National University
of Computer & Emerging Sciences

Packet Sniffing In Broadcast Domain

Group Members:

Muhammad Ammar Rizwan - 16k3862

Muhammad Ahmed Gul - 16k3865

Course:

CS-307 Computer Networks

May 10,2019

Abstract

Packet Sniffer is a packet analyzer software that monitors streams of data packets that flow between computers on a network.It has many applications from administrative purposes to collecting data for security analysis.

This report shows the implementation of packet sniffing of ethernet frames using socket programming.The implementation is done using raw sockets and python3 to sniff off ethernet frame.

The project successfully sniffs off TCP UDP and ICMP packets in the ethernet frame and displays the fields of each packet according to its protocol.

1.Introduction:

1.1 Motivation:

The motivation behind this project is to have a deeper analysis of the packet headers and how packet transmits through the ethernet frame. Packet sniffer analyzer has many applications in real world such as collecting and inspecting data for security purpose.

1.2 Project Description:

This project uses Raw Sockets to sniff packets in the ethernet frame. It unpacks all the fields in frame, converts it into human understandable form and displays packet information like mac addresses, ports , and transport layer protocols including TCP, UDP, ICMP.

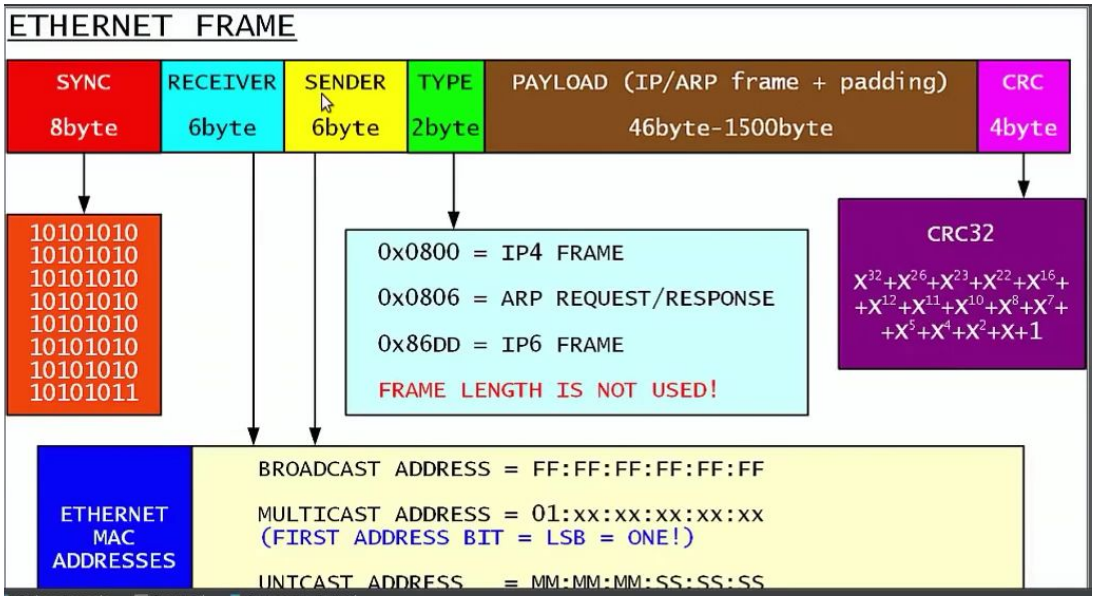
2. Implementation:

2.1 Tools:

- Raw Sockets
- Python

2.2 Sniffing Ethernet Frame:

An ethernet frame consists of 6 fields as shown in the the figure below. As sniffer program receives ethernet frame it sniffs the ethernet header which include receiver address, sender address, type and payload. Firstly the first 14 bytes of the frame is sniffed and then the remaining bytes is unpacked in the next stage. In the next stage the payload is sniffed from 14 bytes till the end of the frame.



2.3 ICMP:

The Internet Control Message Protocol contains three segments type, code and checksum. ICMP is used for sending error messages and operational information. The snippet below unpacks the icmp_type, the code and the checksum.

```
# Unpacks for any ICMP Packet
def icmp_packet(data):
    icmp_type, code, checksum = struct.unpack('! B B H', data[:4])
    return icmp_type, code, checksum, data[4:]
```

2.4 TCP:

The TCP(Transmission Control Protocol) segment contains source port, destination port, sequence number, acknowledgement number and flags. The snippet below unpacks source and destination port, acknowledgment and sequence number and 16 bit offset from TCP segment.

```
# Unpacks for any TCP Packet
def tcp_seg(data):
    (src_port, destination_port, sequence, acknowledgenment, offset_reserv_flag)
= struct.unpack('! H H L L H', data[:14])
    offset = (offset_reserv_flag >> 12) * 4
    flag_urg = (offset_reserved_flag & 32) >> 5
    flag_ack = (offset_reserved_flag & 32) >>4
    flag_psh = (offset_reserved_flag & 32) >> 3
    flag_rst = (offset_reserved_flag & 32) >> 2
    flag_syn = (offset_reserved_flag & 32) >> 1
    flag_fin = (offset_reserved_flag & 32) >> 1

    return src_port, dest_port, sequence, acknowledgement, flag_urg, flag_ack,
flag_psh, flag_rst, flag_syn, flag_fin, data[offset:]
```

OUTPUT:

```
Ethernet Frame:
        - Destination: 84:3A:4B:7A:0D:3E, Source: 00:0C:29:0D:84:F7, Protocol: 8
        - IPV4 Packet:
                - Version: 4, Header Length: 20, TTL: 103
                        - protocol: 6, Source: 108.177.122.189, Target: 172.16.72.16
        - TCP Segment:
                - Source Port: 33850, Destination Port: 19322
                - Sequence: 222167052, Acknowledgment: 688751863
                - Flags:
                        - URG: 2048, ACK: 17664, PSH: 260
                        - RST: 23466, SYN: 0, FIN:26374
Time to sniff 0.00018906593322753906
                - TCP Data:
                        b'\x01\xbb\x8c\xfc>\xe2\xfa\xdf[(\x95\x8b\x80\x18\x01~h\xe5\x00\x00\x01\x01\x08\nK\xc3k\xdb\x19\x871<\x17\x03\x03\x
00\xcb\x00\x00\x00\x00\x00\x00\x00dK\xcc\x89zP(<V?\xe26\x07"X\x1e+\xb1\xfbN\xf8g\xd0\xc8\x94b5=\xf6\x82]^\x98\xd1Q@\x0f\xcch\xbf\xfe`\xfacU9[c~
d\xe8\x18n\x10\x1f\xfa\xe1\xc4\x08\x19\x15C>2\xfa\xea\xe6z\x842\xfe\xf6=\xb2=A\xb4\x9aT}\x88\xce\xb4\x94\xc3\x9fjNZd\x1ddx\x9d\xce\x8dr\x7f\x89
;iT\xa9\x8d\x94\x03\xf8&\xd2\x85\x8e\x8bz<\x13\xdd\x97\xce\xd7f}\xa4\x81q\xf9\xc3\x19\xfd\xd4!\xba\xefSc\x88{$\x84\xd9\xa4\xc34Y\xba\xd7\xbe\xd
1\x85\xd4\x89\xac)\x084\xe0;p-\x1aY~\xb0\xb6\x0e\x01\xf0\xe3\xd8-M\xcf\xbf&\xb7\x14o}(zJ\xccW0\x1a*\x8e\x14\xba\x90\xdf\xcb\x1a^\x81V\x80'
```

2.5 UDP:

UDP(User Datagram Protocol) contains source port ,destination port and size. The snippet below unpacks the src ,dest port and size from from UDP segment.

```
# Unpacks for any UDP Packet
def udp_seg(data):
    src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
    return src_port, dest_port, size, data[8:]
```

OUTPUT:

```
Ethernet Frame:
        - Destination: FF:FF:FF:FF:FF:FF, Source: 84:3A:4B:7A:0D:3E, Protocol: 8
        - IPV4 Packet:
                - Version: 4, Header Length: 20, TTL: 64
                        - protocol: 17, Source: 172.16.72.16, Target: 255.255.255.255
        - UDP Segment:
                - Source Port: 44444, Destination Port: 37020, Length: 9379
Time to sniff 0.0001304149627685547
```

3. Results :

3.1 Sniffing Time TCP versus UDP:

The sniffed packets of two protocols UDP and TCP were compared on the basis of the time taken to sniff off the packets of the two.Following table shows time to unpack a UDP packets;

Packet Number	**Broadcasted Message Size(in bytes)**	**Time to sniff(in seconds)**
1	100	2.973
2	100	2.770
3	100	3.012
4	100	2.532
5	100	2.142
6	100	3.314
7	100	3.183

8	100	2.733

Average Time to sniff a udp packet was found to be 2.832 seconds whereas TCP packet sniffing results are shown in the table below:

Packet Number	**Broadcasted Message Size(in bytes)**	**Time to sniff(in seconds)**
1	100	6.747
2	100	7.176
3	100	6.658
4	100	8.583
5	100	6.651
6	100	6.914
7	100	3.183
8	100	2.733

Upon above mentioned experiment it was concluded that sniffing a UDP packet is 2.51 times faster than sniffing a TCP,possible reasons being the size difference of both the protocols and the number of fields in TCP to bhi unpacked.

3.2 Relationship between Payload Size and Sniffing Time:

Another study was conducted to observe the change in sniffing time based on the size of the broadcasted message.
A range of different sized message were broadcasted from the server as shown below to observe the change in time.

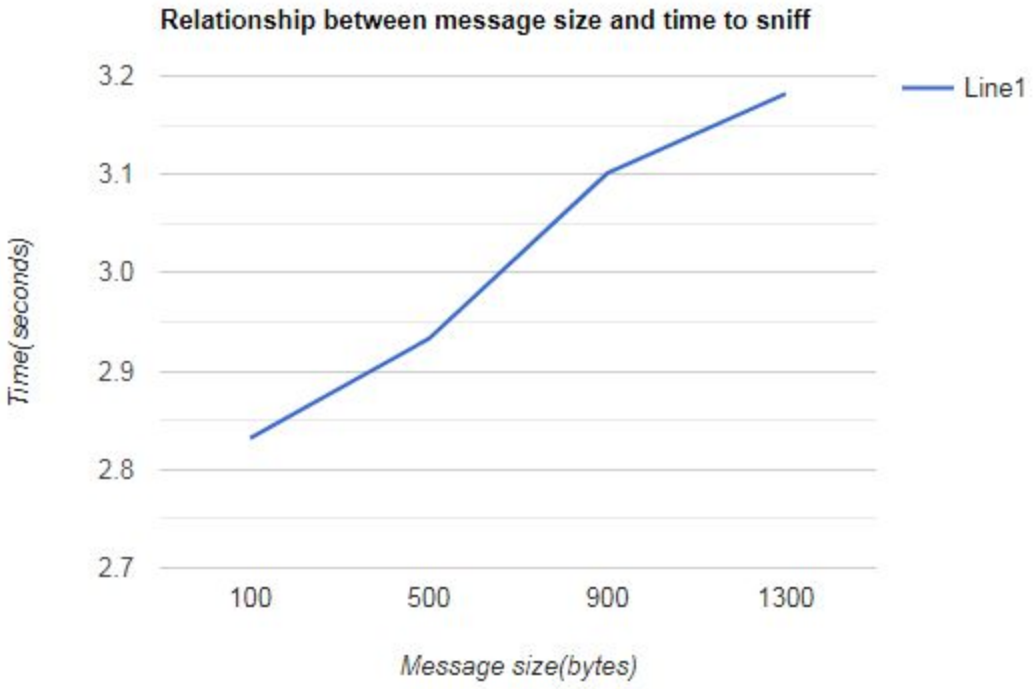
```
server.bind(("", 44444))
message = b"""
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla mattis a odio vitae finibus. Cras e
Fusce sollicitudin elementum eros sed auctor. Morbi iaculis, magna vitae eleifend iaculis, felis d
Proin a finibus urna. Pellentesque habitant morbi tristique senectus et netus et malesuada fames
ac turpis egestas. Duis facilisis at orci nec eleifend. Nunc sit amet leo sit amet dui rutrum
posuere. Sed condimentum dui tortor, non suscipit est tincidunt a. In hac habitasse posuere."""
while True:
```

A 500 bytes broadcast message

```
server.settimeout(0.2)
server.bind(("", 44444))
message = b"""Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Nulla ac molestie ligula. Duis dapibus sed."""
while True:
```

A 100 bytes broadcast message

Packet Number	Broadcasted Message Size(in bytes)	Time to sniff(in seconds)
1	100	2.832
2	500	2.933
3	900	3.101
4	1300	3.199



The above line bar show case that sniffing time increased as the size of the payload was increased.

4. Conclusion

It is concluded that an efficient packet sniffer is implemented that can sniff UDP TCP and ICMP protocol from an ethernet frame.We further came to a conclusion that UDP packet was sniffed more quickly than the TCP packet (i.e about 2.5 times faster).

Another conclusion that we incurred was the relationship between payload size and the speed of sniffing,it is deduced that packet sniffing speed decreases with increase in packet size.

The project in future can be updated to sniff wireless frames as well as IPv6 header of the frames ,which are the two limitations in this project.