packets in temporary buffer.
    - Store received in-order packets in a file.
  - Per-file transfer, write a log file
    - E.g. " fileAAA_receiving_log.txt"

- Write packet & ACK event information with the logging time as shown in Figure 3.
- If the file transfer is finished, display the goodput in the log file :
  #packets / sec (the number of unique packets / the total transfer time).

- Miscellaneous
  - If you are c++ user, please use contained Makefile. Modify it if you want.
  - Time information starts from 0 seconds for each file transfer.
  - The sequence number is started from 0 for each file transfer and increases one per packet.
  - Set the ACK number equal to the data packet sequence number.
  - In the log files, display sentences with proper alignment to improve readability
  - In a sender program, you may use a thread for receiving ACKs. If the receiving thread is started before the sending thread is started, you can see a socket related error message. You have to call bind( ) with the port number of zero (to request any unused source port number)

4. Experimentation

- For a **single** file transfer, make following experimentations.
  1) Draw a goodput graph and average RTT with different probabilities of packet loss.
     - the window size is fixed to 40. The bottleneck link's bandwidth 10Mbps, 25ms one way delay.
     - the loss probability of the bottleneck link is changed to 2%, 4%, 8%, 16%.
  2) Draw a goodput graph and average RTT with different window sizes.
     - the bottleneck link's loss probability is fixed to 2%, bandwidth 10Mbps, 25ms one way delay.
     - Change the window size to 8, 16, 32, 64

  * Each experiment must be measured 10 times to get an average

- Testing file size must be large enough (**more than 10 Mbytes**)

5. Submission

- The deadline is **5.31 (Sun) 23:59**.
  - For delayed submissions, a penalty of -15 points applies every 24 hours. After 72 hours, you get zero points.

- In the case of plagiarism, you will receive 0 points for the first time and ***F*** for the second.

- Submit a zip file including a **report** and two (sender and receiver) program sources(and Makefile if you are c++ user) to iCampus

  - The report file format should be PDF.

  - Name the Report file as follows ***StudentID_Name.pdf*** (ex: 2019001_홍길동.pdf)

  - The report has to include the following things;

    1) Describe your development environment information in detail
    (versions of operating systems, languages, compilers/interpreter versions, compile options)

    2) Present how to design your assignment such as data structures and algorithms.

    3) Show experiment graphs for the experimentation results.

6. Scoring

- Total 100 points

  - 20 points: the sender transmits packets as much as the window size, and transmits the next packet whenever receiving in-order ACK. (Figure 2 & Figure 3)
    - display packet & ACK event information in log files at the sender and the receiver.

  - 20 points: Calculate goodput and avgRTT after the file transfer is finished.
    - display goodput and avgRTT in a log file at the sender.
    - display goodput in a log file at the receiver.

  - 20 points: Detect a timeout of a packet and retransmit the lost packet at a sender.
    - Use the timeout value based on the average RTT.
    - display timeout events in the log file at the sender.
    - display when the packet has sent and current timeout value.

    example)

    ```
    1.063 pkt: 8  |  timeout since 1.010(timeout value 0.052)
    1.063 pkt: 8  |  retransmitted
    ```

  - 10 points: Detect 3-duplicated ACKs and transmit the lost at a sender.
    - display the events in the log file at the sender.

```
1.063 ack: 5  |  received
1.064 ack: 5  |  received
1.065 ack: 5  |  received
1.066 ack: 5  |  received
1.066 pkt: 5  |  3 duplicated ACKs
1.066 pkt: 6  |  retransmitted
```

- 10 points: Valid File

  - The source file and destination file must be exactly same in any circumstances.

- 20 points: Report

  - 10 points for the basic documentation.
  - 10 points for the graphs of two experiments.

7. Q&A

- Leave your questions on the google sheet

```
mininet@mininet-vm:~$ sudo python execute_mn.py 40 1.jpg 1_result.jpg
Starting test...
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
<h1>: receiver program starts...
<h2>: sender program starts...
```

Figure 1. Basic execute_mn.py operation

```
0.000 pkt: 0 | sent
0.001 pkt: 1 | sent
0.025 ACK: 0 | received
0.025 pkt: 2 | sent
0.026 ACK: 1 | received
0.026 pkt: 3 | sent
0.051 ACK: 2 | received
0.052 ACK: 3 | received

File transfer is finished.
Throughput: 137.93 pkts / sec
Average RTT: 65.2 ms
```

Figure 2. "sample.jpg_sending_log.txt"

```
0.000 pkt: 0 | received
0.000 ACK: 0 | sent
0.001 pkt: 1 | received
0.001 ACK: 1 | sent
0.025 pkt: 2 | received
0.025 ACK: 2 | sent
0.026 pkt: 3 | received
0.026 ACK: 3 | sent

File transfer is finished.
Throughput: 137.93 pkts / sec
```

Figure 3. "sample.jpg_receiving_log.txt"