ASU ID: 1223441505
ASURITE ID: vyaganti
GROUP NUMBER: 29

CSE 434

COMPUTER NETWORKS

SOLUTIONS *for* LAB 02

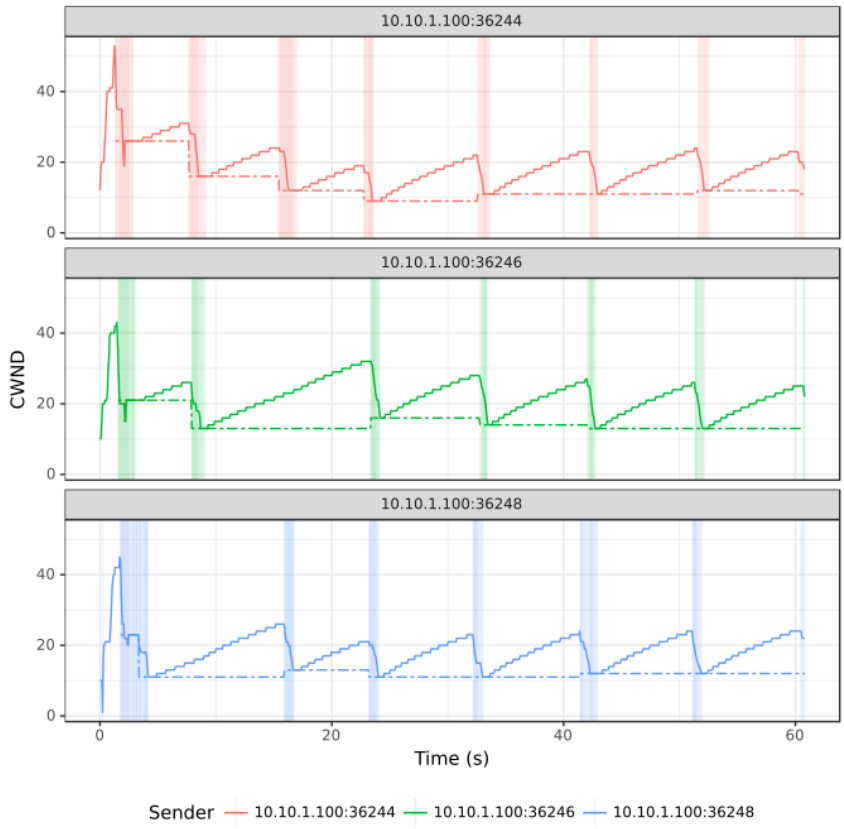
Exercise 1.1: Configure the topology as described in the "Run my experiment" section of the tutorial. Follow the section "Exercise" to create a plot of the congestion window size for each TCP Reno flow over the duration of the experiment (see Fig. 1). Use the script provided to analyze the results of your experiment. Download the `sender-ss.svg` image file to your computer.

Annotate your plot (see Fig. 2) to show the periods of slow start and congestion avoidance, any instances where multiple duplicate ACKs were received (which trigger fast recovery), and any instances of a timeout.

For your lab report:

1. Include your annotated plot of the three flows sharing the bottleneck link using TCP Reno.
2. Using your plot and/or experiment data, explain the behaviour of TCP Reno in the slow start and congestion avoidance phases. In addition, explain what happens to both the congestion window and the slow start threshold when multiple duplicate ACKs are received.

1.1.1



ASU ID: 1223441505
ASURITE ID: vyaganti
GROUP NUMBER: 29

1.1.2.

TCP behavior for slowstart: The actual slow-start mechanism is to increment cwnd by 1 for each ACK received. This seems linear, but that is misleading: after we send a windowful of packets (cwnd many), we have received cwnd ACKs and so have incremented cwnd-many times, and so have set cwnd to (cwnd+cwnd) = 2×cwnd. In other words, cwnd=cwnd×2 after each *windowful* is the same as cwnd+=1 after each *packet*.

TCP behavior congestion avoidance: The central strategy is that when a packet is lost, cwnd should decrease rapidly, but otherwise should increase "slowly". This leads to slow oscillation of cwnd, which over time allows the average cwnd to adapt to long-term changes in the network capacity.

The Fast Retransmit strategy is to resend Data[N] when we have received three(multiple) duplicate ACKs for Data[N-1]; that is, four ACK[N-1]'s in all. Because this represents a packet loss, we also set ssthresh = cwnd/2, set cwnd=1, and begin the threshold-slow-start phase. The effect of this is typically to reduce the delay associated with the lost packet from that of a full timeout, typically 2×RTT, to just a little over a single RTT. The lost packet is now discovered before the TCP pipeline has drained. However, at the end of the next RTT, when the ACK of the retransmitted packet will return, the TCP pipeline will have drained, hence the need for slow start.

Exercise 1.2: Under the "Optional: Other Congestion Control Algorithms" section of the tutorial, repeat the steps in the "Generating Data" section using the TCP Cubic variant. Download the updated `sender-ss.svg` image file to your computer.

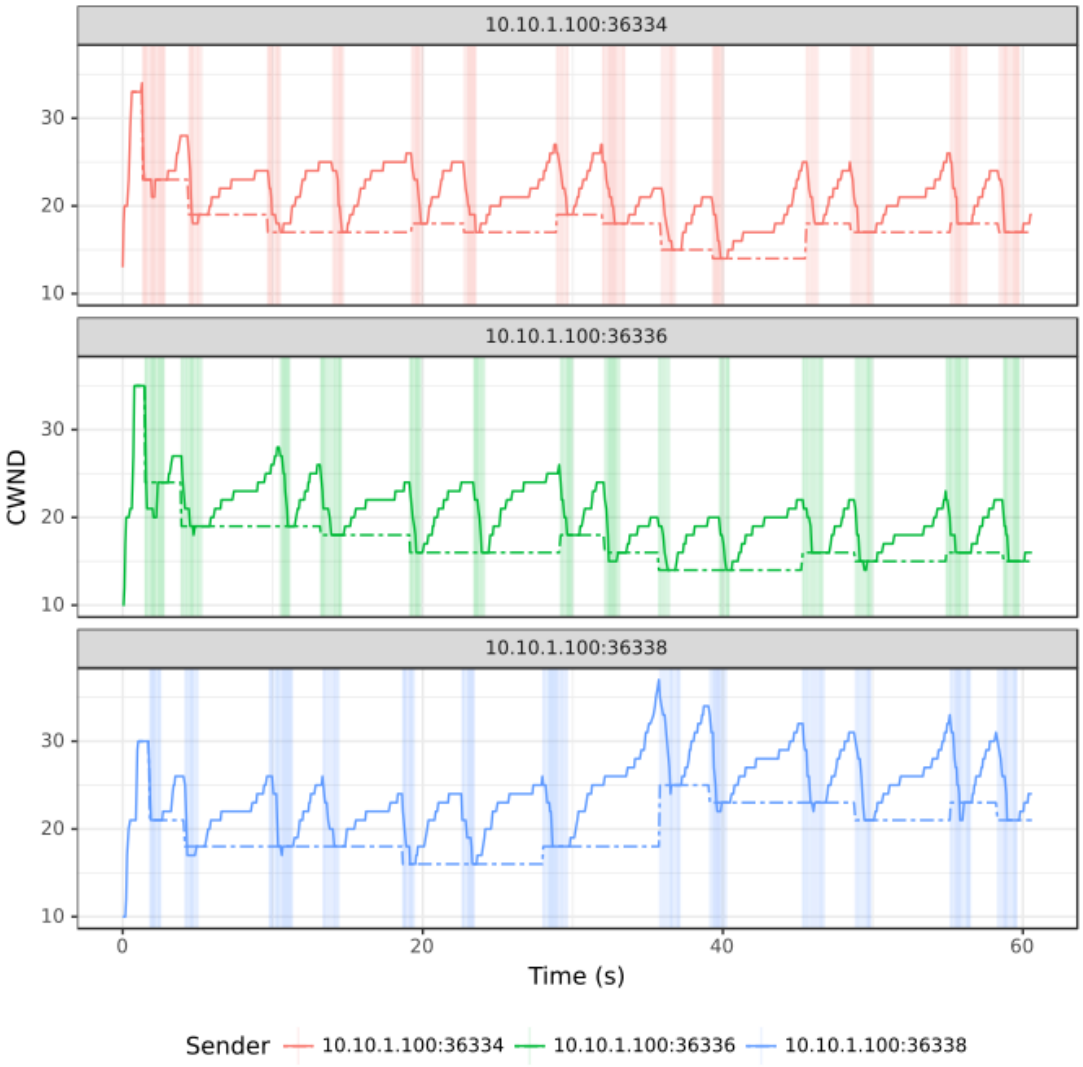
Study the TCP Cubic algorithm and use it to annotate your plot, as you did in Exercise 1.1 for TCP Reno.

For your lab report:

1. Include your annotated plot of the three flows sharing the bottleneck link using TCP Cubic.
2. Using your plot and/or experiment data, explain the behaviour of TCP Cubic.
3. Why does TCP Cubic reach the available bandwidth faster than TCP Reno?

ASU ID: 1223441505
ASURITE ID: vyaganti
GROUP NUMBER: 29

1.2.1.



1.2.2

CUBIC is a less aggressive and more systematic derivative of BIC TCP, in which the window size is a cubic function of time since the last congestion event, with the inflection point set to the window size prior to the event. Because it is a cubic function, there are two components to window growth. CUBIC spends a lot of time at a plateau between the concave and convex growth region which allows the network to stabilize before CUBIC begins looking for more bandwidth. Another major difference between CUBIC and many earlier TCP algorithms is that it does not rely on the cadence of RTTs to increase the window size.

ASU ID: 1223441505
ASURITE ID: vyaganti
GROUP NUMBER: 29

1.2.3

Because Reno uses the traditional linear increase (W=W+1), Cubic implements a binary search increase which can reach the available bandwidth much faster than Reno.

Exercise 1.3: Under "Optional: Low Delay Congestion Control" section of the tutorial, repeat the steps in the "Generating Data" section using the TCP Vegas variant.

Make a note of the `iperf3` throughput and the RTT estimated by `ping` during the TCP Vegas flow.

TCP Vegas does not work well when it shares a bottleneck link with a flow using a loss-based algorithm, e.g., TCP Reno. To help understand the problem, you will send one TCP Reno flow and one TCP Vegas flow through the bottleneck router.

For your lab report:

1. Make a note of the throughput reported by `iperf3` for the TCP Reno flow and the TCP Vegas flow.
2. Using your throughput measurements, and knowledge of the congestion control mechanisms used by each algorithm, can you explain why TCP Vegas does not work well in this scenario?

1.3.1

Throughput reported by iperf3 for the TCP Reno flow:

Interval	Throughput
0-1 sec	3.03 Mbits/sec
30-31 sec	950 Kbits/ sec
59-60 sec	973 Kbits/ sec

Throughput reported by iperf3 for the TCP Vegas flow:

Interval	Throughput
0-1 sec	1.88 Mbits/sec
30-31 sec	34.8 Kbits/ sec
59-60 sec	34.8 Kbits/ sec

1.3.2

TCP Vegas is that it does not work well when it shares a bottleneck link with a TCP Reno flow.

ASU ID: 1223441505
ASURITE ID: vyaganti
GROUP NUMBER: 29

Exercise 1.4: Under "Optional: Explicit Congestion Notification (ECN)" section of the tutorial, use active queue management to configure a queue in both directions on the router, and enable ECN in TCP on each host. Prepare to capture the TCP flow on each host, and then start the experiment. When the experiment finishes, transfer the packet captures (*i.e.*, the `.pcap` files) to your computer using `scp`.

For your lab report:

1. Compare the delay performance of Reno with ECN to that from the experiment (Exercise 1.3) showing the delay performance without ECN.
2. Look for the ECN-related fields in the IP header and TCP header, during connection establishment and during data transfer. Annotate your packet capture files by drawing a circle or a box around the fields to show:
   (a) The ECN handshake, *i.e.*, the `ECN-setup SYN` packet and corresponding `ECN-setup SYN-ACK`.
   (b) Select a data packet and show the two ECN bits in the IP header set to either either 01 or 01.
   (c) Find an instance of a "congestion experienced" signal, *i.e.*, the two ECN bits in the IP header set to 11, and subsequent "congestion window reduced" (CWR) flag in the TCP header of the next packet.

   Note: You may cut/paste select portions from your `.pcap` files; there is no need to include the entirety of each trace file.

1.4.1

Compared to Reno without ECN with ECN, tcp had relatively higher performance. With 3.88 Mbits/sec without ECN and 1.38 MBits/sec with ECN.

1.4.2

a.

b.

c.

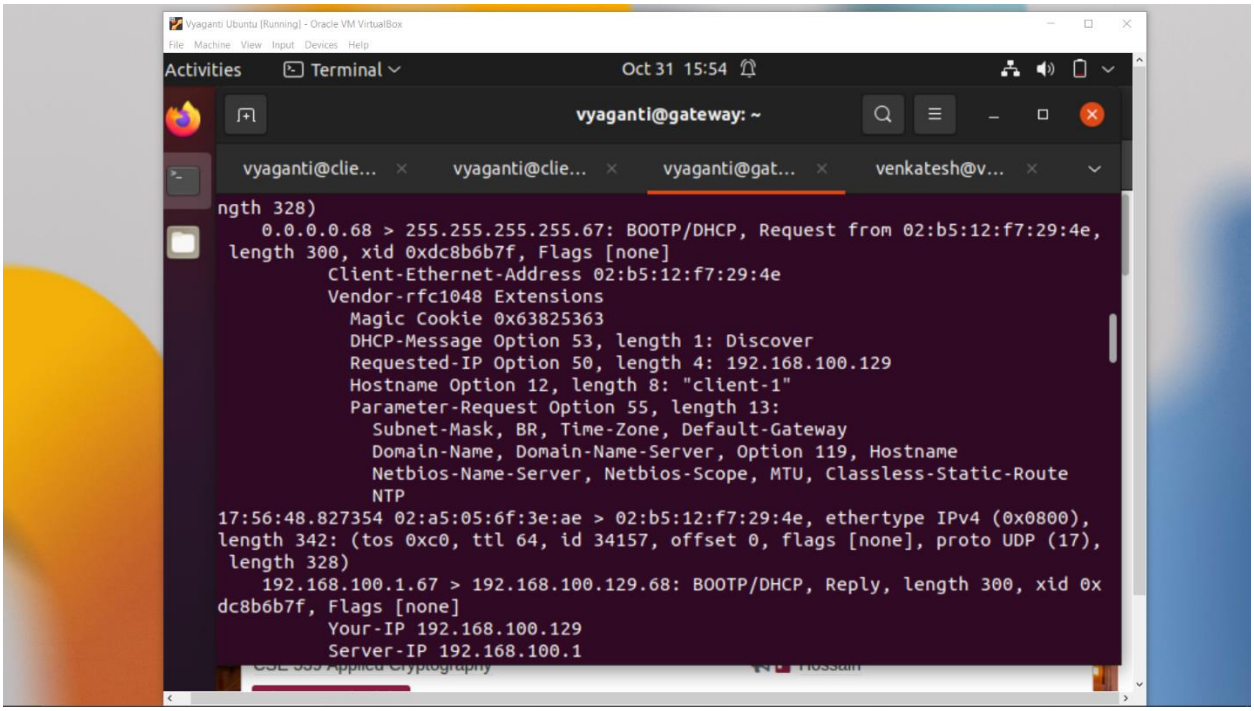
If the queue at the router is starting to fill up - then it sets the two ECN bits in the IP header to 11 before forwarding the packet to the destination. This is a "Congestion Experienced" signal.

Exercise 2.1: Reset the configuration of the clients to mimic connecting to a residential gateway. Follow the instructions in the section "Observe a DHCP Request and Response."

For your lab report:

1. Take a screen shot showing the DHCP discover message sent by the client to try and find DHCP servers on the LAN. What are the source and destination IP addresses in this request? Why are these addresses used?
2. Take a screen shot showing the DHCP offer sent by the server. What IP address does the server offer? What is the range of addresses that the server in our experiment may offer? (You can refer to the `dnsmasq` configuration file.)
3. Take a screen shot showing the DHCP request sent by the client. What is the destination address in this request? Why?
4. Take a screen shot showing the DHCP acknowledgement sent by the server to complete the configuration.
5. Take a screen shot showing the configuration of the client's `eth1` interface after receipt of the DHCP acknowledgement. Annotate your screenshot by drawing a circle or a box around the configuration to show the IP address and subnet mask the client is using. Does this correspond to the IP address in the request and in the acknowledgement? What, if any, other information did the server offer to the client?

2.1.1 Discover

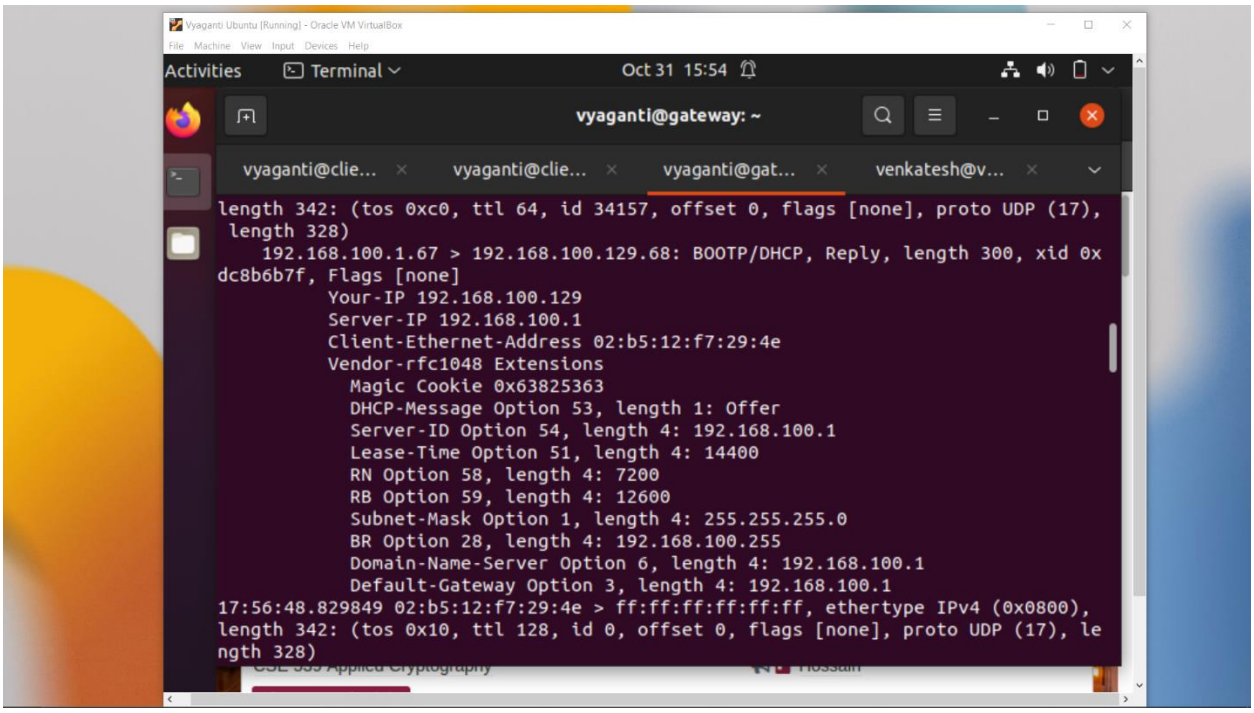


Source: 192.168.100.129

Destination: 192.168.100.1

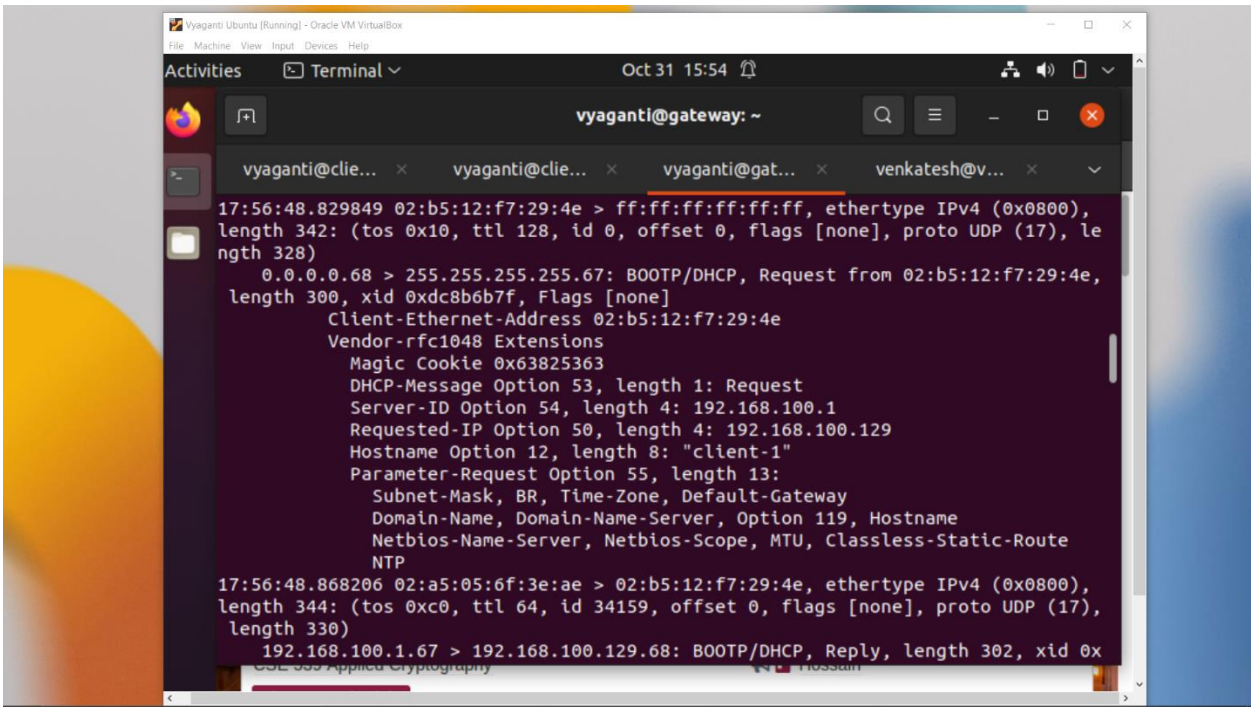
These addresses are used to sent and identify the message that the client sends. That is to discover the message client send to server.

2.1.2 Offer



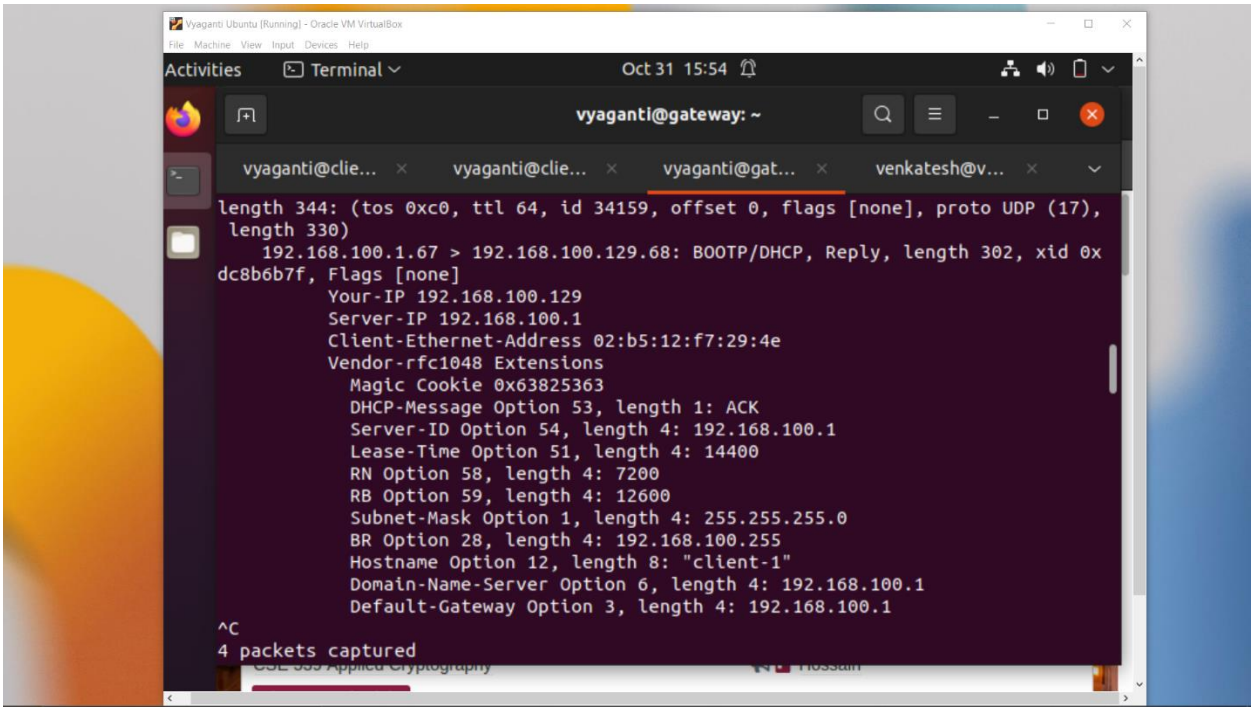
Server IP: 192.168.100.129

2.1.3 Request

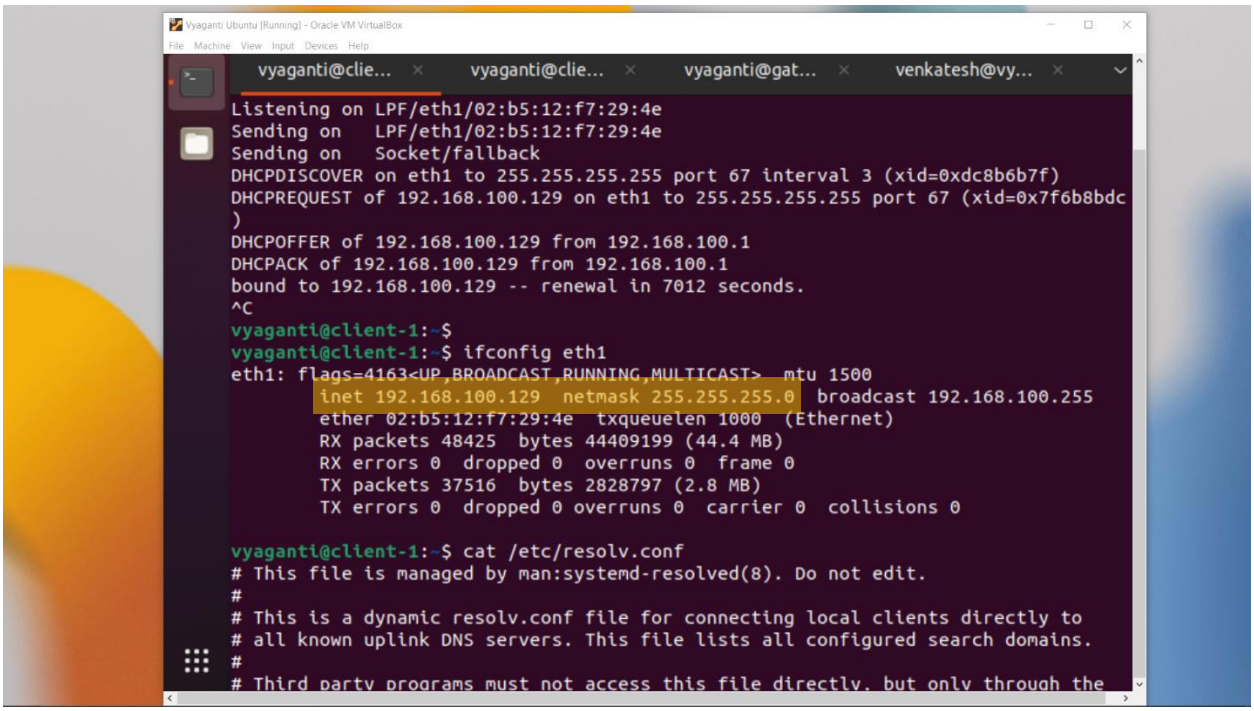


Destination IP: 192.168.100.129

2.1.4 ACK



2.1.5



The IP address is same with Request and ACK (192.168.100.129). If the address is NOT same then the communication would not happen.

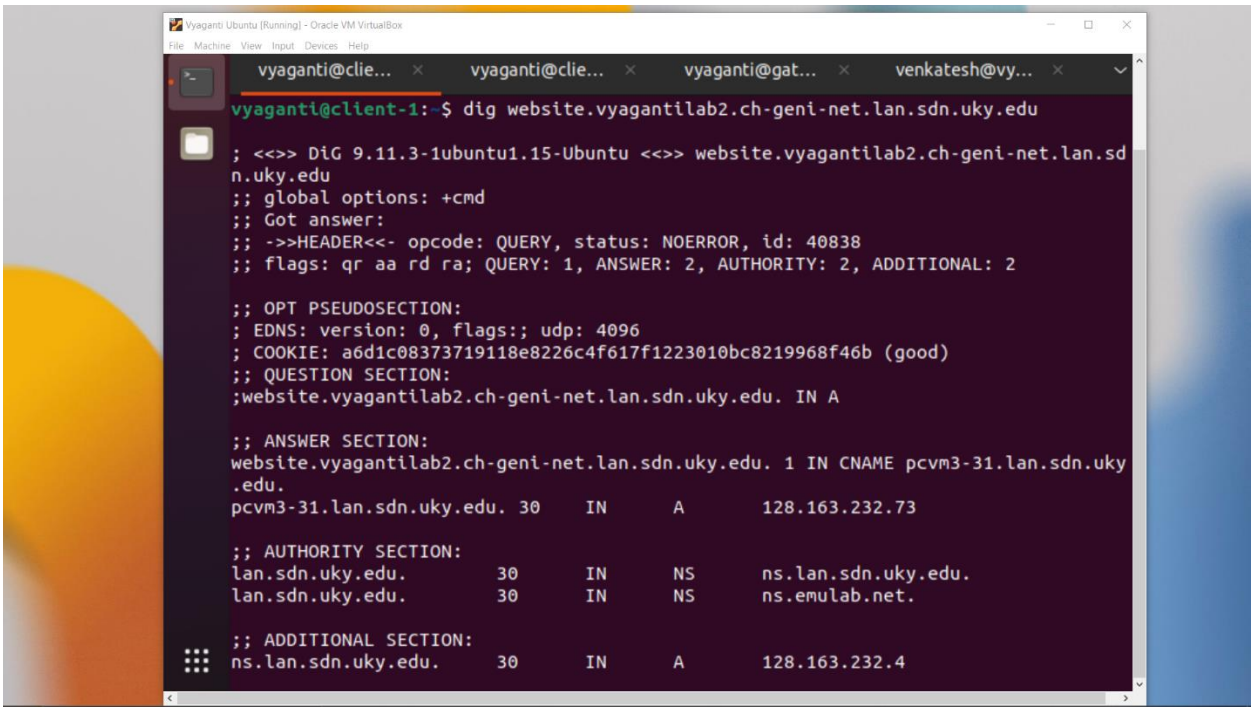
Exercise 2.2: Now follow the instructions in the section "Observe a DNS Query and Response."

For your lab report:

1. For the basic DNS resolution (not the one with `+trace`) take a screen shot showing the `dig` command and its output. Also show the DNS query and response from the `tcpdump` output. Answer the following questions using the output gathered (no explanation is required).
   (a) What is the hostname that you tried to resolve?
   (b) What is the DNS record type associated with your query? (See this list of DNS record types.)
   (c) What is the address for the hostname you asked to resolve?
   (d) Give the name of the first "authoritative" server listed for this name, and the IP address of that "authoritative" server.
   (e) What is the IP address of the server that the DNS response is from?
2. For the hierarchical DNS resolution with `+trace`, take a screen shot of the `dig` command and its output. Draw a diagram showing how the hostname was resolved recursively, starting from the implied "." at the end and moving toward the beginning.
   (a) At the top, show the name servers for the root domain. Highlight the one that you queried for the top-level domain (as shown in the `dig +trace` output).
   (b) At the next level, show the name servers for the top-level domain. Highlight the one that you queried for the second-level domain.
   (c) At the next level, show the name servers for the second-level domain. Highlight the one that you queried for the subdomain.
   (d) Repeat until you have shown how the complete hostname is resolved.

2.2.1

ASU ID: 1223441505
ASURITE ID: vyaganti
GROUP NUMBER: 29



a. vyagantilab2.ch-geni.net.lan.sdn.uky.edu is an alias for the host with CNAME (canonical name) for pcvm3-31.ln.sdn.uky.edu

b. Record type: A (RFC 1035)
c. 192.168.100.129
d. Authoritative hostname: ns.lan.sdn.uky.edu
   Authoritative IP: 128.163.232.4
e. 128.163.232.73

Exercise 2.3: Follow the instructions in the section "Use NAT" set up the `gateway` to use NAT.

For your lab report:

1. Take screen shots to show the three-way TCP handshake for a connection between `client` and `website` as seen by `tcpdump` at the website, and as seen by `tcpdump` at the gateway (on the LAN). Make sure you can see the IP addresses and port numbers used in the connection.
2. Draw a diagram showing how NAT is used between `client` and `website`, similar to this diagram but with the IP addresses, hostnames, and ports from *your* experiment.

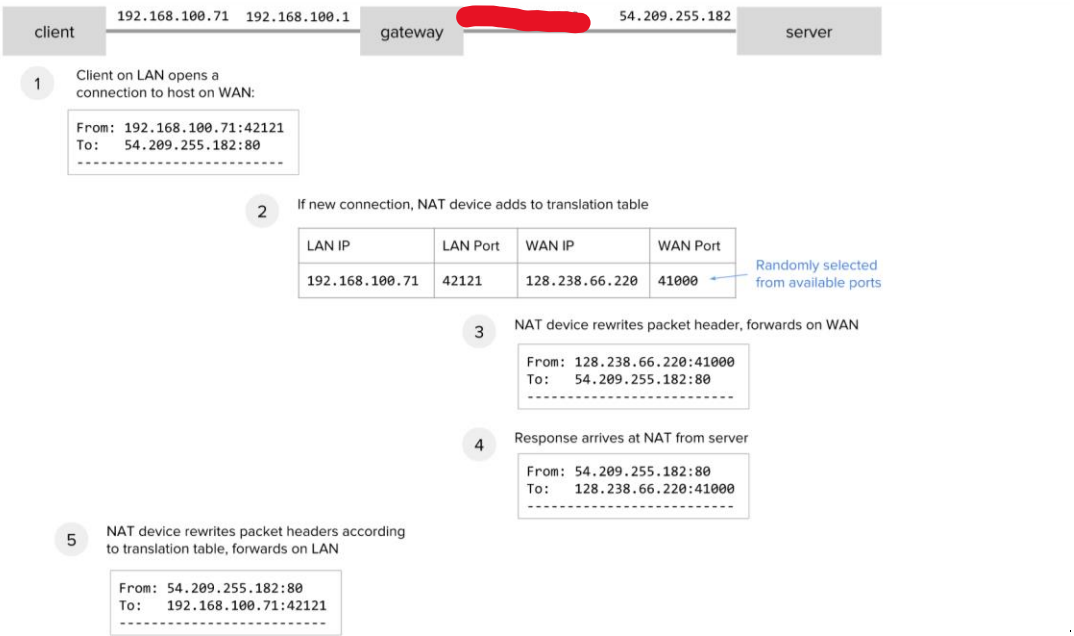
2.3.1

```
}vyaganti@gateway:~$ sudo tcpdump -i eth1 -n "tcp port 80"
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
18:05:50.630522 IP 192.168.100.129.43928 > 128.163.232.73.80: Flags [S], seq 150
7335844, win 64240, options [mss 1460,sackOK,TS val 1218233769 ecr 0,nop,wscale
7], length 0
18:05:50.631378 IP 128.163.232.73.80 > 192.168.100.129.43928: Flags [S.], seq 16
```

```
vyaganti@website:~$ sudo tcpdump -i eth0 -n "tcp port 80"
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:05:50.632948 IP 172.17.2.8.43928 > 128.163.232.73.80: Flags [S], seq 15073358
44, win 64240, options [mss 1460,sackOK,TS val 1218233769 ecr 0,nop,wscale 7], l
ength 0
```

```
vyaganti@client-1:~$ lynx http://pcvm3-31.lan.sdn.uky.edu
vyaganti@client-1:~$ ls
```

2.3.2



128.238.66.220 : 128.163.232.73

192.168.100.71: 192.168.100.129

ASU ID: 1223441505
ASURITE ID: vyaganti
GROUP NUMBER: 29

42121 : 40838

54.209.66.220 : 192.168.100.4

41000 : 67

192.168.100.1 : 192.168.100.1