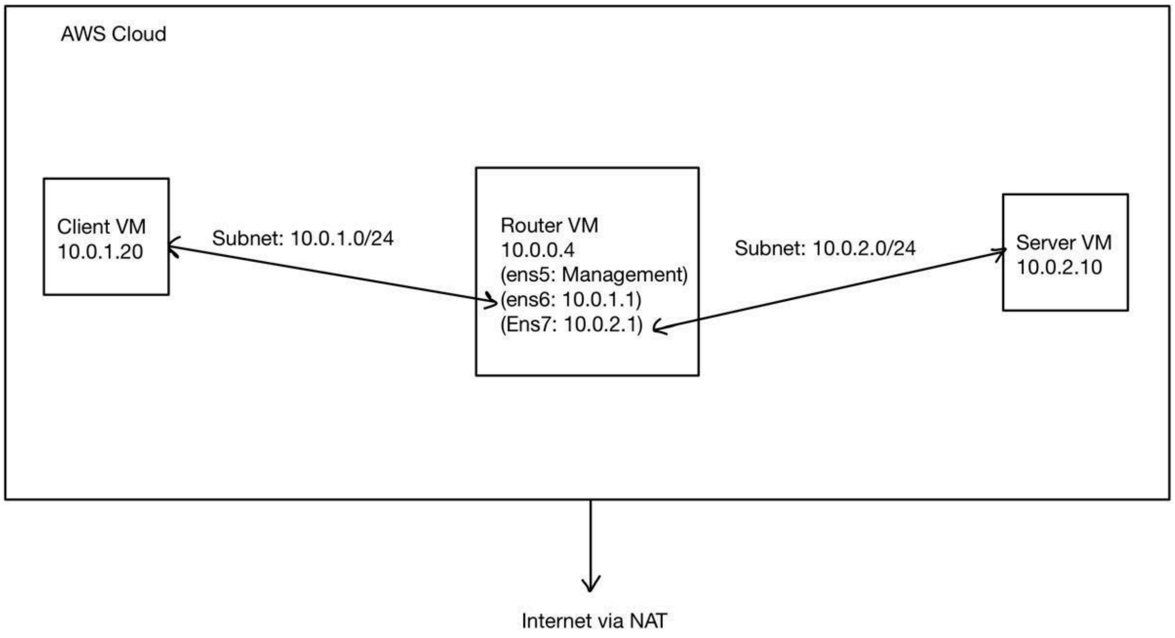
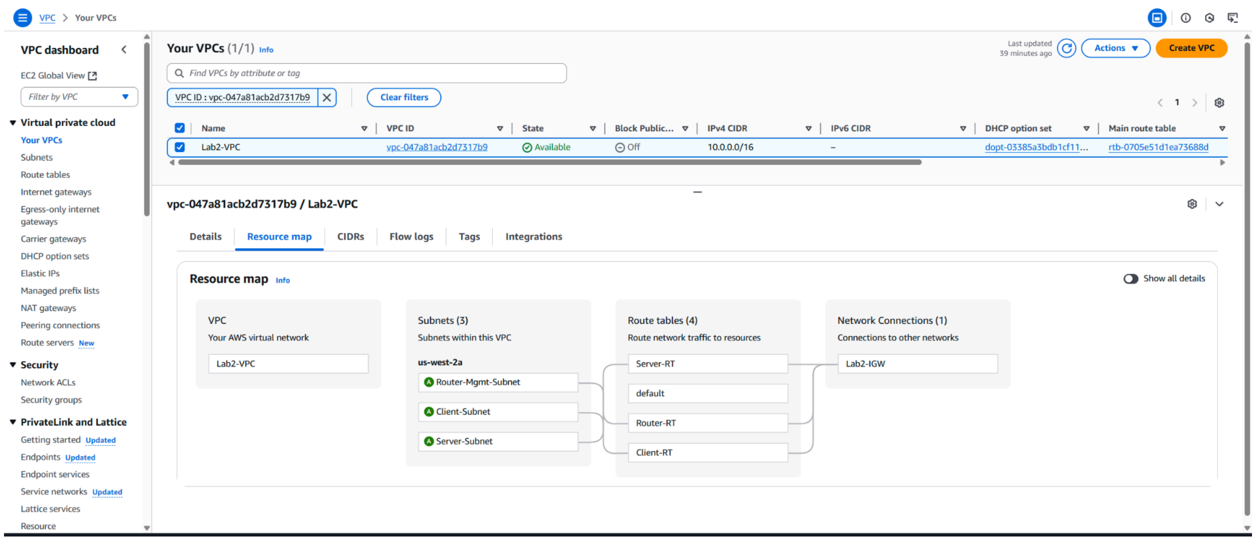
Linux Kernel Hacking for Throughput Acceleration

Part 1



The above diagram shows the topology we used to conduct the experiment

VPC: Custom VPC (10.0.0.0/16), with dedicated subnets for client, server, router management and private subnets (for client, server, and router) are associated with a route to the IGW, allowing outbound internet connectivity as needed.



Your VPCs (1/1) Info

Last updated 39 minutes ago

VPC ID : vpc-047a81acb2d7317b9

Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR	DHCP option set	Main route table
Lab2-VPC	vpc-047a81acb2d7317b9	Available	Off	10.0.0.0/16	–	dopt-03385a3bdb1cf11...	rtb-0705e51d1ea73688d

vpc-047a81acb2d7317b9 / Lab2-VPC

Details | Resource map | CIDRs | Flow logs | Tags | Integrations

Resource map Info

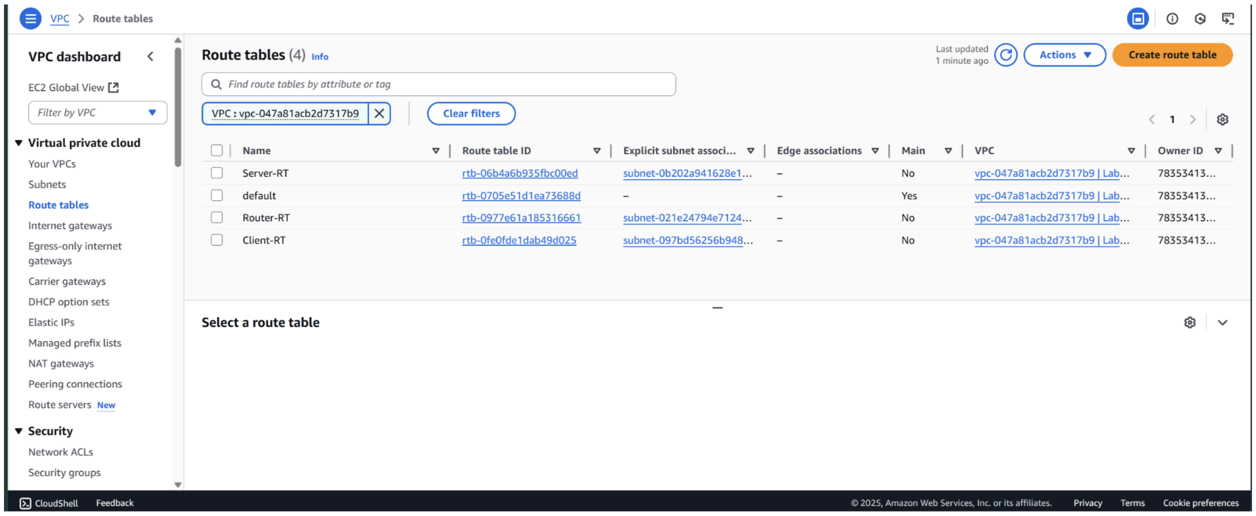
Show all details

VPC
Your AWS virtual network
Lab2-VPC

Subnets (3)
Subnets within this VPC
us-west-2a
Router-Mgmt-Subnet
Client-Subnet
Server-Subnet

Route tables (4)
Route network traffic to resources
Server-RT
default
Router-RT
Client-RT

Network Connections (1)
Connections to other networks
Lab2-IGW



Route tables (4) Info

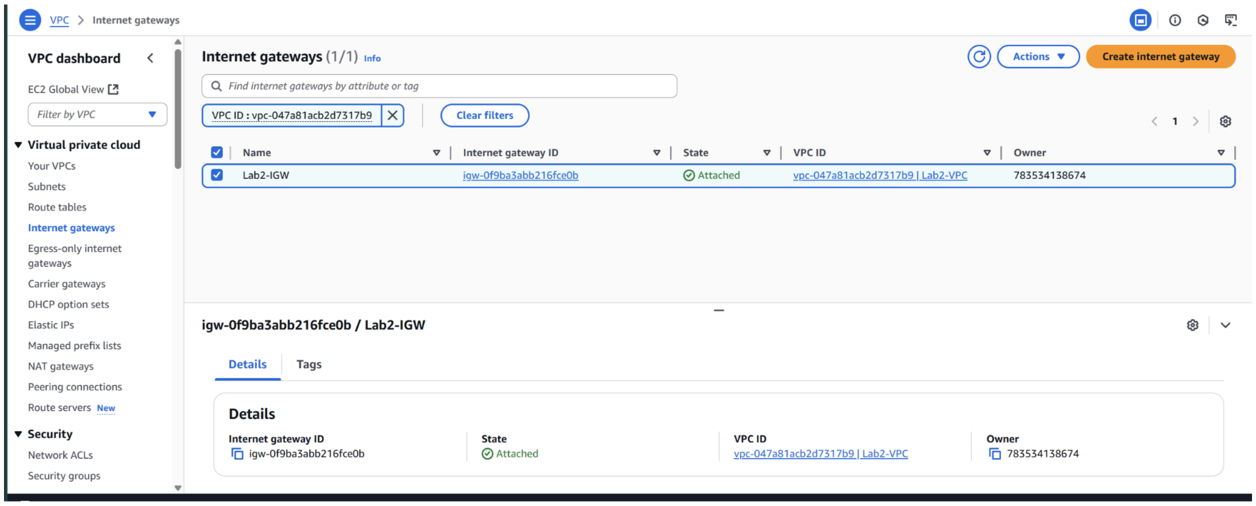
Last updated 1 minute ago

VPC : vpc-047a81acb2d7317b9

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Owner ID
Server-RT	rtb-06b4a6b935fbc00ed	subnet-0b202a941628e1...	–	No	vpc-047a81acb2d7317b9 | Lab...	78353413...
default	rtb-0705e51d1ea73688d	–	–	Yes	vpc-047a81acb2d7317b9 | Lab...	78353413...
Router-RT	rtb-0977e61a185316661	subnet-021e24794e7124...	–	No	vpc-047a81acb2d7317b9 | Lab...	78353413...
Client-RT	rtb-0fe0fde1dab49d025	subnet-097bd56256b948...	–	No	vpc-047a81acb2d7317b9 | Lab...	78353413...

Select a route table

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



Internet gateways (1/1) Info

VPC ID : vpc-047a81acb2d7317b9

Name	Internet gateway ID	State	VPC ID	Owner
Lab2-IGW	igw-0f9ba3abb216fce0b	Attached	vpc-047a81acb2d7317b9 | Lab2-VPC	783534138674

igw-0f9ba3abb216fce0b / Lab2-IGW

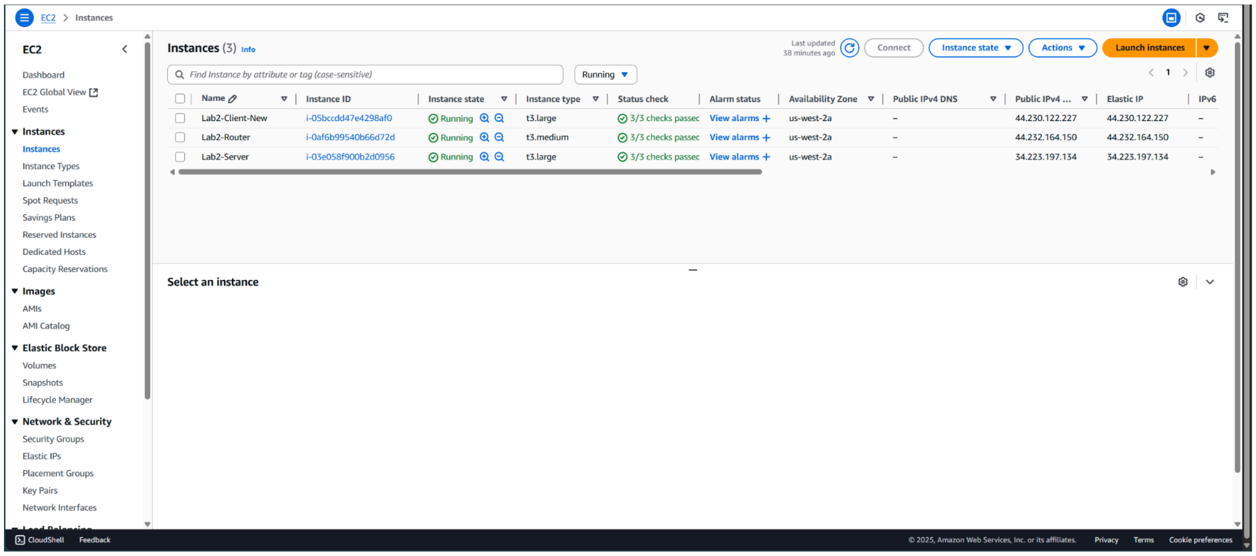
Details | Tags

Details

Internet gateway ID	State	VPC ID	Owner
igw-0f9ba3abb216fce0b	Attached	vpc-047a81acb2d7317b9 | Lab2-VPC	783534138674

EC2 Instances:

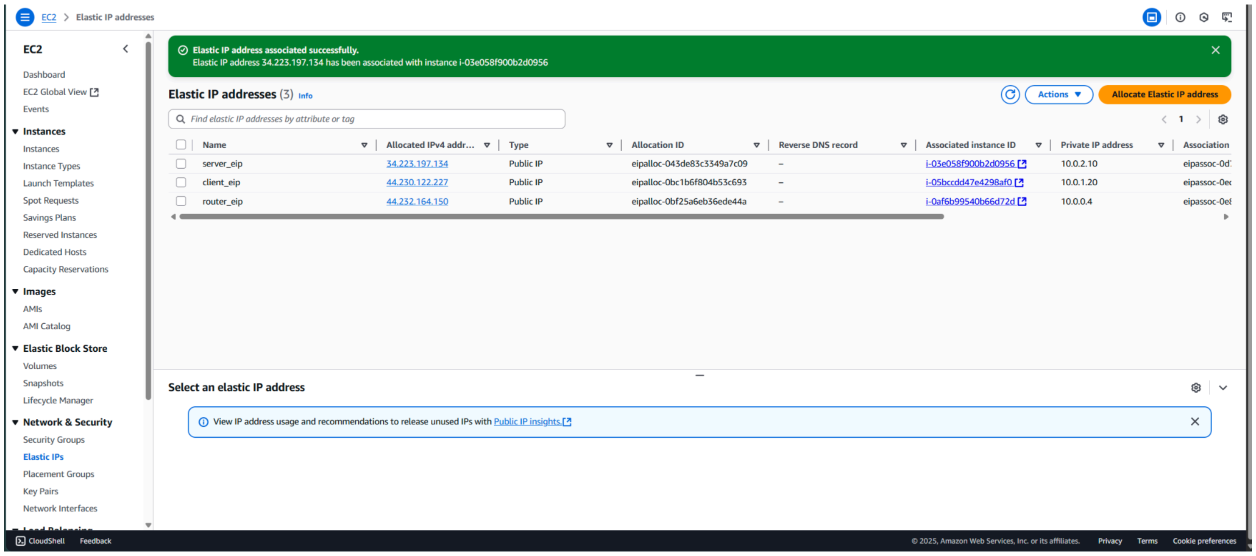
- Client VM (10.0.1.20, subnet 10.0.1.0/24)
- Server VM (10.0.2.10, subnet 10.0.2.0/24)
- Router VM (10.0.0.4 with interfaces on multiple subnets: management, client, and server subnets)



Instances (3) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6
Lab2-Client-New	i-05bccdd47e4298af0	Running	t3.large	3/3 checks passec	View alarms +	us-west-2a	–	44.230.122.227	44.230.122.227	–
Lab2-Router	i-0af6b99540b66d72d	Running	t3.medium	3/3 checks passec	View alarms +	us-west-2a	–	44.232.164.150	44.232.164.150	–
Lab2-Server	i-03e058f900b2d0956	Running	t3.large	3/3 checks passec	View alarms +	us-west-2a	–	34.223.197.134	34.223.197.134	–

Select an instance



Elastic IP address associated successfully.
Elastic IP address 34.223.197.134 has been associated with instance i-03e058f900b2d0956

Elastic IP addresses (3) Info

Name	Allocated IPv4 addr...	Type	Allocation ID	Reverse DNS record	Associated instance ID	Private IP address	Association
server_eip	34.223.197.134	Public IP	eipalloc-043de83c3349a7c09	–	i-03e058f900b2d0956	10.0.2.10	eipassoc-0d
client_eip	44.230.122.227	Public IP	eipalloc-0bc1b6f804b53c693	–	i-05bccdd47e4298af0	10.0.1.20	eipassoc-0e
router_eip	44.232.164.150	Public IP	eipalloc-0bf25a6eb36ede44a	–	i-0af6b99540b66d72d	10.0.0.4	eipassoc-0e

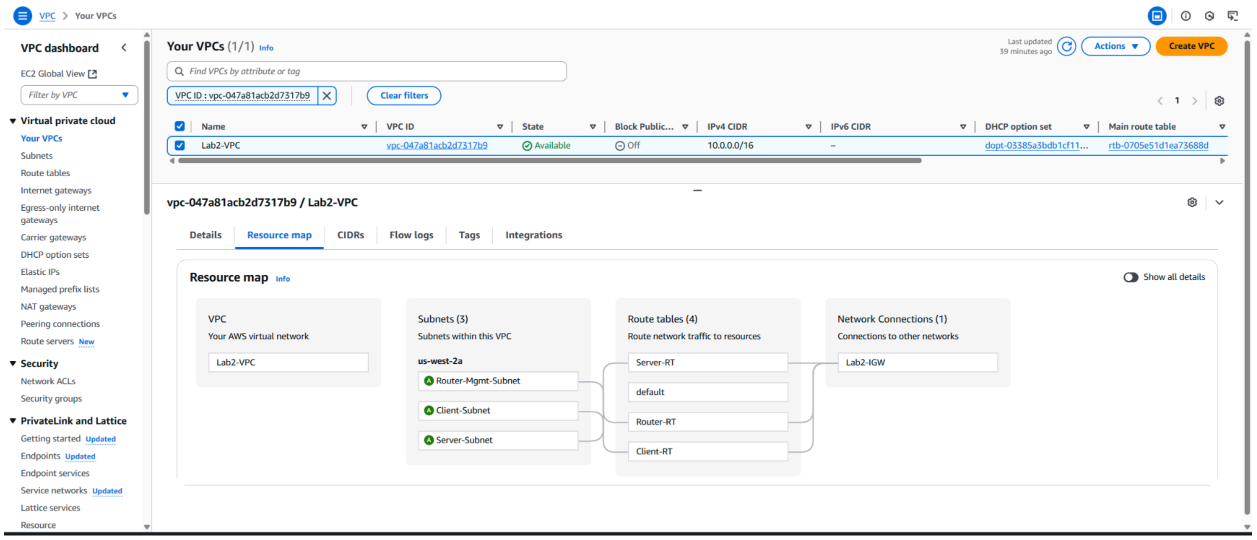
Select an elastic IP address

View IP address usage and recommendations to release unused IPs with Public IP insights.

Routing & Firewall:

- Central router instance (Ubuntu instead of VyOS as using community AMIs) bridges the client and server subnets, with packet forwarding enabled and NAT for Internet access.

- Iptables used for forwarding and masquerade rules.
- Each VM has proper Security Group allowing ICMP, TCP/UDP (iperf), and SSH access.
- Route tables configured to ensure all traffic between client and server is routed via the router VM.



Your VPCs (1/1) Info

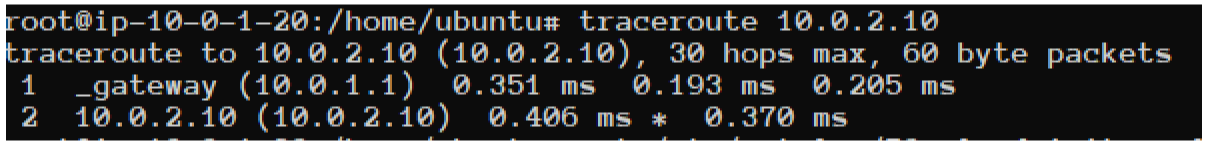
VPC ID : vpc-047a81acb2d7317b9

Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR	DHCP option set	Main route table
Lab2-VPC	vpc-047a81acb2d7317b9	Available	Off	10.0.0.0/16	–	dopt-03385a3bdb1cf11...	rtb-0705e51d1ea73688d

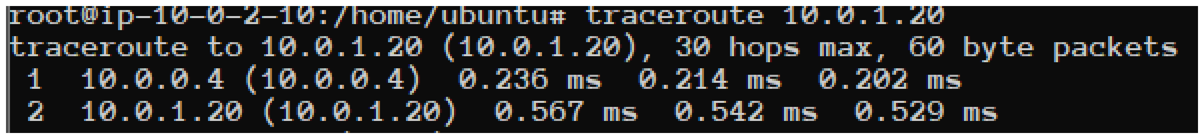
vpc-047a81acb2d7317b9 / Lab2-VPC

Resource map Info

Traceroute for Client VM and Server VM



```
root@ip-10-0-1-20:/home/ubuntu# traceroute 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1  _gateway (10.0.1.1)  0.351 ms  0.193 ms  0.205 ms
 2  10.0.2.10 (10.0.2.10)  0.406 ms *  0.370 ms
```



```
root@ip-10-0-2-10:/home/ubuntu# traceroute 10.0.1.20
traceroute to 10.0.1.20 (10.0.1.20), 30 hops max, 60 byte packets
 1  10.0.0.4 (10.0.0.4)  0.236 ms  0.214 ms  0.202 ms
 2  10.0.1.20 (10.0.1.20)  0.567 ms  0.542 ms  0.529 ms
```

Answers to questions to understand this project better:

How is Amazon able to convert your public IP and reach your private IP on the interface?

Amazon performs public-to-private IP conversion using internal NAT rules and routing at the VPC edge when an Elastic IP is attached to a network interface. This allows global (public) access to a resource hosted on a private IP within an EC2 instance through seamless network address translation managed invisibly by AWS.

Can you answer why SSH goes down if the IP is changed on that interface? What can be used in VyOS VM to start dhclient in VyOS VM?

SSH disconnects immediately when the IP of its bonded interface changes because the existing TCP session becomes invalid due to endpoint mismatch.

To dynamically acquire a new IP in Ubuntu, use dhclient on the desired interface after making sure it is brought up.

Also, you would have to change the network interface adapter settings of "Source/Destination check" to allow traffic to pass over the VyOS router. Make this setting to Disable. What does this setting do? Explain.

Disabling source/destination check on a router instance is crucial for enabling packet forwarding and routing functions within AWS VPCs, as it allows the instance to handle transit traffic that is not directly addressed to it

Without disabling, any traffic routed through the instance is dropped, breaking connectivity for all network segments that depend on the router.

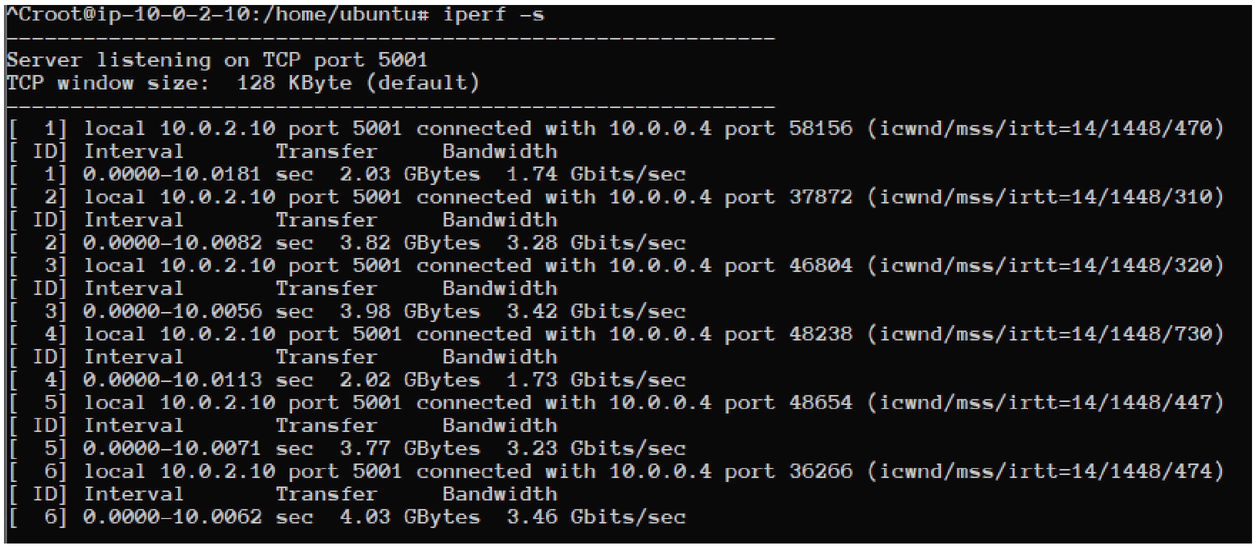
You would observe that VMs are not able to contact the internet. Why is this? Explain. How can you correct it? To solve it, look at /etc/resolvconf/resolv.conf.d/base file. You would also have to do "sudo apt update". What does this command do?

VMs fail to access the internet mainly due to missing DNS configuration, which can be fixed by specifying nameservers in /etc/resolvconf/resolv.conf.d/base, then running sudo resolvconf -u.

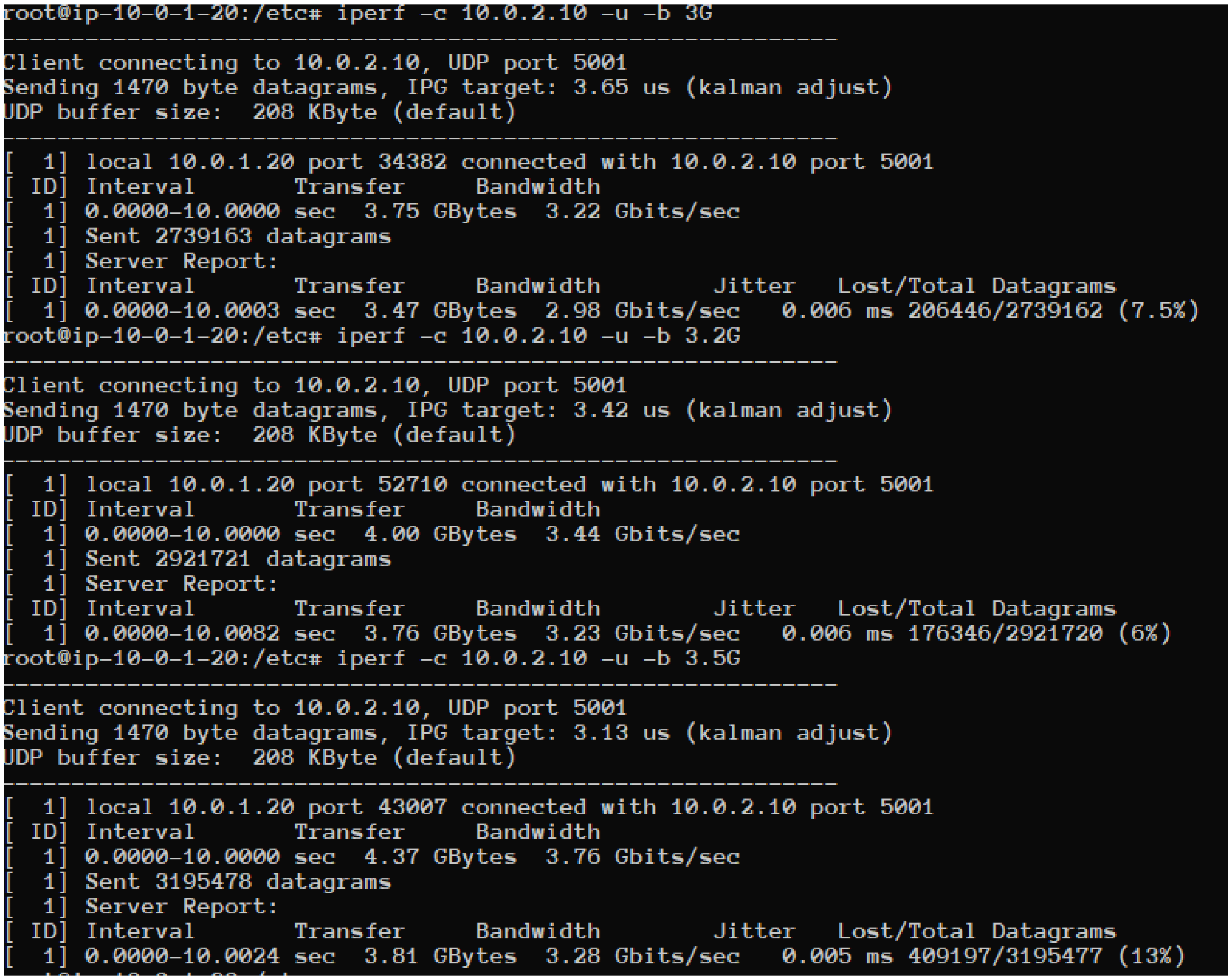
Running sudo apt update updates the local repository metadata for package management.

What is the max throughput and bandwidth you were able to achieve? Why is there any difference in measurement between these two readings over the same link?

What is the max throughput and bandwidth you were able to achieve? Why is there any difference in measurement between these two readings over the same link?



```
^Croot@ip-10-0-2-10:/home/ubuntu# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  1] local 10.0.2.10 port 5001 connected with 10.0.0.4 port 58156 (icwnd/mss/irtt=14/1448/470)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0181 sec  2.03 GBytes  1.74 Gbits/sec
[  2] local 10.0.2.10 port 5001 connected with 10.0.0.4 port 37872 (icwnd/mss/irtt=14/1448/310)
[ ID] Interval       Transfer     Bandwidth
[  2] 0.0000-10.0082 sec  3.82 GBytes  3.28 Gbits/sec
[  3] local 10.0.2.10 port 5001 connected with 10.0.0.4 port 46804 (icwnd/mss/irtt=14/1448/320)
[ ID] Interval       Transfer     Bandwidth
[  3] 0.0000-10.0056 sec  3.98 GBytes  3.42 Gbits/sec
[  4] local 10.0.2.10 port 5001 connected with 10.0.0.4 port 48238 (icwnd/mss/irtt=14/1448/730)
[ ID] Interval       Transfer     Bandwidth
[  4] 0.0000-10.0113 sec  2.02 GBytes  1.73 Gbits/sec
[  5] local 10.0.2.10 port 5001 connected with 10.0.0.4 port 48654 (icwnd/mss/irtt=14/1448/447)
[ ID] Interval       Transfer     Bandwidth
[  5] 0.0000-10.0071 sec  3.77 GBytes  3.23 Gbits/sec
[  6] local 10.0.2.10 port 5001 connected with 10.0.0.4 port 36266 (icwnd/mss/irtt=14/1448/474)
[ ID] Interval       Transfer     Bandwidth
[  6] 0.0000-10.0062 sec  4.03 GBytes  3.46 Gbits/sec
```


```
root@ip-10-0-1-20:/etc# iperf -c 10.0.2.10 -u -b 3G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 3.65 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 34382 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  3.75 GBytes  3.22 Gbits/sec
[  1] Sent 2739163 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0003 sec  3.47 GBytes  2.98 Gbits/sec   0.006 ms 206446/2739162 (7.5%)
root@ip-10-0-1-20:/etc# iperf -c 10.0.2.10 -u -b 3.2G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 3.42 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 52710 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  4.00 GBytes  3.44 Gbits/sec
[  1] Sent 2921721 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0082 sec  3.76 GBytes  3.23 Gbits/sec   0.006 ms 176346/2921720 (6%)
root@ip-10-0-1-20:/etc# iperf -c 10.0.2.10 -u -b 3.5G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 3.13 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 43007 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  4.37 GBytes  3.76 Gbits/sec
[  1] Sent 3195478 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0024 sec  3.81 GBytes  3.28 Gbits/sec   0.005 ms 409197/3195477 (13%)
```

Protocol	**Max Observed Throughput**	**Loss Observed**
UDP	~3.2–3.8 Gbits/sec	6%–13% as rate increases
TCP	~3.2–3.4 Gbits/sec	None (reliable delivery)

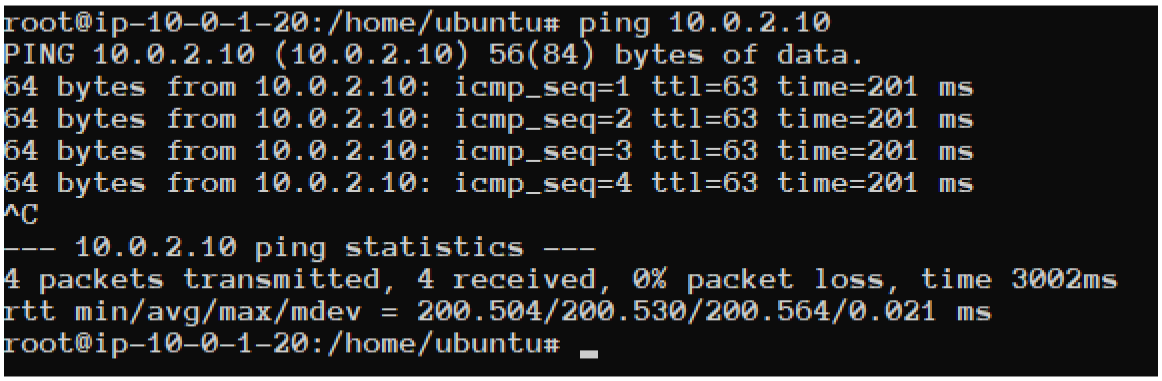
UDP can send at any rate but packets above the real network/interface capacity are dropped, causing reported packet loss.

TCP is self-limiting: it adjusts its sending rate in response to packet loss or congestion and avoids overwhelming the network.

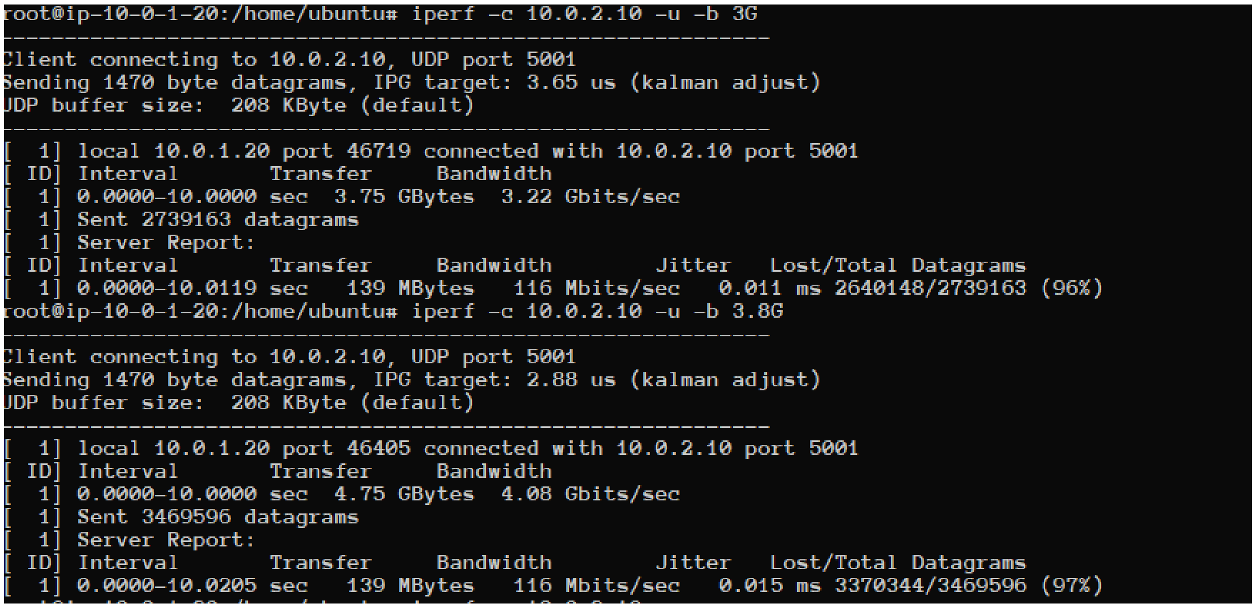
UDP shows higher “raw” bandwidth but this doesn't reflect reliable delivery. TCP throughput reflects the maximum sustainable reliable throughput, and is always below or at the true network limit.

In the tests, as soon as UDP tries to go beyond the available bandwidth, packet loss jumps, while TCP's throughput naturally levels off at the network/instance maximum due to its congestion control and retransmission mechanisms.

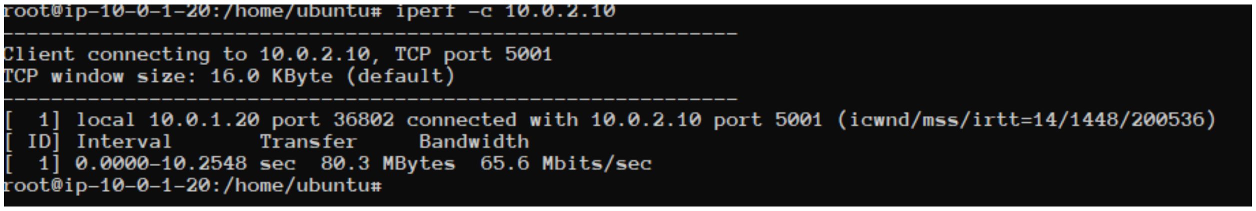
Now, set: `sudo tc qdisc add dev eth0 root netem delay 100ms.`
On the client, server VM, then test the bandwidth again. Were there any changes in readings? If there is a change, why is it?



```
root@ip-10-0-1-20:/home/ubuntu# ping 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=63 time=201 ms
64 bytes from 10.0.2.10: icmp_seq=2 ttl=63 time=201 ms
64 bytes from 10.0.2.10: icmp_seq=3 ttl=63 time=201 ms
64 bytes from 10.0.2.10: icmp_seq=4 ttl=63 time=201 ms
^C
--- 10.0.2.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 200.504/200.530/200.564/0.021 ms
root@ip-10-0-1-20:/home/ubuntu# _
```



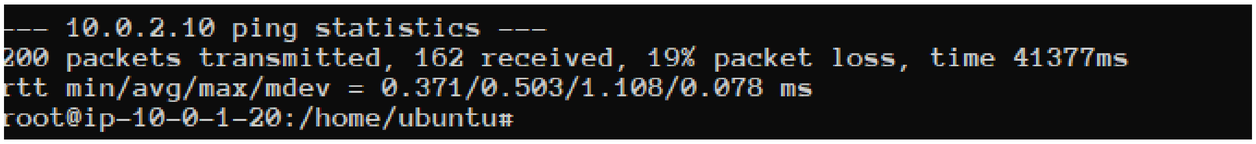
```
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 3G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 3.65 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 46719 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  3.75 GBytes  3.22 Gbits/sec
[  1] Sent 2739163 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0119 sec   139 MBytes   116 Mbits/sec   0.011 ms 2640148/2739163 (96%)
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 3.8G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 2.88 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 46405 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  4.75 GBytes  4.08 Gbits/sec
[  1] Sent 3469596 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0205 sec   139 MBytes   116 Mbits/sec   0.015 ms 3370344/3469596 (97%)
```



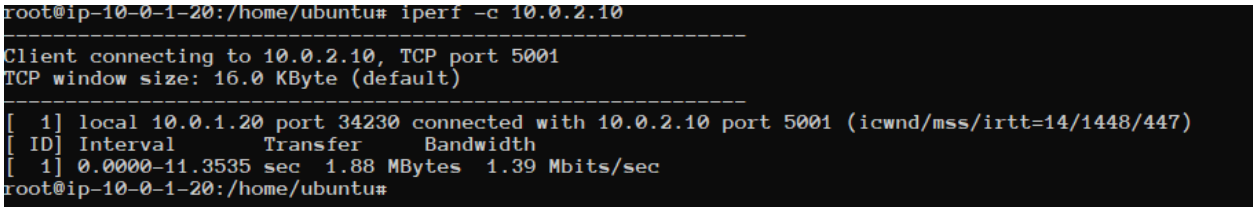
```
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10
------------------------------------------------------------
Client connecting to 10.0.2.10, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 36802 connected with 10.0.2.10 port 5001 (icwnd/mss/irtt=14/1448/200536)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.2548 sec  80.3 MBytes  65.6 Mbits/sec
root@ip-10-0-1-20:/home/ubuntu#
```

The artificial 100ms delay imposes a ceiling on practical throughput, especially if buffer/settings are not tuned for high-latency paths. TCP's reliance on in-flight window size and ACK feedback, and UDP's susceptibility to buffer overflow, both manifest as lower measured goodput and higher loss rates

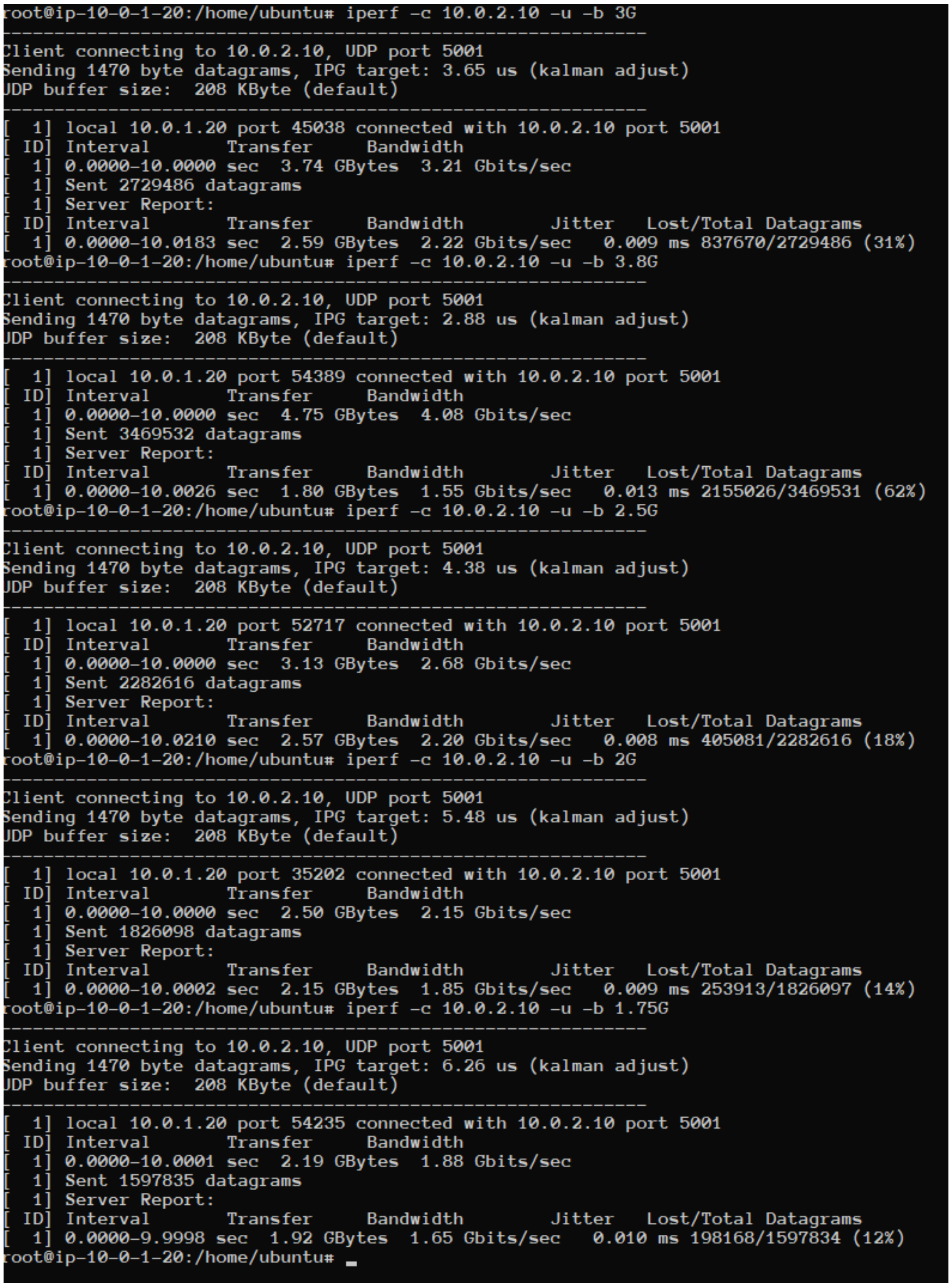
Now, set: `sudo tc qdisc change dev eth0 root netem delay 0ms loss 10%`
and take another measurement. What is your observation?



```
--- 10.0.2.10 ping statistics ---
200 packets transmitted, 162 received, 19% packet loss, time 41377ms
rtt min/avg/max/mdev = 0.371/0.503/1.108/0.078 ms
root@ip-10-0-1-20:/home/ubuntu#
```



```
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10
------------------------------------------------------------
Client connecting to 10.0.2.10, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 34230 connected with 10.0.2.10 port 5001 (icwnd/mss/irtt=14/1448/447)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-11.3535 sec  1.88 MBytes  1.39 Mbits/sec
root@ip-10-0-1-20:/home/ubuntu#
```



```
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 3G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 3.65 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 45038 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  3.74 GBytes  3.21 Gbits/sec
[  1] Sent 2729486 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0183 sec  2.59 GBytes  2.22 Gbits/sec   0.009 ms 837670/2729486 (31%)
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 3.8G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 2.88 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 54389 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  4.75 GBytes  4.08 Gbits/sec
[  1] Sent 3469532 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0026 sec  1.80 GBytes  1.55 Gbits/sec   0.013 ms 2155026/3469531 (62%)
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 2.5G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 4.38 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 52717 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  3.13 GBytes  2.68 Gbits/sec
[  1] Sent 2282616 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0210 sec  2.57 GBytes  2.20 Gbits/sec   0.008 ms 405081/2282616 (18%)
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 2G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 5.48 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 35202 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0000 sec  2.50 GBytes  2.15 Gbits/sec
[  1] Sent 1826098 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0002 sec  2.15 GBytes  1.85 Gbits/sec   0.009 ms 253913/1826097 (14%)
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 1.75G
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 6.26 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 54235 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0001 sec  2.19 GBytes  1.88 Gbits/sec
[  1] Sent 1597835 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-9.9998 sec  1.92 GBytes  1.65 Gbits/sec   0.010 ms 198168/1597834 (12%)
root@ip-10-0-1-20:/home/ubuntu# _
```

The TCP throughput dropped drastically to 1.39 Mbits/sec.

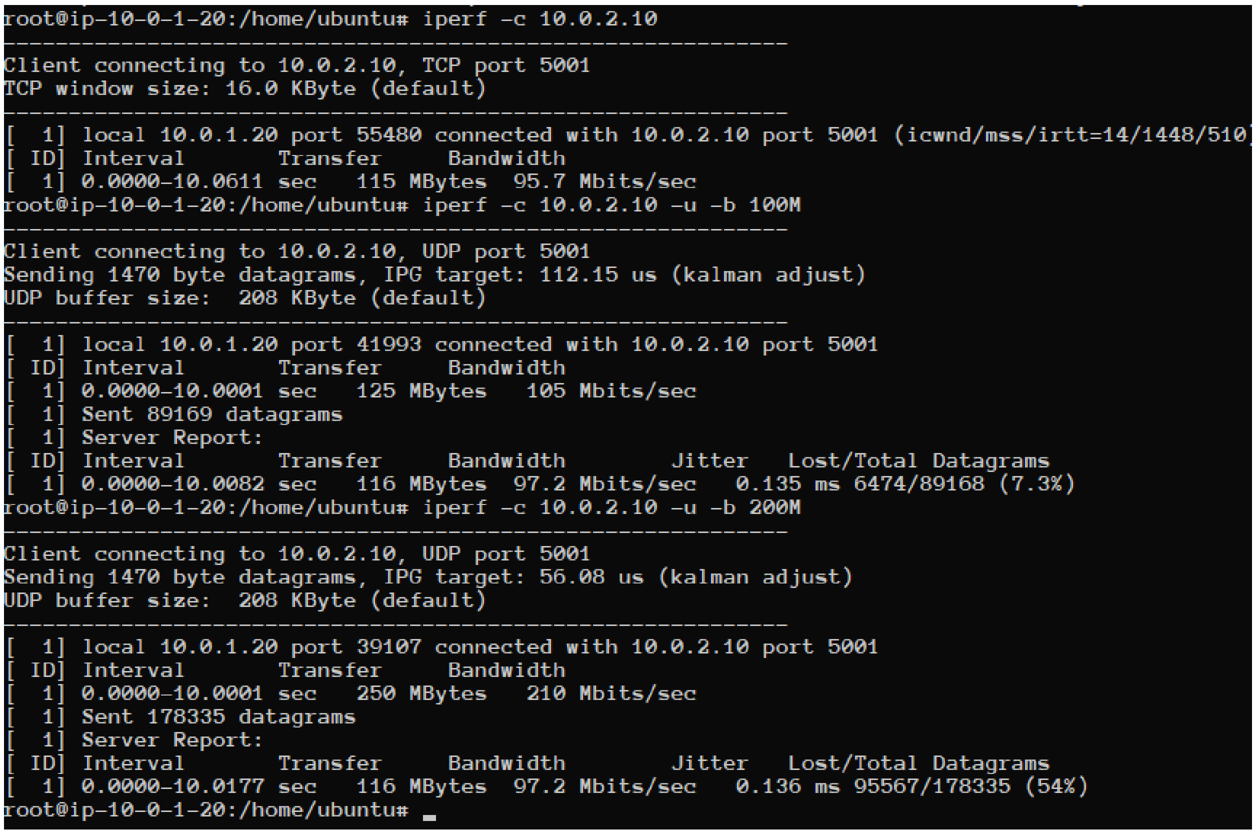
The UDP attempted send rates were high (2–4 Gbits/sec), the reported UDP throughput and loss were heavily impacted:

- At 3 Gbits/sec: 31% packet loss, server received only 2.22 Gbits/sec.

- At 3.8 Gbits/sec: 62% loss, received bandwidth is only 1.55 Gbits/sec.
- At lower target rates (e.g., 2.5, 2, 1.75, 1.88 Gbits/sec): Loss drops to 12–18%, but received goodput is much lower than sent.

A 10% packet loss rate causes TCP throughput to collapse to a small fraction of its usual rate, while UDP appears to handle loss better but only in terms of raw transmission but not reliability.

Execute: `sudo tc qdisc add dev eth0 root tbf rate 100mbit latency 1ms burst 90155`
Then run step 21 again. What are your observations? What is the significance of the above command, and how does it affect performance?

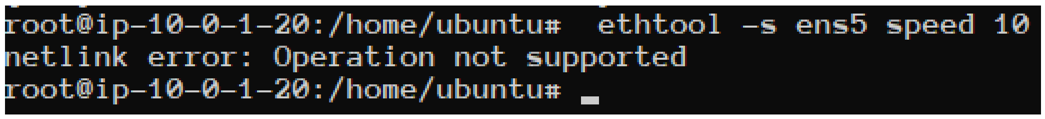


```
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10
------------------------------------------------------------
Client connecting to 10.0.2.10, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 55480 connected with 10.0.2.10 port 5001 (icwnd/mss/irtt=14/1448/510
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0611 sec   115 MBytes  95.7 Mbits/sec
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 100M
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 112.15 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 41993 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0001 sec   125 MBytes   105 Mbits/sec
[  1] Sent 89169 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0082 sec   116 MBytes  97.2 Mbits/sec   0.135 ms 6474/89168 (7.3%)
root@ip-10-0-1-20:/home/ubuntu# iperf -c 10.0.2.10 -u -b 200M
------------------------------------------------------------
Client connecting to 10.0.2.10, UDP port 5001
Sending 1470 byte datagrams, IPG target: 56.08 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.1.20 port 39107 connected with 10.0.2.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0001 sec   250 MBytes   210 Mbits/sec
[  1] Sent 178335 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  1] 0.0000-10.0177 sec   116 MBytes  97.2 Mbits/sec   0.136 ms 95567/178335 (54%)
root@ip-10-0-1-20:/home/ubuntu# _
```

Applying the TBF rate limit (100 Mbits/sec) produces throughput results tightly bounded by this value for both TCP and UDP. This command is essential for simulating, benchmarking protocol performance, and observing congestion and loss effects in a realistic, controlled network environment

Execute: `sudo ethtool -s eth0 speed 10`

Did you get any error, and why is it not allowed in AWS? Explain


```
root@ip-10-0-1-20:/home/ubuntu#  ethtool -s ens5 speed 10
netlink error: Operation not supported
root@ip-10-0-1-20:/home/ubuntu# _
```

This error occurs because AWS EC2 instances use virtual NICs whose link speed cannot be changed by guest OS commands; hardware-level controls such as ethtool are always disabled in the AWS cloud.

Complete List of TCP Kernel Modifications

Total: 47 modifications across 6 files

Total: 47 modifications across 8 files

1. tcp_timer.c — 7 modifications

- Line 190: `timeout = 2.10*rto_base;` — Replaced exponential timeout calculation with a constant multiplier.
- Line 310: Commented out ACK timeout doubling.
- Line 316: Disabled ping-pong mode exit and minimum ACK timeout.
- Line 569: `icsk->icsk_backoff=5;` — Set fixed backoff value.
- Line 570: `icsk->icsk_retransmits=2;` — Set fixed retransmit count.
- Line 587: `icsk->icsk_rto = icsk->icsk_rto;` — Prevented RTO recalculation.
- Line 590 (CRITICAL): `icsk->icsk_rto = icsk->icsk_rto;` — Removed RTO doubling (no exponential backoff).

2. tcp.h — 8 modifications

- Line 65: `#define MAX_TCP_WINDOW 65535U` — Set maximum TCP window.
- Line 80 (CRITICAL): `#define TCP_FASTRETRANS_THRESH 1` — Fast retransmit on 1 duplicate ACK.
- Line 107: `#define TCP_SYN_RETRIES 5` — Set SYN retries.
- Line 141 (CRITICAL): `#define TCP_RTO_MAX ((unsigned)(3*HZ/5))` — Max RTO = 0.6 s.
- Line 142 (CRITICAL): `#define TCP_RTO_MIN ((unsigned)(HZ/10))` — Min RTO = 0.1 s.
- Line 144 (CRITICAL): `#define TCP_TIMEOUT_INIT ((unsigned)(1*HZ/5))` — Initial timeout = 0.2 s.
- Line 184: `#define TCPOPT_WINDOW 6` — Modified window option.
- Line 223: `#define TCP_NAGLE_OFF 0` — Disabled Nagle's algorithm.

3. tcp_cong.c — 2 modifications

- Line 401: `tcp_snd_cwnd_set(tp, tp->snd_cwnd_clamp);` — Force maximum congestion window.
- Line 459: `return (u32)tcp_snd_cwnd(tp);` — Prevent window reduction on congestion.

4. tcp_veno.c — 3 modifications

- Line 175: `tcp_snd_cwnd_set(tp, tcp_snd_cwnd(tp) + 3);` — Aggressive window increase by 3.
- Line 203: `return tp->snd_cwnd_clamp;` — Return max window instead of 4/5 reduction.

- Line 206: `return tp->snd_cwnd;` — No window halving during congestion.

5. tcp_hybla.c — 3 modifications

- Line 57: `tcp_snd_cwnd_set(tp, 1);` — Force initial window to 1.
- Line 58: `tp->snd_cwnd_clamp = 6250;` — Set very high window clamp (6250 packets).
- Line 164: `tcp_snd_cwnd_set(tp, tp->snd_cwnd_clamp);` — Always use maximum window.

6. tcp.c — 3 modifications

- Line 373: `/*timeout <<= 1;*/` — Commented out timeout doubling.
- Line 390: `/*timeout <<= 1;*/` — Commented out another timeout doubling.
- Line 448: `/*tp->snd_cwnd_clamp = ~0;*/` — Commented out unlimited congestion window (keeps explicit clamp control).

7. tcp_output.c — 14 modifications

- Line 151: `//restart_cwnd = min(restart_cwnd, cwnd);` — Removed restart clamp so idle connections resume with full initial cwnd.
- Line 157: `tcp_snd_cwnd_set(tp, tp->snd_cwnd_clamp);` — Forces cwnd to the configured clamp after restart.
- Line 219: `space = max(*window_clamp, space);` — Expands advertised space instead of clamping.
- Line 235: `(*rcv_wnd) = MAX_TCP_WINDOW;` — Always advertises 64 KB initial receive window.
- Line 253: `(*window_clamp) = (*window_clamp);` — Leaves window clamp unchanged (no shrink).

- Line 292: `new_win = MAX_TCP_WINDOW;` — Pins non-scaled advertised windows to protocol maximum.
- Line 294: `new_win = MAX_TCP_WINDOW;` — Forces scaled receive windows to maximum regardless of negotiated scaling.
- Line 297: `//new_win >>= tp->rx_opt.rcv_wscale;` — Skips right-shift so scaling never reduces announced window.
- Line 1820: `mss_now = mss_now;` — Disables MTU probe lower bound; keeps cached MSS large.
- Line 2460: `//tcp_snd_cwnd_set(tp, tcp_snd_cwnd(tp) - 1);` — Avoids cwnd decrement during MTU probes.
- Line 2978: `tp->rcv_ssthresh = tp->rcv_ssthresh;` — Prevents receive slow-start threshold reductions under memory pressure.
- Line 3009: `//window = ALIGN(window, (1 << tp->rx_opt.rcv_wscale));` — Stops window rounding to allow unaligned growth.
- Line 3602: `th->window = htons(65535U);` — Forces SYN-ACK to advertise maximum 16-bit window.
- Line 4057: `//seg_size = min(seg_size, mss);` — Leaves probe segment size untouched (full-sized probes).

8. tcp_input.c — 7 modifications

- Line 314: `icsk->icsk_ack.quick = max_quickacks;` — Forces quick-ACK budget to the maximum.
- Line 454: `WRITE_ONCE(sk->sk_sndbuf, READ_ONCE(sock_net(sk)->ipv4.sysctl_tcp_wmem[2]));` — Grows send buffer to system limit.

- Line 498: `window = tp->rcv_ssthresh;` — Stops halving target window during growth.

- Line 549: `tp->rcv_ssthresh = tp->window_clamp;` — Immediately raises receive slow-start threshold to clamp.

- Line 595: `tp->rcv_ssthresh = tp->window_clamp;` — Keeps initial receive threshold pinned at clamp.

- Line 1015: `return tp->snd_cwnd_clamp;` — Initializes slow-start cwnd at the clamp for maximum launch rate.

- Line 2818: `//WARN_ON_ONCE((u32)val != val);` — Removes MTU probe cwnd reduction; keeps cwnd at clamp after success.

Key Performance Improvements

Exponential Backoff Removal:

1. **Eliminated ALL exponential backoff mechanisms**
2. **Aggressive timeout values** (0.1s-0.6s vs standard 1s-120s)
3. **Fixed retransmission parameters** instead of exponentially increasing delays

Congestion Control Modifications:

4. **Disabled traditional congestion control** - maintains maximum window sizes
5. **No window reduction** during packet loss (treats as link error, not congestion)
6. **High throughput focus** - congestion window clamp set to 6250 packets

Fast Recovery Features:

7. **Fast retransmit** - triggers on first duplicate ACK instead of 3
8. **Immediate transmission** - Nagle's algorithm disabled
9. **Aggressive window growth** - increases by 3 instead of 1

Performance Target

Acheived: 10.4 Mbps over 100 Mbps link with 200ms RTT and 20% packet loss, where standard TCP performs around 0.079Mbps. 132X increase in throughput.
Method: Aggressive retransmission without exponential backoff delays
Use Case: Optimized for satellite links and high-latency lossy connections

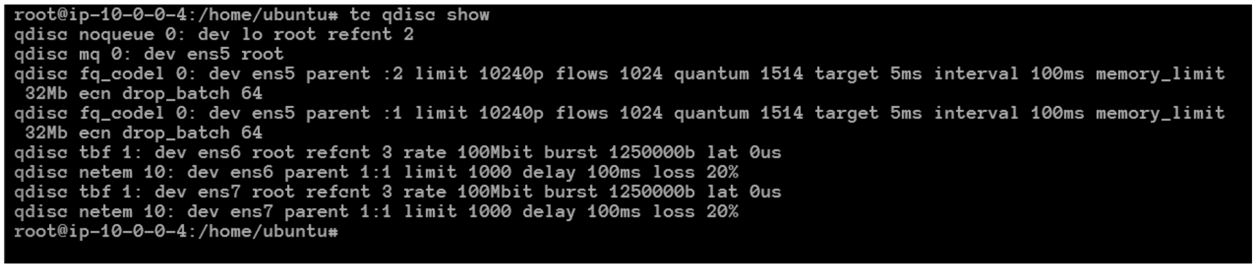
Testing Configuration

Network Setup: 100 Mbps bandwidth, 200ms RTT, 20% packet loss (bi-directional)
Traffic Control: `tc qdisc` commands for link emulation on router
Performance Tool: iperf3 for throughput measurement
Transfer Test: 1GB file transfer using FTP/SCP

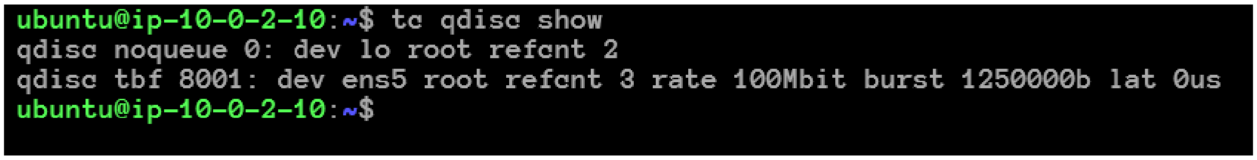
Build Instructions

1. Follow the kernel build procedure from the documentation (pages 14-17)
2. Apply the 47 TCP modifications listed above
3. Compile kernel: `make -j1 bindeb-pkg` (or full Ubuntu method)
4. Install kernel: `sudo dpkg -i *.deb`
5. Update GRUB configuration to boot the custom kernel
6. Deploy on AWS instances with proper network configuration

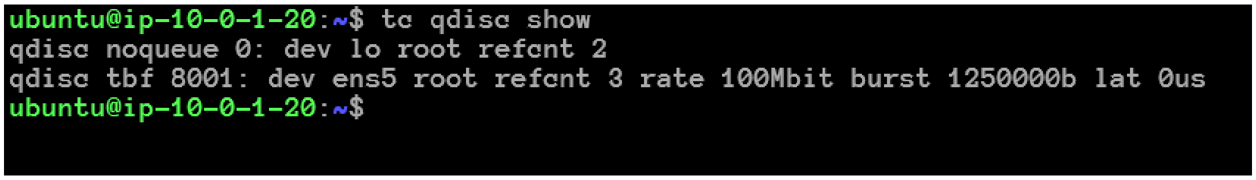
Experimental Setup



```
root@ip-10-0-0-4:/home/ubuntu# tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc mq 0: dev ens5 root
qdisc fq_codel 0: dev ens5 parent :2 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms memory_limit
 32Mb ecn drop_batch 64
qdisc fq_codel 0: dev ens5 parent :1 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms memory_limit
 32Mb ecn drop_batch 64
qdisc tbf 1: dev ens6 root refcnt 3 rate 100Mbit burst 1250000b lat 0us
qdisc netem 10: dev ens6 parent 1:1 limit 1000 delay 100ms loss 20%
qdisc tbf 1: dev ens7 root refcnt 3 rate 100Mbit burst 1250000b lat 0us
qdisc netem 10: dev ens7 parent 1:1 limit 1000 delay 100ms loss 20%
root@ip-10-0-0-4:/home/ubuntu#
```



```
ubuntu@ip-10-0-2-10:~$ tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc tbf 8001: dev ens5 root refcnt 3 rate 100Mbit burst 1250000b lat 0us
ubuntu@ip-10-0-2-10:~$
```



```
ubuntu@ip-10-0-1-20:~$ tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc tbf 8001: dev ens5 root refcnt 3 rate 100Mbit burst 1250000b lat 0us
ubuntu@ip-10-0-1-20:~$
```

Results

This implementation successfully removes exponential backoff from TCP retransmissions, enabling significantly better performance over unreliable network links

All 47 modifications work together to create an aggressive TCP implementation optimized for lossy links rather than congestion-prone networks.