Objective:

Build an IP router, capture a packet sent to that router, determine the next hop and then send the packet to that next hop.

**Design**

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 passed by configuration file

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 hop.

• got\_packet():- Process the 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. Compare whether the next hop address is a local IP or remote IP address.

• 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

• pcap\_inject():- Inject/send the packet to the interface

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

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 the interfaces on this machine which can be sniffed on

Create a pthread and pass it a callback function that captures packets on each interface

Lookup properties of the device

Open the device for sniffing.

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 its an IP packet process the packet else process the ARP packet

Processing a packet:

Open session for outgoing interface

Declare pointers to packet headers (i.e Ethernet and IP)

Define ethernet header

Call the function which lookups the routing table to find next hop IP to reach the destination network and checks if the next hop IP is a remote IP or a local IP.

Call the function 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.

For each entry in the ARP table if IP field matches the given argument then return index of the entry.

For each entry in routing table, find longest prefix match entry for the IP given as argument.

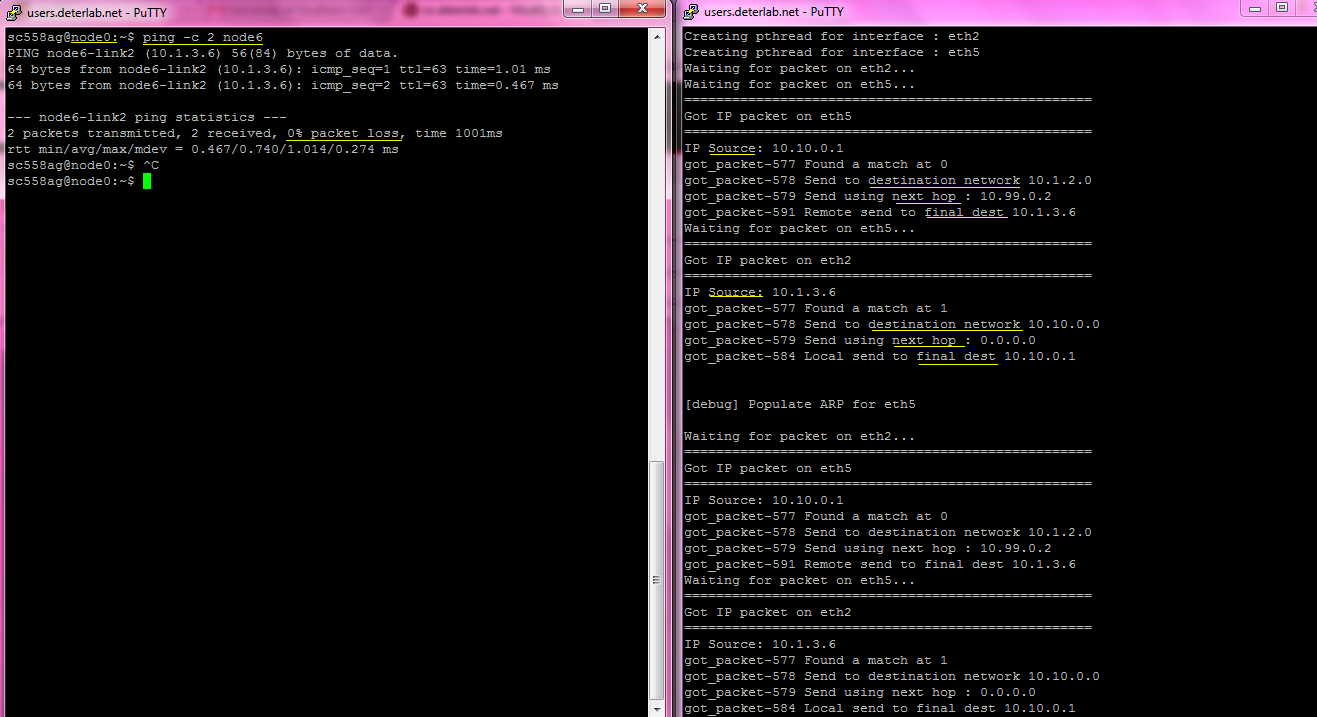
For that entry in routing table, save current next hop IP and next hop mask.

Inject/send the packet

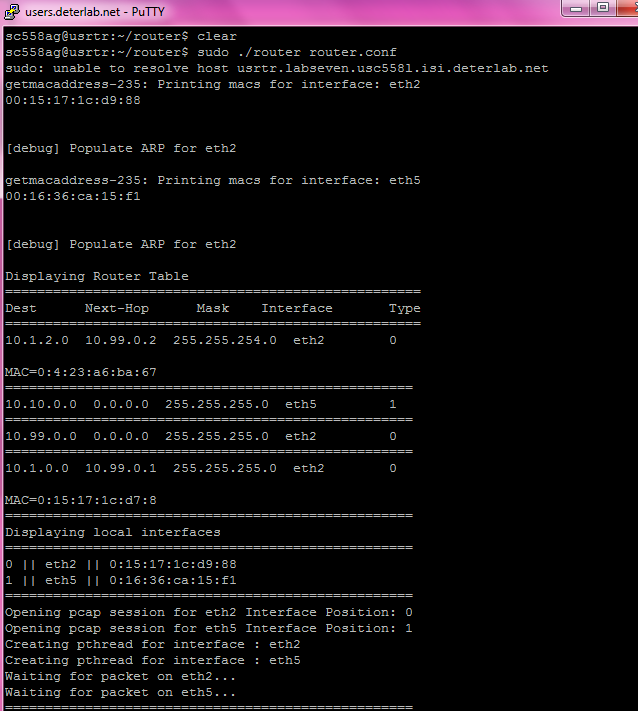
**Observation**

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 other nodes.

For example below is the snapshot of node0 pinging node6.

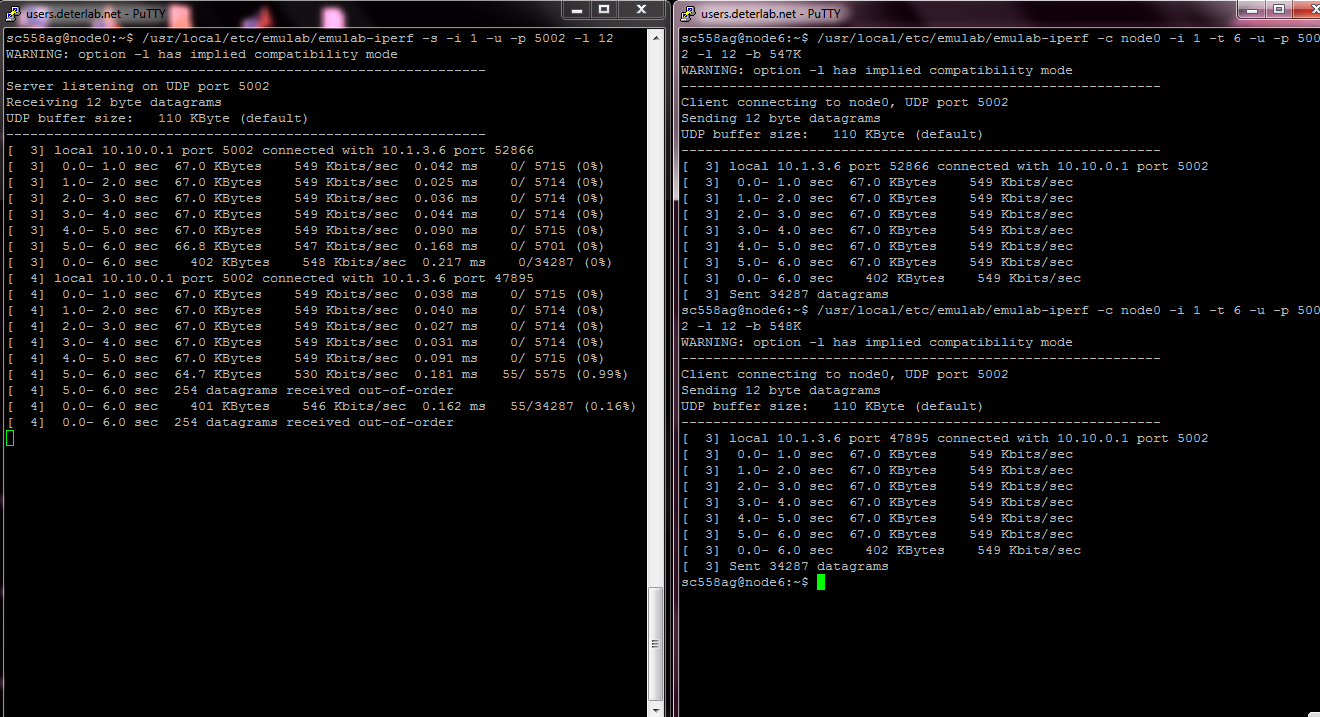


Routing table printed:



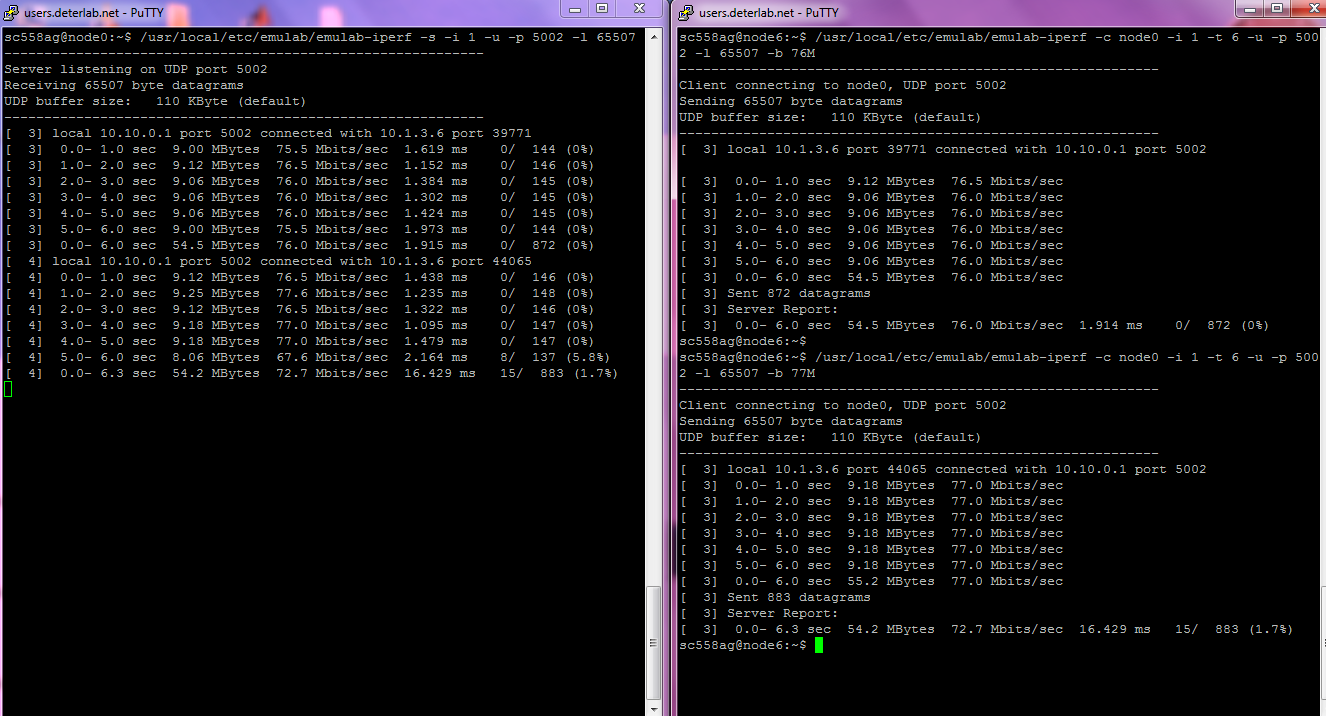
**Packets per second:**

Maximum packets per second measured by the software router as measured by iperf is **5714 packets per second**. Below is the snapshot to support it.



Maximum throughput:

Maximum throughput of the software router as measured by iperf is **76Mbps**. Below is the snapshot to support it.



The file transfer people are more interested in the size of the file transferred end to end which is mostly measured in bytes. Hence they are interested in mega=1024^2. While the network people are more interested in how many bits they transferred over the link hence they use mega=10^6.

**Stop the Linux kernel sending an ICMP \Destination Unreachable" message:**

The ICMP destination unreachable message is generated by a router to inform the source host that the destination unicast address is unreachable. These can cause a problem by the kernel because the kernel has no information related to the destination in its routing table. The routing code is on the userspace in the code. So kernel being ignorant can cause to create this misleading message being displayed “Destination Unreachable”.

To stop the Linux kernel on the usRTR node from sending an ICMP \Destination Unreachable" message to the source of a packet on our network we have used the following. It’s a simple firewall (iptables) rule

iptables -A OUTPUT -p icmp --icmp-type 3 -j DROP

(type 3 for ICMP type “Destination Unreachable”)

**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.

Configuration File:

Router.conf

10.1.2.0 10.99.0.2 255.255.254.0 ethX

10.10.0.0 0.0.0.0 255.255.255.0 ethY

10.99.0.0 0.0.0.0 255.255.255.0 ethX

10.1.0.0 10.99.0.1 255.255.255.0 ethX

Above is the Configuration file we have passed in as an argument for the code in 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://beej.us/guide/bgnet/

3. http://www.cyberciti.biz/tips/linux-iptables-9-allow-icmp-ping.html

**Conclusion**

Thus, we were 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.