Python Sniffer Assignment

Teaching Assistant: Adil Alsuhaim (aalsuha@clemson.edu)
Written for Dr. Andrew Robb

September 25, 2018

Objective

The objective is to create a Python Packet Sniffer to read raw packets and analyzing it. Most of the content of this document is information aimed to help you understand the topic to have an easier time performing the required task. The student's task is highlighted under section 2.2 titled "Your Task". You will need to work on a Linux machine where you have administrative privileges.

1 Background Information

A packet analyzer is a network analysis software that is used to intercept and log data traffic through a network interface.

Data is communicated in a network through physical mediums such as Ethernet cables, or radio waves like WiFi. The signal received is processed by the network interface as a binary sequence of zeros and ones. Network packets are encapsulated in a data-link frame. Consider the case shown in the figure, where Computer A sends a packet to Computer B. The *data* that comes out of the application at Computer A is identical to the data received by the application at Computer B. However, the actual packet received by Computer B will have different MAC header information. Computer A uses a WiFI network interface, the outgoing data is encapsulated in a WiFi MAC frame complaint with WIFI standard IEEE 802.11. This is how the two points, Computer A and its wireless access point, communicate. The MAC frame is only used in the data-link between these two points. It would be different over different physical links.[1]

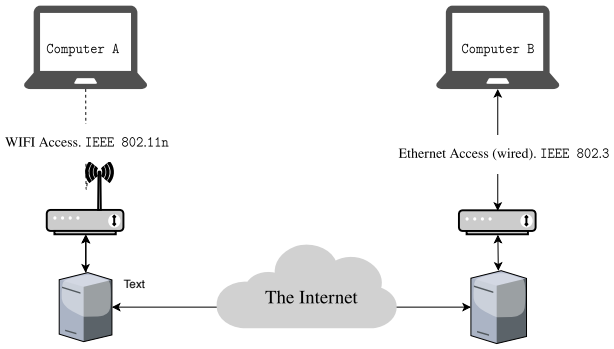
Computer B uses an Ethernet interface, and is wired to its access point. The packet coming into Computer B would have a different MAC frame format since Ethernet is different than WiFi. It will also contains information only relevant to Computer B and its access point, which is specified by the IEEE 802.3 for the Ethernet frame. Even if both Computer A and Computer B were connected to their access respective points with the same standard (i.e. both WiFi or both Ethernet), the MAC addresses on the packets would be different.

In this example, the wireless access point then examine the MAC header to determine if it is the intended recipient[2] of the packet by examining the destination MAC address. In addition,MAC information are used to determine the type of data beyond the MAC header to determine what to do with the packet[3]. For example, if the MAC EtherType is 0x0806 then this is an Address Resolution Protocol (ARP) packet, used by hosts on a network to map IP addresses to MAC addresses.

[1] In this case, the WiFi radio waves are the *physical* link.

[2] It is more accurate to say an intended recipient, since there are broadcast packets intended to all recipients who can read it off-the-air. An example of a wireless broadcast packet is when a WIFI broadcast its SSID. When you work on a laptop and check all available WIFI network, your computer will show you all available WIFI networks available. These networks are the results of receiving beacon messages from multiple WIFI points within range.

[3] It is worth noting that there are different types of WiFi frames. Some contain application data, and some are used for protocol management (like association, authentication, etc). You may come across the details of this in the future.



If this is a data intended from Computer A to Computer B, which is on a different network. The access point will examine the network layer information in the IP header to forward the packet to the internet, or in more accurate terms, to the gateway configured to be its next hop. The gateway is likely a server that is owned by the Internet Service Provider (ISP) of Computer A. Of course, since the data-link is different between the access point and the ISP's gateway, there will be a different MAC information attached to the packet, and the previous MAC information is removed since they are not relevant to this link.

There are more details to what happens every to a packet when their IP header is examined along the path. The easiest to explain is what happens to the Time-To-Live (TTL) value, which gets decremented by 1 at each router. However, there are also other factors, such as examining priority, dropping packets, sending a reply back to sender (like source quench), and others. We will not go through these details in this assignment.

Data between two computers over the network go through multiple servers, we refer to them as *hops*, until they reach their destination. You can have fun trying the traceroute tool on UBUNTU or MAC (On Windows it is called tracert). It is interesting to try to trace a packet's route to servers in different countries, and see what route does your packet take.

1.1 Transport and Network Layer information

When we write network applications to send data over a network, you may have noticed that we are mostly concerned with the data to be sent and the destination's address. As a matter of fact, whenever we make a call to send function, we only provide data as parameters. The Python language API does some work in the background that you do not have to handle with your application. If your application uses send to send a chunk of data of size 100 Bytes, the packet that leaves your computer is slightly larger than that because the network stack within the system attach more information to it. For example the transport protocol in use, such as UDP or TCP, which is added by the Transport layer. The Network layer will attach additional information like the source IP address.

And as we mentioned earlier, MAC header information are then attached, which would include the MAC address of your network interface [4], and some information about the content beyond the MAC header information, like whether the packet contains IPv4 or IPv6 data (specified by EtherType).

Once we decide that a packet is an IP packet, we can look at its content. We can determine what common application the packet belongs to by examining the source and destination port of UDP and TCP packets. A packet with a port number 22 belongs to an ssh application, and a packet with 53 is a Domain Name Service (DNS) packet[5]

1.2 Packet Sniffing on Linux using Python

To capture packets on a Linux system using Python programming language, the heart of the code is a raw socket that listens to incoming/outgoing data at all interfaces.

```
import socket
ETH_P_ALL = 0x0003
s = socket.socket( socket.AF_PACKET , socket.SOCK_RAW , socket.ntohs(ETH_P_ALL)) #Linux

while(True):        #Loop forever, accept all incoming packets.
```

[4] MAC address is sometimes referred to as the physical address
[5] 53 in decimal. You might get confused with this because 53 decimal is 0x35 in hexadecimal

```
        packet = s.recvfrom(65565)
        #Do something with the received packet
```

On a Linux, we use `ETH_P_ALL` to capture packets, even those that are not intended to our machine in what is called "promiscous" mode, as long as they are detectable in the physical medium.
In addition, this code detects packets on all interfaces. You can get a list of interfaces on your computer, use `ifconfig`. One of these interfaces is `lo` for `loopback`, and the other interfaces cooresponds to other interface on the system. In the case of virtual machines, the interface is a virtual connection to the host operating system. On Ubuntu, the interface can have different names like `eth0`, `eno1`, and others.

To create a raw socket that capture low-level packets, the option `AF_PACKET` is used on linux[6]. The function `recvfrom` will read a single packets each time it is called and will return the result in a tuple of `(bytes, address)`. The data is recieved is stored in an object of type `class 'bytes'`, which represent the bytes recieved from the network.and the address is of type `class 'tuple'`. We are going to work mostly with `bytes`, to dissect packets and determine what they are.

2 Your Assignment

2.1 The Setup

We have created few `Python` files to make your work easier.

- `PythonSniffer.py` is a a program that reads packets as shown above and store them in a Python dictionary of the form `(number, byte_array)`
- `PacketInformation.py` which is a class that is instantiated with the tuple info we get from `recvto`. The purpose of this class is to describe recieved packets. Your job is to complete the functions marked with `TODO`. This file contains the main method, and is the file you should execute.
    - If you are reading new packets from the network (not a previously logged file), you should run it using Python 3 **as a super-user**. For example, the command can be python or python3. Check with --version to see which python version are you running. To run the program, run it with sudo as follows:

```
$ sudo python3 PacketInformation.py --num_packets=###
```

2.2 Your Task

Study the format of an Ethernet frame. The first 6-bytes is the `MAC` address of the packet's destination in hexadecimal, followed by the `MAC` address of the source. This is followed by `EtherType`. There are different ways to extract this information in `Python`. Just make sure you're getting the correct information out of your packet.

Your task is to work with the code given to you. You are mainly concerned with the `PacketInformation.py` file, which has empty functions that you should implement to display packet information.

As given to you, the program only prints the packet sizes in bytes and nothing else. You need to use the packet's information to show the following details:

- **MAC layer info.** You should be able to extract source and destination `MAC` addresses from the packets. The information we want are :

[6] It won't work on Mac OSX or Windows

  - **Source MAC Address.** This is a 6-byte data. Typically shown with hyphen or colons. For example, `08:00:27:9b:11:fc` is the MAC address for my virtual machine.
  - **Destination MAC Address.** This is a 6-byte data. You may see packets with MAC address `FF:FF:FF:FF:FF:FF`, which means the packet is a broadcast message.
  - **EtherType.** Gives some information about the packet. For example, if it is an `IPv4` packet or `ARP`. This allows hosts to figure out what to do with the packet.

  I expect you to capture and show at least one `ARP` packet in your assignment. If you struggle getting one packet, try clearing the `ARP` cache on your system.

- **Network layer information.** If the packet is `IPv4` packet, then determine the `IP` header information.
  - **IP Protocol Version.** This is can be 4 or 6. We only need the details of `IPv4` packet. For `IPv6` packet, we will simply print that it is `IPv6` and not go into any further details.
  - **Source IPv4 address.** in dotted decimal format.
  - **Destination IPv4 address.** in dotted decimal format.
  - **Time-To-Live (TTL)** which should be an integer.
  - **Protocol.** The `IP` protocol used by the packet. We will focus on `UDP` and `TCP`. But there is also `ICMP` as well. The protocol numbers (in decimal) are 6 for `TCP`, 17 for `UDP` and `1` for `ICMP`.
- **Application-layer information.** Extract the destination and source port numbers from `UDP` and `TCP` packets. These information are found in the `TCP` and `UDP` headers. The port numbers from 1 to 1023 are reserved. You can which packet belongs to which application by investigatng the number. For example,
  - **Port 22.** Means that the `TCP` packet belongs to an `ssh` client.
  - **Port 53.** Means that the `UDP` packet belonds to a `Domain Name Service` query. If your packet has these ports, print some information about the application. The following table lists some common Application-level services and their protocol numbers.

Port	Service name	Transport protocol
20 and 21	File Transfer Protocol (FTP)	TCP
22	Secure Shell (SSH)	TCP and UDP
25	Simple Mail Transfer Protocol (SMTP)	TCP
53	Domain Name Server (DNS)	TCP and UDP
67 and 68	Dynamic Host Configuration Protocol (DHCP)	UDP
69	Trivial File Transfer Protocol (TFTP)	UDP
80	HyperText Transfer Protocol (HTTP)	TCP
443	HTTP with Secure Sockets Layer (SSL)	TCP and UDP

  You may not be able to get all these ports, but you I expect you to show at least 3 types of packets. If you struggle with this, simply try to generate traffic by opening `ssh` session, and open a web browser.

Final Remark

Not required, but for your information. If you are using a virtual machine, then keep in mind that your virtual machine connects to the internet in different ways. It can be attached to a the host OS with `NAT`, and therefore, the only `MAC` addresses you would see are the virtual `MAC` addresses between your virtual machine and VirtualBox, which is adequate for the assignment.

If you are working on a laptop with `WiFi` connection. You can have more fun (and educational experience) if you configure your virtual machine through VirtualBox setting to attach to `Bridged Adapter` instead of `NAT`. This allows it to access the internet through the `WiFi` device as if the `WiFi` device was attached to it directly, allowing you to see `MAC` addresses with broadcast address of `FF:FF:FF:FF:FF:FF`. However, keep in mind that the `IEEE802.11 MAC` frame is different than Ethernet `MAC` frames.

Sample Output

```
========== PACKET ==========
Size : 66 Bytes
---- MAC frame Information ----
Destination MAC: 10:6f:3f:0e:76:d4
Source MAC: 34:97:f6:8b:25:b9
EtherType: 0x800 (IPv4)

---- Network/Transport Information ----
IPv4 Packet
--------
Source IP Address : 192.168.11.14
Destination IP Address : 130.127.201.248
IP Protocol : TCP
---- Application Information ----
Application Protocol : Secure Shell (SSH)
========== PACKET ==========
Size : 66 Bytes
---- MAC frame Information ----
Destination MAC: 34:97:f6:8b:25:b9
Source MAC: 10:6f:3f:0e:76:d4
EtherType: 0x800 (IPv4)

---- Network/Transport Information ----
IPv4 Packet
--------
Source IP Address : 35.172.175.129
Destination IP Address : 192.168.11.14
IP Protocol : TCP
---- Application Information ----
Application Protocol : HTTP with Secure Sockets Layer (SSL)
========== PACKET ==========
Size : 60 Bytes
---- MAC frame Information ----
Destination MAC: 34:97:f6:8b:25:b9
Source MAC: 10:6f:3f:0e:76:d4
EtherType: 0x806 (Unknown)

---- Network/Transport Information ----
ARP Protocol
========== PACKET ==========
Size : 139 Bytes
---- MAC frame Information ----
Destination MAC: 34:97:f6:8b:25:b9
Source MAC: 10:6f:3f:0e:76:d4
EtherType: 0x800 (IPv4)

---- Network/Transport Information ----
IPv4 Packet
--------
Source IP Address : 192.168.11.1
Destination IP Address : 192.168.11.14
IP Protocol : UDP
---- Application Information ----
Application Protocol : Domain Name Server (DNS)
```