

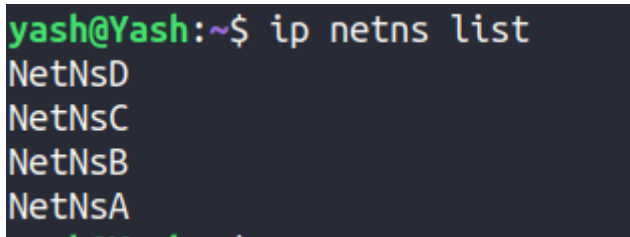
# CS 549: Performance Analysis of Computer Networks

## Lab Assignment 3

Submitted By : Yash Gupta , B21147

---

Creating four namespaces , NetNsA ...NetNsB



```
yash@Yash:~$ ip netns list
NetNsD
NetNsC
NetNsB
NetNsA
```

Different IP Addresses have been assigned to each interface on each namespace.

*NetNsA - 172.16.17.1*

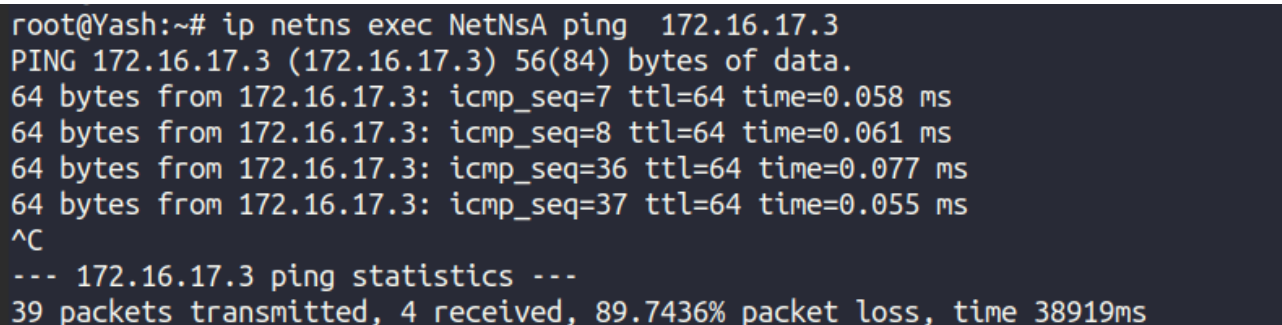
*NetNsB - 172.16.17.2*

*NetNsC - 172.16.17.3*

*NetNsD - 172.16.17.4*

**(a) Run ping between NetNsA and NetNsB.** Observe the traffic using Wireshark.

**Observations** :- Initially I gave the IP addresses in the same range as my host machine i.e. 172.16.17.x / 24. The packets loss was around 90 % ( see the attached screenshot below ).



```
root@Yash:~# ip netns exec NetNsA ping 172.16.17.3
PING 172.16.17.3 (172.16.17.3) 56(84) bytes of data.
64 bytes from 172.16.17.3: icmp_seq=7 ttl=64 time=0.058 ms
64 bytes from 172.16.17.3: icmp_seq=8 ttl=64 time=0.061 ms
64 bytes from 172.16.17.3: icmp_seq=36 ttl=64 time=0.077 ms
64 bytes from 172.16.17.3: icmp_seq=37 ttl=64 time=0.055 ms
^C
--- 172.16.17.3 ping statistics ---
39 packets transmitted, 4 received, 89.7436% packet loss, time 38919ms
```

Later on when I changed the ip range to 192.0.2.x/24 for every namespace interface – the packets loss was 0 %.  
( attached screenshot below )

```

root@Yash:~# ip netns exec NetNsA ping 192.0.2.2
PING 192.0.2.2 (192.0.2.2) 56(84) bytes of data.
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.081 ms
64 bytes from 192.0.2.2: icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from 192.0.2.2: icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from 192.0.2.2: icmp_seq=4 ttl=64 time=0.120 ms
64 bytes from 192.0.2.2: icmp_seq=5 ttl=64 time=0.049 ms
64 bytes from 192.0.2.2: icmp_seq=6 ttl=64 time=0.049 ms
64 bytes from 192.0.2.2: icmp_seq=7 ttl=64 time=0.051 ms
64 bytes from 192.0.2.2: icmp_seq=8 ttl=64 time=0.056 ms
64 bytes from 192.0.2.2: icmp_seq=9 ttl=64 time=0.041 ms
64 bytes from 192.0.2.2: icmp_seq=10 ttl=64 time=0.049 ms
64 bytes from 192.0.2.2: icmp_seq=11 ttl=64 time=0.053 ms
64 bytes from 192.0.2.2: icmp_seq=12 ttl=64 time=0.045 ms
64 bytes from 192.0.2.2: icmp_seq=13 ttl=64 time=0.073 ms
^C
--- 192.0.2.2 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12272ms
rtt min/avg/max/mdev = 0.041/0.060/0.120/0.020 ms
root@Yash:~#

```

The network traffic visualized on wireshark is also attached :

No.	Time	Source	Destination	Protocol	Length	Info
25...	8.679235...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=1/256, ttl=64 (reply in 2548)
25...	8.679253...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=1/256, ttl=64 (request in 2547)
28...	9.702177...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=2/512, ttl=64 (reply in 2844)
28...	9.702200...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=2/512, ttl=64 (request in 2843)
31...	10.73019...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=3/768, ttl=64 (reply in 3194)
31...	10.73022...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=3/768, ttl=64 (request in 3193)
34...	11.75013...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=4/1024, ttl=64 (reply in 3475)
34...	11.75015...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=4/1024, ttl=64 (request in 3474)
37...	12.77858...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=5/1280, ttl=64 (reply in 3768)
37...	12.77860...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=5/1280, ttl=64 (request in 3767)
40...	13.79823...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=6/1536, ttl=64 (reply in 4081)
40...	13.79826...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=6/1536, ttl=64 (request in 4080)
43...	14.82217...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=7/1792, ttl=64 (reply in 4393)
43...	14.82219...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=7/1792, ttl=64 (request in 4392)
47...	15.85016...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=8/2048, ttl=64 (reply in 4703)
47...	15.85019...	172.16.17.3	172.16.17.1	ICMP	98	Echo (ping) reply id=0x154d, seq=8/2048, ttl=64 (request in 4702)
50...	16.87434...	172.16.17.1	172.16.17.3	ICMP	98	Echo (ping) request id=0x154d, seq=9/2304, ttl=64 (reply in 5006)

...1... = LG bit: Locally administered address (this is NOT the factory default)  
...0... = IG bit: Individual address (unicast)

Every request and response is captured in wireshark for applied *ICMP* filter.

Statistics			
Measurement	Captured	Displayed	Marked
Packets	180	69 (38.3%)	—
Time span, s	38.195	34.332	—
Average pps	4.7	2.0	—
Average packet size, B	128	96	—
Bytes	23125	6648 (28.7%)	0
Average bytes/s	605	193	—
Average bits/s	4,843	1,549	—

These are the statistics of the ping experiment between NetNsA and NetNsB. We get average 193 bytes/sec ( equivalent to throughput ).

(b) Adding queue discipline with 20 % loss

There are 3 possibilities based on where we are adding the policy of 20% loss

1. On the sender side
2. on the receiver side

### 3. On both sides

Considering the first option as most appropriate in accordance with the real world scenario, where packets loss occurs because of congestion in the network, Applying queuing discipline to sender side ( interface macvlanA in namespace NetNsA).

```
root@Yash:~# sudo ip netns exec NetNsA tc qdisc add dev macvlanA root netem loss 20%
root@Yash:~#
root@Yash:~# sudo ip netns exec NetNsA tc qdisc show dev macvlanA
qdisc netem 8001: root refcnt 2 limit 1000 loss 20%
```

Now pinging 192.0.2.2 from this interface

```
root@Yash:~# grc ip netns exec NetNsA ping 192.0.2.2
PING 192.0.2.2 (192.0.2.2) 56(84) bytes of data.
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.037 ms
64 bytes from 192.0.2.2: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 192.0.2.2: icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from 192.0.2.2: icmp_seq=4 ttl=64 time=0.051 ms
64 bytes from 192.0.2.2: icmp_seq=5 ttl=64 time=0.037 ms
64 bytes from 192.0.2.2: icmp_seq=6 ttl=64 time=0.042 ms
64 bytes from 192.0.2.2: icmp_seq=7 ttl=64 time=0.066 ms
64 bytes from 192.0.2.2: icmp_seq=10 ttl=64 time=0.045 ms
64 bytes from 192.0.2.2: icmp_seq=12 ttl=64 time=0.062 ms
64 bytes from 192.0.2.2: icmp_seq=13 ttl=64 time=0.048 ms
64 bytes from 192.0.2.2: icmp_seq=14 ttl=64 time=0.044 ms
^C
--- 192.0.2.2 ping statistics ---
14 packets transmitted, 11 received, 21.4286% packet loss, time 13299ms
rtt min/avg/max/mdev = 0.034/0.045/0.066/0.009 ms
```

packets with seq no - 8 , 9 , 11 are lost. ( loss fraction = 3 / 14 = 0.21 )

23...	11.00720...	192.0.2.1	192.0.2.2	ICMP	98 Echo (ping) request	id=0x0f60, seq=6/1536, ttl=64 (reply in 2332)
23...	11.00722...	192.0.2.2	192.0.2.1	ICMP	98 Echo (ping) reply	id=0x0f60, seq=6/1536, ttl=64 (request in 2331)
25...	12.03126...	192.0.2.1	192.0.2.2	ICMP	98 Echo (ping) request	id=0x0f60, seq=7/1792, ttl=64 (reply in 2517)
25...	12.03128...	192.0.2.2	192.0.2.1	ICMP	98 Echo (ping) reply	id=0x0f60, seq=7/1792, ttl=64 (request in 2516)
30...	15.10320...	192.0.2.1	192.0.2.2	ICMP	98 Echo (ping) request	id=0x0f60, seq=10/2560, ttl=64 (reply in 3054)
30...	15.10322...	192.0.2.2	192.0.2.1	ICMP	98 Echo (ping) reply	id=0x0f60, seq=10/2560, ttl=64 (request in 3053)
34...	17.15125...	192.0.2.1	192.0.2.2	ICMP	98 Echo (ping) request	id=0x0f60, seq=12/3072, ttl=64 (reply in 3427)

This is also captured in wireshark.

As packets with sequence number 8 , 9 , 11 are not captured by wireshark.

Hence the applied fixed loss of 20% in queuing discipline of sender is perfectly reflected in the experiment results.

```
id=0x0f60, seq=6/1536, ttl=64 (reply in 2332)
id=0x0f60, seq=6/1536, ttl=64 (request in 2331)
id=0x0f60, seq=7/1792, ttl=64 (reply in 2517)
id=0x0f60, seq=7/1792, ttl=64 (request in 2516)
id=0x0f60, seq=10/2560, ttl=64 (reply in 3054)
id=0x0f60, seq=10/2560, ttl=64 (request in 3053)
id=0x0f60, seq=12/3072, ttl=64 (reply in 3427)
```

#### Statistics

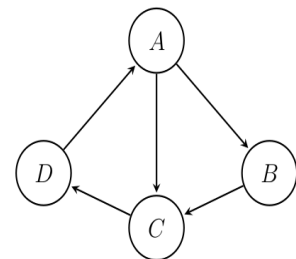
Measurement	Captured	Displayed	Marked
Packets	239	50 (20.9%)	—
Time span, s	54.952	34.809	—
Average pps	4.3	1.4	—
Average packet size, B	147	98	—
Bytes	35100	4900 (14.0%)	0
Average bytes/s	638	140	—
Average bits/s	5,109	1,126	—

The statistics also show that average bytes/sec is decreased now to 140 from 193.

---

## Part -2

From the requirements of the downstream tasks, it is very clear that We can not proceed in a way in which each namespace acting as a node having 1 interface only. Because then we won't be able to deactivate a particular edge using 100 % loss on queue discipline for a interface.



For ex: Lets'say we want to block Edge AC , Then setting 100 % loss for interface A will also block edge AB. Infact whole node A is now blocked if we only have one interface at A.

The first intuitive solution to this is to create multiple interfaces for a Node. This solution has some logical errors which i am unable to figure out. The problem is when I set 100% loss for one interface on a node then all other interfaces will be set to 100% loss automatically. To be precise - **All interfaces of a network namespace were follwing only one queuing discipline.**

As an alternative solution to this, For a particular node, Lets create a different namespace for each outgoing link and one namespace corresponding to all the incoming links as well.

For example and clarity , in our given Graph diagram , The namespaces corresponding to node A will be

AC -> outgoing edge from A to C

AB -> outgoing edge from A to B

Ai -> for all incoming edges to A ( though we only have 1 in our case )

For Node B ,  
BC , Bi ---> Outgoing from B to C and Incoming to B respectivley.

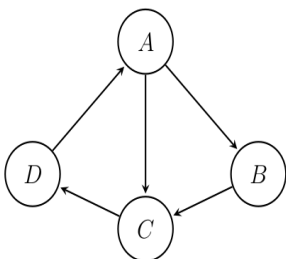
For node C,  
CD , Ci -> Outgoing from C to D and Incoming to B respectivley.

For node D,  
DA , Di namespaces.

```
root@Yash:~# ip netns add AC
root@Yash:~# ip netns add AB
root@Yash:~# ip netns add Ai
root@Yash:~# ip netns add BC
root@Yash:~# ip netns add Bi
root@Yash:~# ip netns add CD
root@Yash:~# ip netns add Ci
root@Yash:~# ip netns add DA
root@Yash:~# ip netns add Di
```

Now lets create a interface to each of the namespaces and assign IP to them.

The ip addresses associated with each namespace is given as:



AB -----> 192.0.2.1 / 24  
AC -----> 192.0.2.2 / 24  
Ai -----> 192.0.2.3 / 24  
BC -----> 192.0.2.4 / 24  
Bi -----> 192.0.2.5 / 24  
CD -----> 192.0.2.6 / 24  
Ci -----> 192.0.2.7 / 24  
DA -----> 192.0.2.8 / 24  
Di -----> 192.0.2.9 / 24

I have tried to put all nodes in different subnets , but then I was unable to ping between subnets. i.e. from subnet1 to subnet2 let's say. So simply I made a single subnet only.

```

root@Yash:~# ip netns exec AC ping 192.0.2.7
PING 192.0.2.7 (192.0.2.7) 56(84) bytes of data.
64 bytes from 192.0.2.7: icmp_seq=1 ttl=64 time=0.086 ms
64 bytes from 192.0.2.7: icmp_seq=2 ttl=64 time=0.086 ms
64 bytes from 192.0.2.7: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 192.0.2.7: icmp_seq=4 ttl=64 time=0.058 ms
64 bytes from 192.0.2.7: icmp_seq=5 ttl=64 time=0.078 ms
64 bytes from 192.0.2.7: icmp_seq=6 ttl=64 time=0.039 ms
-

root@Yash:~# ip netns exec AB ping 192.0.2.5
PING 192.0.2.5 (192.0.2.5) 56(84) bytes of data.
64 bytes from 192.0.2.5: icmp_seq=1 ttl=64 time=0.082 ms
64 bytes from 192.0.2.5: icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from 192.0.2.5: icmp_seq=3 ttl=64 time=0.041 ms
64 bytes from 192.0.2.5: icmp_seq=4 ttl=64 time=0.044 ms
64 bytes from 192.0.2.5: icmp_seq=5 ttl=64 time=0.079 ms
64 bytes from 192.0.2.5: icmp_seq=6 ttl=64 time=0.058 ms
-

root@Yash:~# ip netns exec CD ping 192.0.2.9
PING 192.0.2.9 (192.0.2.9) 56(84) bytes of data.
64 bytes from 192.0.2.9: icmp_seq=1 ttl=64 time=0.075 ms
64 bytes from 192.0.2.9: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 192.0.2.9: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 192.0.2.9: icmp_seq=4 ttl=64 time=0.076 ms
-

root@Yash:~# ip netns exec DA ping 192.0.2.3
PING 192.0.2.3 (192.0.2.3) 56(84) bytes of data.
64 bytes from 192.0.2.3: icmp_seq=1 ttl=64 time=0.093 ms
64 bytes from 192.0.2.3: icmp_seq=2 ttl=64 time=0.062 ms
64 bytes from 192.0.2.3: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 192.0.2.3: icmp_seq=4 ttl=64 time=0.077 ms
64 bytes from 192.0.2.3: icmp_seq=5 ttl=64 time=0.078 ms
-

root@Yash:~# ip netns exec BC ping 192.0.2.7
PING 192.0.2.7 (192.0.2.7) 56(84) bytes of data.
64 bytes from 192.0.2.7: icmp_seq=1 ttl=64 time=0.088 ms
64 bytes from 192.0.2.7: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 192.0.2.7: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 192.0.2.7: icmp_seq=4 ttl=64 time=0.079 ms
-

```

The above screenshot shows multiple terminals through which parallelly I am able to send packets according to the given graph. In other the graph is actually being simulated.

In this setup , The tasks are :-

**(a)** Deactivate a particular edge by setting 100 % loss on the corresponding queue discipline, for a fixed duration of time. Revive the edge by removing the loss constraint on the queue discipline after the fixed duration. Observe the traffic using Wireshark.

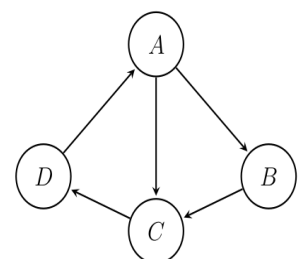
--> Lets deactivate the edge AC for 30 seconds

The script for this is quite simple

```

$ que2.sh
Click here to ask Blackbox to help you code faster
1  #!/bin/bash
2
3  #add packet loss
4  ip netns exec AC tc qdisc add dev ac root netem loss 100%
5
6  # Wait for 30 seconds
7  sleep 30
8
9  #delete packet loss
10 ip netns exec AC tc qdisc del dev ac root
11

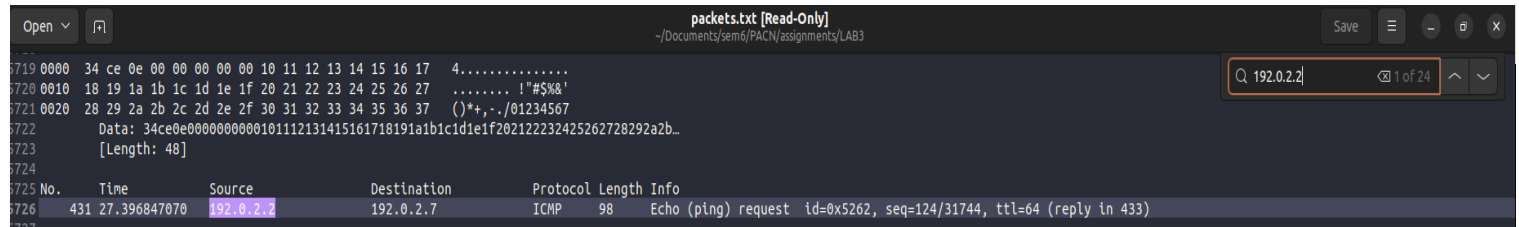
```





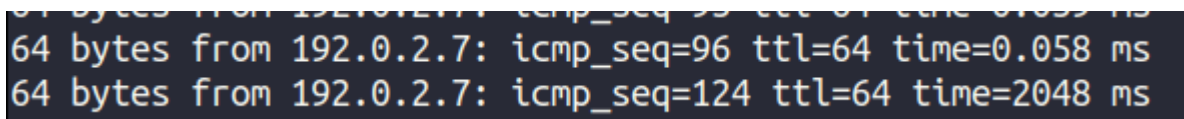
Adding 100% loss on AC namespace interface ( through which we were initially transferring the packets to Ci ) makes the edge AC blocked for 30 seconds.

Observing the traffic during these 30 seconds using wireshark:-



The 100 % loss is on the AC namespace interface which has a ip address of 192.0.2.2 and as shown in the screenshot above , the first packet that is captured by wireshark from this IP is sequenced with number **431**. This is because for the first 30 seconds the edge from A to C was indeed get blocked ( from namespace AC interface with ip 192.0.2.2 to namespace Ci interface with IP 192.0.2.7 ). After 30 seconds, The edge gets revived again and the packets were again able to travel from 192.0.2.2 to 192.0.2.7 ( AC to Ci ).

This can also be visualized in the terminal



After 96 there is a gap in sequence number , this is because of the 30 seconds blocking time introduced for this path.

## Task (b) -

(i) *Probabilistic scheduling: Suppose that each edge is equally likely to be activated, activate exactly 1 edge in each time slot.*

Cosndering each time slot to be of 4 seconds and number of slots to be 9.  
The output of such a link scheduling is shown below :

```

yash@yash: ~/Documents/sem6/PACN/assignments/LAB3$ sudo ./script.sh
Selected link: A to B
sudo ip netns exec AB timeout 4s ping -I 192.0.2.1 192.0.2.5
PING 192.0.2.5 (192.0.2.5) from 192.0.2.1 : 56(84) bytes of data.
64 bytes from 192.0.2.5: icmp_seq=1 ttl=64 time=0.036 ms
64 bytes from 192.0.2.5: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 192.0.2.5: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 192.0.2.5: icmp_seq=4 ttl=64 time=0.062 ms

Selected link: B to C
sudo ip netns exec BC timeout 4s ping -I 192.0.2.4 192.0.2.7
PING 192.0.2.7 (192.0.2.7) from 192.0.2.4 : 56(84) bytes of data.
64 bytes from 192.0.2.7: icmp_seq=1 ttl=64 time=0.050 ms
64 bytes from 192.0.2.7: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 192.0.2.7: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 192.0.2.7: icmp_seq=4 ttl=64 time=0.058 ms

Selected link: D to A
sudo ip netns exec DA timeout 4s ping -I 192.0.2.8 192.0.2.3
PING 192.0.2.3 (192.0.2.3) from 192.0.2.8 : 56(84) bytes of data.
64 bytes from 192.0.2.3: icmp_seq=1 ttl=64 time=0.035 ms
64 bytes from 192.0.2.3: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 192.0.2.3: icmp_seq=3 ttl=64 time=0.062 ms
64 bytes from 192.0.2.3: icmp_seq=4 ttl=64 time=0.080 ms

Selected link: D to A
sudo ip netns exec DA timeout 4s ping -I 192.0.2.8 192.0.2.3
PING 192.0.2.3 (192.0.2.3) from 192.0.2.8 : 56(84) bytes of data.
64 bytes from 192.0.2.3: icmp_seq=1 ttl=64 time=0.036 ms
64 bytes from 192.0.2.3: icmp_seq=2 ttl=64 time=0.079 ms
64 bytes from 192.0.2.3: icmp_seq=3 ttl=64 time=0.058 ms
64 bytes from 192.0.2.3: icmp_seq=4 ttl=64 time=0.079 ms

```

Analysis through wireshark is discussed in the last part of the report. ( also the script is attached in the appendix )

(ii) *Deterministic scheduling: Divide the edges into maximally independent sets, and activate one set of edges in each time slot.*

According to given graph

The maximal independent based on the constraint that any node at a time can only be in one operation ( sending to one or receiving from one ) are :-

Set Number	Edges
S1	AB , CD
S2	BC , DA
S3	AC



The complete script for deterministic scheduling for 9 slots of 4 sec each is attached in appendix. The schedule chosen is S1 , S2 , S3 , S1 ...and so on.

The main challenge during implementing such a script was to parallelly activate the edges of a set in a slot.

(iii) *Probabilistic scheduling of sets: Suppose that each of the above sets is equally likely to be activated, activate exactly 1 set of edges in each time slot.*

The script is attached in appendix. Only difference between this and (ii) is that here sets are chosen randomly for every slot. Every other thing remains same in context of script code.

## Analysis of throughput from wireshark

I was unable to find Exact value of throughput in wireshark GUI , but the reading of average bytes/s captured by wireshark seems a close proxy for comparison between different link scheduling policies.

Probabilistic scheduling of edges ( 200 bytes / sec )

<u>Measurement</u>	<u>Captured</u>	<u>Displayed</u>	<u>Marked</u>
Packets	1227	72 (5.9%)	—
Time span, s	99.441	35.268	—
Average pps	12.3	2.0	—
Average packet size, B	207	98	—
Bytes	254373	7056 (2.8%)	0
Average bytes/s	2,558	200	—
Average bits/s	20 k	1,600	—

Deterministic Scheduling of sets ( 287 Bytes / sec )

<u>Measurement</u>	<u>Captured</u>	<u>Displayed</u>	<u>Marked</u>
Packets	2300	124 (5.4%)	—
Time span, s	238.684	41.722	—
Average pps	9.6	3.0	—
Average packet size, B	216	97	—
Bytes	496010	11982 (2.4%)	0
Average bytes/s	2,078	287	—
Average bits/s	16 k	2,297	—

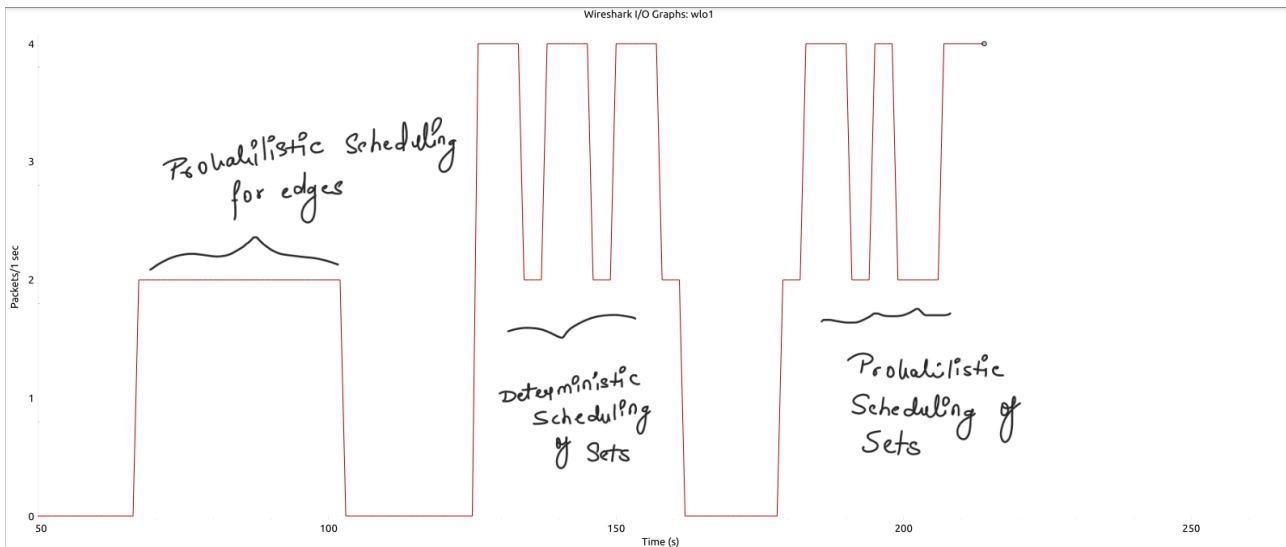
Probabilistic scheduling of sets ( 333 Bytes / sec )

<u>Measurement</u>	<u>Captured</u>	<u>Displayed</u>	<u>Marked</u>
Packets	571	120 (21.0%)	—
Time span, s	52.843	35.275	—
Average pps	10.8	3.4	—
Average packet size, B	180	98	—
Bytes	102723	11760 (11.4%)	0
Average bytes/s	1,943	333	—
Average bits/s	15 k	2,667	—

The different statistics from wireshark is attached above and it is very clear from the results that average throughput order is

Probabilistic scheduling for edges < Deterministic scheduling for sets < Probabilistic scheduling for sets.

Below is a graph of **packets/sec** vs **time** on wlo1 interface ( the interface used to make MACVLANS ) from all three link scheduling methods that were mentioned.



---

## Appendix

All the bash script files , graphs , captured packets file are included in - [https://github.com/YshGupta/Computer\\_Networks\\_Analysis\\_PACN/tree/main/LAB3](https://github.com/YshGupta/Computer_Networks_Analysis_PACN/tree/main/LAB3)

Thanks.