

# Programming Assignment 4

<b>Goal</b>	<b>1</b>
<b>Description</b>	<b>1</b>
"Application" layer	2
Message send/receive application	2
File send/receive application	4
"Transport" layer	5
Message format	5
Protocol stop-and-wait	5
Protocol go-back-n	6
"Network" Layer	7
<b>Grading policy</b>	<b>7</b>
<b>Submission instruction</b>	<b>8</b>

## Goal

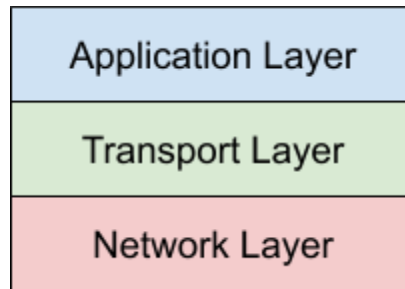
- Gain experience on how to transfer data reliably in an unreliable environment.
- Implement both Stop-And-Wait and Go-Back-N protocols.

## Description

We have studied a few reliable data transfer protocols/algorithms in class (stop-and-wait, go-back-n, selective-repeat), and understood the building blocks needed (sequence number, ack, checksum, timeout, etc.). In this assignment, you will put those techniques into practice.

In the Internet protocol stack (picture below shows application layer to network layer), we know their service models.

- Network layer (e.g. IP): provide unreliable best effort service
- Transport layer (e.g. TCP): provide reliable data transfer by using the unreliable service provided in the network layer.
- Application layer: make use of the reliable data transfer protocol provided in transport layer.



To draw connections to that, here in this assignment we will have three layers as well.

- “Network” layer: provide unreliable data transfer using the model we cover in class, where packets can be lost, corrupted. (Note: order is guaranteed to be the same)
  - Corresponding code is in `udt.py`
- “Transport” layer: provide implementations of reliable data transfer protocols by using the service provided by the unreliable data transfer layer.
  - Protocol stop-and-wait: `sw.py`
  - Protocol go-back-n: `gbn.py`
  - Note: `dummy.py` is an example of how to interact with the “network” layer and the “application” layer. It does not provide any reliable data transfer.
- “Application” layer: use the reliable data transfer layer, and verify it works
  - Application that sends a sequence of messages: `msg_sender.py` and `msg_receiver.py`
  - Application that sends a file: `file_sender.py` and `file_receiver.py`

Details for each layer is described in the following sections.

## “Application” layer

There are two “applications” designed to verify your implementation of the reliable data transfer protocols. You shouldn’t need to change any code here. Just use them to iterate and verify your solution.

### Message send/receive application

In this application, the sender will send 20 messages to the receiver using the “transport” layer protocol of choice. For the demo purpose, we will use the `dummy` protocol, where no reliability is implemented.

First start the receiver using the following command

```
python3 msg_receiver.py dummy
```

Then start the sender using the following command

```
python3 msg_sender.py dummy
```

Now you should see sender output the following information to console, which shows the 20 original messages that have been sent.

```
b'MSG:0'  
b'MSG:1'  
b'MSG:2'  
b'MSG:3'  
b'MSG:4'  
b'MSG:5'  
b'MSG:6'  
b'MSG:7'  
b'MSG:8'  
b'MSG:9'  
b'MSG:10'  
b'MSG:11'  
b'MSG:12'  
b'MSG:13'  
b'MSG:14'  
b'MSG:15'  
b'MSG:16'  
b'MSG:17'  
b'MSG:18'  
b'MSG:19'
```

Go back to the terminal where you started the message receiver, you should see something like the following printed on the console. This shows the messages received by the receiver, where you can clearly see some messages are corrupted, and there is message loss (e.g. message 7 is lost in this case).

```
b'MSG:0'  
b'MSG:\xce'  
b'MSG:2'  
b'MSG:3'  
b'M\xacG:4'  
b'MSG:5'  
b'MSG:6'  
b'MSG:8'  
b'MSG:9'  
b'MSG:\xce1'  
b'MSG:12'  
b'MSG:14'  
b'MS\x8b:15'  
b'MSG:16'  
b'MSG:\xce7'  
b'\xb2SG:18'  
b'MSG:19'
```

## File send/receive application

In this application, the file sender will read the specified file, and chop it into sequence of packets, and send them to the receiver using the “transport” layer protocol of choice. For the demo purpose, we will use the `dummy` protocol, where no reliability is implemented.

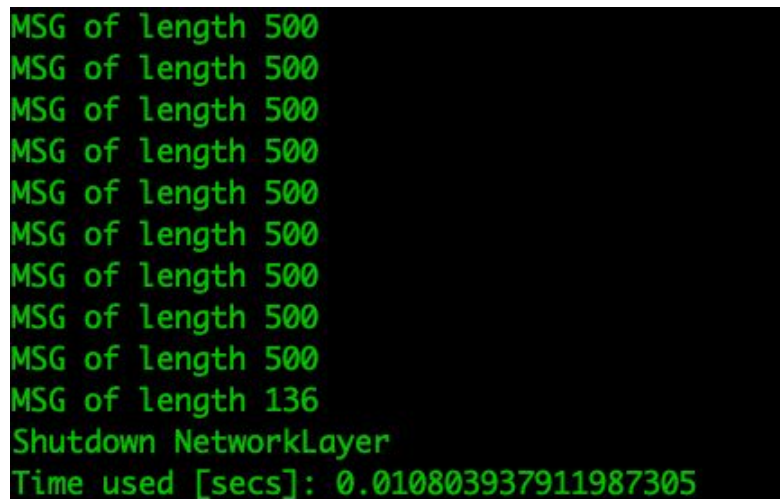
First start the receiver using the following command, where ‘cat.jpg’ just means you want to save the file received as cat.jpg in your current directory.

```
python3 file_receiver.py dummy cat.jpg
```

Then start the sender using the following command, where ‘grumpy\_cat.jpg’ is a file in the local directory that I want to send to the file receiver.

```
python3 file_sender.py dummy grumpy_cat.jpg
```

You should see something like below shown in the file sender’s console. It means, the file sender is dividing the file into sequence of chunks (500 bytes each), and send them sequentially.



```
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 500
MSG of length 136
Shutdown NetworkLayer
Time used [secs]: 0.010803937911987305
```

On the receiver side, you should see that cat.jpg is in your local directory. To make sure these the file received is identical to the file you sent, you need to compare the MD5 signature of these two files. To get the MD5 signature of a file, simply using the command below

```
python3 md5.py grumpy_cat.jpg
```

And you should see something like below shows the signature

```
digest: b'\xdc67\x14\x99BQ\x07U\xdbg|\xb1\xd1WN'
digest as hex: dc3637149942510755db677cb1d1574e
```

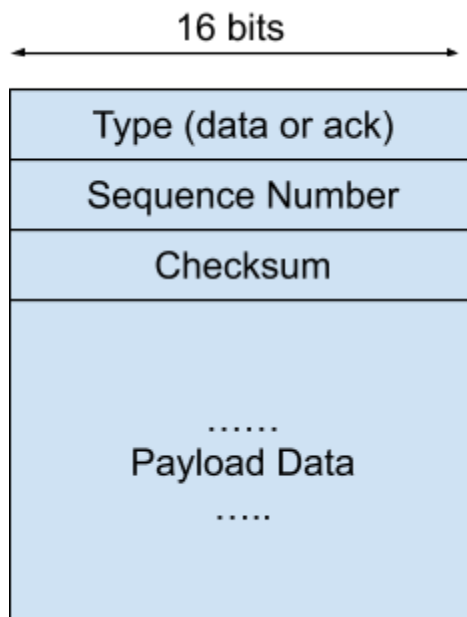
## “Transport” layer

This is the layer you need to implement the reliable data transfer protocol we developed in class (stop-and-wait and go-back-n). Details described in the sections below.

### Message format

We will use the same message format for both stop-and-wait and go-back-n. The header has the following 3 fields.

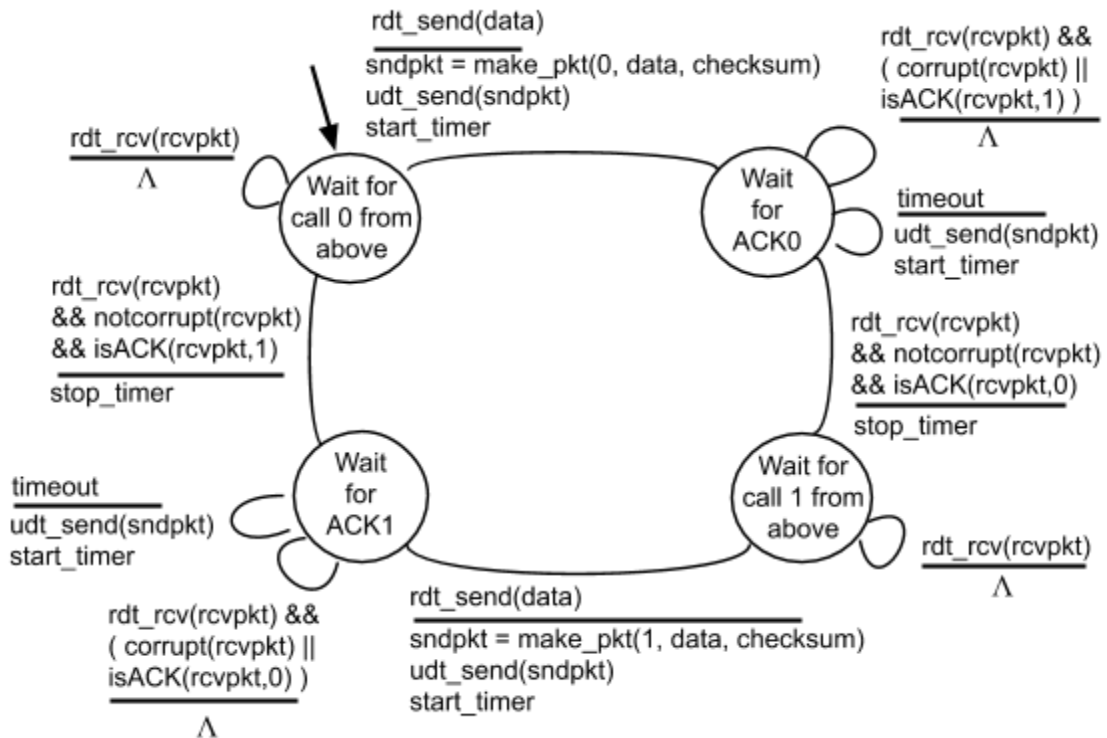
- **Type [16 bits]:** either MSG\_TYPE\_DATA or MSG\_TYPE\_ACK (defined in `config.py`)
- **Sequence Number [16 bits]:** an unsigned 16-bit integer. It can accommodate sequence number from 0 to 65535 range. It depends on your protocol what sequence number to use. For example, in the stop-and-wait protocol, you would only use sequence number 0 and 1.
- **Checksum [16 bits]:** checksum is computed by viewing the whole segment (including type and sequence number in headers) as a list of 16-bit integers.
  - Note: you don't need to worry about implementing this using 1's complement as used in the Internet checksum.
- **Payload:** this is the data sent from the application layer. In this assignment we restrict payload to be no more than 500 bytes per packet (MAX\_MESSAGE\_SIZE defined in `config.py`).



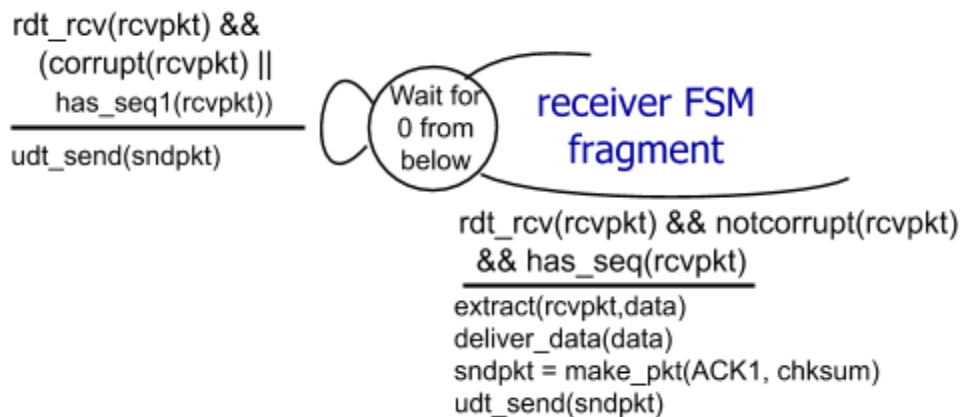
### Protocol stop-and-wait

Please implement this protocol in `sw.py`. Detailed FSM is below.

**Sender**



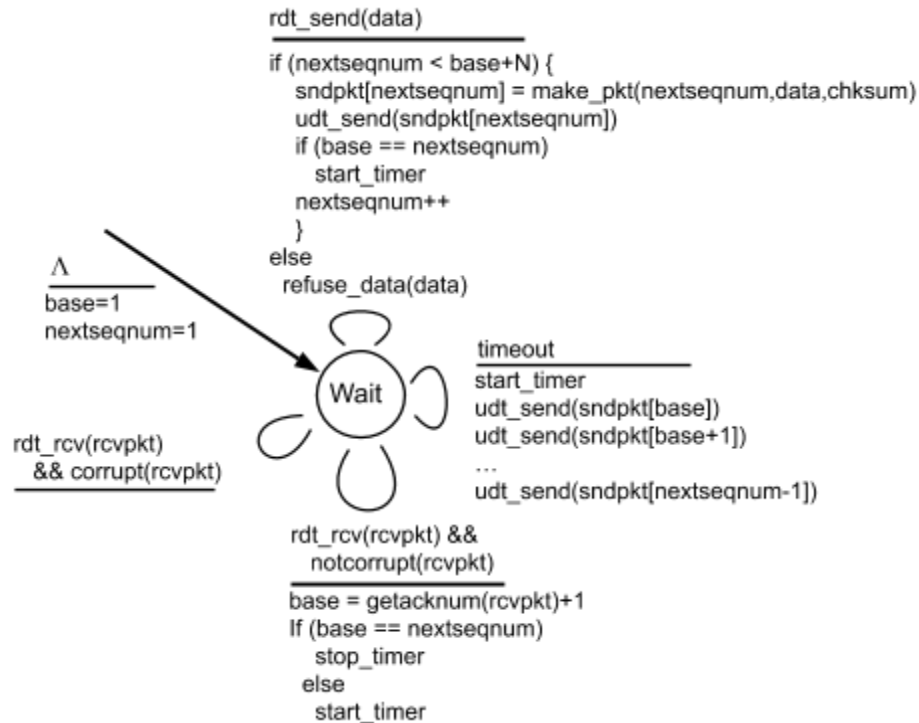
## Receiver



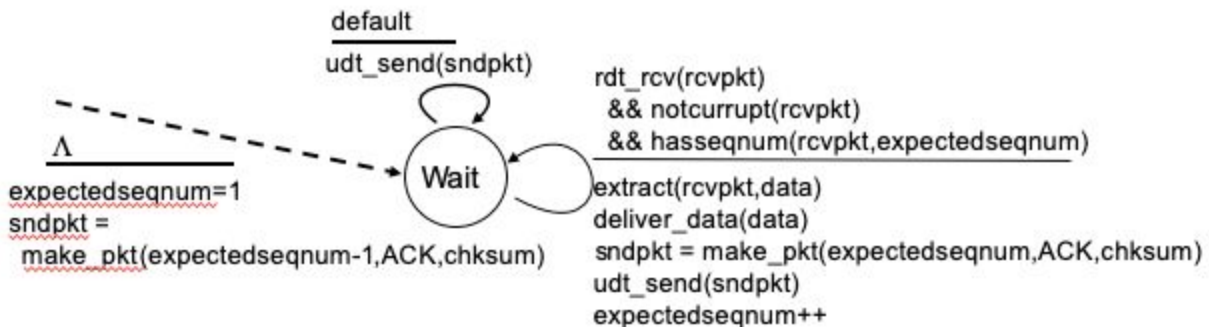
## Protocol go-back-n

Please implement this in `gbn.py`. Detail FSM is below.

## Sender



## Receiver



## “Network” Layer

The internet is “too reliable” for us to conveniently test our implementation. So we inject bit errors and packet loss directly in `udt.py`. You do not need to change any code here.

Both bit error rate and packet loss rate are controlled by the following parameters defined in `config.py`

- **BIT\_ERROR\_PROB**: probability to inject bit errors to the packet that is passed to the “network” layer.
- **MSG\_LOST\_PROB**: probability to lose the packet that is passed to the “network” layer.

## Grading policy

100 points in total.

- [50 pts] stop-and-wait (`sw.py`)
  - [20 pts] pass test using message sender/receiver.
  - [20 pts] pass test using file sender/receiver.
  - [10 pts] proper multithreading synchronization (the `send` and `handle_arrival_msg` function can be called by different threads concurrently, so you need to make your implementation thread safe).
- [50 pts] go-back-n (`gbn.py`)
  - [20 pts] pass test using message sender/receiver.
  - [20 pts] pass test using file sender/receiver.
  - [10 pts] proper multithreading synchronization (the `send` and `handle_arrival_msg` function can be called by different threads concurrently, so you need to make your implementation thread safe).
  - You need to use the **WINDOW\_SIZE** defined in `config.py` as your window size in go-back-n.

Note:

- TAs have the ability of changing parameters defined in `config.py` files at will. For example, change the **BIT\_ERROR\_PROB** to a different value. Your solution should just work without any assumptions on what those values are.
- TAs have the freedom to choose whatever files they want when using file sender/receiver. You can not have any assumption on the size of the files.

## Submission instruction

Please your code and report to all TAs and cc [z.sun@northeastern.edu](mailto:z.sun@northeastern.edu) by the deadline.