

Programming Assignment 3 — Simple Transport Protocol

Assigned: Oct. 17

Due: Oct. 28, 11:59PM.

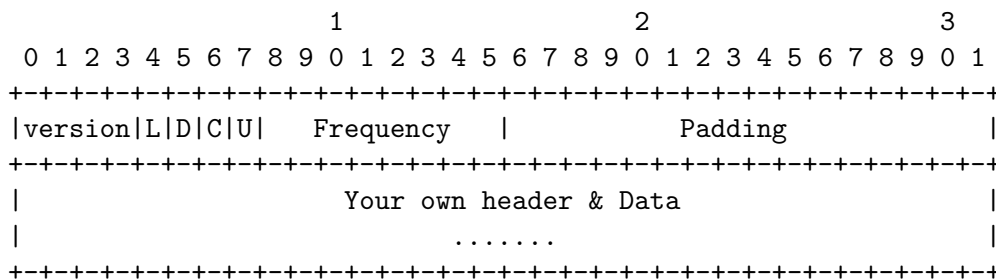
### 1 Introduction

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 an object file (`emulation.o`) for emulating an unreliable network. To use them, you have to include the header file and then link with the object file. There is a macro defined in `emulation.h` to replace all call of `sendto` with `emulate_sendto` which can be configured to drop, damage, or duplicate 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, or may send multiple copies of the same packet. 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, duplicate and forward 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 Packet Format

To make use of the emulation and specify the behavior of the `emulate_sendto` function, the (UDP) data part of each packet will contain the following header:



Version: Always set to 0011

Disposition: The packet disposition consists of four bits (three used) and tells the function how to handle this packet. The bits are:

- L: Loss bit
- D: Duplication bit
- C: Corruption bit

- U: Unused (set to 0)

Any subset of the L, D, and C bits may be set. If set, then the function will act on the packet in the specified manner at some probability. NOTE: these flags will be ignored when your code is graded. You may only use them for debugging.

Frequency: This field controls how often your packets are affected by the emulation server. The contents of this field are treated as an unsigned char and is used as the denominator of the treatment probability.

If Frequency is set to 0, the packet is not harmed.

Examples:

- Suppose Frequency is set to 10 and the L bit is set. Then there is a  $\frac{1}{10}$  chance that your packet will be lost.
- Suppose the L, C and D bits are set, and Frequency is set to 4. Then there is a one quarter chance that your packet may be lost, corrupted or duplicated.
- Suppose the C bit is set and Frequency is set to 1. Then your packet will certainly be corrupted.
- Suppose Frequency is set to 0. The disposition bits won't matter and your packet will not be lost, corrupted, duplicated, or otherwise maligned by the emulation server. (Use for testing).

Padding: Set all 16 bits to zero

After our header, you will almost certainly want to add your own header. Your header 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 L, D, and C bits set as long as the Frequency is not monotonously set to 1.

You may implement any reliability algorithm and congestion control algorithm you choose. However, a more sophisticated algorithm will be given higher credit.

### 3 Executables

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

```
sendfile [-v] -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] segment_id: start (length)
```

where `segment_id` is a local monotonically increasing integer (increase it even in case of retransmission), `start` is the beginning offset of the file sent in the packet, and `length` is the amount of the file sent in that packet.

- When the sender receives a valid ACK packet, it should print:

```
[recv ack]
```

- If the sender receives a corrupt packet, it should print:

```
[recv corrupt packet]
```

- The sender should log every timeout and the segment on whose behalf the sender timed out

```
[timeout] segment_id
```

- If the `-v` option is specified, the sender should log all information about un-acknowledged segments whenever it receives an ack.

```
[holes] start_1 (length_1), start_2 (length_2), ..., start_k (length_k)
```

The command line and output for the receiver is similar:

```
recvfile [-v] -r recv_host: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`.

- When the receiver sends an ack, it should log:

```
[send ack]
```

- If a corrupted packet arrives, the receiver should log:

```
[recv corrupt packet]
```

- If the `-v` option is specified, the receiver should log information about “holes” whenever it receives a segment from the sender using the following format:

```
[holes] start_1 (length_1), start_2 (length_2), ..., start_k (length_k)
```

- 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]
```

## 4 Project Submission

Please submit your code to the Submit Server (<https://submit.cs.umd.edu/>). You should upload a zip file which contains (at least) the following files:

- `sendfile.c` OR `sendfile.cpp`
- `recvfile.c` OR `recvfile.cpp`
- `Makefile`
- `README` - description of the protocol, reliability algorithms and congestion control algorithm (if any)