

# Lab Assignment 2

Start Assignment

---

**Due** Nov 17 by 11:59pm      **Points** 50      **Submitting** a file upload  
**Available** after Oct 17 at 12am

---

## CS640 Programming Assignment 2: Network Emulator and Reliable Transfer

**Due: Nov 17, 2023**

### 1. Overview

In this project you will implement a network emulator and add reliable transfer to your distributed file transfer in the previous assignment. As with the first programming assignment, you can work in teams and write your code in Python. **Please read on carefully.**

#### 1.1. Network Emulator

For this programming assignment you will create a network emulator, which delivers packets between sender(s) and requester(s) you created for the first programming assignment. Your senders and requesters will have additional requirements to support the network emulator.

The network emulator will receive a packet, decide where it is to be forwarded, and, based on the packet priority level, queue it for sending. Upon sending, you will delay the packet to simulate link bandwidth, and randomly drop packets to simulate a lossy link.

We also ask you to implement packet priority queues, a common feature of many packet queueing algorithms. There will be three packet priority levels, and there will be a separate sending queue for each priority level. Each queue will have a fixed size. If the outbound queue for a particular priority level is full, the packet will be dropped. Higher priority packets are **always** forwarded before lower priority packets.

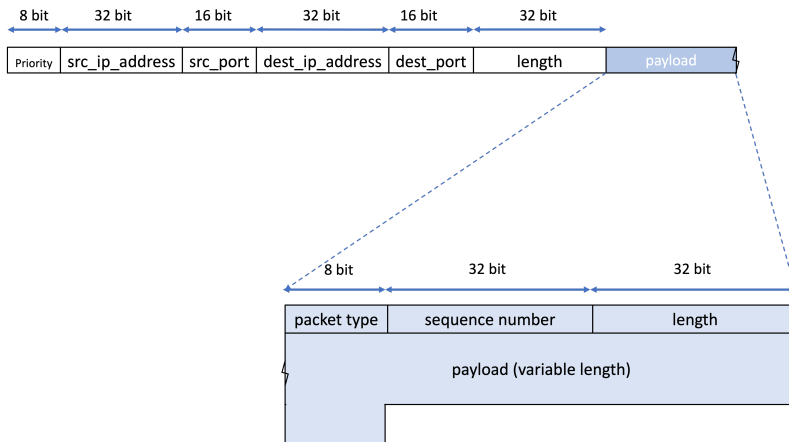
#### 1.2. Reliable Transfer

To achieve the reliable transfer, the requester will advertise a window size (see the requester specification of this write up for more info) to the sender with the request packet. The sender will send a full "window" of packets and wait for ACKs of each packet before sending more packets. After a certain timeout, the sender will retransmit the packets that it has not received an ack for.

## 2. Details + Requirements

### 2.1. Forwarding Encapsulation / Packet structure

In order to implement priority levels and forwarding, you will encapsulate the packet type from the first programming assignment inside a new packet.



Essentially, you are adding an 8-bit priority, a 32-bit source IP address, a 16-bit source port, a 32-bit destination IP address, a 16-bit destination port and a 32-bit length to the front of the packet layout from “programming assignment #1”. Compare this with how UDP datagrams are encapsulated inside IP datagrams (which are then encapsulated in a layer 2 protocol, such as Ethernet). The length field of the outer packet is set to the total size of the inner packet, i.e. inner packet header size + inner packet payload size.

Note that:

- Valid values for priority levels are:
  1. 0x01 - highest priority
  2. 0x02 - medium priority
  3. 0x03 - lowest priority
- For the *ack* packet, the packet type will be A (capital a) and the sequence field will contain the sequence number of the packet that is being acknowledged.
- All the packets sent by the requester should have priority 1.
- The priority of the END packet is the same as the other packets in the flow.

### 2.2. Logical Functions of Emulator

The logical functions of the emulator consist of *routing*, *queueing*, *sending*, and *logging*. Each sub-function is detailed below.

**Static forwarding table:**

The logical functions depend on the static forwarding table provided to the emulators through a file. The file contains lines in the format below, with space as delimiter between the various fields:

<emulator> <destination> <nexthop> <delay> <loss probability>

- **emulator:** a "<Host name> <Port>" pair that identifies the emulator for which the current entry is associated with. Multiple emulators may be specified in a single table, and so they must filter only the lines that apply to them,
- **destination:** a "<Host name> <Port>" pair that identifies the final destination of the packet,
- **next hop:** a "<Host name> <Port>" pair that identifies the next entity to forward the packet to. It can be an emulator, sender or requester.
- **delay:** in milliseconds, and identifies how long the emulator will delay before sending a packet to the corresponding destination.
- **loss probability:** in percentage, and identifies the probability that the emulator will drop a packet when sending packets to the corresponding destination.

Here is an example [table \(https://pages.cs.wisc.edu/~pb/640\\_fl12/PA2/table\)](https://pages.cs.wisc.edu/~pb/640_fl12/PA2/table).

### Routing function:

The routing function is based on the static forwarding table that you provide to your program through the file described above. The destination of an incoming packet is compared with the destination in the forwarding table to find a match. If a match is found, the packet is queued for forwarding to the next hop. If a match is not found, the packet is dropped and the event should be logged (see logging function below).

The emulator reads this file once it starts running and then only refers to its version of the file in memory for every packet. The emulator ignores lines in the table that do not correspond to its own hostname and port. Note that emulator, sender, and requester are all uniquely identified with a "<Host name, Port>" pair and thus multiple of them can run on the same host.

### Queueing function:

The queueing function should examine the priority field on the packet and place the packet in an appropriate queue. All the three queues are of fixed size. This queue size is specified on the command line of the emulator startup. If a queue is full, the packet is dropped and this event is logged (see logging function below).

### Send function:

The send function accepts packets from the three queues defined above and simulates network link conditions *for each destination*. Packets bound for a destination are first delayed to simulate link bandwidth. The delay is defined in the forwarding table and is specified in milliseconds. After a packet has been delayed, it may be dropped to simulate a lossy link based on the loss probability provided in

the forwarding table, and the event is logged (see logging function below). If a packet is not dropped, it is then sent to the network.

### Logging function:

The **logging function** is integral to all functions of the emulator. A packet may be dropped in the emulator in the routing function, the queueing function, or in the send function. Any and all packet drop events must be logged to a file. Loss events must provide a textual reason for the loss (e.g., "no forwarding entry found", "priority queue 1 was full", "loss event occurred.") Each log event must include the source hostname and port, the intended destination host name and port, the time of loss (to millisecond resolution), the priority level of the packet, and the size of the payload.

## 2.3. Forwarding Summary

The order of processing should be similar to the following. Your emulator can simply follow the steps below in an infinite loop and no threading is required for this assignment. Note that logging is not an explicit part of this sequence.

1. Receive packet from network in a non-blocking way. This means that you should not wait/get blocked in the `recvfrom` function until you get a packet. Check if you have received a packet; If not jump to 4,
2. Once you receive a packet, decide whether packet is to be forwarded by consulting the forwarding table,
3. Queue packet according to packet priority level if the queue is not full,
4. If a packet is currently being delayed and the delay has not expired, goto Step 1.
5. If no packet is currently being delayed, select the packet at the front of the queue with highest priority, remove that packet from the queue and delay it,
6. When the delay expires, randomly determine whether to drop the packet,
7. Otherwise, send the packet to the proper next hop.
8. Goto Step 1.

## 2.4. Reliable transfer

The procedure is as follows:

- Upon receipt of a request packet, the sender sends a full window of packets at the rate specified by the user.
- The sender keeps this set of data in a buffer, and keeps a timeout for each of the packets. If it does not receive an ack for a packet and its timeout expires, it will retransmit that packet. The timeout is fixed and is specified by one of the sender's parameters.
- If an ack packet is not received after re-transmitting the packet 5 times, the sender will print an error stating that it gave up on the packet with that specific sequence number, and continue with the next packet.
- Once all packets of that window have been acked the sender sends another window of packets.

- The requester should have a buffer and make sure that it saves the data to the file in the order of the packets' sequence numbers. It should also make sure that it does not print duplicate packets into the file.
- The requester acks every packet that it receives, even if it has already written that packet to the file (may happen if the sender retransmitted a packet due to its timeout, but the original packet actually made it to the requester).

## 2.5. Specification

### Emulator:

The network emulator should be invoked in the following way:

```
python3 emulator.py -p <port> -q <queue_size> -f <filename> -l <log>
```

- **port:** the port of the emulator.
- **queue\_size:** the size of each of the three queues.
- **filename:** the name of the file containing the static forwarding table in the format specified above.
- **log:** the name of the log file.

The network emulator must implement the routing, queueing, sending, and logging logical functions described above. Your program must be able to support forwarding packets through one or multiple emulators (ex. sender to emulator1 to emulator2 to requester). If you have implemented your emulator correctly, this is automatically satisfied.

Note that your emulator should **NOT** drop END packets. This is because testing is made harder when END packets get dropped.

### Sender:

Note that the following requirements for your sender and requester are in addition to requirements stated for programming assignment 1.

You will have to modify your sender to be invoked as follows:

```
python3 sender.py -p <port> -g <requester port> -r <rate> -q <seq_no> -l <length> -f <f_hostname> -e  
<f_port> -i <priority> -t <timeout>
```

- **f\_hostname:** the host name of the emulator.
- **f\_port:** the port of the emulator.
- **priority:** the priority of the sent packets.
- **timeout:** the timeout for retransmission for lost packets in the unit of milliseconds.

The behavior of the sender should be **modified** to:

1. Always start at sequence number 1
2. Increment the sequence number by 1 for each packet sent, instead of by the packet length

3. Print out the observed percentage of packets lost. The loss rate that the sender prints out is not necessarily the same as the loss rate that we identify in the forwarding table since the sender might miss some ACKs. This loss rate is computed by (number of retransmissions / total number of transmissions), where total number of transmissions including both normal transmissions and retransmissions.
4. The end packet is sent after ensuring that all data packets have been received by the receiver (or if max number of retries have reached for sending all packets in the last window).

### Requester:

You will have to modify your requester to be invoked as follows:

```
python3 requester.py -p <port> -o <file option> -f <f_hostname> -e <f_port> -w <window>
```

- **f\_hostname**: the host name of the emulator.
- **f\_port**: the port of the emulator.
- **window**: the requester's window size.

You will also have to **modify** your requester to:

1. The **inner** length field of the request packet will be filled with this window size so that the sender can extract and use this value for sending.
2. Verify that the destination IP address in its received packet (data packets or end packets) is indeed its own IP address, and
3. Suppress display of individual DATA packet information.

## 3. Submission

The python program names must be "sender.py", "requester.py" and "emulator.py". You may also submit a readme.txt file and partner.txt if you work with a partner. partner.txt should contain both partners names as they appear on canvas and your wisc emails.

It is essential that you follow the required packet format and output specification to ensure that your code can be tested. To turn in your code, please submit a .zip file containing sender.py, requester.py, emulator.py, readme.txt and partner.txt file if applicable.

## 4. Examples

Below you can find examples with a description of expected outputs. You can use these examples to understand the requirements better, and test your code before submitting your code. We strongly encourage you to run these examples in order to ensure that you are submitting a complete project. Also, you can write your own testing scripts to validate your code for supporting other features, like the full queue failure, no valid entry in the routing table, change of the window size etc.

- **Example1**  ([https://github.com/Tingjia980311/cs640/blob/main/lab2\\_example1.txt](https://github.com/Tingjia980311/cs640/blob/main/lab2_example1.txt))

- [Example2](https://github.com/Tingjia980311/cs640/blob/main/lab2_example2.txt)  ([https://github.com/Tingjia980311/cs640/blob/main/lab2\\_example2.txt](https://github.com/Tingjia980311/cs640/blob/main/lab2_example2.txt))
- [Example3](https://github.com/Tingjia980311/cs640/blob/main/lab2_example3.txt)  ([https://github.com/Tingjia980311/cs640/blob/main/lab2\\_example3.txt](https://github.com/Tingjia980311/cs640/blob/main/lab2_example3.txt))
- [file.txt](https://pages.cs.wisc.edu/~pb/640_fl12/PA2/file.txt) ([https://pages.cs.wisc.edu/~pb/640\\_fl12/PA2/file.txt](https://pages.cs.wisc.edu/~pb/640_fl12/PA2/file.txt))

## 5. Grading

Grading will mostly be done by running your code through our set of test cases. To ensure you are graded properly, please adhere to the naming conventions detailed above. The test cases we will be tested are listed below, along with their weight. We encourage you to write your own testing procedures and perform your own tests, as we will not be publishing our testing scripts before the due date.

Firstly, we will check if your code can meet the following requirements for each test case:

1. Packets are sent with the new packet structure, and the requester sends ack packets.
2. Packets get forwarded from sender to the requester or inverse through correct emulators according to the forwarding table.
3. Loss probability can be specified at the network emulator; packets get dropped according to that rate.
4. Delay can be specified and works correctly.
5. Packets get sent based on the specified window and are retransmitted if lost (reliable transfer).
6. Emulator queues the packets correctly and packets get dropped if queues are full. Queue size is correctly configured.
7. The sender sends packets with a priority level; the emulator handles priorities correctly.
8. The emulator logs the information correctly.
9. The output file matches the input file.
10. The receivers can receive packets from different senders at the same time.

Test cases:

10 pts: Requesting a file that lives on a single sender through a single emulator.

10 pts: Requesting a file that lives on a single sender through multiple emulators.

10 pts: Requesting a file that is split among 2 different senders through a single emulator.

10 pts: Requesting a file that is split among 2 different senders through different emulators.

10 pts: Requesting a file that is split among 2 different senders, ensuring that packet delays are correctly implemented

10 pts: Requesting a file that is split among 2 different senders with a relatively small queue size in the emulator (expecting some packet drops, looking for correct emulator behavior and correct sender retransmission)

10 pts: Requesting a file that is split among 2 different senders, testing for correct window size implementation

10 pts: Requesting a file that is split among 2 different senders, testing for correct packet loss rate implementation in the emulator



10 pts: Requesting a file that is split among 2 different senders, testing for correct packet priority implementation

10 pts: Requesting a file that is split among 3 different senders through a single emulator.

10 pts: Correct logging information for above testing cases.

## 6. Grading

**Get started very early**, read this description carefully and ask questions. This project has many requirements. Starting early helps you in submitting a complete project. The advice from the first programming assignment to "start small" applies to this project as well.

You can refer to the document of [socket in python](https://docs.python.org/3/library/socket.html) , especially the `setblocking()` function to get knowledge about receiving packets in a non-blocking way. You can also refer to the [logging library](https://docs.python.org/3/howto/logging.html)  in python.

## Programming Assignment 2 FAQ

**Q1** Are we still using UDP, or should we use TCP for this project?

**A1** We are still using UDP, but we are trying to emulate TCP to some extent with our reliable transfer mechanism.

**Q2** There are 2 length fields in the packet. How should I set each of them?

**A2** The inner length field is the same as project 1, except that in case of a request packet it is set to the window size. The outer length field is the inner packet header size + inner packet payload size.

**Q3** How does the requester let the sender know of the window size?

**A3** The length field in the **inner packet** is set to the window size in the request packet.

**Q4** Is the delay value in milliseconds or seconds?

**A4** It is in milliseconds.

**Q5** What are the source and destination fields in the packet set to?



- A5** They are always set to either the address of the sender or the requester. We never set these fields to the address of the emulators.
- Q6** Does the emulator change any field in the packet? For example, does it set the packet source fields to itself?
- A6** No. The emulator does not change any field in the packet. It just forwards the packet untouched.
- Q7** What if the END packet is dropped?
- A7** You have to implement the emulator so that it does not drop the END packets.
- Q8** Does the end pkt belong to the last window or is it treated separately besides any window?
- A8** The end packet is sent after ensuring that all data packets have been received by the receiver (or if max number of retries have reached for sending all packets in the last window).
- Q9** What if the request packet is dropped?
- A9** When testing your code, we will make sure that the table is set so that none of the packets of the requester is dropped.
- Q10** Is the window size in bytes or packets?
- A10** The window size is based on the number of packets.
- Q11** Should the emulator attempt to detect circular routing?
- A11** No, the emulator will act only based on the table that it gets.
- Q12** If the sender always starts with sequence number 1, why is it invoked with the parameter -q ?
- A12** Just to be consistent with project 1. You should overlook this argument, so if someone accidentally invokes your code with 11 for parameter -q, your senders should still ensure the first packet has sequence number 1.
- Q13** Should the emulator append to the log file each time it is run, or overwrite its earlier contents (if there are any)?
- A13** Either way is fine.