CS305 Project: Reliable Data Transfer

November 2020

In this project, you are to build your own **reliable data transfer (RDT) protocol**, and implement the echo server demo basing on it. The network environment is stochastic and asynchronous by its nature, leading to the following situations your RDT needs to deal with: packet loss, delay, data corruption and network congestion.

1 Requirements

1.1 Message Format

Here is an example of how a packet is organized. Add more fields if necessary.

SYN	FIN	ACK	SEQ	SEQACK	LEN	CHECKSUM	PAYLOAD
1 bit	1 bit	1 bit	4 bytes	4 bytes	4 bytes	2 bytes	LEN bytes

1.2 Reliable Data Transfer

Typically for a connection, you are supposed to execute the following steps consecutively

1. Accept and establish a connection;
2. Maintain the connection, keep listening and replying;
3. Close the connection, release the resources.

And the factors that cause the channel to be unreliable are

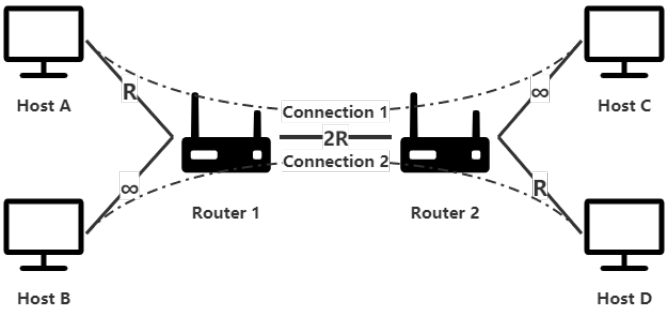
- Delay;
- Loss;
- Corrupt.

Your are responsible for detecting and correcting these abnormalities (through retransmission) when they occur. No restriction is posed on the actual mechanism, you can use GBN, SR or Selective Acknowledge as introduced in the textbook.

1.3 Congestion Control

For a better understanding of the cause and cost of congestion, consider the situation shown in the figure with 4 hosts and 2 routers, where R and ∞ denote link capacity (maximum sending/receiving rate). Two connections are $A \Longleftrightarrow C$, $B \Longleftrightarrow D$. Each of the two routers has finite buffer, and would discard any incoming packet when its buffer is full, thus causing packet loss.

For example, if host A and B take sending rates r_1, r_2 respectively and $r_1 + r_2 > 2R$, then $\frac{r_1+r_2-2R}{r_1+r_2}$ of the packets will be lost before they get on their way to Router 2. It can be shown that in the extreme case, without any congestion control, i.e., each host sends at the maximum rate possible, the network will perform arbitrarily bad and the effective throughput goes to 0.



For this section, you should implement congestion control in addition to the reliable transfer function to address the problem described and achieve a reasonable effective throughput as a fraction of R. The effective throughput is defined as

$$R_T = \frac{\text{correctly transmitted data (excluding retransmission)}}{t}$$

In the code provided, the link rate is interpreted as the reciprocal of transmission delay. The socket delays $\frac{len(data)}{R}s$ before actually transmitting the data. All the packets are sent to a **network** which is a server that simulates the behaviour of Router 1 and Router 2.

1.4 Testing

Build a testing environment to simulate all the potential situations and test your RDT implementation against it to verify the required functionalities.

2 Submission and Presentation

2.1 Submission

rdt.py (and possibly some *utils*) in a *.zip* file. Your RDTSocket will be tested in the scenarios listed below:

1. One connection, infinite buffer, unreliable link.
2. One connection, finite buffer, reliable link.
3. Two connections as described in **Section 1.3**, finite buffer, reliable link.
4. Two connections as described in **Section 1.3**, finite buffer, unreliable link. Slightly more complex behavior on the hosts.

and graded according to its performance (R_T). The application runs on the connections is just the **echo server**.

2.2 Presentation

For the final presentation,

1. Add a *debug* mode to the RDT socket to print out information about the transmission process.
2. Introduce the mechanism you use for RDT and congestion control.
3. Explain how you test and evaluate the performance of your implementation.
4. Explain what are the factors deciding throughput efficiency in the situation in **Section 1.3**, assuming the two connections are symmetric.

you'll possibly need to make a video of about 5∼8 minutes containing the contents described above.

3 Grading

1. Pass the tests (15+20+25+30);

(case 1) (15): correctly transmit data to server and back.
(case 2) (15+5): correctly transmit data and achieve a reasonable R_T for the single connection.
(case 3) (15+10): correctly transmit data and achieve a reasonable R_T for both connections.
(case 4) (15+15): correctly transmit data and achieve a reasonable R_T for both connections.

2. Presentation (30).

4 Notes

- Don't use any existing function that does the work of RDT for you.
- Data sent when testing will be something like a long text.
- Code provided
    1. *USocket.py*: provides sockets with simulated rate;
    2. *rdt.py*: starter code of your RDTSocket;
    3. *network.py*: modify it to do testing.

  You only need to submit *rdt.py*. Make sure it works correctly with the provided *USocket* and *network*.
- Suggested steps:
    1. Assume reliable link and infinite buffer, complete basic connection management.
    2. Turn the link to unreliable, and treat all the exceptions as packet loss.
    3. Differentiate the causes, try to avoid unnecessary retransmission.
    4. Assume finite buffer, implement congestion control.
- For testing, we make the following assumptions:
    1. loss rate $\leq 10\%$;
    2. corruption happens independently across bits with a probability of 1×10^{-5};
    3. delay $|t_d| \sim \mathcal{N}(0, \sigma^2)$ for some $\sigma \propto \sqrt{RTT}$.