In The Name Of God

Sharif University of Technology
Department of Electrical Engineering

Data Network
Instructor: Dr. Pakravan

Linux Networking

Mohammad Javad Amin
401211193

Winter 2024

Contents

Introduction

Network Hypervisors

A network hypervisor, also known as a network virtualization platform, is a software layer that abstracts and virtualizes network resources, similar to how a hypervisor virtualizes computing resources in server virtualization. Network virtualization by decoupling the network control plane from the underlying hardware, allowing for more flexibility, scalability, and efficient resource utilization. [1] Here are some key reasons why network hypervisors are required:

1. **Resource Abstraction:** Network hypervisors abstract physical network devices and resources, making it easier to manage and allocate network resources dynamically. This abstraction enables the creation of virtual networks that operate independently of the underlying physical infrastructure.

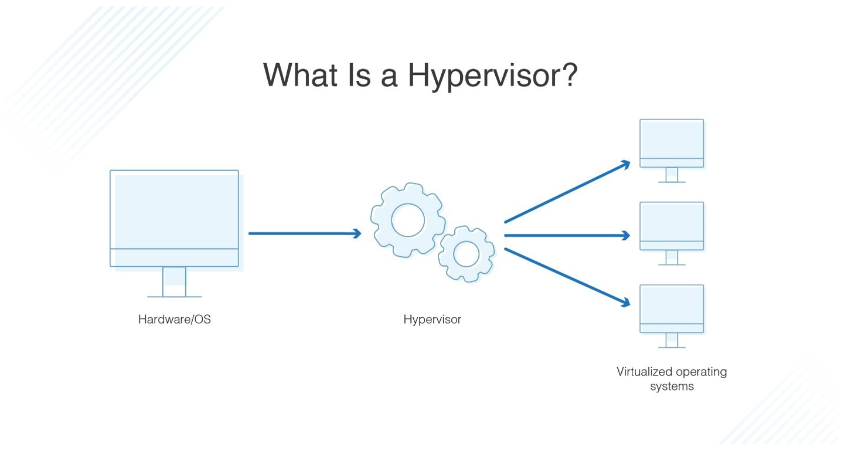
2. **Isolation and Segmentation:** Network hypervisors facilitate the creation of isolated and segmented virtual networks on top of a shared physical network. This is crucial for multi-tenancy scenarios where different users or applications need to operate securely without interfering with each other.

3. **Flexibility and Agility:** By decoupling the network control plane from hardware, network hypervisors provide greater flexibility and agility in managing network configurations. Changes can be made dynamically without requiring modifications to the underlying physical infrastructure.

4. **Centralized Network Management:** Network hypervisors centralize the control and management of network resources, allowing administrators to define and enforce network policies from a central point. This centralized management simplifies network administration and reduces the likelihood of configuration errors.

5. **Scalability:** Virtualized networks created by network hypervisors can easily scale up or down based on changing requirements. This scalability is essential in modern dynamic environments where workloads and network demands can vary rapidly.

6. **Optimized Resource Utilization:** Network hypervisors contribute to optimized resource utilization by allowing multiple virtual networks to share the same physical infrastructure. This ensures that network resources are efficiently used, leading to cost savings and improved overall network performance. [2]

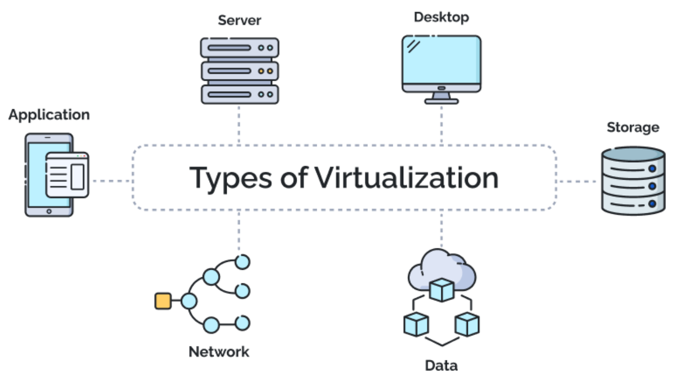


Virtualization in Networks

Virtualization in networks refers to the process of creating virtual instances or representations of physical network resources, such as servers, storage devices, or networks. This technology enables the efficient utilization of resources, better scalability, and flexibility in managing network infrastructure. [3]

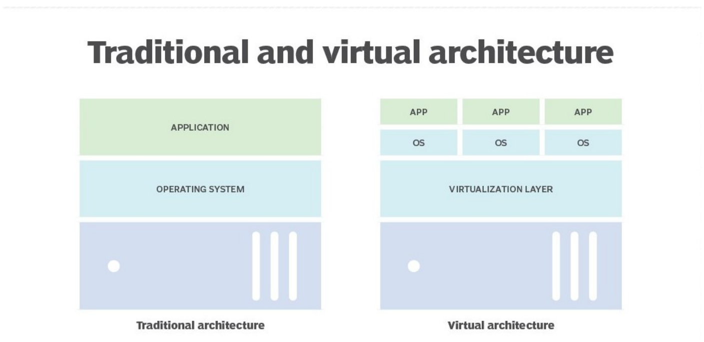
Types of Virtualization in Networks:

1. **Server Virtualization:** Involves creating virtual instances of servers on a single physical server. Each virtual server operates independently with its own operating system.
2. **Network Virtualization:** Focuses on creating virtual networks that operate independently of the physical network infrastructure. It allows for the segmentation of a physical network into multiple virtual networks.
3. **Storage Virtualization:** Involves pooling and managing physical storage devices to create virtual storage resources. It provides flexibility in allocating storage space to different applications or users.
4. **Desktop Virtualization:** Refers to the creation of virtual desktop environments that can be accessed remotely. Users can interact with a virtual desktop hosted on a server rather than a physical machine.
5. **Application Virtualization:** Allows applications to run in isolated environments, separate from the underlying operating system. This enhances compatibility and reduces conflicts between applications.



Role of Virtual Machines (VMs) in Virtualization:

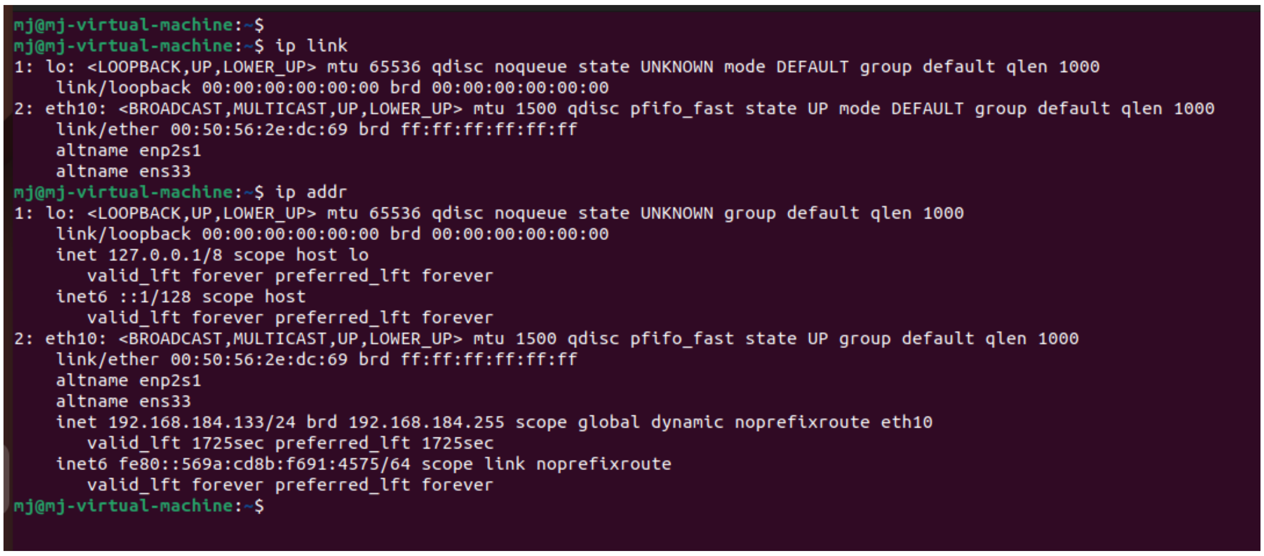
- **Isolation and Independence:** Virtual Machines provide isolation between different operating systems and applications. Each VM operates independently, allowing for the simultaneous running of multiple operating systems on a single physical server.
- **Resource Optimization:** VMs enable efficient utilization of hardware resources by running multiple instances on the same physical server. This leads to improved resource utilization and cost savings.
- **Flexibility and Scalability:** VMs can be easily created, duplicated, or deleted, providing flexibility in managing workloads. This scalability allows for adapting to changing resource requirements without significant hardware changes.
- **Snapshot and Migration:** Virtualization platforms often offer features like snapshots, allowing for the capture of a VM's state at a specific point. This facilitates easy backup, recovery, and migration of VMs between physical servers.
- **Testing and Development:** VMs are widely used for testing and development purposes. Developers can create isolated environments to test applications without affecting the production environment.
- **Disaster Recovery:** Virtualization supports efficient disaster recovery strategies by enabling the quick restoration of VMs from backups. This helps minimize downtime and data loss in case of hardware failures or disasters. [4]



Basics of Linux Network Administration

Network Interfaces

```
$ ip link
$ ip addr
```



```
mj@mj-virtual-machine:~$
mj@mj-virtual-machine:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:50:56:2e:dc:69 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    altname ens33
mj@mj-virtual-machine:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:50:56:2e:dc:69 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    altname ens33
    inet 192.168.184.133/24 brd 192.168.184.255 scope global dynamic noprefixroute eth10
       valid_lft 1725sec preferred_lft 1725sec
    inet6 fe80::569a:cd8b:f691:4575/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
mj@mj-virtual-machine:~$
```

- **lo (Loopback):**
  - IP Address: 127.0.0.1
  - MAC Address: Not explicitly shown, as the loopback interface does not have a physical MAC address associated with it.
- **eth10 (Ethernet):**
  - IP Address: 192.168.184.133
  - MAC Address: 00:50:56:2e:dc:69

The loopback interface (`lo`) is used for local communication within the host itself, and it always has the IP address 127.0.0.1. The Ethernet interface (`eth10`) in a virtual machine is utilized for network communication, facilitating connectivity with the host's physical network and enabling internet access. [5]

How the System Has Received Its IP Configuration:

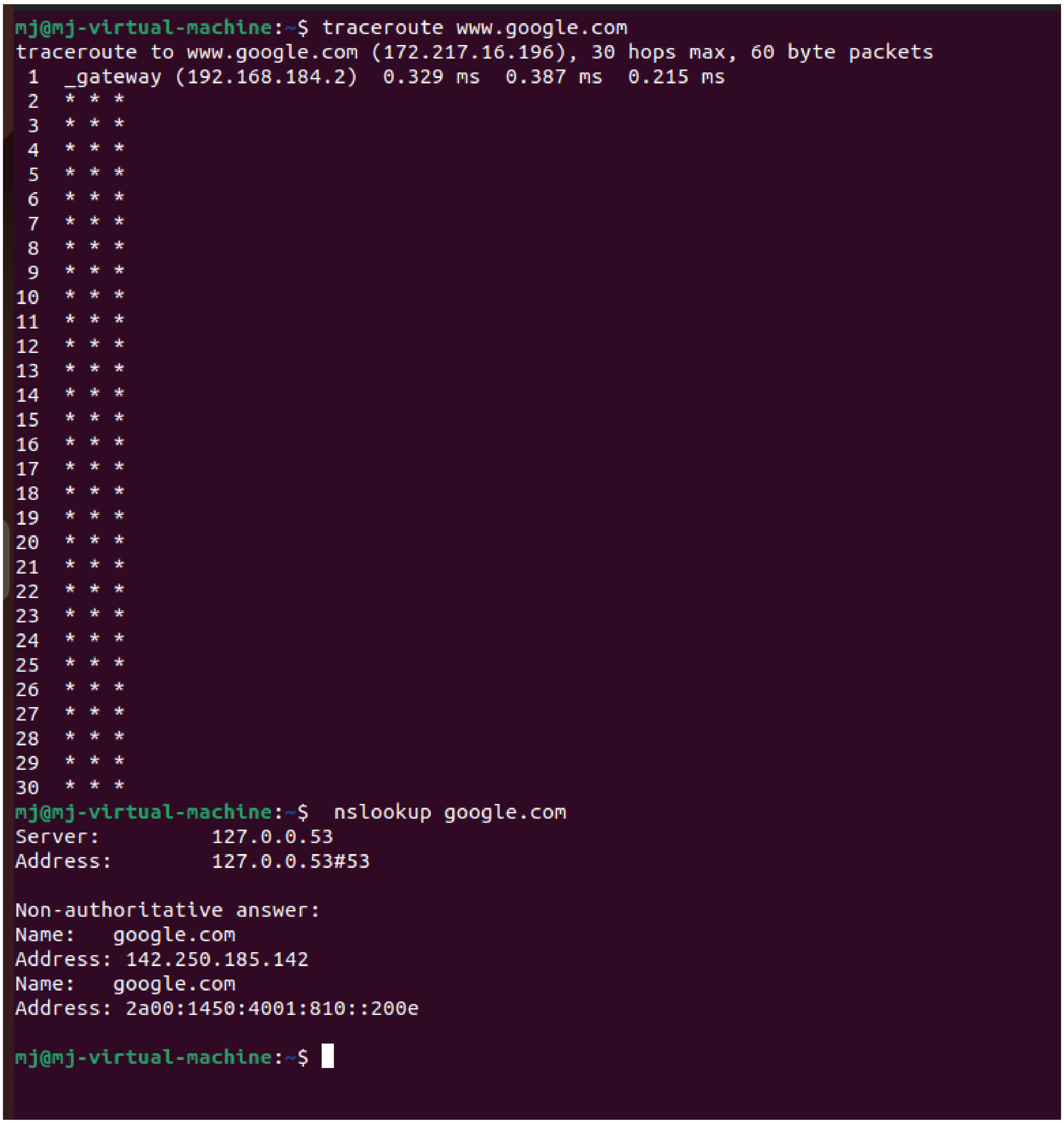
- The loopback interface (`lo`) is assigned the loopback IP address (127.0.0.1) by default and does not require external configuration.
- The Ethernet interface (`eth10`) is assigned an IP address (192.168.184.133) through dynamic configuration (DHCP), as indicated by "dynamic" in the output.

Probing the network between the local system and a destination

a. Traceroute

```
 $ traceroute www.google.com
```

This command will display the route that packets take to reach the google.com, showing the IP addresses of routers along the path.



```
mj@mj-virtual-machine:~$ traceroute www.google.com
traceroute to www.google.com (172.217.16.196), 30 hops max, 60 byte packets
 1  _gateway (192.168.184.2)  0.329 ms  0.387 ms  0.215 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
mj@mj-virtual-machine:~$  nslookup google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.185.142
Name:   google.com
Address: 2a00:1450:4001:810::200e

mj@mj-virtual-machine:~$ 
```

What Do Three Stars Represent?

We only see ***. This indicates that the router at that hop did not respond to the traceroute probe.

- Some Routers Don't Respond Because of the Overload:
  A router may be too busy to respond to the traceroute probe, treating it as a low-priority event. To check if this is the case, repeating the traceroute for the same destination multiple times may reveal whether the absence of *** was due to a temporary overload. If the path remains the same but *** disappears, it suggests a temporary overload. If the path changes, it may be because the overloaded router is temporarily avoided by other routers in the network.

- Some Routers Don't Want to Respond:
  Certain organizations configure their routers not to respond to traceroute requests to avoid revealing internal network details. [6]

b. Routers Shown by Name:

The routers shown by name instead of their IP addresses are resolved using reverse DNS lookup. When performing a traceroute, the tool attempts to resolve the IP addresses to corresponding hostnames. If the routers have reverse DNS entries, their names will be displayed.

c. Benefit of Knowing Routers in the Path:

Being aware of the routers in the path provides several benefits:

- **Network Troubleshooting:** It helps diagnose network issues by identifying the specific routers where packet loss or latency might be occurring.
- **Optimization:** Knowing the network path allows for optimization of routes, potentially improving network performance.
- **Security:** Understanding the route helps in identifying potential security risks or unauthorized network paths.

d. Find IP Address Record for 'google.com':

```
$ nslookup google.com
```

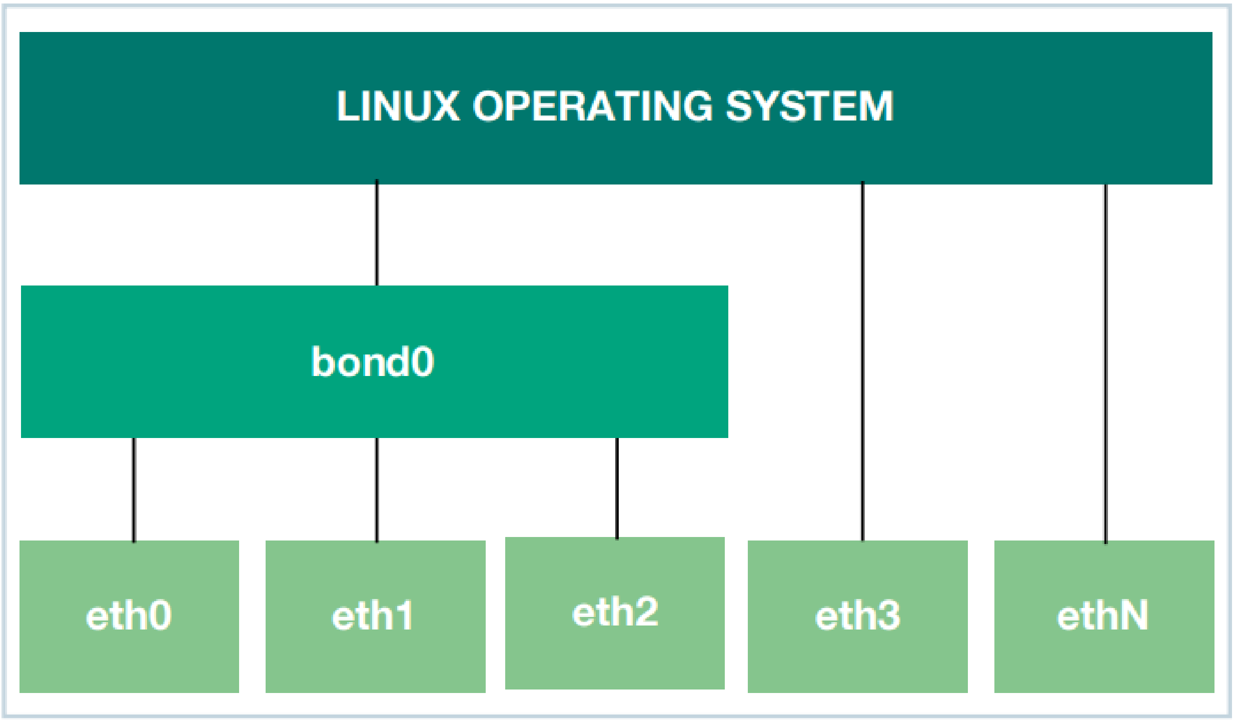
or

```
$ dig +short google.com
```

This commands will provide the IP address associated with the 'google.com' server.

Network Interface Bonding

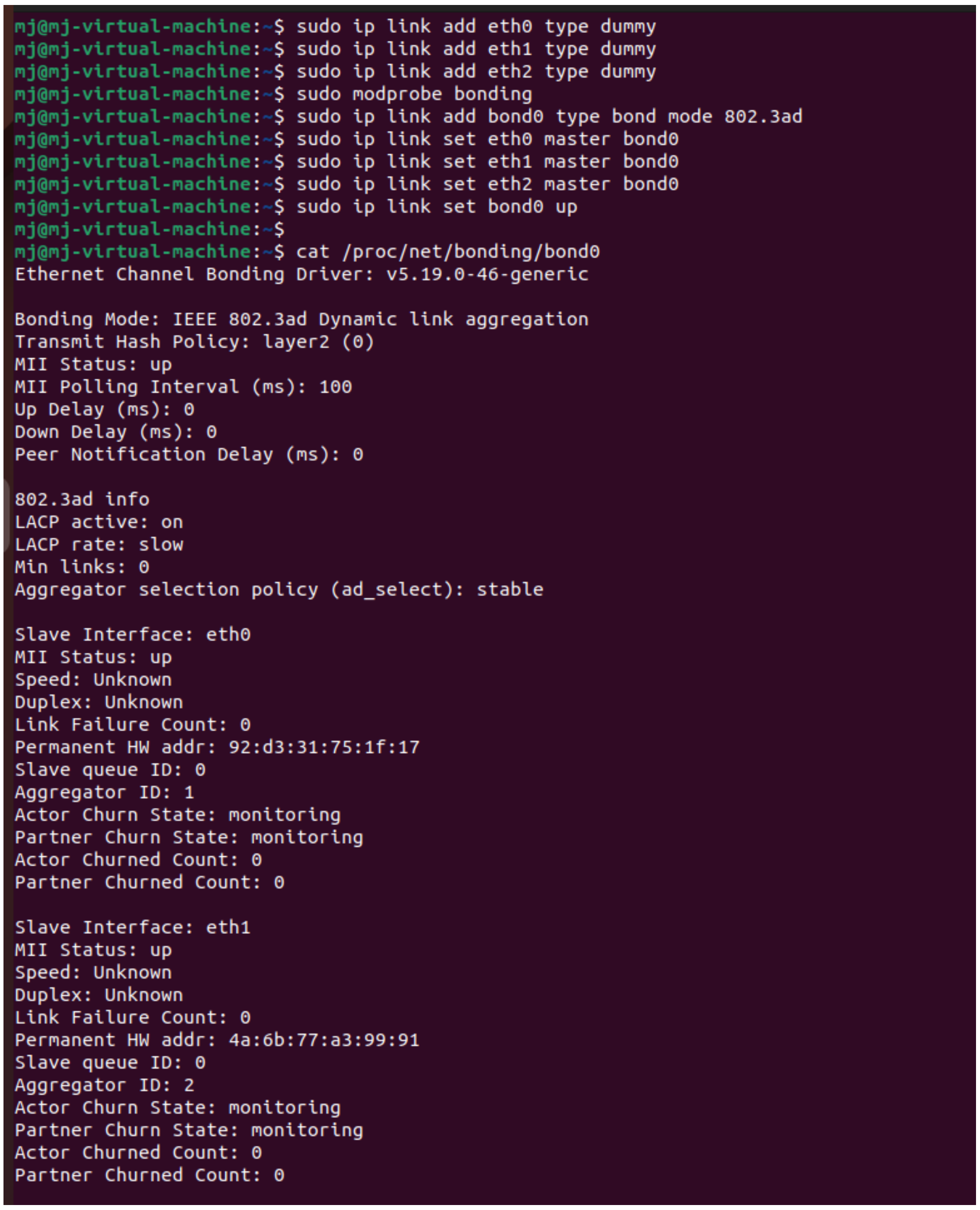
a. Create a bond



Multiple Ethernet interfaces bonded into a single network interface. [5]

```
# Create Interfaces
$ sudo ip link add eth0 type dummy
$ sudo ip link add eth1 type dummy
$ sudo ip link add eth2 type dummy

# All interface for adding to bond must be down

$ sudo modprobe bonding
$ sudo ip link add bond0 type bond mode 802.3ad

# Add eth0, eth1, and eth2 to the bond
$ sudo ip link set eth0 master bond0
$ sudo ip link set eth1 master bond0
$ sudo ip link set eth2 master bond0

# Activate the bond interface
$ sudo ip link set bond0 up

# To check the status of a bonded interface
$ cat /proc/net/bonding/bond0
```

This will create a bond interface named ‘bond0‘ consisting of the three physical interfaces (‘eth0‘, ‘eth1‘, and ‘eth2‘).



```
mj@mj-virtual-machine:~$ sudo ip link add eth0 type dummy
mj@mj-virtual-machine:~$ sudo ip link add eth1 type dummy
mj@mj-virtual-machine:~$ sudo ip link add eth2 type dummy
mj@mj-virtual-machine:~$ sudo modprobe bonding
mj@mj-virtual-machine:~$ sudo ip link add bond0 type bond mode 802.3ad
mj@mj-virtual-machine:~$ sudo ip link set eth0 master bond0
mj@mj-virtual-machine:~$ sudo ip link set eth1 master bond0
mj@mj-virtual-machine:~$ sudo ip link set eth2 master bond0
mj@mj-virtual-machine:~$ sudo ip link set bond0 up
mj@mj-virtual-machine:~$
mj@mj-virtual-machine:~$ cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v5.19.0-46-generic

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

802.3ad info
LACP active: on
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable

Slave Interface: eth0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 92:d3:31:75:1f:17
Slave queue ID: 0
Aggregator ID: 1
Actor Churn State: monitoring
Partner Churn State: monitoring
Actor Churned Count: 0
Partner Churned Count: 0

Slave Interface: eth1
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 4a:6b:77:a3:99:91
Slave queue ID: 0
Aggregator ID: 2
Actor Churn State: monitoring
Partner Churn State: monitoring
Actor Churned Count: 0
Partner Churned Count: 0
```

b. Linux Network Bonding Modes

Bonding modes determine how network traffic is distributed across the member interfaces and how failover is handled. Linux supports various bonding modes, each with specific characteristics:

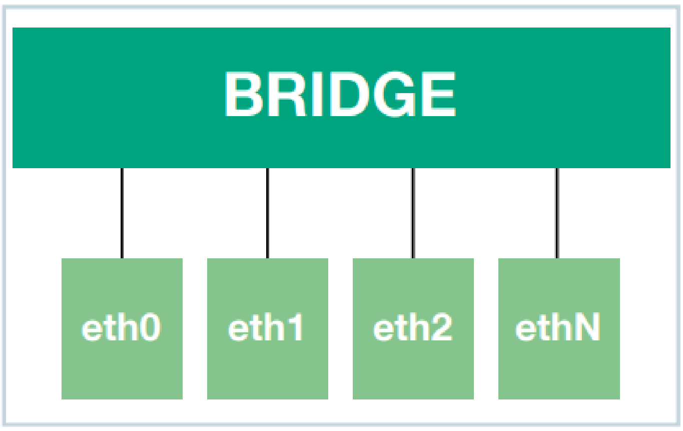
- **balance-rr:** The default mode is round-robin bonding, providing load balancing and fault tolerance.
- **active-backup:** Provides fault tolerance, allowing only one slave to be active at a time. If the active slave fails, another takes over.
- **balance-xor:** Provides fault tolerance and load balancing by transmitting based on hash.
- **broadcast:** Provides fault tolerance by transmitting everything on all slave interfaces.
- **802.3ad:** Follows the IEEE 802.3ad standard for dynamic link aggregation. It creates aggregation groups for links with the same speed and duplex to provide fault tolerance and load balancing. 802.3ad uses LACP to communicate with the other side of the bond.
- **balance-tlb:** Adaptive transmit load balancing that doesn't require any special switch support.
- **balance-alb:** Adaptive load balancing that doesn't require any special switch support due to its use of ARP negotiation. [5]

The most common bonding modes are **active-backup** and **802.3ad**.

Understanding Linux Internetworking

Layer 2 Internetworking on Linux Systems

a. Create a bridge



Linux bridge configuration. [5]

```
# Alreade create interfaces in last section
# Create the bridge

# Create the bridge interface (br0)
$ sudo ip link add name br0 type bridge

# Add eth0, eth1, and eth2 to the bridge
$ sudo ip link set eth0 master br0
$ sudo ip link set eth1 master br0
$ sudo ip link set eth2 master br0

# Activate the bridge
$ sudo ip link set dev br0 up
```

This will create a bridge interface named ‘br0‘ and add the three physical interfaces (‘eth0‘, ‘eth1‘, and ‘eth2‘) to it.

b. MAC Address Table for the Bridge

```
 # Show MAC address table:
 $ sudo bridge fdb show
```

The following is the MAC address table for the network:

```
33:33:ff:91:45:75 dev eth10 self permanent
01:00:5e:00:00:01 dev eth10 self permanent
33:33:00:00:00:fb dev eth10 self permanent
01:00:5e:00:00:fb dev eth10 self permanent
33:33:00:00:00:01 dev eth10 self permanent
92:d3:31:75:1f:17 dev eth0 vlan 1 master br0 permanent
92:d3:31:75:1f:17 dev eth0 master br0 permanent
33:33:00:00:00:01 dev eth0 self permanent
4a:6b:77:a3:99:91 dev eth1 vlan 1 master br0 permanent
4a:6b:77:a3:99:91 dev eth1 master br0 permanent
33:33:00:00:00:01 dev eth1 self permanent
1e:7b:4b:09:ac:85 dev eth2 vlan 1 master br0 permanent
```

```
1e:7b:4b:09:ac:85 dev eth2 master br0 permanent
33:33:00:00:00:01 dev eth2 self permanent
33:33:00:00:00:01 dev br0 self permanent
01:00:5e:00:00:6a dev br0 self permanent
33:33:00:00:00:6a dev br0 self permanent
01:00:5e:00:00:01 dev br0 self permanent
a2:97:4c:80:5c:4a dev br0 vlan 1 master br0 permanent
a2:97:4c:80:5c:4a dev br0 master br0 permanent
```

Analysis of the Table

The MAC address table provides information about which ports can reach specific MAC addresses in the bridge. Here are some key points from the table:

- The table includes entries for different MAC addresses associated with various network interfaces (e.g., eth0, eth1, eth2, br0).
- Entries indicate the relationship between VLANs (virtual LANs) and the bridge (br0).
- The "permanent" status indicates that these entries are manually configured and will persist even after a reboot.
- The "self" designation implies that the MAC address is associated with the device itself.

c. Preventing Loops in Bridges

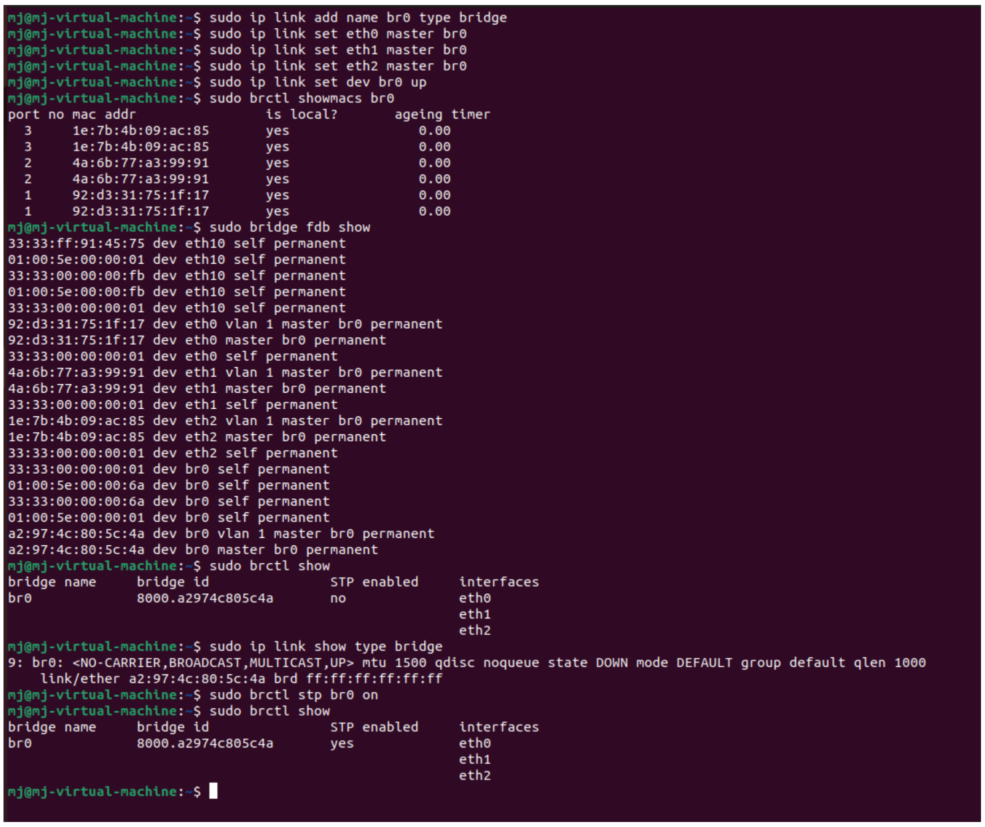
To prevent loops in bridges, you can use the Spanning Tree Protocol (STP). STP is a protocol that prevents loops in network topologies by blocking redundant paths. It's designed to identify and block any redundant paths to avoid loops.

```
# To enable STP on a bridge:
$ sudo brctl stp br0 on
```

This command enables STP on the bridge ‘br0‘. STP will then work to identify and block any redundant paths in the network.

Beneficial Fields in IP Packets for Loop Prevention

The Time-To-Live (TTL) field in the IP header is crucial for loop prevention. The TTL field is a counter that is decremented by each router that processes the packet. If the TTL reaches zero, the packet is discarded. In the presence of a loop, the TTL would be decremented multiple times, and the packet would be dropped before causing network congestion or a broadcast storm.



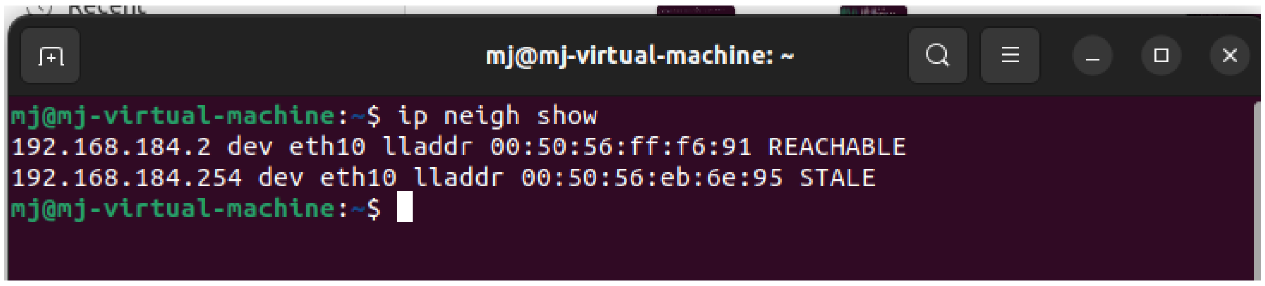
```
mj@mj-virtual-machine:~$ sudo ip link add name br0 type bridge
mj@mj-virtual-machine:~$ sudo ip link set eth0 master br0
mj@mj-virtual-machine:~$ sudo ip link set eth1 master br0
mj@mj-virtual-machine:~$ sudo ip link set eth2 master br0
mj@mj-virtual-machine:~$ sudo ip link set dev br0 up
mj@mj-virtual-machine:~$ sudo brctl showmacs br0
port no mac addr                is local?       ageing timer
  3     1e:7b:4b:09:ac:85       yes                0.00
  3     1e:7b:4b:09:ac:85       yes                0.00
  2     4a:6b:77:a3:99:91       yes                0.00
  2     4a:6b:77:a3:99:91       yes                0.00
  1     92:d3:31:75:1f:17       yes                0.00
  1     92:d3:31:75:1f:17       yes                0.00
mj@mj-virtual-machine:~$ sudo bridge fdb show
33:33:ff:91:45:75 dev eth10 self permanent
01:00:5e:00:00:01 dev eth10 self permanent
33:33:00:00:00:fb dev eth10 self permanent
01:00:5e:00:00:fb dev eth10 self permanent
33:33:00:00:00:01 dev eth10 self permanent
92:d3:31:75:1f:17 dev eth0 vlan 1 master br0 permanent
92:d3:31:75:1f:17 dev eth0 master br0 permanent
33:33:00:00:00:01 dev eth0 self permanent
4a:6b:77:a3:99:91 dev eth1 vlan 1 master br0 permanent
4a:6b:77:a3:99:91 dev eth1 master br0 permanent
33:33:00:00:00:01 dev eth1 self permanent
1e:7b:4b:09:ac:85 dev eth2 vlan 1 master br0 permanent
1e:7b:4b:09:ac:85 dev eth2 master br0 permanent
33:33:00:00:00:01 dev eth2 self permanent
33:33:00:00:00:01 dev br0 self permanent
01:00:5e:00:00:6a dev br0 self permanent
33:33:00:00:00:6a dev br0 self permanent
01:00:5e:00:00:01 dev br0 self permanent
a2:97:4c:80:5c:4a dev br0 vlan 1 master br0 permanent
a2:97:4c:80:5c:4a dev br0 master br0 permanent
mj@mj-virtual-machine:~$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.a2974c805c4a       no              eth0
                                                        eth1
                                                        eth2
mj@mj-virtual-machine:~$ sudo ip link show type bridge
9: br0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default qlen 1000
    link/ether a2:97:4c:80:5c:4a brd ff:ff:ff:ff:ff:ff
mj@mj-virtual-machine:~$ sudo brctl stp br0 on
mj@mj-virtual-machine:~$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.a2974c805c4a       yes             eth0
                                                        eth1
                                                        eth2
mj@mj-virtual-machine:~$ 
```

Layer 3 Internetworking View on Linux Systems

Neighbor Table

```
# Show Neighbor table
$ ip neigh show
```

The following is the neighbor table in the system:



```
mj@mj-virtual-machine:~$ ip neigh show
192.168.184.2 dev eth10 lladdr 00:50:56:ff:f6:91 REACHABLE
192.168.184.254 dev eth10 lladdr 00:50:56:eb:6e:95 STALE
mj@mj-virtual-machine:~$
```

Explanation of Fields in a Row

Each row in the neighbor table provides information about a specific neighbor. Here's an explanation of the fields:

- **192.168.184.2:** The IP address of the neighbor.
- **dev eth10:** The network interface through which the neighbor is reachable.
- **lladdr 00:50:56:ff:f6:91:** The Link Layer (MAC) address of the neighbor.
- **REACHABLE/STALE:** The state of the neighbor entry.
  - **REACHABLE:** The neighbor is currently reachable.
  - **STALE:** The neighbor entry is stale, meaning it might be outdated.

IP Routing

a. Show the Routing Table

```
# Show routing table
$ ip route show
```

The following is the routing table in the system:

```
default via 192.168.184.2 dev eth10 proto dhcp src 192.168.184.133 metric 100
192.168.184.0/24 dev eth10 proto kernel scope link src 192.168.184.133 metric 100
```

Explanation of Fields in a Row:
Each row in the routing table provides information about a specific route. Here's an explanation of the fields:

- **default:** This route is the default route, used when no more specific route is found for a destination. It serves as the gateway of last resort.
- **via 192.168.184.2 dev eth10:** Traffic to destinations not covered by more specific routes will be sent via the gateway with IP address 192.168.184.2 through the network interface eth10.
- **proto dhcp:** The protocol used to obtain this route. In this case, it's obtained through DHCP.
- **src 192.168.184.133:** The source IP address associated with this route.
- **metric 100:** The metric or cost associated with the route. Lower values indicate higher priority.
- **192.168.184.0/24 dev eth10 proto kernel scope link src 192.168.184.133 metric 100:** This row represents a route to the local network (192.168.184.0/24) directly connected to the eth10 interface.

b. Role of the Default Route

The **default route** serves as the gateway of last resort. When a system needs to send a packet to a destination for which it doesn't have a more specific route, it uses the default route.

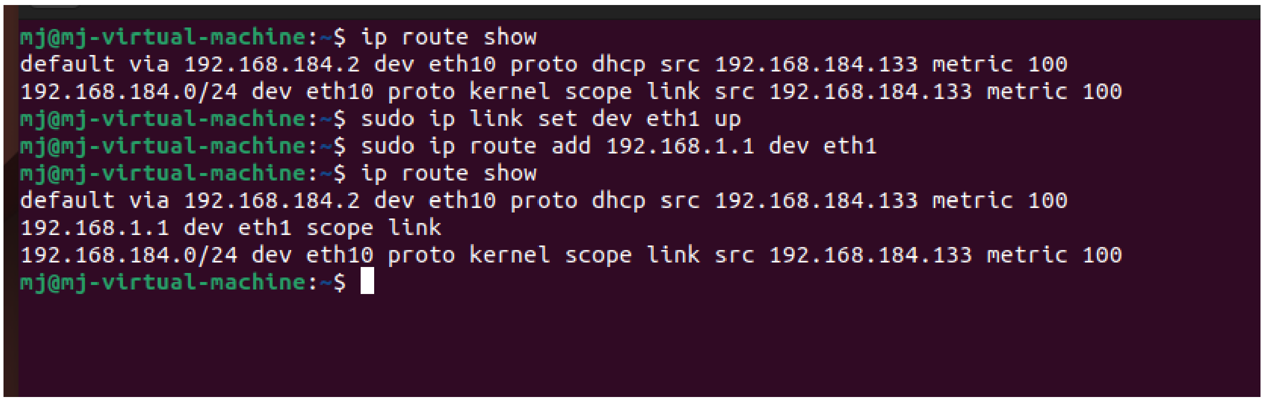
c. Creating a Static Route and Making it Persistent

```
# Create a static route
$ sudo ip link set dev eth1 up
$ sudo ip route add 192.168.1.1 dev eth1
```

This adds a static route saying that the network at 192.168.1.1 is reachable via the eth1 interface.
To make this route persistent after restarting the host, you can add the route information to a configuration file.

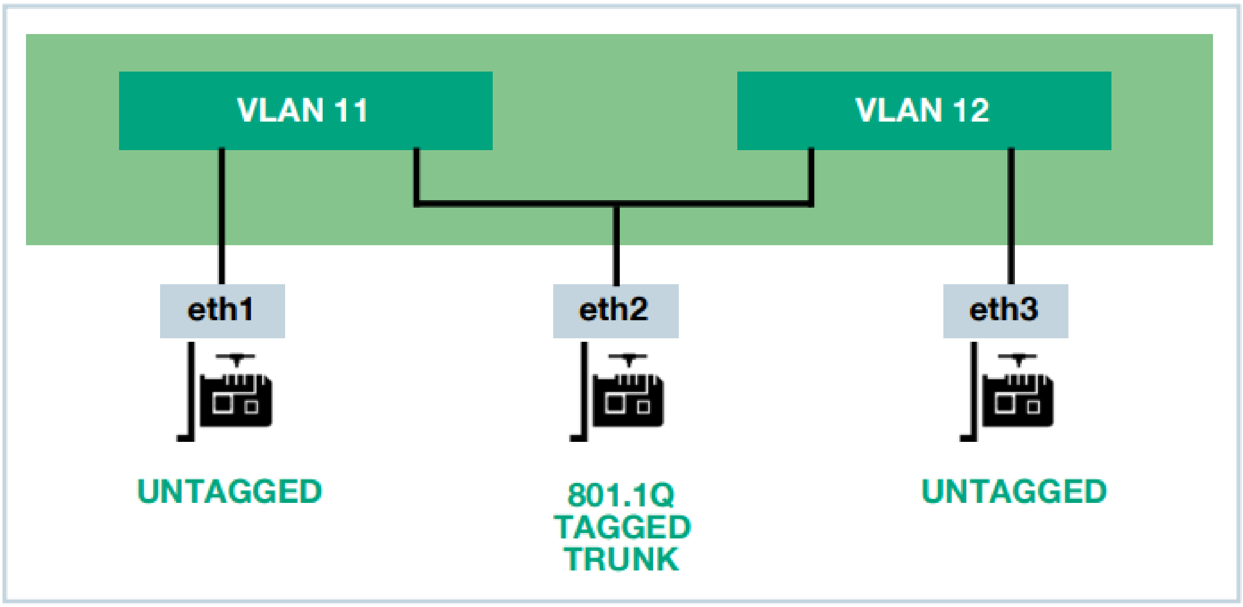
```
 # Write a script
 $ sudo nano /etc/network/if-up.d/custom-routes
 $ sudo chmod +x /etc/network/if-up.d/custom-routes
```

This script will be executed whenever a network interface is brought up.



```
mj@mj-virtual-machine:~$ ip route show
default via 192.168.184.2 dev eth10 proto dhcp src 192.168.184.133 metric 100
192.168.184.0/24 dev eth10 proto kernel scope link src 192.168.184.133 metric 100
mj@mj-virtual-machine:~$ sudo ip link set dev eth1 up
mj@mj-virtual-machine:~$ sudo ip route add 192.168.1.1 dev eth1
mj@mj-virtual-machine:~$ ip route show
default via 192.168.184.2 dev eth10 proto dhcp src 192.168.184.133 metric 100
192.168.1.1 dev eth1 scope link
192.168.184.0/24 dev eth10 proto kernel scope link src 192.168.184.133 metric 100
mj@mj-virtual-machine:~$
```

Virtual LANs (VLANs)



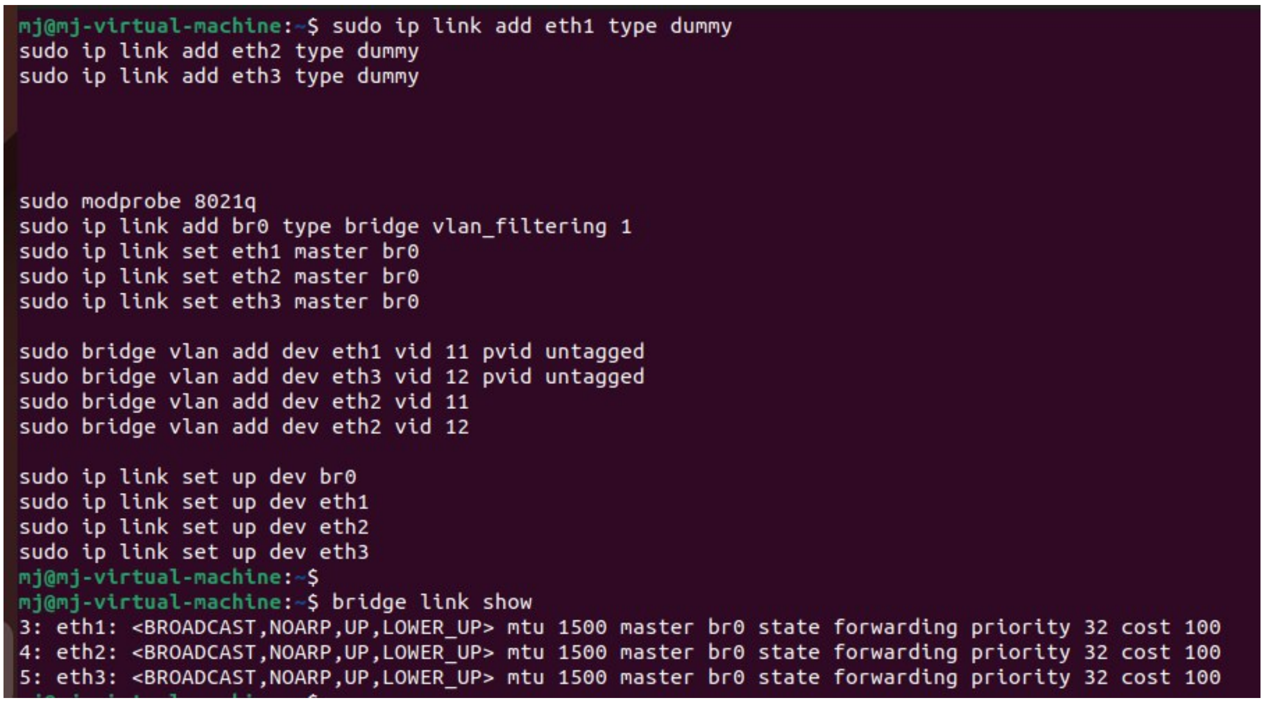
Tagged and untagged VLAN traffic. [5]

a. Creating VLANs with Bridges

```
# Load the 8021q module
sudo modprobe 8021q

# Create a bridge named br0 with VLAN filtering enabled
sudo ip link add br0 type bridge vlan_filtering 1

# Attach eth1, eth2, and eth3 to the br0 bridge
sudo ip link set eth1 master br0
sudo ip link set eth2 master br0
sudo ip link set eth3 master br0

```

```
# Add VLAN 11 to eth1 with untagged traffic
sudo bridge vlan add dev eth1 vid 11 pvid untagged

# Add VLAN 12 to eth3 with untagged traffic
sudo bridge vlan add dev eth3 vid 12 pvid untagged

# Add VLAN 11 to eth2
sudo bridge vlan add dev eth2 vid 11

# Add VLAN 12 to eth2
sudo bridge vlan add dev eth2 vid 12

# Bring up the bridge and associated interfaces
sudo ip link set up dev br0
sudo ip link set up dev eth0
sudo ip link set up dev eth1
sudo ip link set up dev eth2
sudo ip link set up dev eth3
```

This setup places eth1 in VLAN11, eth3 in VLAN12, and eth2 in both VLANs [5]



```
mj@mj-virtual-machine:~$ sudo ip link add eth1 type dummy
sudo ip link add eth2 type dummy
sudo ip link add eth3 type dummy




sudo modprobe 8021q
sudo ip link add br0 type bridge vlan_filtering 1
sudo ip link set eth1 master br0
sudo ip link set eth2 master br0
sudo ip link set eth3 master br0

sudo bridge vlan add dev eth1 vid 11 pvid untagged
sudo bridge vlan add dev eth3 vid 12 pvid untagged
sudo bridge vlan add dev eth2 vid 11
sudo bridge vlan add dev eth2 vid 12

sudo ip link set up dev br0
sudo ip link set up dev eth1
sudo ip link set up dev eth2
sudo ip link set up dev eth3
mj@mj-virtual-machine:~$
mj@mj-virtual-machine:~$ bridge link show
3: eth1: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 master br0 state forwarding priority 32 cost 100
4: eth2: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 master br0 state forwarding priority 32 cost 100
5: eth3: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 master br0 state forwarding priority 32 cost 100
```

b. Trunks

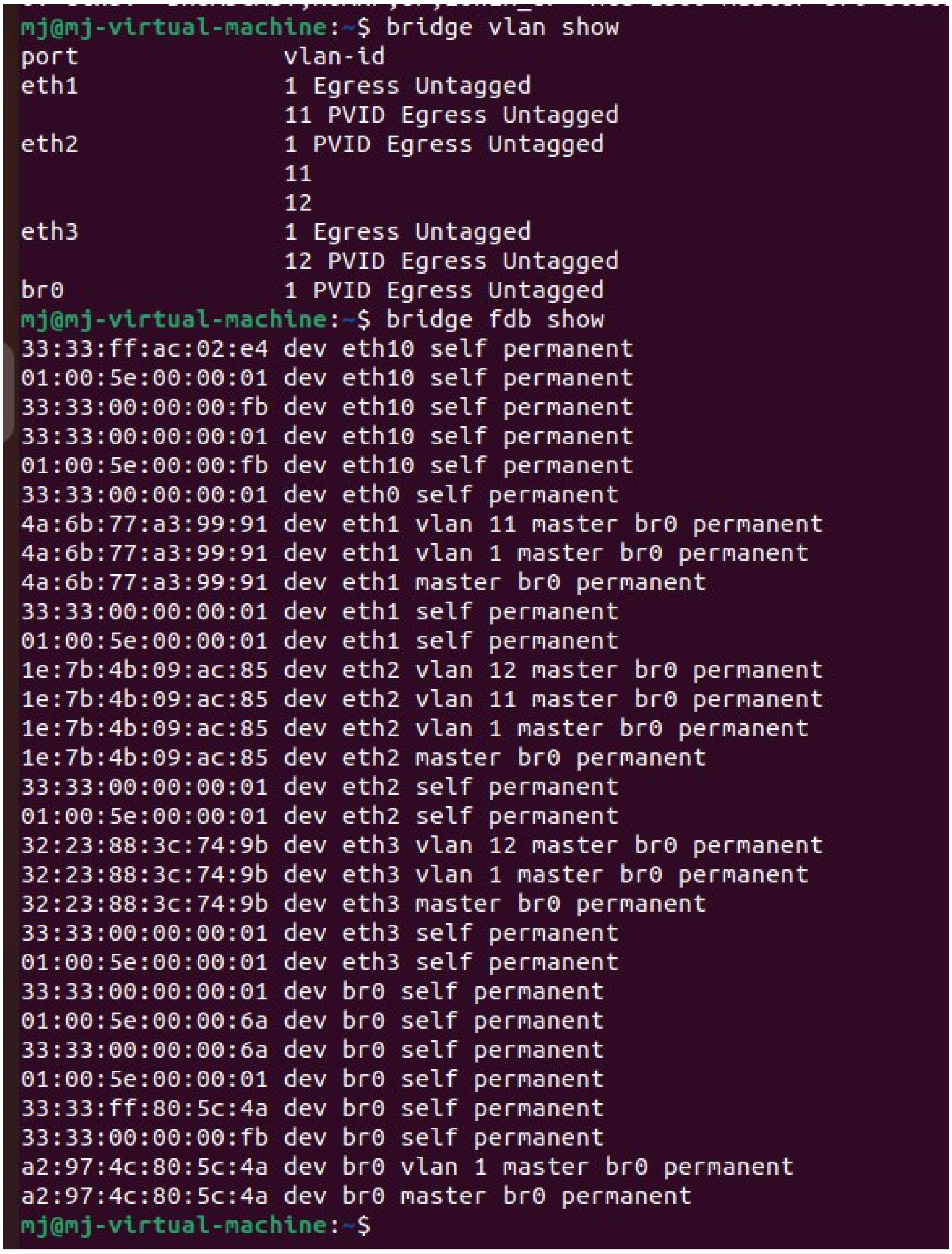
In networking, a trunk is a communication path between two network devices where multiple VLANs can travel. Trunks are often used to carry traffic for multiple VLANs over a single physical link.

In the given scenario, eth2 is configured as a tagged trunk because it carries traffic for both VLAN11 and VLAN12. The VLAN tagging allows the switch or other network devices to distinguish between the VLANs.

c. MAC Address Tables in 3 VLANs

The MAC address tables for VLANs 11, 12, and the default/native VLAN are different. Each VLAN operates as a separate broadcast domain, and the MAC address table is specific to each VLAN. Devices in one VLAN are not aware of devices in other VLANs, and the switches maintain separate MAC address tables for each VLAN. So, the MAC address tables for VLAN11, VLAN12, and the default VLAN are contain the MAC addresses of devices within their respective VLANs, but not devices in other VLANs.

d. Screenshots



```
mj@mj-virtual-machine:~$ bridge vlan show
port                vlan-id
eth1                1 Egress Untagged
                    11 PVID Egress Untagged
eth2                1 PVID Egress Untagged
                    11
                    12
eth3                1 Egress Untagged
                    12 PVID Egress Untagged
br0                 1 PVID Egress Untagged
mj@mj-virtual-machine:~$ bridge fdb show
33:33:ff:ac:02:e4 dev eth10 self permanent
01:00:5e:00:00:01 dev eth10 self permanent
33:33:00:00:00:fb dev eth10 self permanent
33:33:00:00:00:01 dev eth10 self permanent
01:00:5e:00:00:fb dev eth10 self permanent
33:33:00:00:00:01 dev eth0 self permanent
4a:6b:77:a3:99:91 dev eth1 vlan 11 master br0 permanent
4a:6b:77:a3:99:91 dev eth1 vlan 1 master br0 permanent
4a:6b:77:a3:99:91 dev eth1 master br0 permanent
33:33:00:00:00:01 dev eth1 self permanent
01:00:5e:00:00:01 dev eth1 self permanent
1e:7b:4b:09:ac:85 dev eth2 vlan 12 master br0 permanent
1e:7b:4b:09:ac:85 dev eth2 vlan 11 master br0 permanent
1e:7b:4b:09:ac:85 dev eth2 vlan 1 master br0 permanent
1e:7b:4b:09:ac:85 dev eth2 master br0 permanent
33:33:00:00:00:01 dev eth2 self permanent
01:00:5e:00:00:01 dev eth2 self permanent
32:23:88:3c:74:9b dev eth3 vlan 12 master br0 permanent
32:23:88:3c:74:9b dev eth3 vlan 1 master br0 permanent
32:23:88:3c:74:9b dev eth3 master br0 permanent
33:33:00:00:00:01 dev eth3 self permanent
01:00:5e:00:00:01 dev eth3 self permanent
33:33:00:00:00:01 dev br0 self permanent
01:00:5e:00:00:6a dev br0 self permanent
33:33:00:00:00:6a dev br0 self permanent
01:00:5e:00:00:01 dev br0 self permanent
33:33:ff:80:5c:4a dev br0 self permanent
33:33:00:00:00:fb dev br0 self permanent
a2:97:4c:80:5c:4a dev br0 vlan 1 master br0 permanent
a2:97:4c:80:5c:4a dev br0 master br0 permanent
mj@mj-virtual-machine:~$
```

Namespaces in linux

a. Linux Namespaces

In Linux, namespaces are a feature that allows different processes to have isolated views of various system resources. They provide a way to partition system resources among different processes, creating a form of process-level virtualization. Here are some of the main namespaces in Linux and a brief explanation of each:

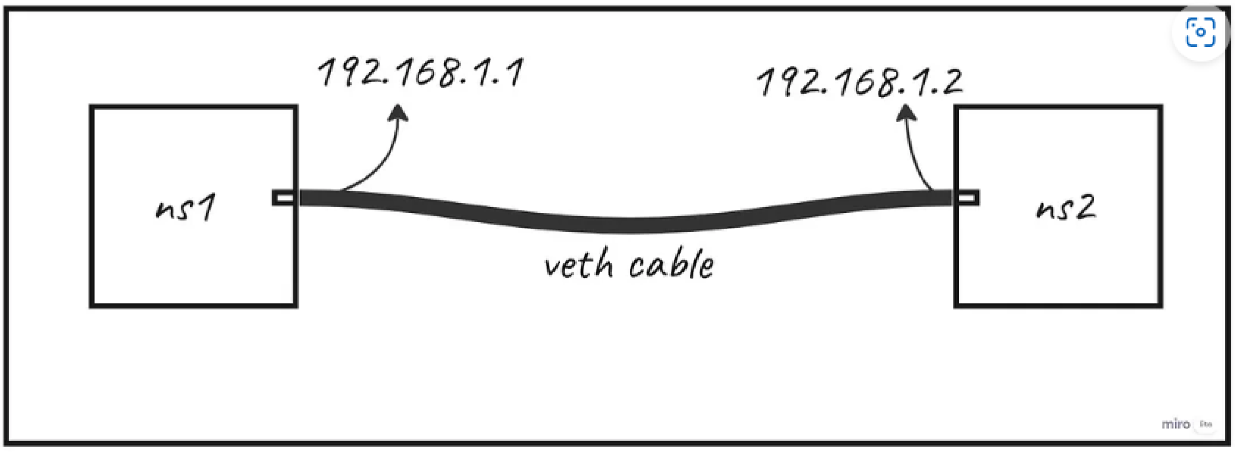
- **PID Namespace (pid):**
  - **Application:** Provides isolation for process IDs (PIDs).
  - **Explanation:** Processes in different PID namespaces have their own set of PIDs, meaning they may see different processes as having PID 1, for example. This can be useful for process isolation and management.
- **Network Namespace (net):**
  - **Application:** Isolates network-related resources.
  - **Explanation:** Processes in different network namespaces have their own network stack, interfaces, routing tables, and firewall rules. This is useful for creating network-isolated environments, similar to virtual machines.
- **Mount Namespace (mnt):**
  - **Application:** Isolates the filesystem mount points.
  - **Explanation:** Processes in different mount namespaces have their own view of the filesystem. Changes to mounts in one namespace do not affect others. This is commonly used in containerization to provide a private filesystem space for each container.
- **IPC Namespace (ipc):**
  - **Application:** Provides isolation for Inter-Process Communication (IPC) resources.
  - **Explanation:** Processes in different IPC namespaces have their own System V IPC objects (like message queues, semaphores, and shared memory segments). This allows isolation and independence between processes using these communication mechanisms.
- **UTS Namespace (uts):**
  - **Application:** Isolates the hostname and NIS domain name.
  - **Explanation:** Processes in different UTS namespaces have their own hostname and NIS domain name. This is useful for providing a distinct identity to processes running in separate namespaces.
- **User Namespace (user):**
  - **Application:** Provides isolation for user and group IDs.
  - **Explanation:** Processes in different user namespaces have their own view of user and group IDs. This is crucial for user namespace mapping, especially in the context of containerization, where it allows processes to have different UID/GID mappings inside and outside a container.
- **Cgroup Namespace (cgroup):**
  - **Application:** Provides isolation for control group (cgroup) hierarchies.
  - **Explanation:** Processes in different cgroup namespaces have their own view of the cgroup hierarchy. This is used to isolate resource management policies among different groups of processes.

These namespaces collectively contribute to the lightweight process isolation and containerization features in Linux. Tools like Docker and Kubernetes extensively leverage these namespaces to create isolated and portable environments for applications. Each namespace provides a way to separate and control specific aspects of a process's view of the system. [7]

Comparison between VLANs and Network Namespaces(additional information)

- **Layer of Operation:** VLANs operate at Layer 2, dealing with MAC addresses and broadcast domains, while network namespaces operate at Layer 3 and above, dealing with IP addresses and complete network stacks.
- **Scope of Isolation:** VLANs primarily isolate broadcast domains, allowing devices within the same VLAN to communicate. Network namespaces provide a more comprehensive isolation, separating entire network stacks.
- **Configuration:** VLANs are configured on network switches. In contrast, network namespaces are configured and managed by the operating system kernel.
- **Use Case:** VLANs are commonly used in large-scale enterprise networks to segment traffic and improve network efficiency. Network namespaces find application in containerization and virtualization for process isolation, creating independent network environments for applications or services. [8]

b. Create Two Network Namespaces



Two namespaces connected witha a veth cable. [8]

```
# Create network namespaces
$ sudo ip netns add ns1
$ sudo ip netns add ns2
```

c. Create and Configure Veth Cable

```
# Create a virtual Ethernet pair
$ sudo ip link add eth1 type veth peer name eth2
```

d. Connect Veth Cable with Each Namespace

```
# Move each end of the pair to its respective namespace
$ sudo ip link set eth1 netns ns1
$ sudo ip link set eth2 netns ns2
```

e. Bind IP Address with Interfaces

```
# Assign IP addresses to the interfaces
$ sudo ip netns exec ns1 ip addr add 192.168.1.1/24 dev eth1
$ sudo ip netns exec ns2 ip addr add 192.168.1.2/24 dev eth2
```

Now, attempt to ping one node from another:

```
# Ping
$ sudo ip netns exec ns1 ping 192.168.1.2
```

Observation: The nodes are not reachable because we haven't activated the interfaces yet.

f. Activate Interfaces

```
# Bring up the interfaces
$ sudo ip netns exec ns1 ip link set eth1 up
$ sudo ip netns exec ns2 ip link set eth2 up
```

Now, attempt to ping again:

```
# Ping
$ sudo ip netns exec ns1 ping 192.168.1.2
```

Observation: The nodes are reachable now.

g. Write Bash Script

Create a bash script named `my-script.sh` use nano:

```
  $ nano my_script.sh
```

```
  #!/bin/bash

  if [ "$#" -ne 2 ]; then
      echo "Usage: $0 <node1> <node2>"
      exit 1
  fi

  node1=$1
  node2=$2

  # Create network namespaces
  sudo ip netns add $node1
  sudo ip netns add $node2

  # Create a virtual Ethernet pair
  sudo ip link add eth1 type veth peer name eth2

  # Move each end of the pair to its respective namespace
  sudo ip link set eth1 netns $node1
  sudo ip link set eth2 netns $node2

  # Assign IP addresses to the interfaces
  sudo ip netns exec $node1 ip addr add 192.168.1.1/24 dev eth1
  sudo ip netns exec $node2 ip addr add 192.168.1.2/24 dev eth2

  # Bring up the interfaces
  sudo ip netns exec $node1 ip link set eth1 up
  sudo ip netns exec $node2 ip link set eth2 up

  # Ping from node1 to node2
  sudo ip netns exec $node1 ping 192.168.1.2
```

Make it executable:

```
chmod +x my_script.sh
```

Run the script:

```
./my_script.sh node1 node2
```

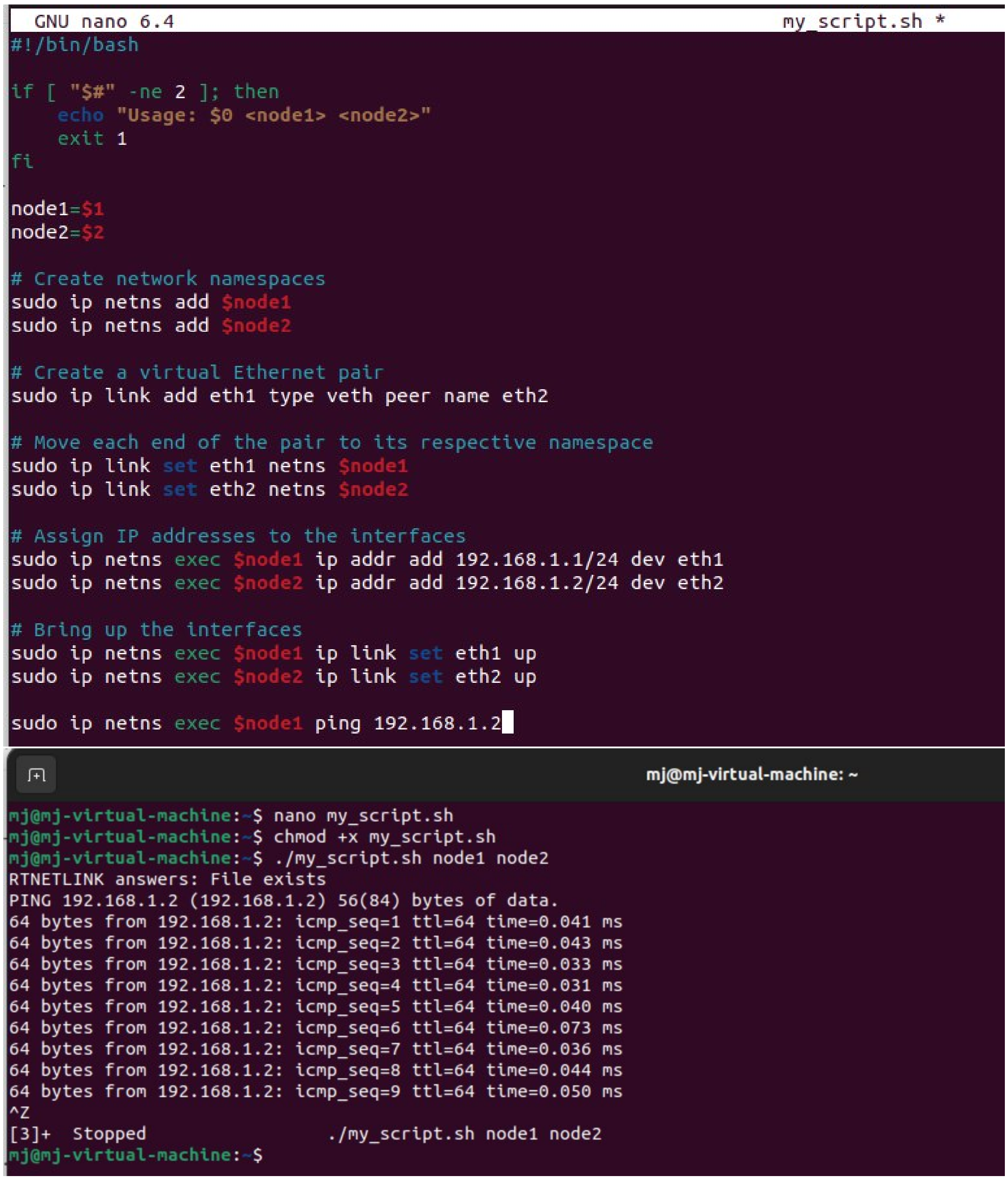
h. Ping 8.8.8.8 from Each Namespace

```
sudo ip netns exec ns1 ping 8.8.8.8
sudo ip netns exec ns2 ping 8.8.8.8
```

Observation: 8.8.8.8 may not be reachable. The issue is due to the lack of internet access in the isolated namespaces.

Solution for Internet Access in Namespaces

To enable internet access, you need to set up network address translation (NAT) and forwarding in the main network namespace. [8]



```
  GNU nano 6.4                                                      my_script.sh *
#!/bin/bash

if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <node1> <node2>"
    exit 1
fi

node1=$1
node2=$2

# Create network namespaces
sudo ip netns add $node1
sudo ip netns add $node2

# Create a virtual Ethernet pair
sudo ip link add eth1 type veth peer name eth2

# Move each end of the pair to its respective namespace
sudo ip link set eth1 netns $node1
sudo ip link set eth2 netns $node2

# Assign IP addresses to the interfaces
sudo ip netns exec $node1 ip addr add 192.168.1.1/24 dev eth1
sudo ip netns exec $node2 ip addr add 192.168.1.2/24 dev eth2

# Bring up the interfaces
sudo ip netns exec $node1 ip link set eth1 up
sudo ip netns exec $node2 ip link set eth2 up

sudo ip netns exec $node1 ping 192.168.1.2
```

```
mj@mj-virtual-machine: ~
mj@mj-virtual-machine:~$ nano my_script.sh
mj@mj-virtual-machine:~$ chmod +x my_script.sh
mj@mj-virtual-machine:~$ ./my_script.sh node1 node2
RTNETLINK answers: File exists
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.041 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.033 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.031 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.040 ms
64 bytes from 192.168.1.2: icmp_seq=6 ttl=64 time=0.073 ms
64 bytes from 192.168.1.2: icmp_seq=7 ttl=64 time=0.036 ms
64 bytes from 192.168.1.2: icmp_seq=8 ttl=64 time=0.044 ms
64 bytes from 192.168.1.2: icmp_seq=9 ttl=64 time=0.050 ms
^Z
[3]+  Stopped                 ./my_script.sh node1 node2
mj@mj-virtual-machine:~$
```

Acknowledgments

I extend my sincere appreciation to ChatGPT for its invaluable assistance in the writing process of this report. The ability to generate LaTeX-formatted content and provide insightful answers to questions has significantly contributed to the completion of this work. I express my gratitude for the support provided throughout this endeavor.

References

[1] DNSstuff. What is a hypervisor? [Online]. Available: https://www.dnsstuff.com/what-is-hypervisor

[2] Virtasant. Hypervisors: A comprehensive guide. [Online]. Available: https://www.virtasant.com/blog/hypervisors-a-comprehensive-guide

[3] S. Software. Virtualization. [Online]. Available: https://www.starwindsoftware.com/blog/virtualization

[4] TechTarget. Virtualization definition. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/virtualization

[5] D. M. Davis, *Linux Networking 101*, H. Kirchner, C. Guthrie, and M. Emery, Eds. Okatie Village Ste 103-157: ActualTech Media, 2017, layout and Design: Scott D. Lowe.

[6] baeldung. (2022) Traceroute: What are the three stars in the output? [Online]. Available: https://www.baeldung.com/linux/traceroute-three-stars

[7] Wikipedia. Linux namespaces. Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Linux_namespaces

[8] R. Masud. (2023) Create network namespace and ping between them through veth cable (virtual ether cable). Medium. [Online]. Available: https://medium.com/@reduanmasudcse/create-network-namespace-and-ping-between-them-through-veth-cable-virtual-ether-cable-143ba5da1da0