Project 3. Developing a Reliable Data Transfer Protocol

Due: 11/18/2022

Goal

The IP protocol enables addressing, packet routing and forwarding in the network as a best-effort service, which does not guarantee the arrival of a packet at the destination. In other words, packets may get lost, delayed, out of order, and corrupted. TCP provides reliable data transfer to ensure packet delivery as an end-to-end service. In this project, you will use datagrams in UDP to send and receive data (with the possibility of packet loss) and develop your own TCP-like protocol to provide reliable data transfer. You will develop a simple file transfer application that uses your own reliable transfer protocol, not socket system calls, for file transfer. You can use Java, C++, C, or Python; if you want to use a language other than these, consult the instructor.

Turnin

Submit your program through myCourses with readme that contains how to compile and execute.

Part 1. Reliable Data Transfer

TCP does not attempt to correct the bits of missed or corrupted packets at the receiver, but retransmits those packets when the sender detects that something goes wrong with those packets. The TCP sender infers those errors based on acknowledgments from the receiver and timeout (TO). To cope with packet loss and delay, you need to implement a mechanism that uses acknowledgments, retransmission, and TO like TCP. As noted, the sender transmits packets to the receiver via UDP. You also need to implement a data integrity checking mechanism using something similar to the TCP checksum. In addition, each TCP packet must have a sequence number that can be used to reorder arrived packets at the receiver. Define the header format of your own TCP protocol, and prepend this header to each datagram generated from UDP.

Your TCP will support a congestion control mechanism that works like TCP-Tahoe. Specifically, the mechanism has two phases: 1) slow start and 2) congestion avoidance, and uses sliding window-based control. This means that the sender can send a window of data at each RTT starting from one packet, and increase or decrease in response to network congestion conditions. The sender also implements fast retransmission for faster response to suspected packet losses. Of course, there should be TO as well, so that the sender can respond quickly in case there are not many packets in transit to trigger fast retransmission. The sender should not send more than what the receiver can handle at a time (flow control).

Part 2. File Transfer Application

Write your own file transfer program that uses the reliable data transfer protocol that you develop in Part 1. The protocol can read and write files from/to a remote server, but cannot list directories or provide user authentication. Any transfer begins with a request to read or write a file, which also serves to request a connection. The file is sent in fixed length blocks of 512 bytes, and each data packet (one block) is acknowledged by an ack packet before the next packet can be sent. A data packet less than 512 bytes indicates termination of a transfer. One typical initiation for write a file is following:

```
Host A sends a WRQ to host B with source = A's TID, dest = 69.
Host B sends an ACK (with block number = 0) to host A with source = B's TID
and destination = A's TID.
```

where WRQ means write request and TID is the end host's ID also used as a port.

```
myftp> ?
Commands may be abbreviated.  Commands are:

connect          connect to remote myftp
put              send file
get              receive file
quit             exit tftp
?                print help information
```

Note: If any function not specified in this writeup, it's at your own discretion. That is, as long as your program does what is required here, feel free to create and use whatever utility functions as needed.