Layer 2 Raw Socket Programming

We assume the environment introduced in the previous lecture. In this hands-on session, we will create data link raw sockets using Linux and Python 3 and try to send/receive data using sockets.

Hands-On with Sample Code

Step 1. Preparation

Here is the sample code. (This session is based on this Python sample code. If you prefer C language, please refer to the C version.)

Open the terminal and type:

```
$ git clone https://github.com/y-sira/pyng.git
```

You can find network interface names and MAC addresses on your virtual machine using `ip` command as follows:

```
$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
    link/ether 08:00:27:dd:d7:43 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
    link/ether 08:00:27:b0:d6:ff brd ff:ff:ff:ff:ff:ff
4: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
    link/ether 08:00:27:e6:4d:39 brd ff:ff:ff:ff:ff:ff
5: enp0s10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mo
    link/ether 08:00:27:8e:75:44 brd ff:ff:ff:ff:ff:ff
```

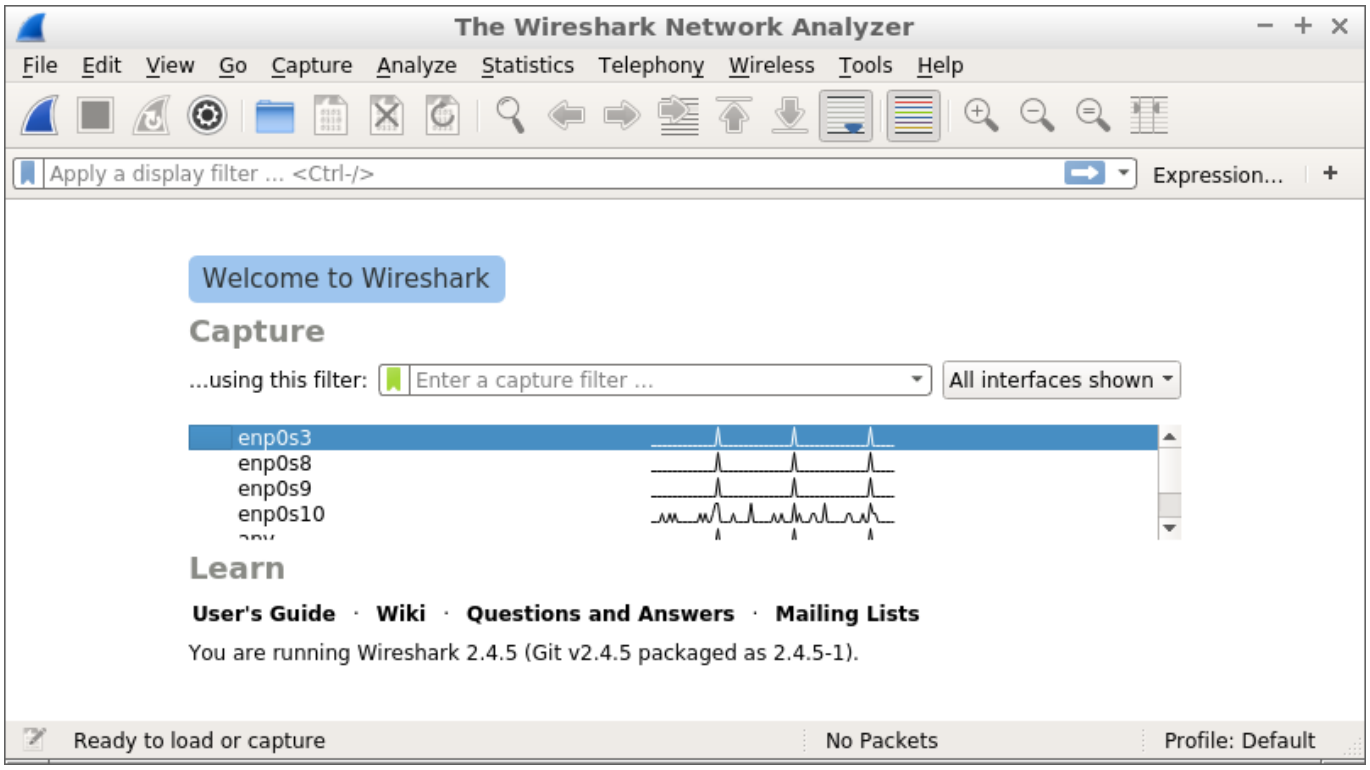
Please remember interface names and corresponding MAC addresses.

Step 2. Start Packet Capture Using Wireshark

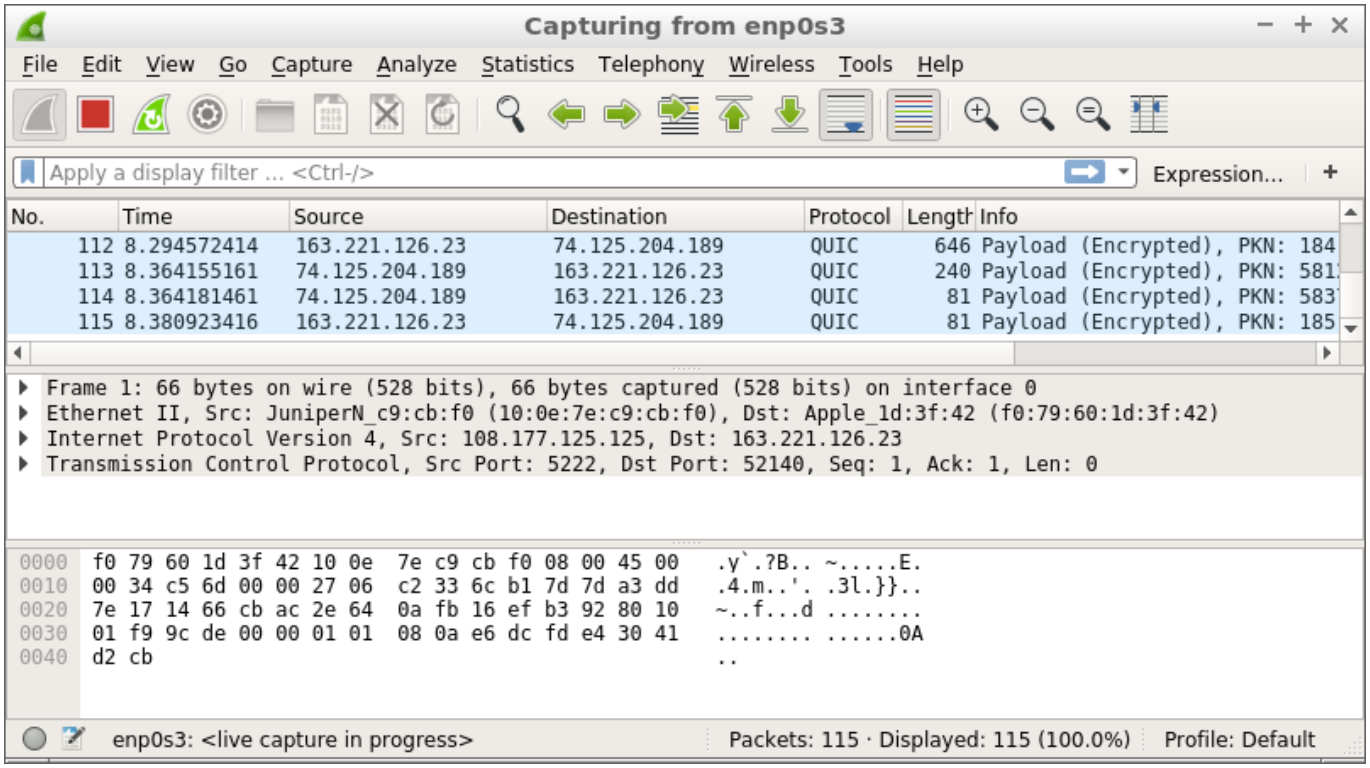
Open the terminal and type:

```
$ sudo wireshark
```

Wireshark screen will be appeared.



Double click the interface name you want to monitor, then you can see packets sent to the interface.



Step 3. Send/Receive Frames Using Layer 2 Raw Sockets

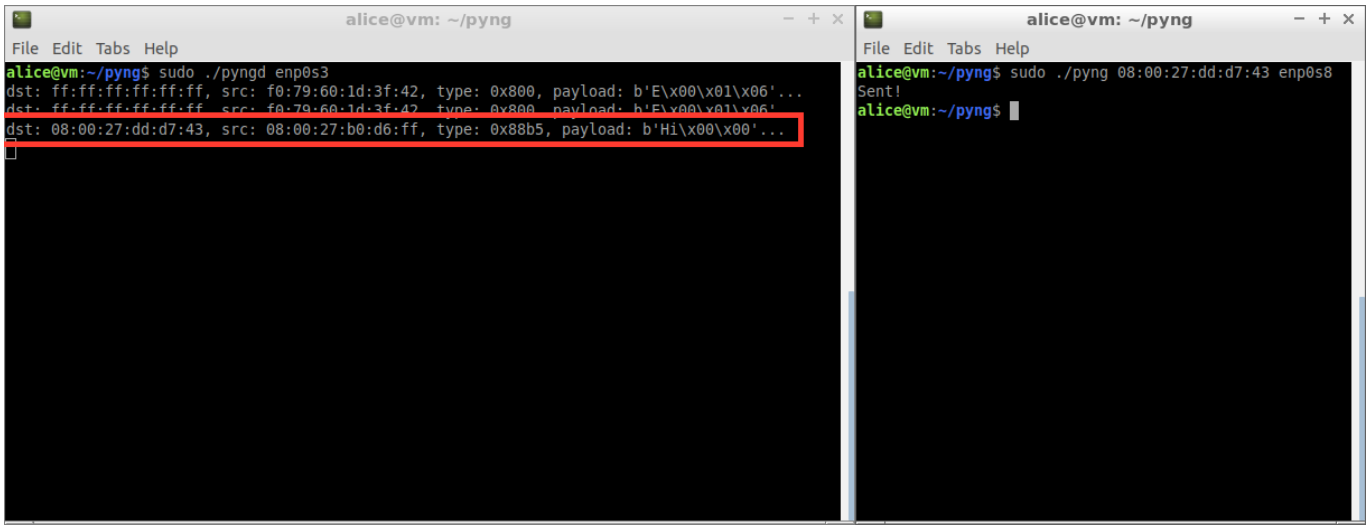
Move to `pyng` directory and run the server script:

```
$ cd /path/to/pyng
$ sudo ./pyngd $SERVER_INTERFACE_NAME
```

Open another terminal, move to `pyng` directory, and run the client script to send the frame to the server:

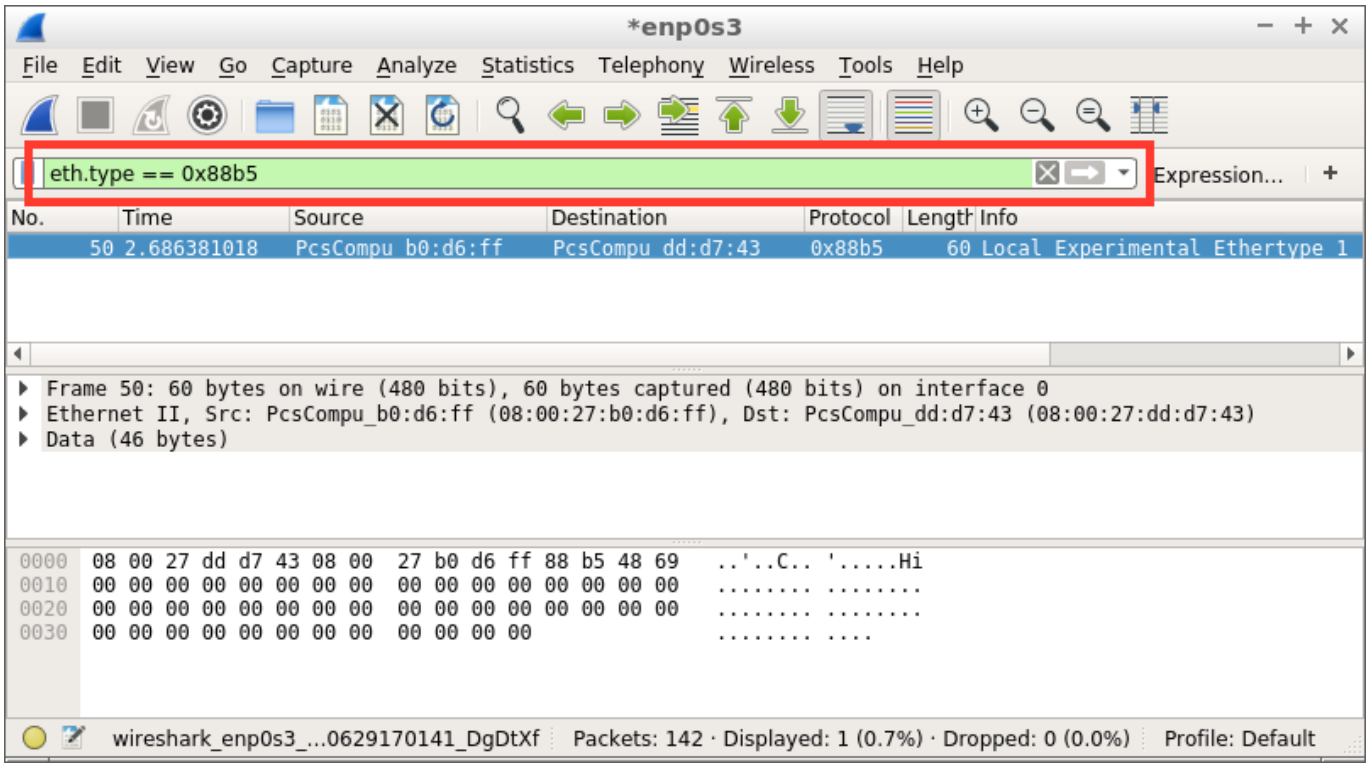
```
$ cd /path/to/pyng
$ sudo ./pyng $DESTINATION_MAC_ADDRESS $CLIENT_INTERFACE_NAME
Sent!
```

Make sure that the data client script sent is appeared in the server side terminal.



Step 4. Check Frames with Wireshark

Stop packet capture. You can filter the packet by protocol, source address, destination address, ..., etc. Please enter `eth.type == 0x88b5` in the filter text box to check frames you sent.



Verify the Ethernet header and the payload.

Step 5. Considerations

Please consider below.

- What data is sent?
  - Why does the payload contain many zero?
- How do we send/receive data in the sample code?
  - What is `struct.pack('!6s6sH', ...)` in the server side script?
- What does the magic number `0x88b5` mean?
- What kinds of packets are displayed in Wireshark?
- What does the MAC address `ff:ff:ff:ff:ff:ff` mean?
- What is `select` in the server side sample code?
  - Why do not use `recv()` directly?
  - How can we handle massive requests?
- ...

For Beginners

Socket APIs

The socket APIs are APIs for interprocess communication and network communication from applications. It is available on UNIX and Linux. In Linux, it is easy to handle the data link layer by

using the socket API. Let's explain Linux sockets later.

Data Link Raw Sockets

In order to send and receive data using socket APIs, we have to generate a socket descriptor. The socket descriptor can be generated using `int socket (int family, int type, int protocol)` defined in `sys/socket.h`. The first argument `family` specifies the address family. For the data link layer, specify `AF_PACKET` defined in `sys/socket.h`. The second argument `type` specifies the socket type. For the data link layer, specify `SOCK_RAW` defined in `sys/socket.h`. The third argument `protocol` specifies the protocol. Specify `ETH_P_ALL` to retrieve any Ethernet frames, and specify `ETH_P_IP` to retrieve Ethernet frames containing a IP packet. The protocol types of the Ethernet frame is defined in `linux/if_ether.h`.

In Python, there is a standard library that wraps the OS level socket APIs, so we will use it to create a data link raw socket. Please be careful to `close()` after using the socket.

```
import socket
ETH_P_ALL = 3
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ETH_P_ALL))
s.close()
```

We use `bind()` to bind the network interface to the socket. You can check the network interface available on your computer with the following command.

```
$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
    link/ether 08:00:27:dd:d7:43 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
    link/ether 08:00:27:b0:d6:ff brd ff:ff:ff:ff:ff:ff
4: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
    link/ether 08:00:27:e6:4d:39 brd ff:ff:ff:ff:ff:ff
5: enp0s10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mo
    link/ether 08:00:27:8e:75:44 brd ff:ff:ff:ff:ff:ff
```

For example, to bind the network interface enp0s3 to the socket, write as follows.

```
import socket
ETH_P_ALL = 3
interface = 'enp0s3'
```

```
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ETH_P_ALL))
s.bind((interface, 0))
# do something
s.close()
```

Now you are ready to send and receive raw data on the data link layer using the network interface enp0s3.

Receive Data Using Raw Sockets

Use `recv()` to receive data using a socket. Specify the buffer size in first argument of `recv()`. `recv()` will block until ready to receive data from the socket. The return value of `recv()` is the received byte sequence. This byte string consists of the header and the payload of the Ethernet frame.

When receiving data sent to enp0s3, the script `server.py` is as follows:

```
import socket
ETH_P_ALL = 3
ETH_FRAME_LEN = 1514  # Max. octets in frame sans FCS
interface = 'enp0s3'
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ETH_P_ALL))
s.bind((interface, 0))
data = s.recv(ETH_FRAME_LEN)
print(data)           # => b'\x08\x00\x27\xdd\xd7\x43\x08\x00\x27\x8e\x75\x44\x88
s.close()
```

Note that the data link raw sockets require a root permission. Please run the above script as root.

When the data is sent to enp0s3, you can get the following output:

```
$ sudo python3 server.py
b"\x08\x00'\xdd\xd7C\x08\x00'\x8euD\x88\xb5Hi"
```

Send Data Using Raw Sockets

To send data using a socket, use `sendall()`. The data to be sent must be a byte sequence containing the header and payload of the Ethernet frame.

When sending data “Hi” from enp0s10 (08:00:27:8e:75:44) to enp0s3 (08:00:27:dd:d7:43), the script `client.py` is as follows:

```
import socket
ETH_P_ALL = 3
interface = 'enp0s10'
dst = b'\x08\x00\x27\xdd\xd7\x43'  # destination MAC address
src = b'\x08\x00\x27\x8e\x75\x44'  # source MAC address
proto = b'\x88\xb5'                # ethernet frame type
payload = 'Hi'.encode()            # payload
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ETH_P_ALL))
s.bind((interface, 0))
s.sendall(dst + src + proto + payload)
s.close()
```

Then,

```
$ sudo python3 client.py
Sent!
```

Make sure that the server side terminal outputs the data you sent.

See Also

- A. S. Tanenbaum and D. J. Wetherall, Computer Networks, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010.
- “Ethernet frame - Wikipedia,” https://en.wikipedia.org/wiki/Ethernet_frame
- “IEEE 802 Numbers,” https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml
- “Introduction to RAW-sockets,” http://tuprints.ulb.tu-darmstadt.de/6243/1/TR-18.pdf
- “Promiscuous mode - Wikipedia,” https://en.wikipedia.org/wiki/Promiscuous_mode
- “GitHub - torvalds/linux: Linux kernel source tree,” https://github.com/torvalds/linux
  - “linux/socket.h at master · torvalds/linux · GitHub,” https://github.com/torvalds/linux/blob/master/include/linux/socket.h
  - “linux/if_ether.h at master · torvalds/linux · GitHub,” https://github.com/torvalds/linux/blob/master/include/uapi/linux/if_ether.h
- “The Python Standard Library — Python 3.7.0 documentation,” https://docs.python.org/3/library/index.html
  - “19.1. socket — Low-level networking interface — Python 3.7.0 documentation,” https://docs.python.org/3/library/socket.html
  - “7.1. struct — Interpret bytes as packed binary data — Python 3.7.0 documentation,” https://docs.python.org/3/library/struct.html
  - “19.3. select — Waiting for I/O completion — Python 3.7.0 documentation,” https://docs.python.org/3/library/select.html

    - “19.4. selectors — High-level I/O multiplexing — Python 3.7.0 documentation”, https://docs.python.org/3/library/selectors.html
- “GitHub - python/cpython: The Python programming language,” https://github.com/python/cpython
    - “cpython/socketserver.py at master · python/cpython,” https://github.com/python/cpython/blob/master/Lib/socketserver.py

---

Internet Engineering (4007)

IPLab Teaching Assistants
internet-eng-2018@is.naist.jp

This page is maintained by teaching assistants. Please feel free to contact us by email if you have any questions or comments.