Project II: Implement the Network

1. Description

   In this project, you are required to implement several layers of the network. As you know, the whole network can be divided into five layers: application layer, transport layer, network layer, link layer and physical layer.

   In this project, we will provide a unreliable link layer and physical layer interface. You are required to implement the network layer, transport layer and then create an applet on the Application layer over the unreliable link layer emulator we provided.

2. Instruction

   This instruction assumes that you are using python3, pip-python3 on Linux.

   a) Data Link Layer

      Our link layer emulator runs on the UDP/IPv4. It's in a package called LinkLayer.

      You can install this package from pypi or github (latest)

      ```
      (Github)# pip install git+https://github.com/lightsing/Data-Link-Layer-Emulator
      (pypi)# pip install DataLinkLayerEmulator
      ```

      i. Frame Format(LinkLayer.util.frame)

      ```
      +-----------------+-----------------+---------+---------------+
      | src MAC Address | dst MAC Address | length  |    payload    |
      +-----------------+-----------------+-------------------------+
      |     6 bytes     |     6 bytes     | 4 bytes | 0 - 560 bytes |
      +-----------------+-----------------+---------+---------------+
      ```

      1. MAC address:

         6 bytes length. Example:02:00:0A:14:27:AA

         The default MAC address is generated from your IP address by 前插 perpend 0x0200 of your IPv4 address bytes. (LinkLayer.util.ip_mac)

      2. MTU: the maximum transmission unit is 560 bytes.

      ii. Config File

      You need to create a config file (config.json) before you use this library. If you don't know what's JSON, read this.

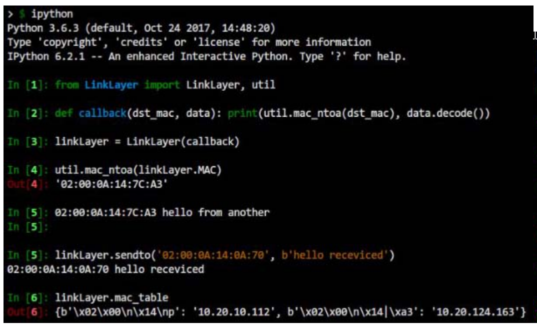
      Here's a sample config.

```
{
   "test": {
      "ip": "172.18.1.3",
      "port": 80
   },
   "port": 5000,
   "loss": 0.1,
   "devices": [
      "10.20.124.163"
   ]
}
```

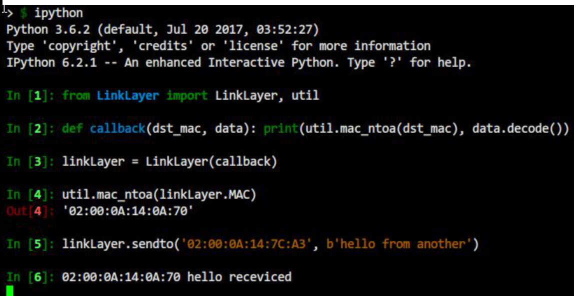
test part defines which server is used to autodetect your local IP address. Keep this in your file if your program works in the campus.

port defines which port to listen and to send frame.
loss defines the loss rate of the frame transmission.
devices defines the address of other nodes in the network.

iii. Getting Start



```
> $ ipython
Python 3.6.3 (default, Oct 24 2017, 14:48:20)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from LinkLayer import LinkLayer, util

In [2]: def callback(dst_mac, data): print(util.mac_ntoa(dst_mac), data.decode())

In [3]: linkLayer = LinkLayer(callback)

In [4]: util.mac_ntoa(linkLayer.MAC)
Out[4]: '02:00:0A:14:7C:A3'

In [5]: 02:00:0A:14:7C:A3 hello from another
In [5]:

In [5]: linkLayer.sendto('02:00:0A:14:0A:70', b'hello receviced')
02:00:0A:14:0A:70 hello receviced

In [6]: linkLayer.mac_table
Out[6]: {b'\x02\x00\n\x14\np': '10.20.10.112', b'\x02\x00\n\x14|\xa3': '10.20.124.163'}
```



```
-> $ ipython
Python 3.6.2 (default, Jul 20 2017, 03:52:27)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from LinkLayer import LinkLayer, util

In [2]: def callback(dst_mac, data): print(util.mac_ntoa(dst_mac), data.decode())

In [3]: linkLayer = LinkLayer(callback)

In [4]: util.mac_ntoa(linkLayer.MAC)
Out[4]: '02:00:0A:14:0A:70'

In [5]: linkLayer.sendto('02:00:0A:14:7C:A3', b'hello from another')

In [6]: 02:00:0A:14:0A:70 hello receviced
```

Here's a simple example of this library.

1. callback

   Once a frame arrives at the LinkLayer, it will pass the frame to the next layer or other handler. Thus, we need to register a callback function to the LinkLayer object. In other words, your network layer need to register it's callback to the LinkLayer.

2. sendto

   sendto is the function to send frame to other nodes in the network.

iv. Switch

With a traditional switch works on the second layer, you can implement a switch based on this library with simple combination of the LinkLayer objects.

b) Network Layer

In this project, you need to implement a IPv4-like network layer.

i. IP packet format

```
+---------+---------+--------+----------+---------+----------------+
|src IP   | dst IP  |  ttl   | protocol | length  |    payload     |
+---------+---------+--------+----------+---------+----------------+
| 4 bytes | 4 bytes | 1 byte |  1 byte  | 4 bytes | 0 - 546 bytes  |
+---------+---------+--------+----------+---------+----------------+
```

1. IP address is a standard IPv4 address with 32 bits length
2. ttl (time-to-live):

   When the TTL field hits zero, the router discards the packet and typically sends an ICMP Time Exceeded message to the sender.
3. protocol (protocol in payload field)
   a) ICMP-like: protocol = 1
   b) TCP-like: protocol = 2
   c) UDP-like: protocol = 3
4. length

   size of the payload in bytes
5. Fragmentation: no fragmentation

ii. Protocols

1. ICMP-like (An Internet Control Message Protocol like protocol)

```
+--------+--------+
|  type  |  code  |
+--------+--------+
| 1 byte | 1 byte |
+--------+--------+
```

message need to be implemented:

1. type=8 code=0 echo request; type=0 code=0 echo reply (ping)
2. type=3 destination unreachable
   a) code=0 Destination network unreachable
   b) code=1 Destination host unreachable
   c) code=2 Destination protocol unreachable
   d) code=3 Destination port unreachable
3. type=11 Time Exceeded; code=0 TTL expired in transit

2. TCP-like (see next section)
3. UDP-like (see next section)

iii. Router?

Yes! You need to design a router in this layer.You need to implement NAT (Network Address Translation) in the router part.

c) Transport Layer

i. TCP-like

1. packet format

```
+----------+---------------+
|  header  |    payload    |
+----------+---------------+
| 19 bytes | 0 - 527 bytes |
+----------+---------------+
```

header: 19 bytes

```
+----------+----------+---------+---------+--------+-------------+---------+
| src port | dst port | seq num | ack num | flags  | window size | length  |
+----------+----------+---------+---------+--------+-------------+---------+
| 2 bytes  | 2 bytes  | 4 bytes | 4 bytes | 1 byte |   2 bytes   | 4 bytes |
+----------+----------+---------+---------+--------+-------------+---------+
```

2. port: 0-65535
   seq num: sequence number
   ack num (if ack in flags is set): acknowledgment number
   flags: FIN: 0x1; SYN: 0x2; RST: 0x3; ACK: 0x5, other bits should be zero
   window size: The size of the receive window, which specifies the number of window size units (by default, bytes)
3. design other parts based on TCP related RFCs and things you learn on

ii. UDP-like

1. packet format

```
+----------+----------+---------+---------------+
| src port | dst port | length  | payload       |
+----------+----------+---------+---------------+
| 2 bytes  | 2 bytes  | 4 bytes | 0 - 538 bytes |
+----------+----------+---------+---------------+
```

3. What to implement

a) Network Layer

A NetworkLayer class with IP-like Protocol, ICMP-like Protocol implemented

A router class with route table, NAT

In this project, using static IP address, config your router table before your program runs which means you have already known how many nodes in your network and who are their gateway.

Your NetworkLayer should respond the invalid packets with ICMP, while your NetworkLayer should also deal with the income ICMP packets. Example: Destination network unreachable, Destination host unreachable, etc.

An implementation of ping utility (optional)

b) Transport Layer

A TransportLayer class with TCP-like Protocol and UDP-like Protocol implemented.

Socket API for Application Layer (reference python socket API):

- create socket (SOCK_STREAM, SOCK_DGRAM)
- bind address (ip, port)
- listen, accept (for SOCK_STREAM)
- send, recv (for SOCK_STREAM)
- recvfrom, sendto (for SOCK_DGRAM)
- close

raise exception when error occurs.

c) Application Layer

TCP and UDP socket programming based on your socket API.

Example: An echo server and client.

4. Tips:

a) What should your program behave?

Assume that you are an IETF expert, read the RFCs.

Think what will happen if it's in the real Internet. For example, trying to TCP handshake with a closed port, what will happen?

b) What if there are bugs with the Data-Link-Layer-Emulator library?

Recheck your code and confirm the bug, then report it as an issue immediately!