Assignment 2 - Understanding the principles of RDT (Reliable Data Transfer)

Deadline: 01/11/2024, 23:59
Marks: 100
Penalty: 5% marks will be deducted for each day after the due date.

The assignment gives you an opportunity to implement and evaluate different protocols for achieving end-to-end reliable data transfer at the application layer over the unreliable datagram protocol (UDP) transport protocol. In particular, it is required that you implement the following two protocols:

1. Stop-and-wait **(50 marks)**
2. Go Back N **(50 Marks)**

For each of the above protocols, you will implement the two protocol endpoints referred to as the sender (h1) and receiver (h2), respectively. These endpoints also act as application programs. Data communication is unidirectional, requiring the transfer of a large file from the sender to the receiver over a link as a sequence of smaller messages. The underlying link is assumed to be symmetric in terms of bandwidth and delay characteristics.

1. VM instructions

The VM previously provided for assignment 1 can be reused. Link to download the VM: https://drive.google.com/file/d/1maiWDRqWK213oLYxGXQyY3AJXdc4ldGi/view?usp=sharing

2. Setup Instructions

1. Start a mininet instance using the command “sudo mn”
2. Download and paste the UDP template code into any folder of your choice:
   Sender.py:
   https://drive.google.com/file/d/11DThUyqwZRZ38RyiBgOGnZaqjFJkNsRw/view?usp=share_link

   Reciever.py:
   https://drive.google.com/file/d/1iozOQ2HbSlh5PLu5LK_FjR7xTNLwE-Gt/view?usp=share_link

3. xterm into host h1 and h2 using commands “xterm h1” and “xterm h2” respectively.
4. Execute “python3 reciever.py” from host h2
5. Execute “python3 sender.py” from host h1

Observe that the template program sends a string taken as input and sends it as a UDP message. Wireshark can be used to visualise the packets in the respective interfaces.

3. Network conditions:

The network conditions need to be modified for the assignment. Here we describe template code that should be modified as per the instructions to implement and evaluate the protocols.

a. Bandwidth: Bandwidth describes the capacity of links. In this mininet assignment, we have only two links (h1-eth0 and h2-eth0). We need to test the protocols under varying bandwidth conditions. So if one would want to set the bandwidth of both links to 10Mbps, the following commands can be used:

> sudo tc qdisc add dev h1-eth0 root netem rate **10Mbit** limit 100 (execute from xterm h1)
> sudo tc qdisc add dev h2-eth0 root netem rate **10Mbit** limit 100 (execute from xterm h2)

Here “100” is the buffer/queue size of the interface. Feel free to vary them as per the observed behaviour.

b. Propagation delay: Propagation delay is the time taken for a packet to traverse the link. To set the RTT to ~10ms, set the propagation delay of the two links to 5ms. The following command is to be executed from their respective terminals (h1 and h2, respectively) to achieve this.

> sudo tc qdisc change dev h1-eth0 root netem delay **5ms** (execute from xterm h1)
> sudo tc qdisc change dev h2-eth0 root netem delay **5ms** (execute from xterm h2)

c. Packet loss: Real links may lose packets due to congestion or various other factors. Since we are working on emulated links, we need to induce packet loss. The following commands will ensure that **5% of packets** are lost.

> sudo tc qdisc change dev h1-eth0 root netem loss 5% (execute from xterm h1)
> sudo tc qdisc change dev h2-eth0 root netem loss 5% (execute from xterm h2)

4. Detailed assignment specification:

With the VM and the template files set up, the objective is to modify the contents of both sender and receiver to implement “stop-and-wait” and ”Go Back N” protocols. Before implementing the logic for the protocols:

1. implement sender and receiver endpoints for transferring a large file given here from the sender to the receiver over UDP as a sequence of small messages with 1KB maximum payload (NB. 1KB = 1024 bytes).
2. Network conditions: **10Mbps bandwidth**, **5ms propagation delay** and **0% packet loss rate** (i.e., no packet loss).

3. In the sender code, insert, at a minimum, a 10ms gap (i.e., sleep for 10ms, also read the **Note** below) after each packet transmission. The buffer is set to a buffer/queue size of 100 as per the above commands for bandwidth. Because the sending rate from the sender is typically larger than the link speed (10Mbps) specified here, the queue is likely to overflow, and hence packet losses are unavoidable.
4. To allow the test of an ideal, reliable channel case, the 10ms gap is suggested. If packet losses continue to occur, increase the time gap.
5. The exchanged data messages must also have a header part for the sender to include a 16-bit message sequence number (for duplicate filtering at the receiver in the next and subsequent parts) and a bit/byte flag to indicate the last message (i.e., end-of-file).
6. Let us call the resulting files as sender.py and receiver.py

Note: From section(A) onwards, you should remove the sleeping part (i.e., without the time gap) from the sender and follow new network conditions as specified.

(A) Stop-and-wait:

a. Extend sender and receiver applications from above to implement a stop-and-wait protocol (specifically rdt3.0) described in section 3.4.1 in (J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach" (6th edition), Pearson Education, 2013)
b. Rename the resulting two programs, yourRollNo_senderStopWait.py and yourRollNo_receiverStopWait.py, respectively.
c. This part requires you to define an acknowledgement message the receiver will use to inform the sender about receiving the data message. Discarding duplicates at the receiver end using sequence numbers put in by the sender is also required. You can test the working of duplicate detection functionality in your implementation by using a small retransmission timeout on the sender side.
d. Network conditions: **10Mbps bandwidth**, **5ms propagation delay** and **5% packet loss rate**
e. Experiment with different retransmission timeouts and the corresponding number of retransmissions and throughput. Tabulate your observations in the space provided under Stop-and-wait Question 1 in the report.
f. For this, your sender implementation should measure average throughput (in KB/s), which is defined as the ratio of file size (in KB) to the transfer time (in seconds). Transfer time, in turn, can be measured at the sender as the interval between the first message transmission time and acknowledgement receipt time for the last message. Before the sender application finishes and quits, print the average throughput value to the standard output.
g. Under Stop-and-wait Question 2 in the report, discuss the impact of retransmission timeout on the number of retransmissions and throughput. Also indicate the optimal timeout value from a communication efficiency viewpoint (i.e., the timeout that minimises the number of retransmissions).

(B) Go back N:

a. Extend sender.py and receiver.py to implement the Go-Back-N protocol as described in section 3.4.3 in (J. F. Kurose and K. W. Ross, “Computer Networking: A Top-Down Approach” (6th edition), Pearson Education, 2013), by allowing the sender window size to be greater than 1.
b. Name the sender and receiver implementations from this part as yourRollNo_senderGBN.py and yourRollNo_receiverGBN.py, respectively.
c. Network conditions: **10Mbps bandwidth** and **0.5% packet loss rate**
d. Experiment with different window sizes at the sender (increasing in powers of 2 starting from 1) and different **propagation delay values (5ms, 50ms and 150ms)**.
e. For the 5ms case, use the “optimal” value for the retransmission timeout identified from the previous part. Adjust the timeout suitably for the other two cases (i.e., 50ms and 150ms cases).
h. Tabulate your results under Question 1 and answer Question 2 in the second subsection provided in the report.

What to submit:

- Readme file for instructions to run your program
- Zip file containing two folders: (1) stopandwait (2) go-back-n and name the zip file as <ROLLNO>_Assignment2.
- Each folder will contain the respective python files along with the readme
- The zip file should also contain your report in pdf format
- Failure to follow the instructions will result in difficulty on our side for evaluation and will result in penalties.

PLAGIARISM STATEMENT <Include it in your report>

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand my responsibility to report honor violations by other students if we become aware of it.

Name:
Date:
Signature: <keep your initials here>