

## Programming Assignment 5 — Simple Transport Protocol

*Assigned: Mar. 28**Due: Apr. 8, 23:59:59.*

## 1 Introduction

You will design a simple transport protocol that provides reliable datagram service. Your protocol will be responsible for ensuring data is delivered in order, without duplicates or errors. Since the local area networks in the university are far too reliable, we will provide a header file (`emulation.h`) and a source file (`emulation.c`) for emulating an unreliable network. To use them, you have to include the header file and compile with the source file. There is a macro defined in `emulation.h` to replace all calls of `sendto` with `emulate_sendto` which can be configured to drop, damage, duplicate, or delay packets on demand. If the packet is not dropped, then it is sent to (using UDP) the destination node as if `sendto` is invoked. Note that the `emulate_sendto` may corrupt a packet and then send it, may send multiple copies of the same packet, or may delay for an amount of time and then send it. While debugging, you can select the behavior of the function, and use it to test parts of your code. However, your selection will be ignored in grading, and the function will randomly drop, damage, duplicate, or delay packets.

For the assignment, you will transfer a named file reliably from between two nodes (a sender and a receiver). You do NOT have to implement connection open/close etc. You may assume that the receiver is run first and will wait indefinitely, and the sender can just send a named file to the receiver.

## 2 Emulation Setup

Both of your sender and receiver programs have to include the header file `emulation.h` and use only `sendto()` function to send all packets. Including `emulation.h` will replace all instances of `sendto()` with `emulate_sendto()`, which is implemented in `emulation.c`. You may configure the emulated network conditions by calling the following function:

```
void emulate_config(char drop, char damage, char duplicate,
                  char delay, int frequency);
```

- drop/damage/duplicate/delay flags: If you do not want `emulate_sendto()` to drop, damage, duplicate, or delay any packet, set the corresponding flags to 0; otherwise, set them to a nonzero value. When the delay flag is on, packets may be deferred for 200 milliseconds – 1 second at random.
- frequency: Indicate how often your packets are affected by `emulate_sendto()`. The content of this field is used as the denominator of the treatment probability. If frequency is set to 0, no packets will be harmed.
- Examples:

- Suppose frequency is set to 10 and the drop flag is on. Then each of your packets may be lost with probability  $\frac{1}{10}$ .
  - Suppose all flags are on, and frequency is set to 4. Then there is a one quarter chance for each of your packets to be lost, corrupted, duplicated, or delayed.
  - Suppose the damage flag is on and frequency is set to 1. Then each of your packets will certainly be corrupted.
  - Suppose frequency is set to 0. The flags won't matter and your packet will not be affected. (Use for testing).
- If this function is never called, all flags are on and the frequency is set to 10 by default.

You may modify emulation.c and emulation.h for testing, but make sure your submitted program works with the given emulation code. The emulation code you submit will be ignored.

### 3 Packet Format

You have to design your own packet format and use UDP as a carrier to transmit packets. Your packet might include fields for packet type, acknowledgement number, advertised window, data, etc. This part of the assignment is entirely up to you. Your code MUST:

- Transfer the file name reliably.
- Transfer the file contents reliably.
- The receiver must write the contents it receives onto stable storage (disk) with the appropriate file name.
- Your sender and receiver must gracefully exit.
- Your code must be able to transfer a file with any number of the drop, damage, duplicate, and delay flags set as long as the frequency is not monotonously set to 1.

You may implement any reliability algorithm you choose. However, a more sophisticated algorithm will be given higher credit. For example, some desired properties include (but are not limited to):

- Fast: Require little time to transfer a file.
- Low overhead: Require low data volume to be exchanged over the network, including data bytes, headers, retransmissions, acknowledgements, etc.

### 4 Executables

The command line syntax and output for the sender executable is as follows:

```
sendfile -r recv_host:recv_port -f name_of_file_to_be_transmitted
```

- When a sender sends a packet (including retransmission), it should print the following:

```
[send data] start (length)
```

where start is the beginning offset of the file sent in the packet, and length is the amount of the file sent in that packet.

- You may also print some messages of your own to indicate receiving acknowledgement, time-out, etc, depending on your design, but make it concise and readable.

The command line and output for the receiver is similar:

```
recvfile -r recv_port
```

- When the receiver receives a valid data packet, it should print:  
[recv data] start (length) status  
where status is one of ACCEPTED(in-order), ACCEPTED(out-of-order), or IGNORED.
- Similar to sendfile, you may add your own output messages.
- The receiver should store a file with name “x” as “x.recv”. (This will allow you to use the same directory for both the receiver and the sender).

Both the sender and the receiver should print out a message after completion of file transfer:  
[completed]

## 5 Submission

- Please submit your code as described in the forum.
- What to turn in: All of your source code along with a Makefile and a README text file. Your Makefile must produce two executables named `sendfile` and `recvfile`. The README file should include a description and explanation of the packet formats, protocols, and algorithms you use, and a list of properties/features of your design that you think is good.