

EE542 - Lab02

Building a Server-Router-Client topological model on AWS

Topics:

- Initializing Virtual Private Cloud (VPC) service and EC2 instances on AWS
- Using AWS VPC services and EC2 instances to build a Server-Router-Client network model
- Exploring to configure routing settings for different components in Computer Networks
- Learning to install iPerf and iPerf3 tools on Linux to measure the bandwidths of computers' network access

Author: [Boyang Xiao](#)

- **USC id:** 3326-7302-74
- **Email:** boyangxi@usc.edu
- **Github:** [here](#)

Dev Environment:

- **AWS account:** Free tier
- **Installed VM OS:** [vyOS 1.3](#), [Ubuntu 18.04 LTS](#)

Video Introductions

Creating VPC and EC2 instances on AWS

VPC configuration

Click on AWS VPC console and create a new VPC by clicking the button on the upper-right corner

Configurations for VPC (as shown in the following picture):

- **Name:** ee542-lab02
- **IPv4 CIDR:** 10.0.0.0/16
- **Disable** IPv6 CIDR

Click on the subnets tab on the left side-bar. Add four subnets to the VPC, partitioning the IPv4 CIDR into four domains:

- Subnet-1: 10.0.1.0/24
- Subnet-2: 10.0.2.0/24
- Subnet-3: 10.0.3.0/24
- Subnet-4: 10.0.4.0/24

VPC > Your VPCs > vpc-081501e5310406ca9

vpc-081501e5310406ca9 / ee542-lab02 Actions

Details Info

VPC ID vpc-081501e5310406ca9	State Available	DNS hostnames Disabled	DNS resolution Enabled
Tenancy Default	DHCP option set dopt-0f37ee9f080c1b4c1	Main route table rtb-0c7699dd249390132	Main network ACL acl-085e97c7983ee477f
Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool -	IPv6 CIDR (Network border group) -
Route 53 Resolver DNS Firewall rule groups -	Owner ID 949927252648		

Get Elastic IP addresses assigned by AWS

An [elastic ip](#) address is a static IPv4 address designed for dynamic cloud computing. An Elastic IP address is allocated to your AWS account, and is yours until you release it.

Click the Elastic IP tab on the left side-bar. Allocate **three elastic ip addresses** for further use.

These three elastic ip addresses can be the public IPv4 addresses for the ServerVM, RouterVM and ClientVM, which can be used to access the open Internet by the virtual machines.

! When the elastic ip addresses are associated with an EC2 instance, it will be billed a little amount of fee, even if the instance is closed. Please release the elastic ip addresses if they are not in use anymore to avoid more billings!!

Elastic IP addresses (3/3) Actions Allocate Elastic IP address

Filter Elastic IP addresses

<input checked="" type="checkbox"/>	Name	Allocated IPv4 address	Type	Allocation ID	Reverse DNS record
<input checked="" type="checkbox"/>	lab02-ip-vyos	52.37.148.54	Public IP	eipalloc-06ecaebfaa18c1d6	-
<input checked="" type="checkbox"/>	lab02-ip-ubuntu-client	44.238.64.196	Public IP	eipalloc-0b95293ad955c27eb	-
<input checked="" type="checkbox"/>	lab02-ip-ubuntu-server	52.38.44.116	Public IP	eipalloc-07cc118a4990f9127	-

Elastic IP addresses: 52.38.44.116, 52.37.148.54, 44.238.64.196

Create EC2 (Elastic computing clouds) instances

[Amazon Elastic Compute Cloud \(Amazon EC2\)](#) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

- Create one instance and install vyOS 1.3 in it as a RouterVM.
- Create two instances and install Ubuntu 18.04 in them as a ClientVM and a ServerVM.

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State
<input type="checkbox"/>	ubuntu-lab02-server	i-0269f425e12e7f0a8	t2.micro	us-west-2a	running
<input type="checkbox"/>	ubuntu-lab02-client	i-02f89f11a5753b5c6	t2.micro	us-west-2a	running
<input type="checkbox"/>	vyOS-lab02-test	i-0f4ae597a0ebfe84d	t3.small	us-west-2a	running

Building a Server-Router-Client model on AWS

Configure routing relations for EC2 instances

Configure two network interfaces for each instance, one for SSH access, the other for internal networks access.

- The network interfaces which are connecting to the outside internet **should be associated with one of the elastic IP addresses** we got. The instance can use this interface to connect to the open Internet domains and can also be accessed by outside SSH clients.
- The network interfaces which are used for internal communications should **share the same subnet domains** for three instances so that they can access to each other in this same subnet domains. Here, we chose 10.0.2.0/24 domain as the internal networks domain.

The mapping relations between each instance's network interfaces is shown below.

Instance name	OS	Network interface #	Subnet domain	Mapping to	Usage
vyOS-lab02-router	vyOS 1.3	eth0	10.0.1.0/24	Elastic ip #1: lab02-ip-vyos	SSH access from outside
...	...	eth1	10.0.2.0/24	Client and Server VMs	Internal communications
Ubuntu-lab02-Client	Ubuntu 18.04	eth0	10.0.2.0/24	Router VM	Internal communications
...	...	eth1	10.0.3.0/24	Elastic ip #2: lab02-ip-client	SSH access from outside
Ubuntu-lab02-Server	Ubuntu 18.04	eth0	10.0.2.0/24	Router VM	Internal communications
...	...	eth1	10.0.4.0/24	Elastic ip #3: lab02-ip-server	SSH access from outside

Launch the instances and access them by SSH

Create and launch three instances and wait for them to be prepared. When the instances are ready and the elastic ip addresses are configured right, we should be able to ping these three elastic ip addresses well. [Here](#) is a little online Ping tool we can use to test that.

Authentication: When creating the instances, AWS should create a key pair for them. This is for authentication when logging into the instances, so we don't need to input user/passwd. This should be a [*.pem] file and please keep it safe on the local end.

Open three terminal windows on the local end. Enter the folder where the .pem file is, and input the following command to SSH into the instances' OS:

```
ssh -i "lab02.pem" <OS name>@<Elastic ip address>
```

For example: If we want to ssh into the vyOS router vm, we can execute:

```
ssh -i "lab02.pem" vyos@52.37.148.54
```

Or if we want to ssh into the Server VM, we can :

```
ssh -i "lab02.pem" ubuntu@52.38.44.116
```

In this way, we can access and operate on the instances operating systems we created.

⚠ **There is one thing we should be careful with:**

If the Elastic IP is associated with a eth1 interface of a instance, we may not be able to access to it through SSH, since the initialized Linux VM has no open& running eth1 interface but only has eth0. We can associate the elastic ip address with eth0 and get into the instance first. Than we open up eth1 and configure its IPv4 address as allocated by AWS and set the routing rules. We can refer to [this post](#) to set the rules.

Test networks connections for the model and set routing rules

Goal:

- Make these three Vms can connect with each other under the same subnet domain.
- Build the routing model: When Server and Client communicates with each other, the traffic should be like:
 - Server -> Router -> Client
 - Client -> Router -> Server

Test Networks connection using Ping cmd and tcpdump cmd

Tcpdump is a very useful tool on Linux which can help us to monitor TCP/UDP packets switching status through a host. We can open up tcpdump on vyOS router VM to monitor if there is any network in/out through the router:

```
tcpdump -i eth1 host <ServerVM eth0 IP address> or host <ClientVM eth0 IP address>
```

If we directly ping ServerVM on the client VM like below.

```
ping <Server eth0 IP address>
```

We can surely success because the ServerVM and ClientVM is under the same subnet and they can surely access each other. But we can also notice that there is no networks activity on the router machine, since the Server and Client communicate with each other directly, but don't go through the router.

```

vyos@ip-10-0-1-7:~$ ssh -i "lab02.pem" vyos@52.37.148.54
Welcome to VyOS!

Check out project news at https://blog.vyos.io
and feel free to report bugs at https://phabricator.vyos.net

You can change this banner using "set system login banner post-login" command.

VyOS is a free software distribution that includes multiple components,
you can check individual component licenses under /usr/share/doc/*/copyright
Last login: Fri Sep  2 11:13:02 2022 from cpe-76-175-124-70.socal.res.rr.com
vyos@ip-10-0-1-7:~$ tcpdump -i eth1 host 10.0.2.86 or host 10.0.2.163
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes

```

```

ubuntu@ip-10-0-2-86:~$ ping 10.0.2.163
PING 10.0.2.163 (10.0.2.163) 56(84) bytes of data.
64 bytes from 10.0.2.163: icmp_seq=1 ttl=64 time=0.397 ms
64 bytes from 10.0.2.163: icmp_seq=2 ttl=64 time=0.459 ms
64 bytes from 10.0.2.163: icmp_seq=3 ttl=64 time=0.599 ms
64 bytes from 10.0.2.163: icmp_seq=4 ttl=64 time=0.534 ms
64 bytes from 10.0.2.163: icmp_seq=5 ttl=64 time=0.412 ms
64 bytes from 10.0.2.163: icmp_seq=6 ttl=64 time=0.448 ms
64 bytes from 10.0.2.163: icmp_seq=7 ttl=64 time=0.440 ms
64 bytes from 10.0.2.163: icmp_seq=8 ttl=64 time=0.485 ms
64 bytes from 10.0.2.163: icmp_seq=9 ttl=64 time=0.460 ms
64 bytes from 10.0.2.163: icmp_seq=10 ttl=64 time=0.470 ms
64 bytes from 10.0.2.163: icmp_seq=11 ttl=64 time=0.495 ms
64 bytes from 10.0.2.163: icmp_seq=12 ttl=64 time=0.390 ms
64 bytes from 10.0.2.163: icmp_seq=13 ttl=64 time=0.500 ms
64 bytes from 10.0.2.163: icmp_seq=14 ttl=64 time=0.908 ms
64 bytes from 10.0.2.163: icmp_seq=15 ttl=64 time=0.502 ms
64 bytes from 10.0.2.163: icmp_seq=16 ttl=64 time=0.474 ms
64 bytes from 10.0.2.163: icmp_seq=17 ttl=64 time=0.469 ms
64 bytes from 10.0.2.163: icmp_seq=18 ttl=64 time=0.419 ms
64 bytes from 10.0.2.163: icmp_seq=19 ttl=64 time=0.493 ms
64 bytes from 10.0.2.163: icmp_seq=20 ttl=64 time=0.510 ms
64 bytes from 10.0.2.163: icmp_seq=21 ttl=64 time=0.434 ms

```

```

Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1078-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Sep  2 11:16:43 UTC 2022

System load:  0.0               Processes:    93
Usage of /:   21.0% of 7.58GB   Users logged in:  0
Memory usage: 21%              IP address for eth0: 10.0.2.163
Swap usage:   0%                IP address for eth1: 10.0.4.37

47 updates can be applied immediately.
37 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

New release '20.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Sep  2 11:14:55 2022 from 76.175.124.74
ubuntu@ip-10-0-2-163:~$

```

Configure Routing relations on ServerVM and ClientVM

To make the networks between ServerVM and ClientVM passing through the RouterVM and be routed by the router, we should configure the Routing table on both the ServerVM and ClientVM. If we look into any one of these two 's routing table using:

```
route -n
```

We can find that there is no special routing info. All the networks are routed by default.

```

Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.0.2.1        0.0.0.0         UG    100    0      0 eth0
10.0.2.0         0.0.0.0         255.255.255.0   U      0      0      0 eth0
10.0.2.1         0.0.0.0         255.255.255.255 UH     100    0      0 eth0
10.0.3.0         0.0.0.0         255.255.255.0   U      0      0      0 eth1

```

To be specific, we should add a **Gateway** for the destination IP address on the both end:

- For the Client, the destination ip is Server's ip and the gateway is the Router's ip.

- For the Server, the destination ip is the client's ip, and the gateway is the Router's ip.

For example, we execute this on the Client side:

```
sudo route add -host <ServerVM eth0 IP address> gw <RouterVM eth1 IP address>
```

And we execute this on the Server side:

```
sudo route add -host <ClientVM eth0 IP address> gw <RouterVM eth1 IP address>
```

Then we can find an extra rule on the routing table:

```
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.0.2.1        0.0.0.0         UG        100    0      0 eth0
10.0.2.0         0.0.0.0         255.255.255.0   U         0      0      0 eth0
10.0.2.1         0.0.0.0         255.255.255.255 UH        100    0      0 eth0
10.0.2.163       10.0.2.64       255.255.255.255 UGH       0      0      0 eth0
10.0.3.0         0.0.0.0         255.255.255.0   U         0      0      0 eth1
```

Then we Ping ServerVm from ClientVm again and monitor Router's networks activities. We can find that the Router has networks activities from both Server side and Client side, as shown in the picture below. And we can say that **the network communication has been configured as we expected.**

The screenshot displays three terminal windows illustrating network configuration and connectivity:

- Top Left Window (VyOS):** Shows the VyOS command-line interface. A red box highlights the output of the `tcpdump -i eth1 host 10.0.2.86 or host 10.0.2.163` command, which captures ICMP echo requests and replies between the Router and the Client/Server VMs.
- Top Right Window (Client VM):** Shows the Client VM terminal. It displays the command `sudo route add -host 10.0.2.163 gw 10.0.2.64` and the resulting Kernel IP routing table, which includes a new entry for 10.0.2.163 via 10.0.2.64. A red box highlights the successful ping results to 10.0.2.163.
- Bottom Window (Server VM):** Shows the Server VM terminal. It displays the command `sudo route add -host 10.0.2.86 gw 10.0.2.64` and the resulting Kernel IP routing table, which includes a new entry for 10.0.2.86 via 10.0.2.64.

Annotations on the image:

- A red arrow points to the ping results in the Client VM terminal with the text: "We ping ServerVM from the Client side and it succeeds".
- A red arrow points to the tcpdump output in the VyOS terminal with the text: "Router has network activities from both the Server and Client sides".
- A red arrow points to the routing table in the Server VM terminal with the text: "This is the server VM terminal".

Other Questions and Answers

Q: How is amazon able to convert your public IP and reach your private IP on the interface?

Answer: The elastic ip addresses are randomly assigned by AWS and AWS server actually works as a router when an elastic ip is allocated to the user. The AWS server will create a mapping table to route all the network traffic from the public internet from this elastic IP to the instances' private ip address.

Q: Why SSH goes down if IP is changed on that interface

Answer: When we use SSH to access an EC2 instance, we firstly connect to the elastic ip address associated with this instance, and AWS will route us from this elastic ip to the instance's private ip, then we can control the instance's OS. If we change any IP configurations on the console interface while SSH is still on, SSH connections will lose because we are actually changing the mapping table from the elastic ip to the instance and AWS will have errors routing the networks between us and the instances.

Networks measurement with iPerf & iPerf3

[iPerf](#) is a tool for network performance measurement and tuning. It is a cross-platform tool that can produce standardized performance measurements for any network. Iperf has client and server functionality, and can create data streams to measure the throughput between the two ends in one or both directions. Typical iperf output contains a time-stamped report of the amount of data transferred and the throughput measured.

Install iPerf/iPerf3 on Ubuntu VMs

For either Client VM or Server VM: First, we enter root:

```
sudo su
```

If we directly install iPerf/iPerf3, it will fail:

```
sudo apt-get install -y iperf
```

We should install some dependancies ahead:

```
apt-get install linux-tools-common
apt-get update
apt-get install linux-tools-generic linux-cloud-tools-generic
apt-get install linux-tools-5.4.0-77-generic
apt-get install linux-cloud-tools-5.4.0-77-generic
```

Then we install iPerf/iPerf3:

```
sudo apt-get install -y iperf
```

It may still fail, and we follow the instructions on the terminal:

```
--fix-broken install
```

If there is an error named "subprocess was killed by signal (Broken pipe)" or something like that, we can refer to [this post](#) (Btw, this dude literally saved my ass when installing the dumbasss iPerf 😊) and execute:

```
sudo dpkg -i --force-overwrite <PATH to the overwritten files>
sudo apt -f install
```

Find every file that is said to be overwritten on the terminal and fill them into the commands above. For example, on my virtual machines:

```
sudo dpkg -i --force-overwrite /var/cache/apt/archives/linux-azure-5.4-tools-
5.4.0-1078_5.4.0-1078.81~18.04.1_amd64.deb
sudo dpkg -i --force-overwrite /var/cache/apt/archives/linux-gcp-5.4-tools-5.4.0-
1078_5.4.0-1078.84~18.04.1_amd64.deb
sudo dpkg -i --force-overwrite /var/cache/apt/archives/linux-gke-5.4-tools-5.4.0-
1078_5.4.0-1078.84~18.04.1_amd64.deb
sudo dpkg -i --force-overwrite /var/cache/apt/archives/linux-oracle-5.4-tools-
5.4.0-1078_5.4.0-1078.86~18.04.1_amd64.deb
sudo apt -f install
```

Then we can install all we want:

```
sudo apt-get install -y iperf
sudo apt-get install -y iperf3
```

Measure the networks using iPerf3

First, we configure the **ServerVM** as an iPerf3 server:

```
iPerf3 -s
```

Note the portnum assigned to the Server and use iPerf3 on the **Client VM** to communicate with the Server:

```
iperf3 -u -c 10.0.2.163 -b <portnum>
```

The Client VM will send/receive an amount of packets with the Server and this process will last for 10 secs. Both ends will show the info:


```

Connecting to host 10.0.2.163, port 5201
[ 4] local 10.0.2.86 port 35417 connected to 10.0.2.163 port 5201
[ ID] Interval            Transfer            Bandwidth          Total Datagrams
[ 4] 0.00-1.00 sec      88.0 KBytes        721 Kbits/sec      11
[ 4] 1.00-2.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 2.00-3.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 3.00-4.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 4.00-5.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 5.00-6.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 6.00-7.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 7.00-8.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 8.00-9.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 9.00-10.00 sec     80.0 KBytes        655 Kbits/sec      10
- - - - -
[ ID] Interval            Transfer            Bandwidth          Jitter    Lost/Total
Datagrams
[ 4] 0.00-10.00 sec     808 KBytes        662 Kbits/sec      0.060 ms  0/100 (0%)
[ 4] Sent 100 datagrams

```

We can then read the Bandwidth, jitter and datagrams loss from the measurement results. If we open tcpdump on the router, we can also find that all the networks traffics pass through the router and they are all using UDP/IP protocol.

All network traffics pass through the router, using UDP/IP protocol.

```

08:25:29.689939 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:29.689971 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:29.709049 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:29.709082 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:29.809952 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:29.809985 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:29.909947 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:29.909984 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.009945 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.009980 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.109933 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.109968 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.209971 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.209991 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.309918 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.309955 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.409915 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.409951 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.509908 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.509942 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.609911 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.609945 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.709914 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.709948 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.809939 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.809975 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.909944 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:30.909979 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.008995 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.009932 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.109921 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.109954 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.209950 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.209983 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.309905 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.309936 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.409968 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.409992 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.509950 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.509986 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.609939 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.609978 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.709931 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.709964 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.809946 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.809984 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.909915 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:31.909950 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:32.009933 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:32.009971 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192
08:25:32.109936 IP 10.0.2.86.60403 > 10.0.2.163.5201: UDP, length 8192

```

```

[ 4] Sent 100 datagrams
iperf Done.
ubuntu@ip-10-0-2-86:~$ iperf3 -u -c 10.0.2.163 -b 5201
Connecting to host 10.0.2.163, port 5201
[ 4] local 10.0.2.86 port 60403 connected to 10.0.2.163 port 5201
[ ID] Interval            Transfer            Bandwidth          Total Datagrams
[ 4] 0.00-1.00 sec      88.0 KBytes        721 Kbits/sec      11
[ 4] 1.00-2.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 2.00-3.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 3.00-4.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 4.00-5.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 5.00-6.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 6.00-7.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 7.00-8.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 8.00-9.00 sec      80.0 KBytes        655 Kbits/sec      10
[ 4] 9.00-10.00 sec     80.0 KBytes        655 Kbits/sec      10
- - - - -
[ ID] Interval            Transfer            Bandwidth          Jitter    Lost/Total
Datagrams
[ 4] 0.00-10.00 sec     808 KBytes        662 Kbits/sec      0.116 ms  0/100 (0%)
[ 4] Sent 100 datagrams
iperf Done.
ubuntu@ip-10-0-2-86:~$ |

```

```

Server listening on 5201
Accepted connection from 10.0.2.86, port 43558
[ 5] local 10.0.2.163 port 5201 connected to 10.0.2.86 port 60403
[ ID] Interval            Transfer            Bandwidth          Jitter    Lost/Total
Datagrams
[ 5] 0.00-1.00 sec      80.0 KBytes        655 Kbits/sec      0.052 ms  0/10 (0%)
[ 5] 1.00-2.00 sec      80.0 KBytes        655 Kbits/sec      0.052 ms  0/10 (0%)
[ 5] 2.00-3.00 sec      80.0 KBytes        655 Kbits/sec      0.048 ms  0/10 (0%)
[ 5] 3.00-4.00 sec      80.0 KBytes        655 Kbits/sec      0.055 ms  0/10 (0%)
[ 5] 4.00-5.00 sec      80.0 KBytes        655 Kbits/sec      0.041 ms  0/10 (0%)
[ 5] 5.00-6.00 sec      80.0 KBytes        655 Kbits/sec      0.052 ms  0/10 (0%)
[ 5] 6.00-7.00 sec      80.0 KBytes        655 Kbits/sec      0.051 ms  0/10 (0%)
[ 5] 7.00-8.00 sec      80.0 KBytes        655 Kbits/sec      0.055 ms  0/10 (0%)
[ 5] 8.00-9.00 sec      80.0 KBytes        655 Kbits/sec      0.057 ms  0/10 (0%)
[ 5] 9.00-10.00 sec     80.0 KBytes        655 Kbits/sec      0.116 ms  0/10 (0%)
[ 5] 10.00-10.00 sec     0.00 Bytes         0.00 bits/sec      0.116 ms  0/0 (0%)
- - - - -
[ ID] Interval            Transfer            Bandwidth          Jitter    Lost/Total
Datagrams
[ 5] 0.00-10.00 sec     0.00 Bytes         0.00 bits/sec      0.116 ms  0/100 (0%)
Server listening on 5201

```

Add delay/loss to the virtual machines and measure the networks

To emulate the network delay and network loss scenarios, we can manually add some delay and loss to the ip config and explore the iPerf measurement results again.

First, we add delay to both the Server and the Client with the following commands:

```
sudo tc qdisc add dev eth0 root netem delay 100ms
```

And we do the measurements again. The results are as following:

```
Connecting to host 10.0.2.163, port 5201
[ 4] local 10.0.2.86 port 55948 connected to 10.0.2.163 port 5201
[ ID] Interval          Transfer      Bandwidth      Total Datagrams
[ 4]  0.00-1.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  1.00-2.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  2.00-3.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  3.00-4.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  4.00-5.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  5.00-6.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  6.00-7.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  7.00-8.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  8.00-9.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  9.00-10.00   sec   80.0 KBytes    655 Kbits/sec    10
- - - - -
[ ID] Interval          Transfer      Bandwidth      Jitter      Lost/Total
Datagrams
[ 4]  0.00-10.00   sec   800 KBytes    655 Kbits/sec  0.143 ms    0/99 (0%)
[ 4] Sent 99 datagrams
```

We can find that, the bandwidth has not changed at all. But the jitter can be twice as much. **Therefore**, the networks delay can influence the jitter but not the bandwidth.

Then, we remove the delay and add loss to both virtual machines (note that this time we use **change** command but not **add**):

```
sudo tc qdisc change dev eth0 root netem delay 0ms loss 10%
```

And we get the measurements results again:

```
Connecting to host 10.0.2.163, port 5201
[ 4] local 10.0.2.86 port 35747 connected to 10.0.2.163 port 5201
[ ID] Interval          Transfer      Bandwidth      Total Datagrams
[ 4]  0.00-1.00    sec   88.0 KBytes    721 Kbits/sec    11
[ 4]  1.00-2.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  2.00-3.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  3.00-4.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  4.00-5.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  5.00-6.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  6.00-7.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  7.00-8.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  8.00-9.00    sec   80.0 KBytes    655 Kbits/sec    10
[ 4]  9.00-10.00   sec   80.0 KBytes    655 Kbits/sec    10
```

```

- - - - -
[ ID] Interval          Transfer    Bandwidth    Jitter    Lost/Total
Datagrams
[  4]  0.00-10.00  sec   808 KBytes   662 Kbits/sec  0.084 ms   10/100 (10%)
[  4] Sent 100 datagrams

```

As we expected, the jitter gets back to normal because we set the delay back to 0ms. And the datagrams loss gets up to 10%, because we set the loss to be 10%. As always, the bandwidth is still the same.

Next, we delete the configurations we set just now:

```
sudo tc qdisc del dev eth0 root
```

And we set another scenario: we limit the networks rate to 100mbit, latency to 1 ms and burst 9015:

```
sudo tc qdisc add dev eth0 root tbf rate 100mbit latency 1ms burst 9015
```

We measure again and the results:

```

Connecting to host 10.0.2.163, port 5201
[  4] local 10.0.2.86 port 45441 connected to 10.0.2.163 port 5201
[ ID] Interval          Transfer    Bandwidth    Total Datagrams
[  4]  0.00-1.00    sec   88.0 KBytes   721 Kbits/sec    11
[  4]  1.00-2.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  2.00-3.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  3.00-4.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  4.00-5.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  5.00-6.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  6.00-7.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  7.00-8.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  8.00-9.00    sec   80.0 KBytes   655 Kbits/sec    10
[  4]  9.00-10.00   sec   80.0 KBytes   655 Kbits/sec    10
- - - - -
[ ID] Interval          Transfer    Bandwidth    Jitter    Lost/Total
Datagrams
[  4]  0.00-10.00   sec   808 KBytes   662 Kbits/sec  0.071 ms    0/100 (0%)
[  4] Sent 100 datagrams

```

We can find that all the results remain the same, because the existing bandwidth is way lower than the limitations we set, so the limitations do not affect the results. If we set the rate to be really small, and we will see that the data transferring rate will descend fast.

Last, we set speed for the instance:

```
ubuntu@ip-10-0-2-86:~$ sudo ethtool -s eth0 speed 10
Cannot get current device settings: Operation not supported
not setting speed
```

We can find that AWS does not allow us to execute this operation, because every instance has a fixed bandwidth resource and it is not allowed to modify.

Add delay / loss on the Router and measure again

Let's do the same thing on the & only on **router**, launch the Server and Client using iPerf3 again and see what will happen.

First, we add 100ms delay for the router.

! Note that: Router uses eth1 as the internal network interface. **We should add delay to eth1 but not eth0!!!**

Here are the results:

```
Connecting to host 10.0.2.163, port 5201
[ 4] local 10.0.2.86 port 59975 connected to 10.0.2.163 port 5201
[ ID] Interval            Transfer      Bandwidth      Total Datagrams
[ 4] 0.00-1.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 1.00-2.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 2.00-3.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 3.00-4.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 4.00-5.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 5.00-6.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 6.00-7.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 7.00-8.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 8.00-9.00 sec      80.0 KBytes   655 Kbits/sec   10
[ 4] 9.00-10.00 sec     80.0 KBytes   655 Kbits/sec   10
- - - - -
[ ID] Interval            Transfer      Bandwidth      Jitter      Lost/Total
Datagrams
[ 4] 0.00-10.00 sec     800 KBytes   655 Kbits/sec   0.059 ms    0/99 (0%)
[ 4] Sent 99 datagrams
```

We can find that the results are similar as we got from the Server/Client settings: **the Jitter can be higher due to the higher delay**. If we look into the info from the Server end, we can even find the jitter varies a lot within these 10 secs and the delay can really affect the network stability.

Then we set Router delay to 0ms and loss to 10%. Here are the results:

```
Connecting to host 10.0.2.163, port 5201
[ 4] local 10.0.2.86 port 35957 connected to 10.0.2.163 port 5201
[ ID] Interval            Transfer      Bandwidth      Total Datagrams
[ 4] 0.00-1.00 sec      88.0 KBytes   721 Kbits/sec   11
```

```

[ 4] 1.00-2.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 2.00-3.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 3.00-4.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 4.00-5.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 5.00-6.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 6.00-7.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 7.00-8.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 8.00-9.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 9.00-10.00 sec 80.0 KBytes 655 Kbits/sec 10
- - - - -
[ ID] Interval          Transfer      Bandwidth      Jitter      Lost/Total
Datagrams
[ 4] 0.00-10.00 sec 808 KBytes 662 Kbits/sec 0.087 ms 11/100 (11%)

```

Same, the results from iPerf3 say that the average loss comes up to approximately 10%, as we expected.

Next, we limit the networks rate to 100mbit, latency to 1 ms and burst 9015. Here are the results:

```

Connecting to host 10.0.2.163, port 5201
[ 4] local 10.0.2.86 port 54763 connected to 10.0.2.163 port 5201
[ ID] Interval          Transfer      Bandwidth      Total Datagrams
[ 4] 0.00-1.00 sec 88.0 KBytes 721 Kbits/sec 11
[ 4] 1.00-2.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 2.00-3.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 3.00-4.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 4.00-5.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 5.00-6.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 6.00-7.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 7.00-8.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 8.00-9.00 sec 80.0 KBytes 655 Kbits/sec 10
[ 4] 9.00-10.00 sec 80.0 KBytes 655 Kbits/sec 10
- - - - -
[ ID] Interval          Transfer      Bandwidth      Jitter      Lost/Total
Datagrams
[ 4] 0.00-10.00 sec 808 KBytes 662 Kbits/sec 0.031 ms 0/100 (0%)
[ 4] Sent 100 datagrams

```

Same, it does not affect the networks at all, as expected.

Conclusion:

- Either we add delay / loss / rate limit on the Server, Client or Router end, the network measurement results can be similar. If we add to them at the same time, the limit of network can be the bottleneck among these three components.
- Delay: affect the network connections' stability and add jitters.
- Loss: affect the datagrams loss rate.
- Rate limit: affect the bandwidth.

The End

This concludes the whole network emulations on AWS. We can do more on AWS, such as building internal networks structures based on these topological relations, or building some CDN servers or machine learning computing servers and using parallel computing on AWS. Anyway, this way to build a network is pretty fun!

If you have any concerns, please contact me at boyangxi@usc.edu ❤️

Fight on Trojans!