Laboratory 5: Software IP Router

Vaibhav Maheshwari

Malhar Trivedi

Harsh Desai

(We submitted the assignment whatever was completed before the deadline through email. Later, Professor Cho extended the deadline).So, please consider our latest deadline. Also, the earlier submitted code contained certain part from online reference (like ip header), but the new code is absolutely ours.

Objective

Build a simple IP router in user space which can capture a packet sent to it, determine the next hop and then send the packet to that next hop.

Design & Implementation

Our implementation was based on following components:

1) **Tcpdump/libpcap** to capture/sniff on the interface for packets
2) Determine the next hop for the packet using the **routing table**
3) **Tcpdump packet injection** used to send the packet to next hop

Functions:

1) Following functions in our code relate to the **capturing process**:

- pcap_findalldevs():- Find interfaces on the machine to sniff on
- pthread_create():- Create an independent pthread for each interface/device
- interfacedetail():- Callback function for each pthread created above
- pcap_lookupnet():- Get device properties
- pcap_open_live():- Open a session on given device and return its handler
- pcap_next():- Grab a packet from the session handler

2) Following functions deal with **processing the packet** once received. One important task here is extracting the source and destination addresses and looping up in the routing and ARP table for the destination IP and mac in order to send it to the next how.

- got_packet():- Process and received IP packet and send it to the appropriate destination. This functions involves below mentioned sub functions.

- pcap_open_live(): Open a session to send on the outgoing interface
- comparedetail():- For the destination IP in the packet, look up in the routing table and return the next hop IP to reach that network

- getarpentry():- Get the mac address of the next hop IP obtained above from the ARP table

3) Following function deals with **sending the packet** to the next hop

- memcpy():- Modify the source mac and destination mac fields in the Ethernet header of the to be sent packet

- pcap_inject():- Inject/send the packet to the interface whose session was opened

Data structures:

1) Ethernet header

```
struct sniff_ethernet
{
 u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination host address
u_char  ether_shost[ETHER_ADDR_LEN];    /* source host address */
u_short ether_type;                     /* IP? ARP? RARP? etc */
};
```

2) Routing table

```
struct Rtable
{
 bpf_u_int32 mask;    //mask for next hop IP
 bpf_u_int32 dest;    //destination network add
 bpf_u_int32 next;    //next hop router IP
 char interface[10]; //interface number
u_char mac[6];
 int interfaceposn;
};
```

3) ARP table

```
struct arptable
{
bpf_u_int32 destn;
 u_char macaddr[6];
 char interface[10];
};
```

Pseudo code of the each of the functions with the program flow follows:

```
int main()
{

 Read config file and populate routing table based on that

 Display current routing table
 Displace current ARP table
 Display interfaces we will be capturing packets on

 Find all devices on this machine which can be sniffed on ( we just
 concentrate on first 3 interfaces)

 for(each interface found above)
 {
      Create a pthread and pass it a callback function that deals with
      capturing packets on that interface

 }


void interfacedetail()
{


Lookup properties of the device
Open the device for sniffing.

for(loop N times)
{
    Grab a packet from current session handler
    Initialize pointers to Ethernet and IP header to access their
    fields
    Check whether it is an IP or an ARP packet and remember it

      If(IP packet)
      {
            Call got_packet() which processes the packet
      }
```

```
      Else
      {
            Call got_arp() which processes ARP packet
      }

 }


void got_packet()
{

      Open session for outgoing interface
      Declare pointers to packet headers (i.e Ethernet and IP)
      Define ethernet header


      Call comparedetail() which lookups the routing table to find next
      hop IP to reach the destination network

      Call getarpentry() which gets the mac address of the next hop IP
      from the ARP table

      Make packet. Modify source and destination mac addresses in the
      ethernet header.

      Call pcap_inject() and inject/send the packet
}


int getarpentry()
{

      for(each entry in the ARP table)
      {
            if(IP field matches the given argument)
            return index of the entry;
      }
 If nothing found,return -1
}

comparedetail()
{

      for(each entry in routing table)
    {
            Find longest prefix match entry for the IP given as
    argument

            if(entry found in routing table)
            {
                  If(First match no need to compare)
                  {
```

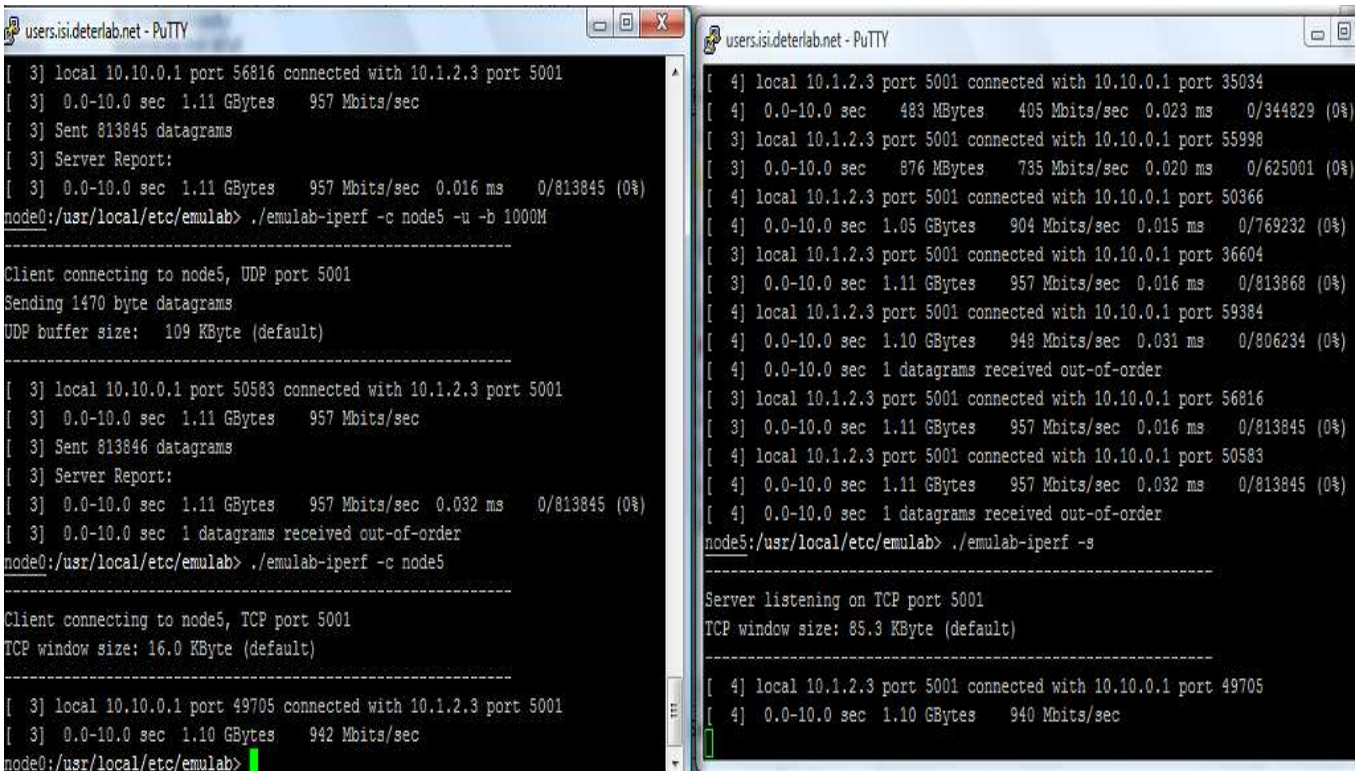
```
                        Save current next hop IP
                        Save current next hop mask
                  }
                  else
                  {
                        if(New match is longer)
                        {
                              Save next hop IP
                              Save next hop mask
                        }

                  }

            }
      }
      if(nothing found)
            return 0
      else
            Return next hop IP
   }
}
```

Observation

Only networks connected to rtr1 and rtr2 in the topology are able to ping each other before running our code. However, after running the code node0 is also able to ping node5.

TCP: 942Mbits/sec
UDP: 957Mbits/sec.

(Snapshot Attached)



CIDR

We used one CIDR route for node5 and node6. These nodes have different network address. Normally this will take one entry for each network in routing table.

In this case network is not so large so there is not much problem but when network grows rapidly, there could be serious issue of storage in routing table to accommodate these entries.

CIDR can alleviate this problem by reducing entries in router. This way it saves memory in router and operation becomes faster.

Destination	Mask	Next hop	Interface
10.1.0.0	255.255.255.0	Rtr1(10.99.0.1)	x
10.1.2.0	255.255.254.0	Rtr2(10.99.0.2)	x
10.10.0.0	255.255.255.0	-	y

Above is the example of entries in routing table at usRTR. One can notice mask for Rtr2 is not default but modified to aggregate route of node5 and node6. This will help to reduces number of entries in routing table.

References

1. http://www.tcpdump.org/#documentation
2. http://security-freak.net/packet-injection/packet-injection.html#1
3. http://beej.us/guide/bgnet/

Conclusion

Thus, we were partially successful in building a software IP router using libpcap and raw sockets. Libpcap served as a useful tool to capture the packets and dissect the headers to extract the fields. Raw sockets were also useful to create a packet from scratch by bypassing the network stack.

The assignment helped us to understand the network stack in greater detail by analyzing (and replicating in our code) the headers and the routing process.