Capturing Packets using Raw Socket

Kotaro Kataoka
Dept. of CSE, IIT Hyderabad

Overview

- Let’s make a simple packet capturing software like tcpdump
  - Extremely restricted feature
  - Command Line Interface
- How do you prepare your NIC to capture packets?
- How do you show the contents of packet headers?
  - Casting a pointer of buffer to the structure of specific protocol header

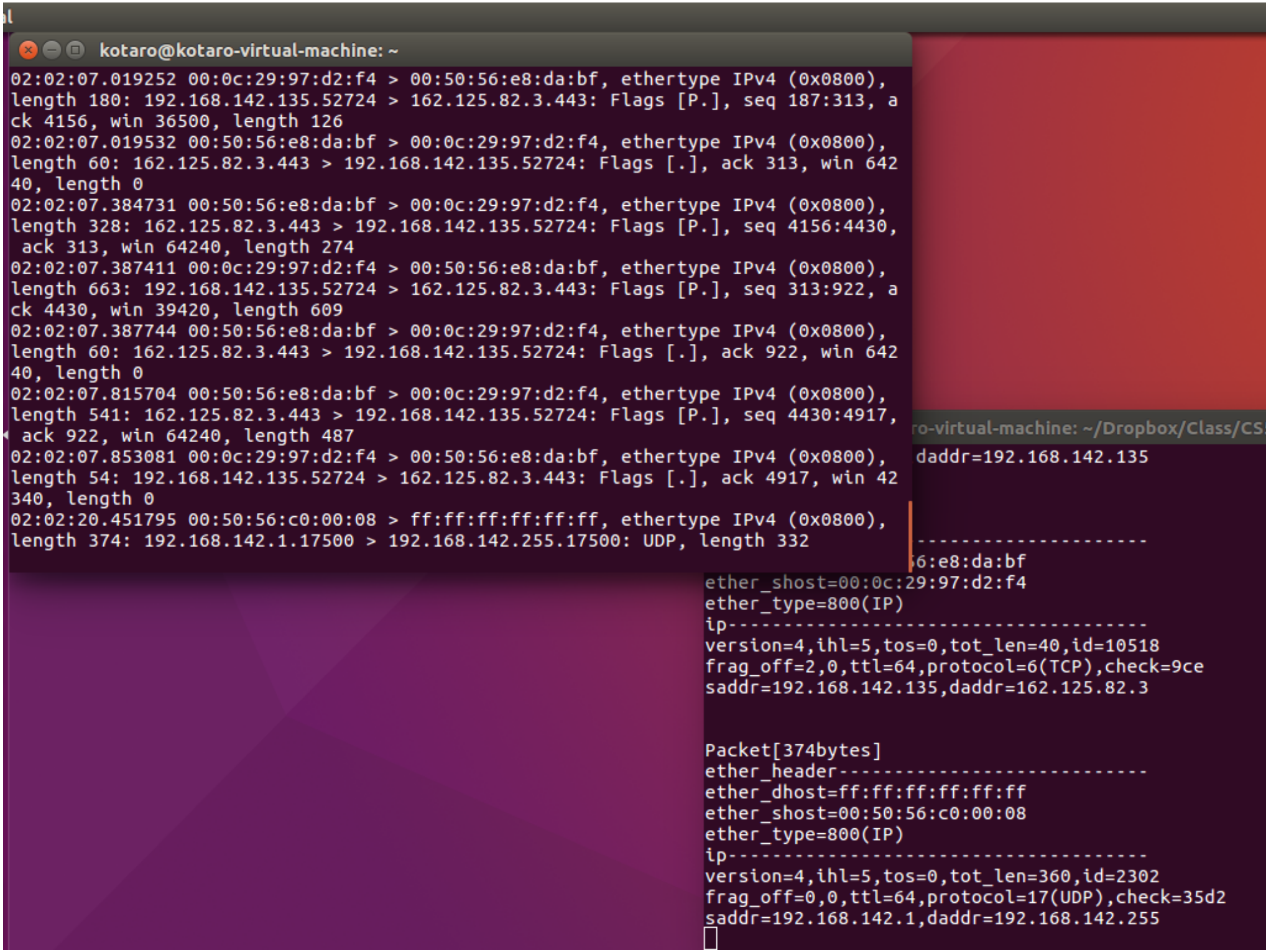
Main Function in Brief

```
int
main(NIC Name)
{
   Setup NIC to Promiscuous Mode;

   While(1) {
      Read a packet from NIC;
      Analyze and show packet Contents;
   }

}
```

Demo


kotaro@kotaro-virtual-machine: ~
02:02:07.019252 00:0c:29:97:d2:f4 > 00:50:56:e8:da:bf, ethertype IPv4 (0x0800), length 180: 192.168.142.135.52724 > 162.125.82.3.443: Flags [P.], seq 187:313, ack 4156, win 36500, length 126
02:02:07.019532 00:50:56:e8:da:bf > 00:0c:29:97:d2:f4, ethertype IPv4 (0x0800), length 60: 162.125.82.3.443 > 192.168.142.135.52724: Flags [.], ack 313, win 64240, length 0
02:02:07.384731 00:50:56:e8:da:bf > 00:0c:29:97:d2:f4, ethertype IPv4 (0x0800), length 328: 162.125.82.3.443 > 192.168.142.135.52724: Flags [P.], seq 4156:4430, ack 313, win 64240, length 274
02:02:07.387411 00:0c:29:97:d2:f4 > 00:50:56:e8:da:bf, ethertype IPv4 (0x0800), length 663: 192.168.142.135.52724 > 162.125.82.3.443: Flags [P.], seq 313:922, ack 4430, win 39420, length 609
02:02:07.387744 00:50:56:e8:da:bf > 00:0c:29:97:d2:f4, ethertype IPv4 (0x0800), length 60: 162.125.82.3.443 > 192.168.142.135.52724: Flags [.], ack 922, win 64240, length 0
02:02:07.815704 00:50:56:e8:da:bf > 00:0c:29:97:d2:f4, ethertype IPv4 (0x0800), length 541: 162.125.82.3.443 > 192.168.142.135.52724: Flags [P.], seq 4430:4917, ack 922, win 64240, length 487
02:02:07.853081 00:0c:29:97:d2:f4 > 00:50:56:e8:da:bf, ethertype IPv4 (0x0800), length 54: 192.168.142.135.52724 > 162.125.82.3.443: Flags [.], ack 4917, win 42340, length 0
02:02:20.451795 00:50:56:c0:00:08 > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800), length 374: 192.168.142.1.17500 > 192.168.142.255.17500: UDP, length 332
ro-virtual-machine: ~/Dropbox/Class/CS
daddr=192.168.142.135
6:e8:da:bf
ether_shost=00:0c:29:97:d2:f4
ether_type=800(IP)
ip----------
version=4,ihl=5,tos=0,tot_len=40,id=10518
frag_off=2,0,ttl=64,protocol=6(TCP),check=9ce
saddr=192.168.142.135,daddr=162.125.82.3
Packet[374bytes]
ether_header----------
ether_dhost=ff:ff:ff:ff:ff:ff
ether_shost=00:50:56:c0:00:08
ether_type=800(IP)
ip----------
version=4,ihl=5,tos=0,tot_len=360,id=2302
frag_off=0,0,ttl=64,protocol=17(UDP),check=35d2
saddr=192.168.142.1,daddr=192.168.142.255

Code: Setting up NIC

Setting-up NIC (1)

- Creating a RAW socket for capturing any Ethernet Frame (PF_PACKET, ETH_P_ALL)

```
s = socket(PF_PACKET,
           SOCK_RAW,
           htons(ETH_P_ALL))
```

Description of NIC: struct ifreq <if.h>

```
struct ifreq
    {
    # define IFHWADDRLEN6
    # define IFNAMSIZ   IF_NAMESIZE
    union
    {
        /* Interface name, e.g. "en0".  */
        char ifrn_name[IFNAMSIZ];
    } ifr_ifrn;

    union
    {
        Omitting some detail…
        short int ifru_flags;
    } ifr_ifru;
};
```

Finding NIC to Listen using ioctl()

- SIOC**G*****: Getting information / parameters
- SIOC**S*****: Setting information / parameters

```
/* GET the index of corresponding NIC name */
memset(&ifreq, 0, sizeof(struct ifreq));
strncpy(ifreq.ifr_name, device,
      sizeof(ifreq.ifr_name) - 1);
if(ioctl(s, SIOCGIFINDEX, &ifreq) < 0) {
   perror("ioctl");
   close(s);
   return(-1);
}
```

Parameters for Packet Handling at a Socket

```
struct sockaddr_ll
{
   unsigned short int sll_family;
   unsigned short int sll_protocol;
   int sll_ifindex;
   unsigned short int sll_hatype;
   unsigned char sll_pkttype;
   unsigned char sll_halen;
   unsigned char sll_addr[8];
};
```

Setting the property of packet handling

```
sa.sll_family = PF_PACKET;
sa.sll_protocol = htons(ETH_P_ALL);
sa.sll_ifindex = ifreq.ifr_ifindex;

/* Reflecting the property to the raw socket */
if(bind(s, (struct sockaddr *) &sa, sizeof(sa)) < 0)
{
   perror("bind");
   close(s);
   return(-1);
}
```

Enabling “Promiscuous Mode” on the raw socket using ioctl()

- Want to Process any packet that comes and goes through the NIC
- Promiscuous Mode is set as a Flag

```
ifreq.ifr_flags = ifreq.ifr_flags|IFF_PROMISC;
if(ioctl(s, SIOCSIFFLAGS, &ifreq) < 0) {
    perror("ioctl");
    close(s);
    return(-1);
}
```

Analyzing a packet
(Selecting the printing method based on the packet type)

Getting a packet from Socket

- Whenever a packet is seen on the socket, you get it with read()

```
if((len = read(s ,buf, sizeof(buf))) <= 0){
   perror("read");
}
```

What is that “packet”?

- A packet captured at NIC is an Ethernet Frame

```
struct ether_addr
{
  u_int8_t ether_addr_octet[ETH_ALEN];
} __attribute__ ((__packed__));

struct ether_header
{
  u_int8_t  ether_dhost[ETH_ALEN]; /* destination eth addr */
  u_int8_t  ether_shost[ETH_ALEN]; /* source ether addr */
  u_int16_t ether_type;            /* packet type ID field */
} __attribute__ ((__packed__));
```

Analyzing an Ethernet Frame

- From where to where is a packet going?
- What is the contents (payload) in the frame?

- AnalyzePacket() gets the pointer to the head of "**data**" read at NIC
- Cast the data into the shape of Ethernet Frame

```
struct ether_header *eh;
eh = (struct ether_header *)ptr;

if(ntohs(eh->ether_type) == ETHERTYPE_ARP) {
    Show information in Ethernet Header
    (I'm skipping to show the contents of ARP message :P)
} else if(ntohs(eh->ether_type) == ETHERTYPE_IP) {
    Show information in Ethernet and IPv4 Header
}
```

Printing the contents of packet header

Printing 6 byte data of u_char as Ethernet Address using Hex Ascii

```
char *
my_ether_ntoa_r(u_char *hwaddr,
                char *buf, socklen_t size)
{
   snprintf(buf, size,
      "%02x:%02x:%02x:%02x:%02x:%02x",
      hwaddr[0], hwaddr[1], hwaddr[2],
      hwaddr[3], hwaddr[4], hwaddr[5]);

   return(buf);
}
```

Printing the contents of Ethernet Header

```
fprintf(fp, "ether_dhost=%s\n",
    my_ether_ntoa_r(eh->ether_dhost, buf, sizeof(buf)));
fprintf(fp, "ether_shost=%s\n",
    my_ether_ntoa_r(eh->ether_shost, buf, sizeof(buf)));

fprintf(fp, "ether_type=%02X", ntohs(eh->ether_type));
switch(ntohs(eh->ether_type)){
    case ETH_P_IP:
        fprintf(fp, "(IP)\n");
        break;
    case ETH_P_IPV6:
        fprintf(fp, "(IPv6)\n");
        break;
    case ETH_P_ARP:
        fprintf(fp, "(ARP)\n");
        break;
    default:
        fprintf(fp, "(unknown)\n");
        break;
}
```

What about IP Header?

- Similar process with Printing Ethernet Header
- Cast to IPv4 Header Structure
- Convert u_char data to the meaning full values
- Show as human readable text

Summary

- Covered
  - Setting up the NIC
  - Choosing the printing method based on the packet type (Analyze)
  - Printing the protocol specific information (Print)
- Skipped
  - Calculation of checksum
  - Many protocols that’s interesting to show
    - TCP, UDP, ARP, IPv6, etc.

Hints for Firewall Assignment

- Making my firewall using raw socket
  - This time we covered only read()
  - What to do to bridge the packet?
- Detecting port scans and DoS
  - Watching only individual packets may not help
  - How do you understand the trend of incoming traffic?

Main Function in Brief

```
int
main(NIC Name)
{
    Setup 2 NICs to Promiscuous Mode;

    While(1) {
        Read a packet from NIC1;
        Analyze the packet Contents;
        Decide forward or drop the packet;
        If (forward) {
            Write the packet from NIC2;
        } else {
            Discard the packet;
        }
    }

}
```