Programming Assignment 2
Implementing a Reliable Data Transport Protocol

1 Objectives

Since you've learned about the socket interface and how it is used by an application; by now, you're pretty much an expert in how to use the socket interface over a reliable transport layer, so now seems like a good time to implement your own socket layer and reliable transport layer! You'll get to learn how the socket interface is implemented by the kernel and how a reliable transport protocol like TCP runs on top of an unreliable delivery mechanism (which is the real world, since in real world networks nothing is reliable). This lab should be fun since your implementation will differ very little from what would be required in a real-world situation.

The network communication in last assignment was provided through a reliable transfer protocol (TCP/IP). In this assignment, you are required to implement a reliable transfer service on top of the UDP/IP protocol. In other words, you need to implement a service that guarantees the arrival of datagrams in the correct order on top of the UDP/IP protocol, along with congestion control.

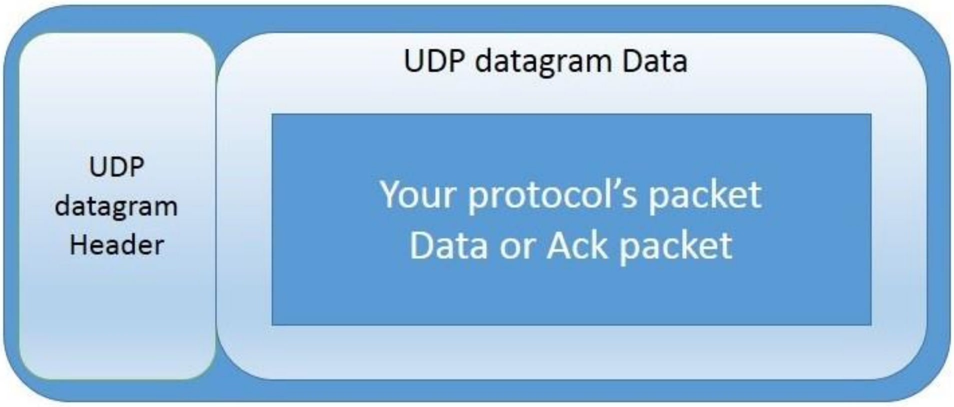
2 Reliability Specifications

2.1 Specifications

Suppose you've a file and you want to send this file from one side to the other (server to client). You will need to split the file into chunks of data of fixed length and add the data of one chunk to a UDP datagram packet in the data field of the packet. You need to implement TCP with congestion control.

2.2 Packet types and fields

There are two kinds of packets: Data packets and Ack-only packets. You can tell the type of a packet by its length. Ack packets are 8 bytes, while Data packets vary from 8 to 512 bytes (This is just an example but you're free to choose the header/data size for the packets).




```
        /* Data-only packets */
    struct packet {
        /* Header */
        uint16_t cksum; /* Optional bonus part */
        uint16_t len;
        uint32_t seqno;
        /* Data */
        char data[500]; /* Not always 500 bytes, can be less */
    };
```

```
    /* Ack-only packets are only 8 bytes */
    struct ack_packet {
        uint16_t cksum; /* Optional bonus part */
        uint16_t len;
        uint32_t ackno;
    };
```

3 Congestion Control

Congestion is informally: "too many sources sending too much data too fast for the network to handle". There are two main indicators for congestion: lost packets (buffer overflow at routers) long delays (queuing in router buffers). Not handling network congestion may lead to unneeded retransmissions (link carries multiple copies of a packet, since when a packet is dropped, any upstream transmission capacity used for that packet is wasted)! To Implement congestion control, follow the FSM of TCP congestion control given below.

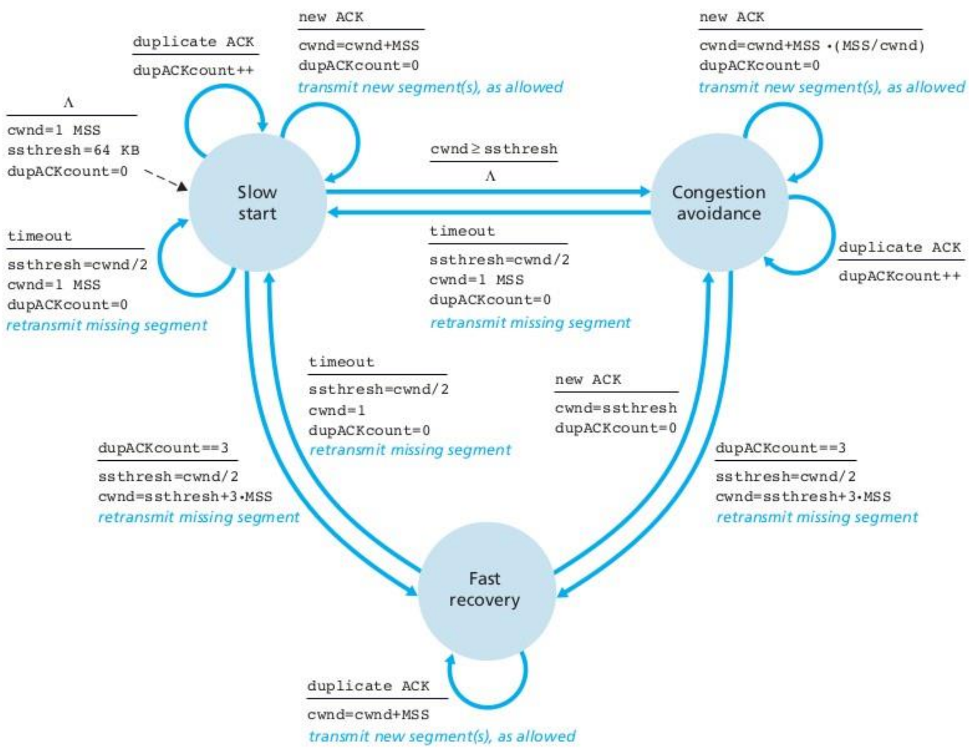


Figure 2: FSM description of TCP congestion control (http://kuroseross.com/)

4 Packet Loss Simulation

Your implementation will run in a simulated environment, this means that we will simulate the packet loss probability (PLP) since packet loss is infrequent in a localhost or LAN environment.

4.1 Specifications

PLP ranges from 0 to 1. For PLP=0 your implementation should behave with no packet loss. For non-zero values of PLP, you should simulate packet loss (by dropping datagrams given as parameters to the send() method) with the corresponding probability - i.e. a datagram given as a parameter to the send() method is transmitted only 100*(1 - PLP)% of the time (A value of 0.1 would mean that one in ten packets (on average) are lost).

5 Work flow between server and client

5.1 Flow of data

The main steps are:

1. The client sends a datagram to the server to get a file giving the its filename. This send needs to be backed up by a timeout in case the datagram is lost.
2. The server forks off a child process to handle the client.
3. The server (child) creates a UDP socket to handle file transfer to the client.
4. Server sends its first datagram, the server uses some random number generator random() function to decide with probability p if the datagram would be passed to the method send() or just ignore sending it
5. Whenever a datagram arrives, an ACK is sent out by the client to the server.
6. If you choose to discard the package and not to send it from the server the timer will expire at the server waiting for the ACK that it will never come from the client (since the packet wasn't sent to it) and the packet will be resent again from the server.
7. Update the window, and make sure to order the datagrams at the client side.
8. repeat those steps till the whole file is sent and no other datagrams remains.
9. close the connection.

5.2 Handling time-out

You're supposed to use a timer(one timer per datagram) at the server side to handle time-out events.

5.3 Arguments for the client

The client is to be provided with an input file client.in from which it reads the following information, in the order shown, one item per line :

IP address of server.
Well-known port number of server.
Filename to be transferred (should be a large file).

5.4 Arguments for the server

You should provide the server with an input file server.in from which it reads the following information, in the order shown, one item per line :

Well-known port number for server.
Random generator seed value.
Probability p of datagram loss (real number in the range [0.0 , 1.0]

6 Network system analysis

You should provide a comparison between stop-and-wait and selective repeat strategies in terms of throughput, based on a series of transfers of large files (e.g.: 4 MBytes, or higher). Your test runs should be performed with at least the following PLP values: 1%, 5%, 10% and 30%. For each PLP value, you should report the average of at least 5 (consecutive) runs. For the congestion control, you should print a graph showing like the one in the lecture's slides.

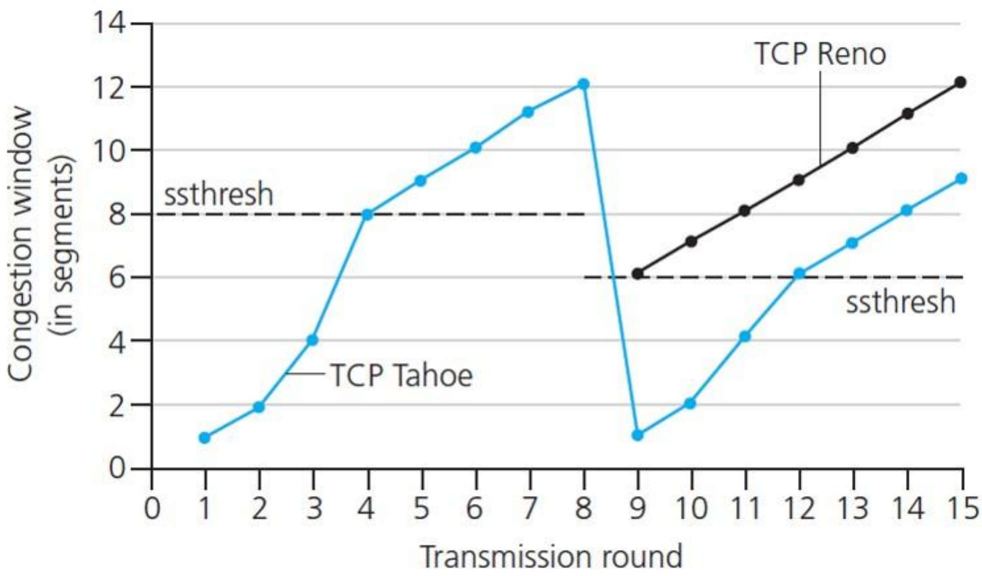


Figure 3: Congestion-control graph.

7 Bonus

- Error detection and checksumming - You will need some kind of checksumming to detect errors. The Internet checksum is a good candidate here. Remember that no checksumming can detect all errors but your checksum should have sufficient number of bits (e.g.

  16 bit in Internet checksum) to make undetected errors very rare. We cannot guarantee completely error-free delivery because of checksumming's limitation. But you should be convinced that this should happen very rarely with a good checksumming technique. Furthermore, you will need to simulate a channel will errors with some probability, just as done for loss, where you corrupt the data of a given packet before sending it.

8 Notes

- You must provide a suitable way for observing the loss of packets, time-outs and resending packets.
- Don't limit yourself with the problem statement, you've the reference, the lecture slides and the mighty google, those protocols are working in the real-world not theoretical proposals, if you think you've a better/more right implementation for what is required please do it.
- START SIMPLE. Set the probabilities of loss and corruption to zero and test out your routines. Better yet, design and implement your procedures for the case of no loss and no corruption, and get them working first. Then handle the case of one of these probabilities being non-zero, and then finally both being non-zero.
- You must understand thoroughly the main difference between UDP and TCP, so you can provide the functionalities of TCP-like protocol using UDP, from the main differences is the ordering of packets, reliability and Congestion control.

9 Policy

- You should develop your application in C/C++ programming language.
- You should work individually.
- You are required to submit external documentation for your program and to comment your code thoroughly and clearly. Your documentation should describe the overall organization of your programs, the major functions and data structures.