Report on Programming Assignment 3

Gaurang Naik
123079009
Collaborator: Aniruddh Rao

October 7, 2014

1 Experiment Setup

The setup for the assignment is as shown in figure 1. There are two systems connected to each other over the CSE LAN interface. These machines are designated the names C (for Client) and S (for Server) respectively. The IP address of C is 10.129.5.195 and that of S is 10.129.5.194. At each machine, three virtual interfaces are created (using ifconfig eth0:x 192.168.y.x, where y=6 for S and y=7 for C and x=1,2,3). Thus, there are three virtual interfaces at each machine C (C1, C2 and C3) and S (S1, S2 and S3).

With no soultion implemented, the virtual interfaces at C cannot connect to the virtaul interfaces at S. This is because C has no idea as to what the private IPs of S mean, and vice versa. The objective of this assignment is to implement a solution at C and at S, so the virtual interfaces at C and S can communicate with each other.

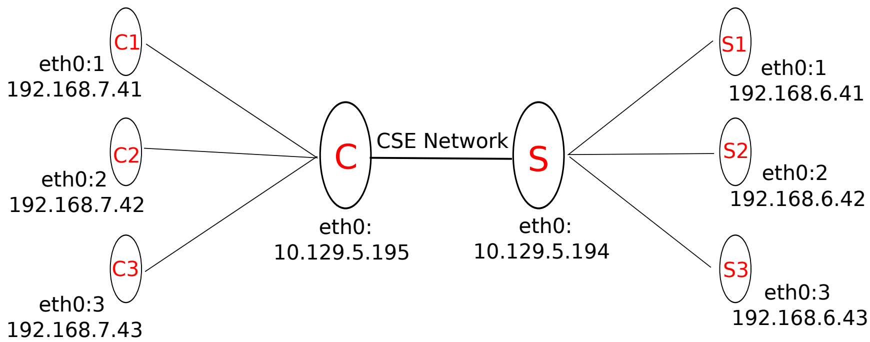

Figure 1: Topology of clients and servers

For testing the communication between the Clients and the Servers at C1, C2, C3 and S1, S2, S3, I have tested the Client Server application implemented in PA1. As mentioned in the assigment description, the client and server were modified to operate on a particular IP interface.

1.1 Solution using iptables

The first solution for providing connectivity between C1, C2, C3 and S1, S2, S3 has been implemented using iptables. The iptables commands are executed at nodes C and S.

The idea behind the iptables implementation is shown in figure 2. The entire solution can be broken down into three steps.

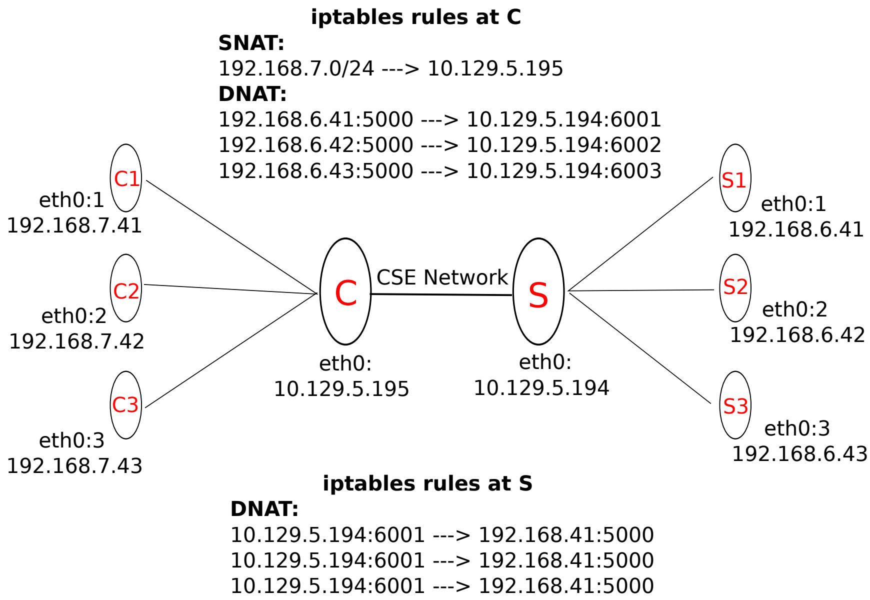

Figure 2: Solution using iptables

1. In order that the virtual interfaces of C (C1, C2 and C3) can communicate with the outside network, we first need to perform a Source Network Address Translation (SNAT) at node C. The same thing can be achieved by using MASQUERADE at node C. Once this is done, virtual interfaces of C can communicate with S.

2. Now, C has no clue what to do with packets destined to S1, S2 and S3. Therefore, a Destination NAT (DNAT) is implemented at C, which translates all packets destined to S1, S2 and S3 to the gateway S. At this stage, C1, C2 and C3 appear to communicate with S1, S2 and S3. However, they simply talk to the gateway S. Each virtual interface is identified uniquely by using different port numbers while performing DNAT. As shown in figure 2, any packet arriving from 192.168.6.41:5000 will be translated to 10.129.5.194:6001; any packet arriving from 192.168.6.42:5000 will be translated to 10.129.5.194:6002 and so on.

3. Now that packets destined to S1, S2 and S3 reach S successfully, a DNAT is implemented at S. Any packet arriving at port 6001 of S is translated to 192.168.6.41:5000, any packet arriving at port 6002 of S is translated to 192.168.6.42:5000 and so on. Thus, packets destined to S1, S2 and S3 arrive at the respective interfaces correctly.

Note that the solution is very rigid. I was unable to make a more generic solution. The servers have to be necessarily listening on port 5000 to make communication possible.

1.2 Solution using tun

The second solution for providing connectivity between C1, C2, C3 and S1, S2, S3 has been implemented using the tun interface. tun interfaces were created nodes C and S.

An overview of the working of this solution is shown in figure 3.

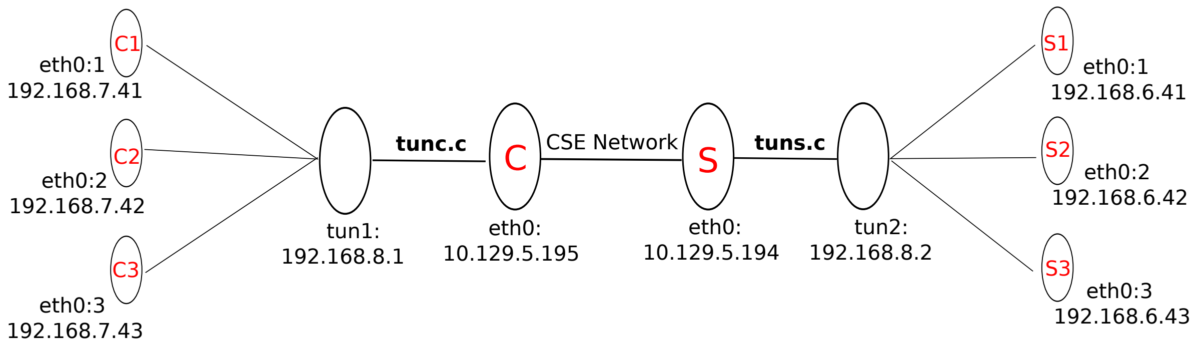

Figure 3: Soultion using tun

The virtual interfaces now do not communicate with the inteface eth0. Instead, *all packets from the virtual interfaces are routed to the tun interface using the route command.* At C and S, a user space application runs which simply exchanges data between the tun interface and the eth0 interface. This application is simply a modification of the sample code provided as reference in the assignment description.

A script file is used to create a persistent tun interface at C and S. A *select()* loop runs in the application which checks if any data arrives on the tun or eth0 interface. The application uses the tun file descriptor and the socket descriptor and to check data arriving on either interface. Any data received from the tun interface is sent to the eth0 interface and vice versa.

Any IP packet sent to the tun interface is captured by the user space application, which sends this IP datagram over a TCP socket. As a result, one IP datagram gets encapsulated into another IP datagram, hence the tunneling. At the receiver side, the user space application reads data from eth0 interface over the TCP socket and sends this data to the tun interface. This is how the decapsulation occurs. The tun interface sends the packet to the appropriate destination virtual interface.

2 Wireshark Analysis of iptables implementation

The Wireshark capture for the iptables implementation at S is as shown in figure 4.

As seen in figure 2, all packets to be sent to 192.168.6.4x are sent to port 600x (x = 1, 2, 3) of S. This can clearly be seen in figure 4. The communication happens between ports 6002 and 6003 (6001 not seen in image) of S and randomly allocated ports of C. This is as a result of iptables DNAT implementation at C. Also, due to the SNAT implementation at C, all packets arriving from C1, C2 and C3 seem to arrive from C (10.129.5.195). Finally, the DNAT implementation at S sends the packet to appropriate interface (S1, S2 or S3).

The highlighted packet in the trace contains a response from the server (of PA1). The contents at the bottom show the response (Ok anirudh).

3 Wireshark Analysis of tun implementation

The Wireshark capture for the tun implementation at S is as shown in figure 5 and 6.

Here, the captures have been taken on the tun interface and also on the eth0 interface. The eth0 interface capture (figure 5) shows that the TCP connection was set up only once between C and S. At the tun interface (figure 6), a TCP connection using SYN, SYN ACK and ACK was set up for each flow (C1-S1, C2-S2, C3-S3).

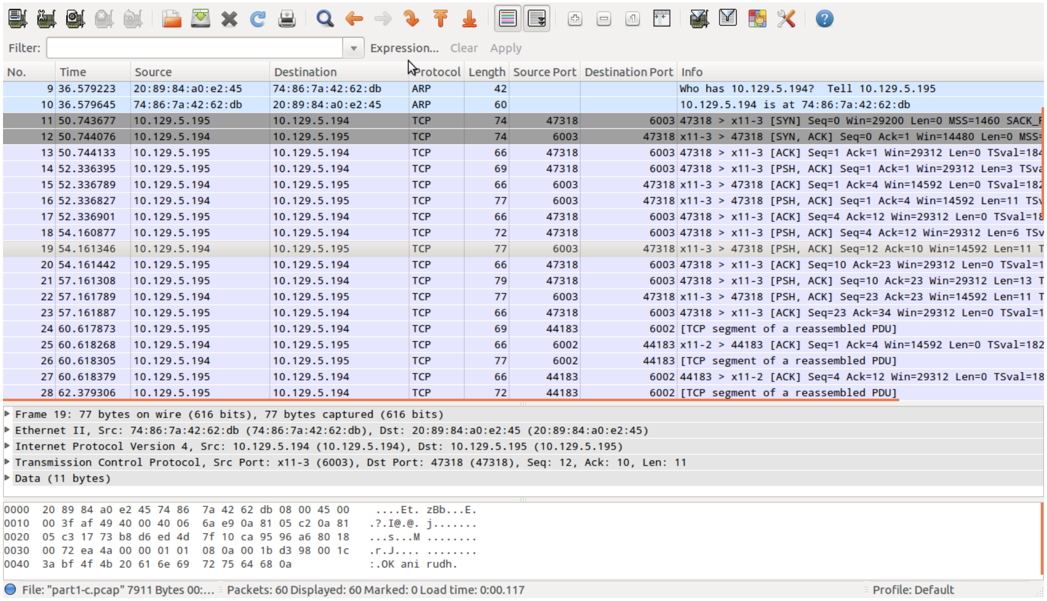

Figure 4: Wireshark capture for iptables (client)

Figure 6 clearly shows communication occuring between the virtual interfaces of C and S. Packets are sent and received by IP addresses 192.168.6.4x and 192.168.7.4x (x = 1, 2, 3).

4 Comparison of two solutions

As mentioned before, the solution using iptables is rigid. The code written allows for communication between the client and server only on port 5000 of the virtaul interfaces. That is, S1, S2 and S3 can hear only on port 5000. And packets sent from C1, C2 and C3 will go through only for those destined to port 5000 of S1, S2 and S3. However, a more generic code could be written for the same purpose making the code more flexible. Also, the solution provided would only work for TCP connections. In order to specify the ports, iptables command require you to sepcify the protocol. In order to use another protocol, say ping, instead of TCP, the protocol needs to be changed to icmp instead of TCP.

As far as the signalling overheads are concerned, when the iptables solution is used, C and S would have to establish a TCP connection for each client-server pair. If there are large number of clients and servers, then this overheads would be significant. On the other hand, using the tun interface, C and S set up a TCP connection only once. TCP connections between the virtual interfaces of C and S are established through the tunneled packets. So all TCP SYN, SYN ACK and ACK packets between say C1 and S1 appear as normal IP packets at C and S eth0 interface. The actual number of TCP connections established in the tun-based solution is one more than the ip-tables based solution (1 TCP connection between C and S, and 1 TCP connection for each virtual client-server pair). As a result, the total number of packets sent and received by C and S is more in the tun based solution than the IP based solution. Thus, if signalling overheads at C and S are concerned, the overheads are larger in iptables-based solution. However, the actual number of packets exchanged between C and S are more in the tun-based solution.

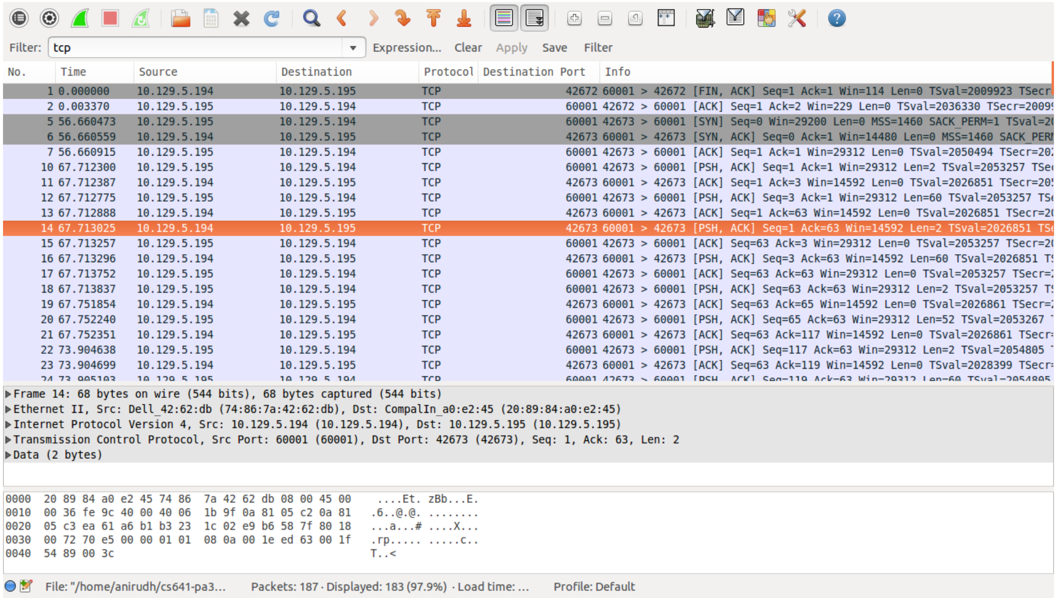

Figure 5: Wireshark capture for tun (client) at eth0 interface

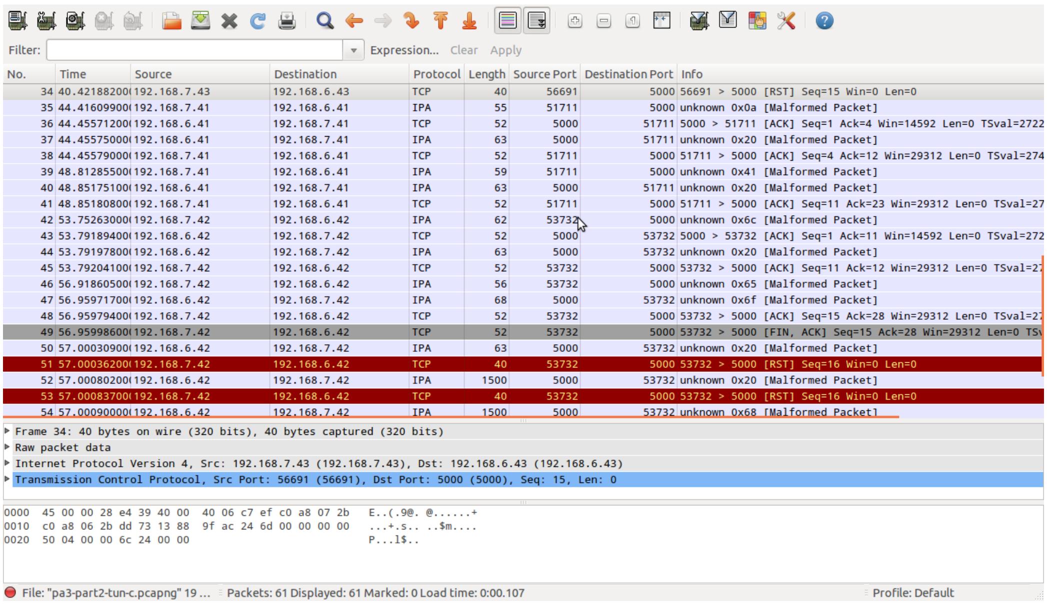

Figure 6: Wireshark capture for tun (client) at tun2 interface