

# IP routing & troubleshooting

Lecture 5.2

Module 5. Linux Networking

Serhii Zakharchenko

# Agenda

---

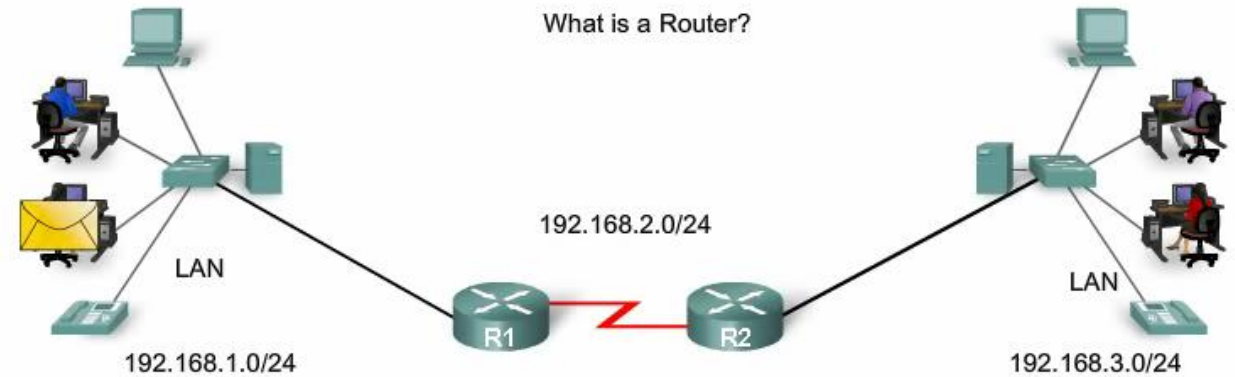
- IP routing configuration
- General troubleshooting procedures
- Linux monitoring and troubleshooting tools
- Q&A

---

# IP routing configuration

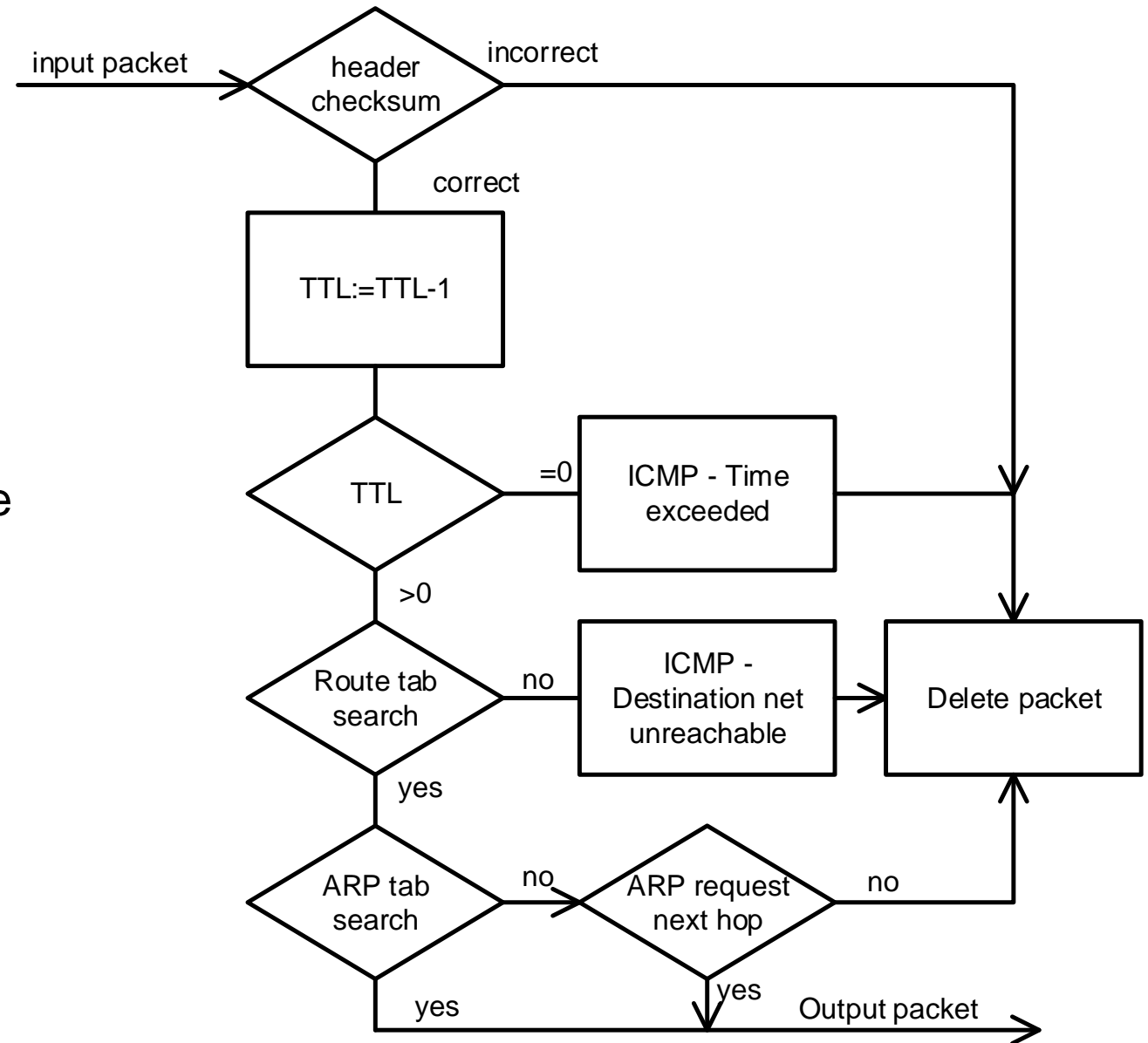
# Functions of a router

- When the router receives a packet, it examines the **destination address** of the packet and uses the **routing table** to search for the best path to that network.
- The routing table also includes the **interface** to be used to **forward packets** for each known network.
- When a match is found, the router **encapsulates** the packet into the data link frame of the outgoing or **exit interface**, and the packet is forwarded toward its destination.
- It is possible for a router to receive a packet that is encapsulated in one type of data link frame, and to forward the packet out of an interface that uses a different type of data link frame.



# Routing principles

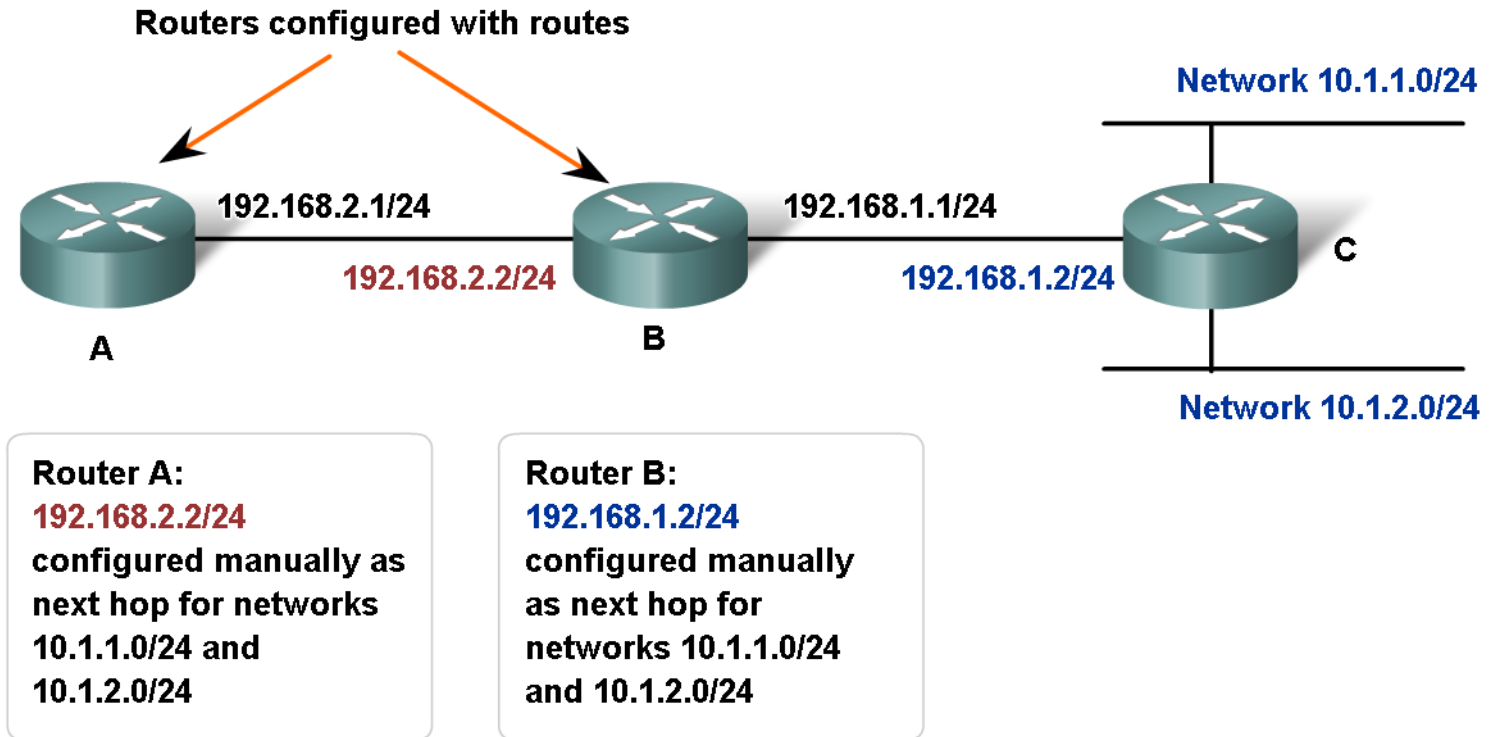
- **Principle 1:** "Every router makes its decision alone, based on the information it has in its own routing table."
- **Principle 2:** "The fact that one router has certain information in its routing table does not mean that other routers have the same information."
- **Principle 3:** "Routing information about a path from one network to another does not provide routing information about the reverse or return path."



# Static Routing

Static routing has three primary uses:

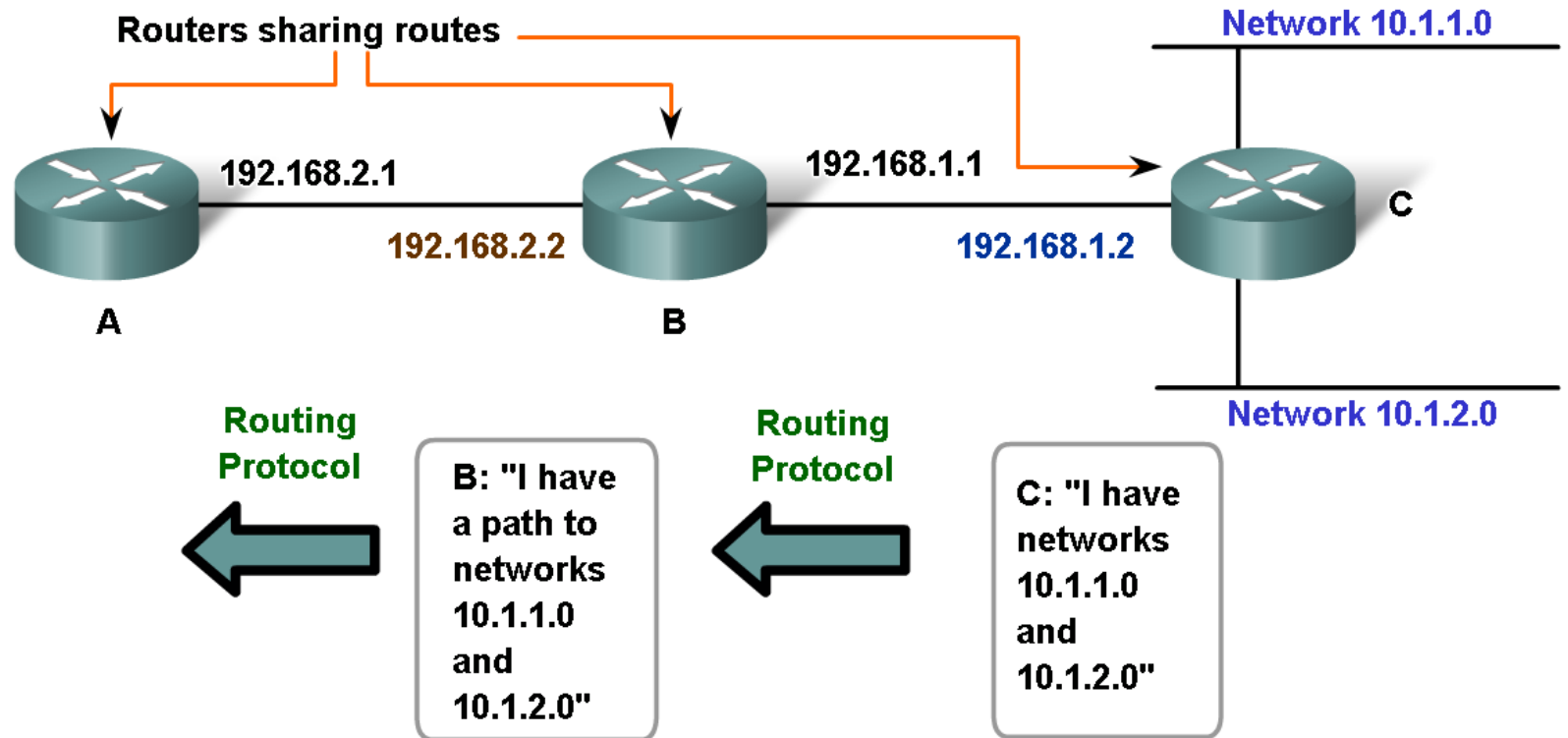
- Providing ease of routing table maintenance in **smaller networks** that are not expected to grow significantly.
- Routing to and from **stub networks**. A stub network is a network accessed by a single route, and the router has no other neighbors.
- Using a **single default route** to represent a path to any network that does not have a more specific match with another route in the routing table. Default routes are used to send traffic to any destination beyond the next upstream router.



# Dynamic Routing


Dynamic routing protocols functions:

- Exchange of routing information between routers.
- Automatic update of the routing table when changing the route.
- Determining the best path to the destination.



Router B learns about Router C's networks dynamically.  
Router B's next hop to 10.1.1.0 and 10.1.2.0 is **192.168.1.2** (Router C).  
Router A learns about Router C's networks dynamically from Router B.  
Router A's next hop to 10.1.1.0 and 10.1.2.0 is **192.168.2.2** (Router B).

# Logical Topology example

**N450 Wi-Fi роутер**  
Модель TL-WR940N

Состояние

Быстрая настройка

WPS

Рабочий режим

Сеть

Беспроводной режим

Гостевая сеть

DHCP

Переадресация

Безопасность

Родительский контроль

Яндекс.DNS

Контроль доступа

Настройки маршрутизации

Список статических маршрутов

ID	IP-адрес назначения	Маска подсети	Основной шлюз	Состояние
1	10.0.0.0	255.255.0.0	192.168.1.105	Включено

Добавить...

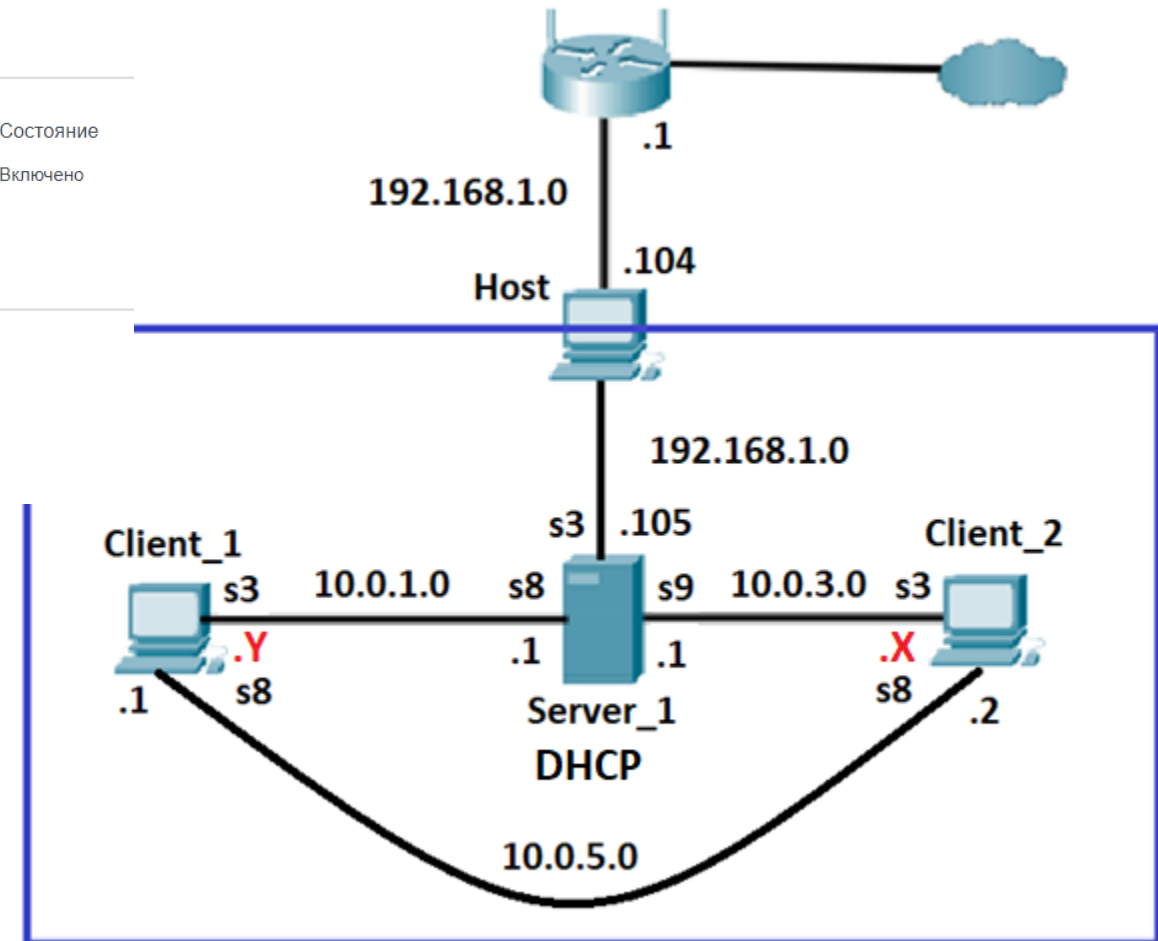
Включить все

Отключить все

Удалить все

Предыдущая

Следующая





# Linux Routing switch on

- Routing is switched on in Linux Server, but it is switched off in Linux Workstation.
- Switch on routing is needed only on **transit** devices.
- To check out routing enable use *sysctl net.ipv4.conf.all.forwarding* command
- To switch “on” or “off” routing you must edit /etc/sysctl.conf file
- To review routing table: *\$ip route show*

```
sergey@Server1:/etc/ssh$ ip route show
default via 192.168.1.1 dev enp0s3 proto static metric 100
10.0.1.0/24 dev enp0s8 proto kernel scope link src 10.0.1.1 metric 101
10.0.3.0/24 dev enp0s9 proto kernel scope link src 10.0.3.1 metric 102
169.254.0.0/16 dev enp0s3 scope link metric 1000
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.105 metric 100
```

```
sergey@Server1:~$ cat /proc/sys/net/ipv4/ip_forward
1
sergey@Server1:~$ sysctl net.ipv4.conf.all.forwarding
net.ipv4.conf.all.forwarding = 1
```

```
# Uncomment the next two lines to enable Spoof protection (reverse-
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
# Enabling this option disables Stateless Address Autoconfiguratio
# based on Router Advertisements for this host
net.ipv6.conf.all.forwarding=1

#####
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
```

# Routing Tables Review

```
sergey@Client1:~$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default _gateway 0.0.0.0 UG 100 0 0 enp0s3
10.0.1.0 0.0.0.0 255.255.255.0 U 100 0 0 enp0s3
10.0.5.0 0.0.0.0 255.255.255.0 U 101 0 0 enp0s8
link-local 0.0.0.0 255.255.0.0 U 1000 0 0 enp0s8
```

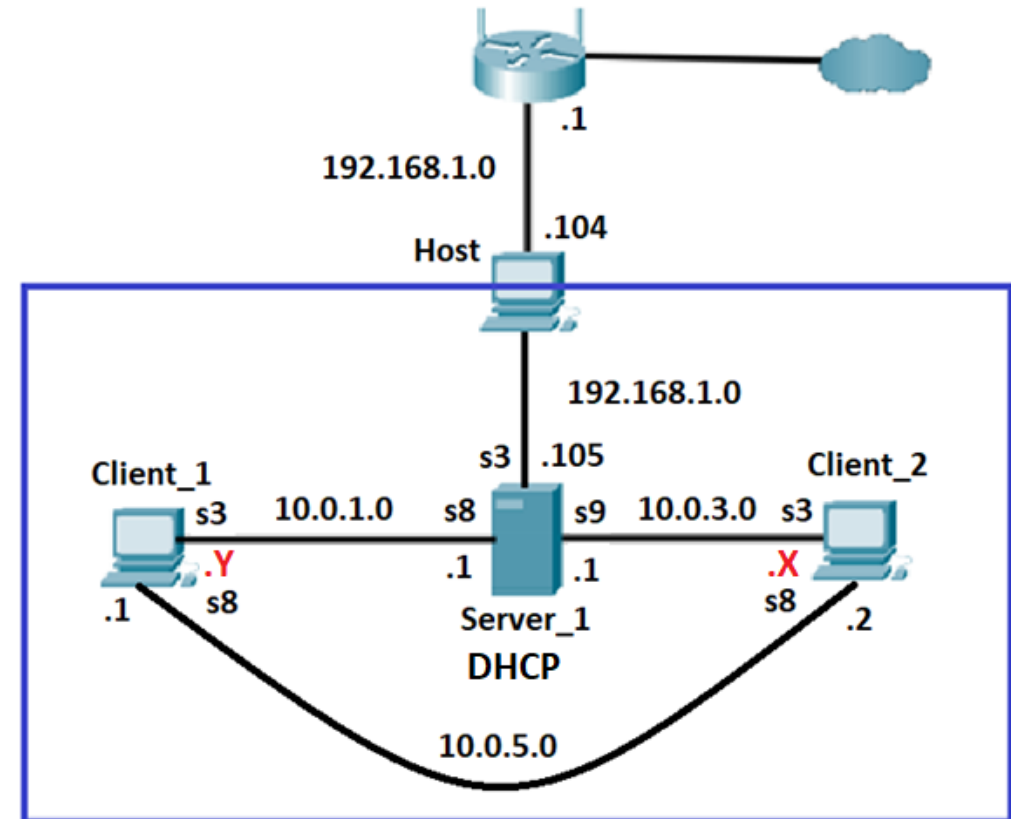
```
sergey@Server1:/etc/ssh$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default _gateway 0.0.0.0 UG 100 0 0 enp0s3
10.0.1.0 0.0.0.0 255.255.255.0 U 101 0 0 enp0s8
10.0.3.0 0.0.0.0 255.255.255.0 U 102 0 0 enp0s9
link-local 0.0.0.0 255.255.0.0 U 1000 0 0 enp0s3
192.168.1.0 0.0.0.0 255.255.255.0 U 100 0 0 enp0s3
```

```
sergey@Client2:~$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default _gateway 0.0.0.0 UG 20100 0 0 enp0s3
10.0.3.0 0.0.0.0 255.255.255.0 U 100 0 0 enp0s3
10.0.5.0 0.0.0.0 255.255.255.0 U 101 0 0 enp0s8
link-local 0.0.0.0 255.255.0.0 U 1000 0 0 enp0s8

sergey@Client2:~$ ip route sh
default via 10.0.3.1 dev enp0s3 proto dhcp metric 100
10.0.3.0/24 dev enp0s3 proto kernel scope link src 10.0.3.10 metric 100
10.0.5.0/24 dev enp0s8 proto kernel scope link src 10.0.5.2 metric 101
169.254.0.0/16 dev enp0s8 scope link metric 1000
```

*\$route*

*\$ip route sh*



# Connectivity Check

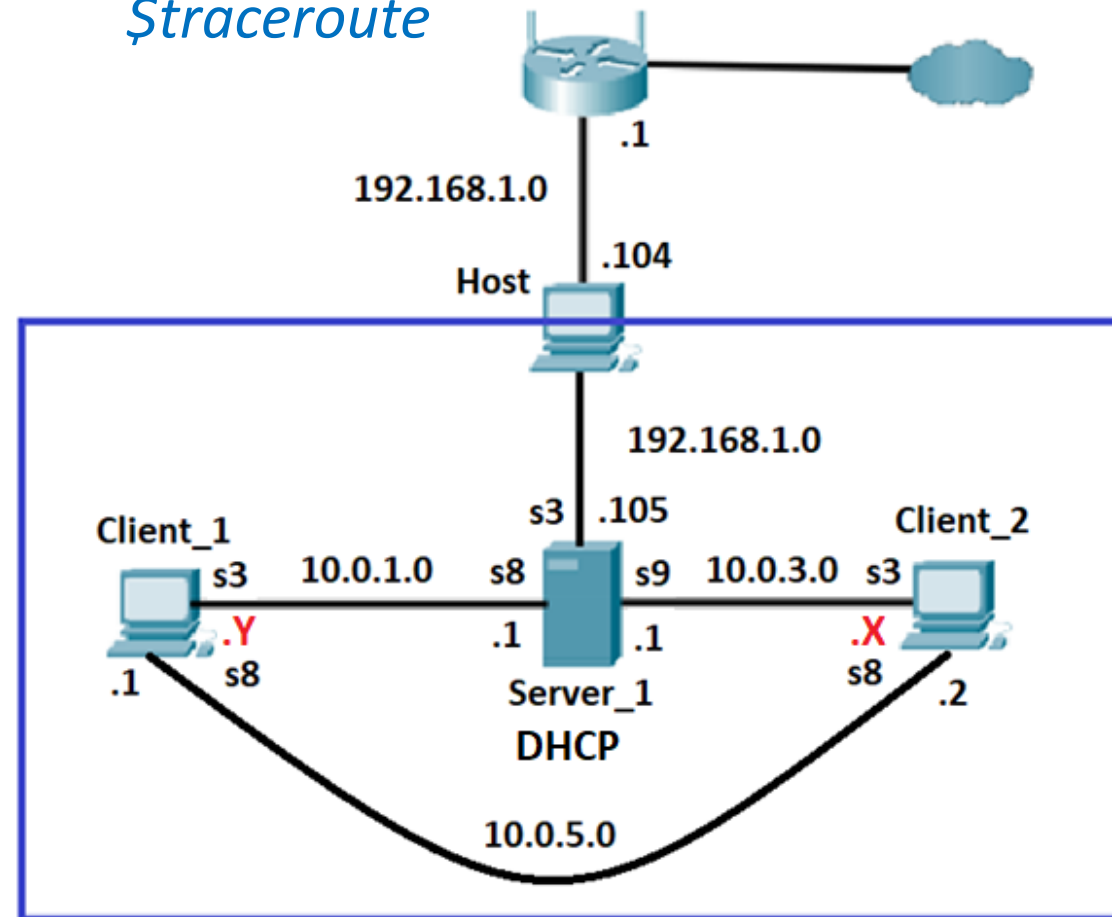
```
sergey@Client1:~$ ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_seq=1 ttl=63 time=1.28 ms
64 bytes from 10.0.3.10: icmp_seq=2 ttl=63 time=1.91 ms
64 bytes from 10.0.3.10: icmp_seq=3 ttl=63 time=1.17 ms
^C
--- 10.0.3.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.170/1.454/1.910/0.325 ms
sergey@Client1:~$ ping 10.0.5.2
PING 10.0.5.2 (10.0.5.2) 56(84) bytes of data.
64 bytes from 10.0.5.2: icmp_seq=1 ttl=64 time=0.929 ms
64 bytes from 10.0.5.2: icmp_seq=2 ttl=64 time=1.17 ms
64 bytes from 10.0.5.2: icmp_seq=3 ttl=64 time=1.46 ms
```

For more details see ping(8).

```
sergey@Client1:~$ traceroute 10.0.5.2
traceroute to 10.0.5.2 (10.0.5.2), 30 hops max, 60 byte packets
 1 10.0.5.2 (10.0.5.2) 0.596 ms 0.535 ms 0.510 ms
sergey@Client1:~$ traceroute 10.0.3.10
traceroute to 10.0.3.10 (10.0.3.10), 30 hops max, 60 byte packets
 1 _gateway (10.0.1.1) 0.454 ms 0.395 ms 0.373 ms
 2 10.0.3.10 (10.0.3.10) 0.687 ms 0.609 ms 0.584 ms
```

```
sergey@Server1:/etc/ssh$ ping 10.0.5.1
PING 10.0.5.1 (10.0.5.1) 56(84) bytes of data.
From 192.168.1.1 icmp_seq=2 Redirect Host(New nexthop: 105.1.168.192)
From 192.168.1.1 icmp_seq=3 Redirect Host(New nexthop: 105.1.168.192)
From 192.168.1.1 icmp_seq=4 Redirect Host(New nexthop: 105.1.168.192)
```

*\$ping*  
*\$tracert*



# Routing Configuration

---

- Temporary routing configuration
  - *route*
  - *ip route*
- Permanent routing configuration
  - config files editing
  - nmcli utility using
- Dynamic routing configuration
  - quagga

## *ip route* versus *route*

---

- The *ip route* suite is set to replace the net-tools suite (with *route* command) of network configuration tools. There are "synonym" commands that perform similar function in each.
- *route* is a fairly simple tool, perfect for creating static routes. It's still present in many distributions for compatibility.
- *ip route* is much more powerful, it has much more functionality, and can create more specialized rules.

# *route* command

---

- **route** command in Linux is used when you want to work with the IP/kernel routing table. It is mainly used to set up static routes to specific hosts or networks via an interface. It is used for showing or update the IP/kernel routing table.
- Many Linux distributions do not have route command pre-installed. To install it:

Debian/Ubuntu

*\$sudo apt-get install net-tools*

CentOS/RedHat

*\$sudo yum install net-tools*

Fedora OS

*\$sudo dnf install net-tools*



# Temporary route adding

*ip route add <network\_ip>/<cidr> via <gateway\_ip> [metric <metric>].*

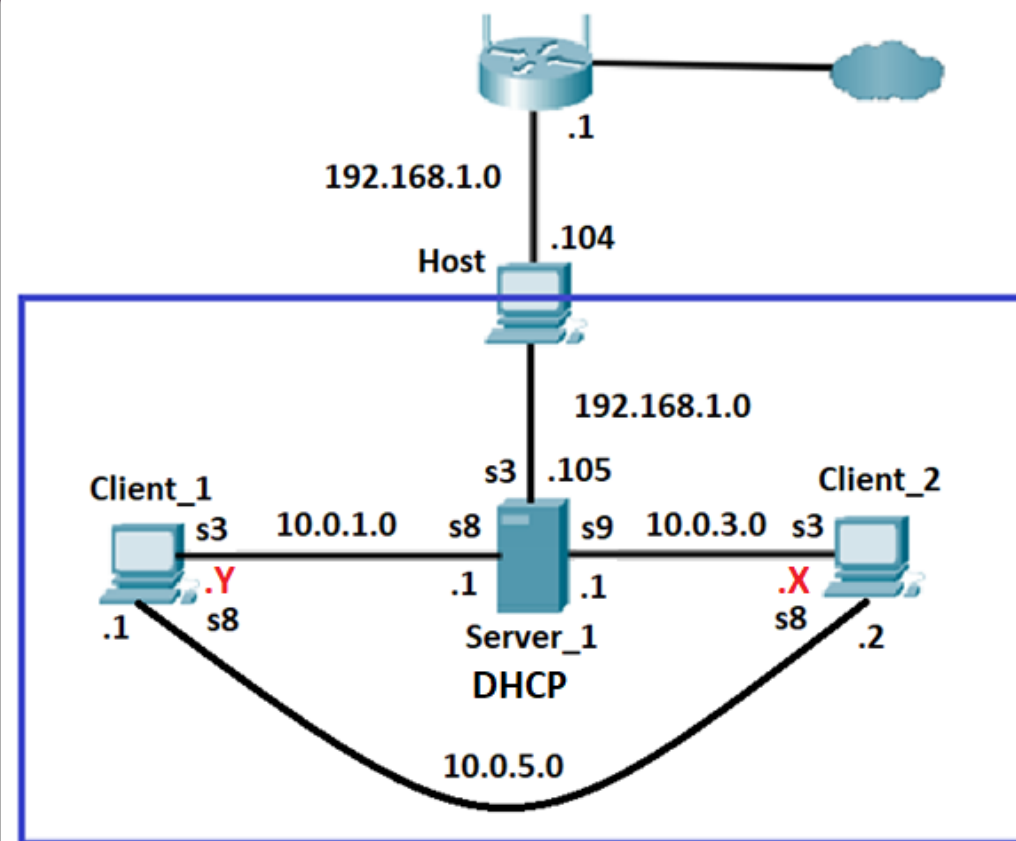
```
sergey@Client1:~$ route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
default          _gateway        0.0.0.0          UG    100    0      0 enp0s3
10.0.1.0         0.0.0.0         255.255.255.0    U     100    0      0 enp0s3
10.0.5.0         0.0.0.0         255.255.255.0    U     101    0      0 enp0s8
link-local      0.0.0.0         255.255.0.0      U     1000   0      0 enp0s8

sergey@Client1:~$ traceroute 10.0.3.12
traceroute to 10.0.3.12 (10.0.3.12), 30 hops max, 60 byte packets
 1  _gateway (10.0.1.1)  0.759 ms  0.720 ms  0.705 ms
 2  10.0.3.12 (10.0.3.12) 42.309 ms 42.426 ms 42.330 ms

sergey@Client1:~$ ip route add 10.0.3.0/24 via 10.0.5.2
RTNETLINK answers: Operation not permitted

sergey@Client1:~$ sudo ip route add 10.0.3.0/24 via 10.0.5.2
[sudo] password for sergey:
sergey@Client1:~$ route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
default          _gateway        0.0.0.0          UG    100    0      0 enp0s3
10.0.1.0         0.0.0.0         255.255.255.0    U     100    0      0 enp0s3
10.0.3.0         10.0.5.2        255.255.255.0    UG     0      0      0 enp0s8
10.0.5.0         0.0.0.0         255.255.255.0    U     101    0      0 enp0s8
link-local      0.0.0.0         255.255.0.0      U     1000   0      0 enp0s8

sergey@Client1:~$ traceroute 10.0.3.12
traceroute to 10.0.3.12 (10.0.3.12), 30 hops max, 60 byte packets
 1  10.0.3.12 (10.0.3.12) 1.013 ms 0.955 ms 0.944 ms
```



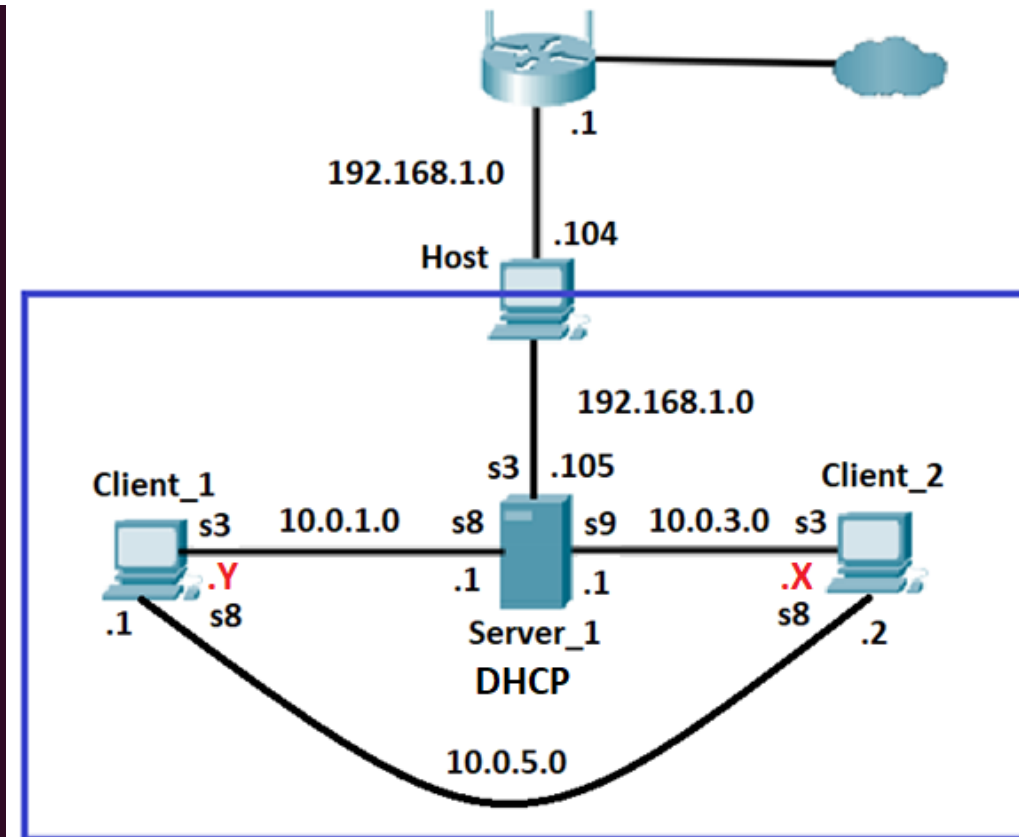
# Temporary route delating

*ip route del <network\_ip>/<cidr> via <gateway\_ip>*

```

sergey@Client1:~$ route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
default          _gateway        0.0.0.0          UG     100    0      0 enp0s3
10.0.1.0         0.0.0.0         255.255.255.0    U      100    0      0 enp0s3
10.0.3.0         10.0.5.2        255.255.255.0    UG     0      0      0 enp0s8
10.0.5.0         0.0.0.0         255.255.255.0    U      101    0      0 enp0s8
link-local       0.0.0.0         255.255.0.0      U      1000   0      0 enp0s8
sergey@Client1:~$ traceroute 10.0.3.12
traceroute to 10.0.3.12 (10.0.3.12), 30 hops max, 60 byte packets
 1  10.0.3.12 (10.0.3.12)  1.013 ms  0.955 ms  0.944 ms
sergey@Client1:~$ sudo ip route del 10.0.3.0/24 via 10.0.5.2
sergey@Client1:~$ route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
default          _gateway        0.0.0.0          UG     100    0      0 enp0s3
10.0.1.0         0.0.0.0         255.255.255.0    U      100    0      0 enp0s3
10.0.5.0         0.0.0.0         255.255.255.0    U      101    0      0 enp0s8
link-local       0.0.0.0         255.255.0.0      U      1000   0      0 enp0s8
sergey@Client1:~$ traceroute 10.0.3.12
traceroute to 10.0.3.12 (10.0.3.12), 30 hops max, 60 byte packets
 1  _gateway (10.0.1.1)  0.459 ms  0.406 ms  0.385 ms
 2  10.0.3.12 (10.0.3.12)  0.824 ms  0.741 ms  0.718 ms

```





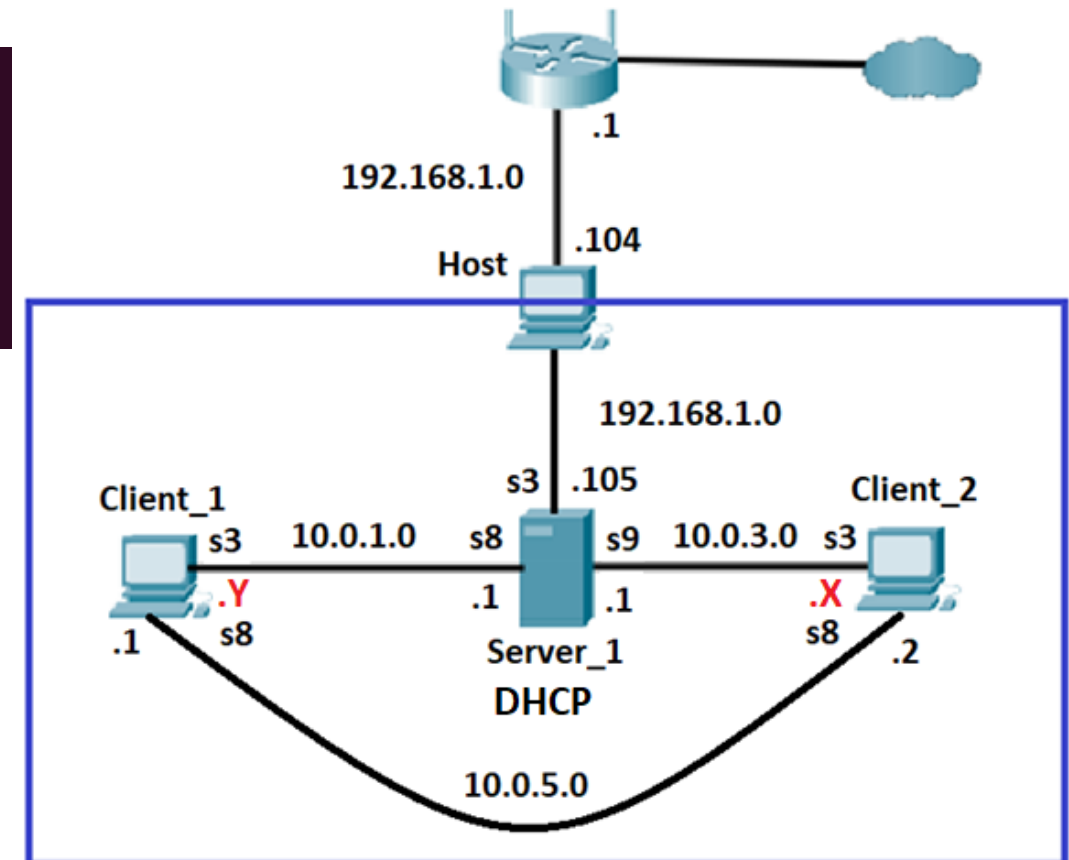
# Neighbors addresses resolution

## *ip neighbors*

```
sergey@Server1:~$ ip neigh
10.0.3.12 dev enp0s9 lladdr 08:00:27:bc:3b:5e STALE
10.0.1.12 dev enp0s8 lladdr 08:00:27:7c:97:87 STALE
192.168.0.1 dev enp0s3 lladdr b0:be:76:44:30:c4 REACHABLE
fe80::a00:27ff:fe7c:9787 dev enp0s8 lladdr 08:00:27:7c:97:87 STALE
fe80::a00:27ff:feb3b5e dev enp0s9 lladdr 08:00:27:bc:3b:5e STALE
sergey@Server1:~$
```

```
sergey@Client1:~$ ip neigh
10.0.5.2 dev enp0s8 lladdr 08:00:27:d0:cd:da STALE
10.0.1.1 dev enp0s3 lladdr 08:00:27:ec:6a:64 REACHABLE
sergey@Client1:~$
```

```
sergey@Client2:~$ ip neigh
10.0.5.1 dev enp0s8 lladdr 08:00:27:45:a0:da STALE
10.0.3.1 dev enp0s3 lladdr 08:00:27:1f:20:d2 REACHABLE
sergey@Client2:~$
```



# Priorities of routes

---

If the routing table contains two or more routes for a particular network, the following rules will be used for the final route selection:

- The “longest match” rule
- The “lowest metric” rule
- The “equal cost load balancing” rule

# The “longest match” rule

The most priority has the route with the biggest prefix

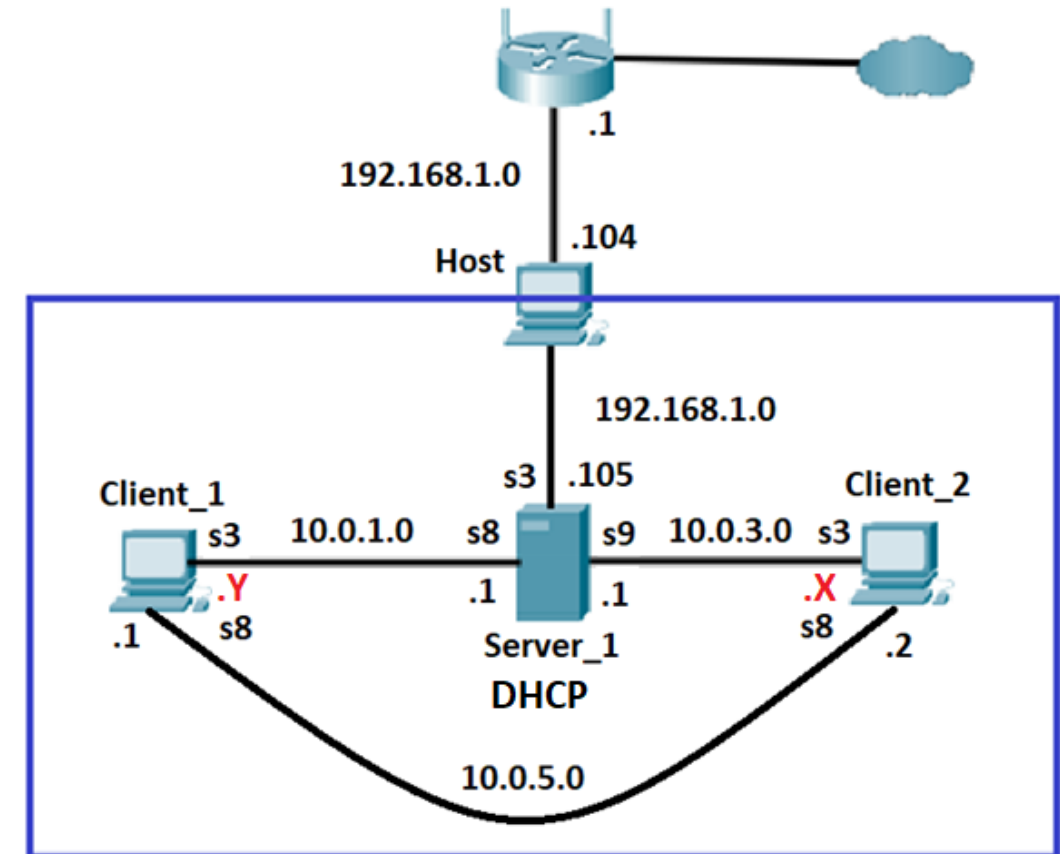
IP Packet Destination	172.16.0.10	10101100.00010000.00000000.00001010
Route 1	172.16.0.0/12	10101100.00010000.00000000.00000000
Route 2	172.16.0.0/18	10101100.00010000.00000000.00000000
Route 3	172.16.0.0/26	10101100.00010000.00000000.00000000

Longest Match to IP Packet Destination 

# The “longest match” and “lowest metric” rules

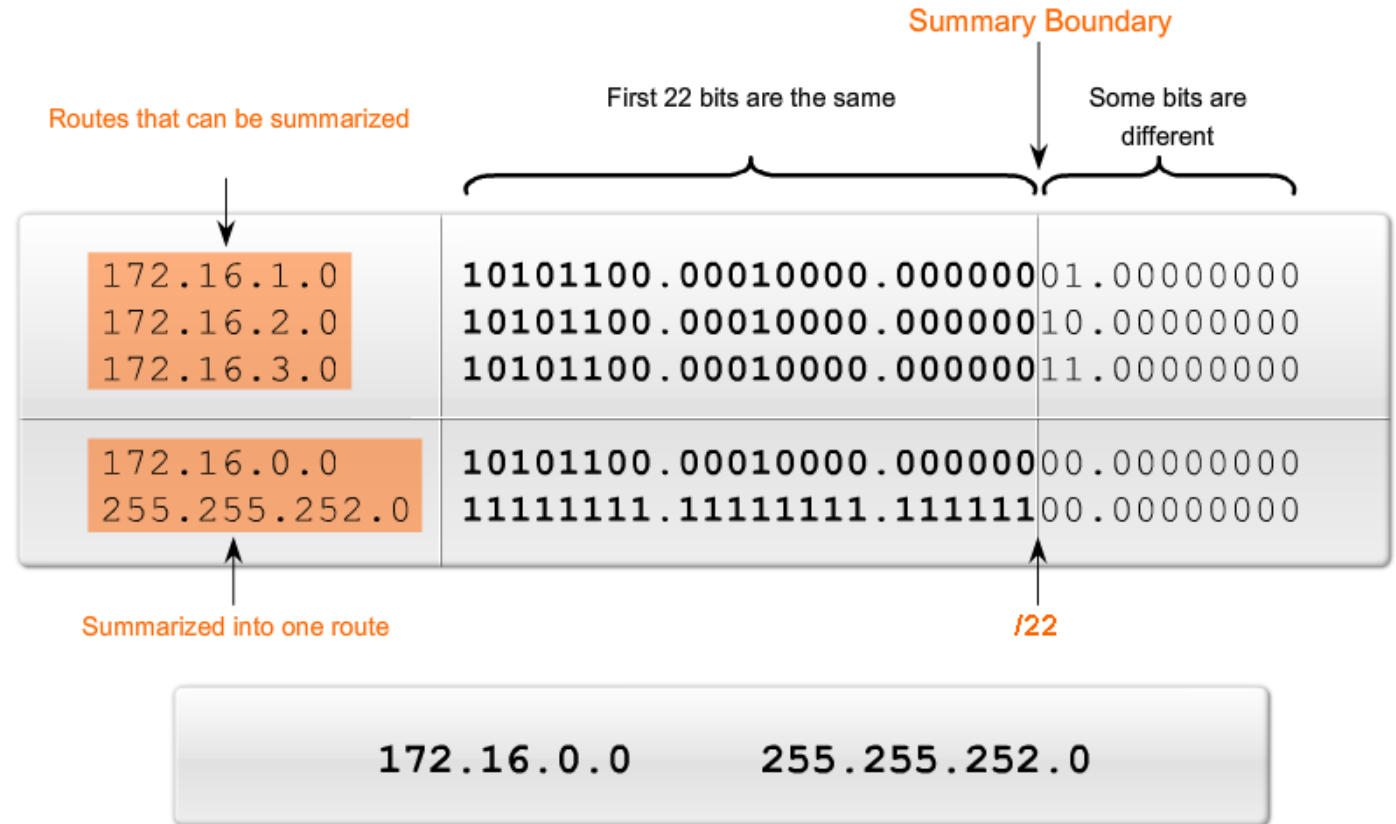
```
sergey@Client1:~$ sudo ip route add 10.0.3.0/25 via 10.0.1.1 metric 10
sergey@Client1:~$ route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
0.0.0.0          10.0.1.1         0.0.0.0          UG    20100  0      0 enp0s3
10.0.1.0         0.0.0.0          255.255.255.0    U     100    0      0 enp0s3
10.0.3.0         10.0.1.1         255.255.255.128 UG     10    0      0 enp0s3
10.0.3.0         10.0.5.2         255.255.255.0    UG     0      0      0 enp0s8
10.0.5.0         0.0.0.0          255.255.255.0    U     101    0      0 enp0s8
169.254.0.0      0.0.0.0          255.255.0.0      U     1000   0      0 enp0s8
sergey@Client1:~$ traceroute 10.0.3.12
traceroute to 10.0.3.12 (10.0.3.12), 30 hops max, 60 byte packets
 1  10.0.1.1 (10.0.1.1)  1.174 ms  1.161 ms  1.141 ms
 2  10.0.3.12 (10.0.3.12)  0.729 ms  0.697 ms  0.672 ms
```

```
sergey@Client1:~$ sudo ip route del 10.0.3.0/25 via 10.0.1.1 metric 10
sergey@Client1:~$ sudo ip route add 10.0.3.0/24 via 10.0.1.1 metric 10
sergey@Client1:~$ route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
0.0.0.0          10.0.1.1         0.0.0.0          UG    20100  0      0 enp0s3
10.0.1.0         0.0.0.0          255.255.255.0    U     100    0      0 enp0s3
10.0.3.0         10.0.5.2         255.255.255.0    UG     0      0      0 enp0s8
10.0.3.0         10.0.1.1         255.255.255.0    UG    10     0      0 enp0s3
10.0.5.0         0.0.0.0          255.255.255.0    U     101    0      0 enp0s8
169.254.0.0      0.0.0.0          255.255.0.0      U     1000   0      0 enp0s8
sergey@Client1:~$ traceroute 10.0.3.12
traceroute to 10.0.3.12 (10.0.3.12), 30 hops max, 60 byte packets
 1  10.0.3.12 (10.0.3.12)  0.521 ms  0.602 ms  0.584 ms
```



# Summary Static Routes

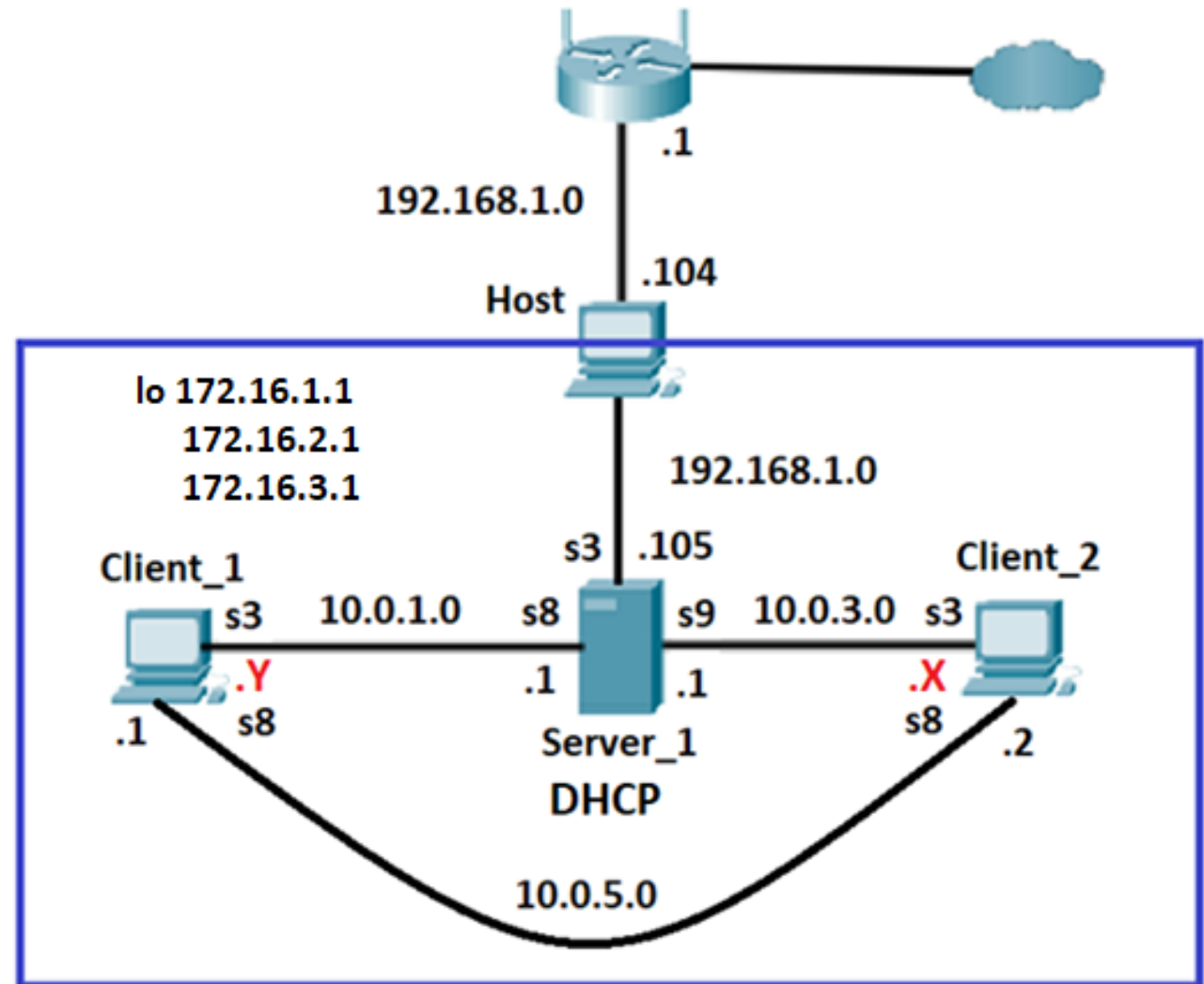
- Summarizing routes **reduces** the size of the routing table.
- Route summarization is the process of combining a number of static routes into a single static route.
- Multiple static routes can be summarized into a single static route if:
  - The destination networks can be summarized into a single network address
  - The multiple static routes all use the same exit-interface or next-hop IP address



# Summary Static Routes

```
sergey@Client1:~$ sudo ip addr add 172.16.2.1/24 dev lo
sergey@Client1:~$ sudo ip addr add 172.16.3.1/24 dev lo
sergey@Client1:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
    UP lt 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
    inet 172.16.1.1/24 scope global lo
        valid_lft forever preferred_lft forever
    inet 172.16.2.1/24 scope global lo
        valid_lft forever preferred_lft forever
    inet 172.16.3.1/24 scope global lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
```

```
sergey@Server1:~$ sudo ip route add 172.16.0.0/22 via 10.0.1.12
sergey@Server1:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
0.0.0.0         192.168.0.1    0.0.0.0         UG    100    0
10.0.1.0        0.0.0.0        255.255.255.0   U     101    0
10.0.3.0        0.0.0.0        255.255.255.0   U     102    0
169.254.0.0     0.0.0.0        255.255.0.0     U     1000   0
172.16.0.0      10.0.1.12      255.255.252.0   UG     0     0
192.168.0.0     0.0.0.0        255.255.255.0   U     100    0
sergey@Server1:~$ ping 172.16.2.1
PING 172.16.2.1 (172.16.2.1) 56(84) bytes of data.
64 bytes from 172.16.2.1: icmp_seq=1 ttl=64 time=0.396 ms
64 bytes from 172.16.2.1: icmp_seq=2 ttl=64 time=0.403 ms
64 bytes from 172.16.2.1: icmp_seq=3 ttl=64 time=0.589 ms
```

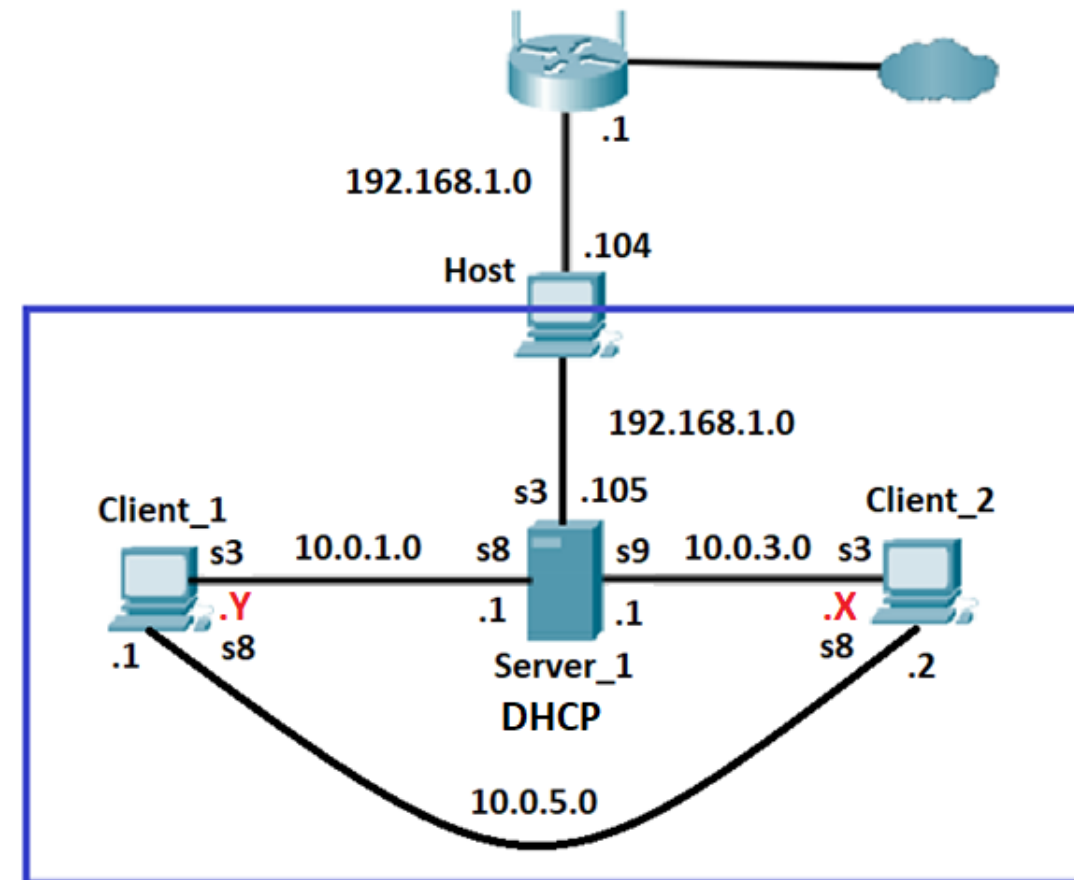


# Permanent routing configuration

To add a permanent route in Ubuntu use Netplan.

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    enp0s3:
#     addresses: [10.0.4.2/24]
#     dhcp4: true
#     routes:
#     - to: 10.0.3.0/24
#       via: 10.0.5.2
#       metric: 50
    enp0s8:
      addresses: [10.0.5.1/24]
      routes:
        - to: 10.0.3.0/24
          via: 10.0.5.2
          metric: 50
```

```
sergey@Client1:~$ route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
default          _gateway        0.0.0.0          UG    100    0      0 enp0s3
10.0.1.0         0.0.0.0         255.255.255.0    U     100    0      0 enp0s3
10.0.3.0         10.0.5.2        255.255.255.0    UG    50     0      0 enp0s8
10.0.5.0         0.0.0.0         255.255.255.0    U     101    0      0 enp0s8
link-local      0.0.0.0         255.255.0.0      U     1000   0      0 enp0s8
sergey@Client1:~$ traceroute 10.0.3.12
traceroute to 10.0.3.12 (10.0.3.12), 30 hops max, 60 byte packets
 1  10.0.3.12 (10.0.3.12)  2.259 ms  2.208 ms  2.202 ms
```



# Permanent routing configuration in another Linux distributives

Ununtu legacy (before 18.04 version): **edit** the “/etc/network/interfaces”:

*auto eth0*

*iface eth0 inet static*

*address 10.0.2.2*

*netmask 255.255.255.0*

*up route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.2.1*

On RHEL and CentOS distributions,  
you need to **create** a file named  
“route-<device>” in the  
“/etc/sysconfig/network-scripts” folder:

*\$ sudo vi /etc/sysconfig/network-scripts/  
route-enp0s3*

```
losboxes@osboxes network-scripts]$ route -n
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.0.4.2        0.0.0.0         UG    102    0      0 enp0s9
0.0.0.0          10.0.1.1        0.0.0.0         UG    103    0      0 enp0s3
10.0.1.0         0.0.0.0         255.255.255.0   U     103    0      0 enp0s3
10.0.3.0         10.0.1.1        255.255.255.0   UG    103    0      0 enp0s3
10.0.4.0         0.0.0.0         255.255.255.0   U     102    0      0 enp0s9
10.0.5.0         0.0.0.0         255.255.255.0   U     104    0      0 enp0s8
losboxes@osboxes network-scripts]$ cd /etc/sysconfig/network-scripts
losboxes@osboxes network-scripts]$ dir
ifcfg-enp0s3  ifcfg-enp0s8  ifcfg-test  ifcfg-test-1  route-enp0s3
losboxes@osboxes network-scripts]$ cat route-enp0s3
ADDRESS0=10.0.3.0
NETMASK0=255.255.255.0
GATEWAY0=10.0.1.1
losboxes@osboxes network-scripts]$
```



# Permanent routing configuration via *nmcli* in CentOS

---

To add the new route in routing table:

*nmcli connection modify <conn-name> ipv4.routes "<network ip-addr>/<prefix> <gateway>"*

*systemctl restart network*

Example:

*nmcli con mod test ipv4.routes "10.0.6.0/24 10.0.5.1"*

*systemctl restart network*

```
[osboxes@osboxes ~]$ cd /etc/sysconfig/network-scripts
[osboxes@osboxes network-scripts]$ dir
ifcfg-enp0s3  ifcfg-test  route-enp0s3
[osboxes@osboxes network-scripts]$ nmcli con mod test ipv4.routes "10.0.6.0/24 10.0.5.1"
[osboxes@osboxes network-scripts]$ dir
ifcfg-enp0s3  ifcfg-test  route-enp0s3  route-test
[osboxes@osboxes network-scripts]$ cat route-test
ADDRESS0=10.0.6.0
NETMASK0=255.255.255.0
GATEWAY0=10.0.5.1
[osboxes@osboxes network-scripts]$ _
```

# Dynamic routing in Linux

- [Quagga](#) is a routing software suite, providing implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4 for Unix platforms.
- The Quagga architecture consists of a core daemon, *zebra*, which acts as an abstraction layer to the underlying Unix kernel and presents the Zserv API over a Unix or TCP stream to Quagga clients.
- It is these Zserv clients which typically implement a routing protocol and communicate routing updates to the zebra daemon. Existing Zserv implementations are:

IPv4	IPv6	
zebra		- kernel interface, static routes, zserv server
ripd	ripngd	- RIPv1/RIPv2 for IPv4 and RIPng for IPv6
ospfd	ospf6d	- OSPFv2 and OSPFv3
bgpd		- BGPv4+ (including address family support for multicast and IPv6)
isisd		- IS-IS with support for IPv4 and IPv6

- Quagga daemons are each configurable via a network accessible CLI (called a 'vty'). The CLI follows a style similar to that of other routing software.

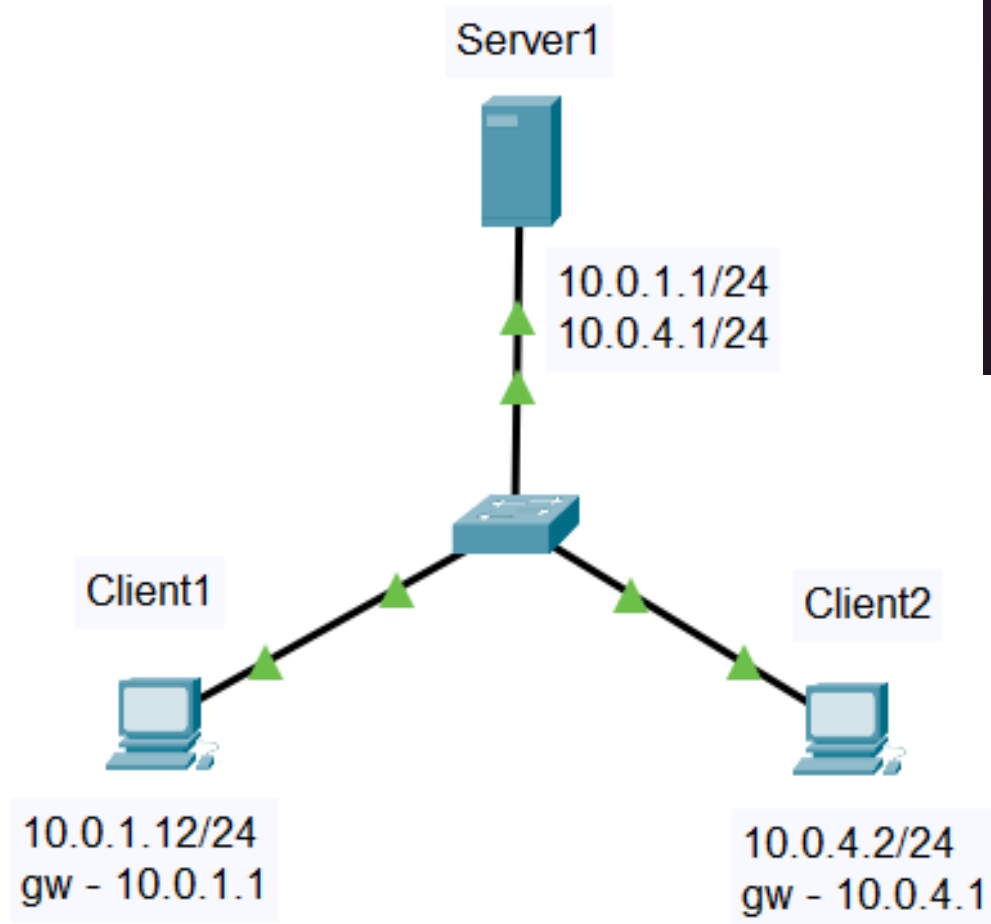
# Quagga installation and configuration

---

- Ubuntu Quagga installation: `sudo apt install quagga-core`
- Configuration files should be in `/etc/quagga`
- Each router daemon gets its own configuration file, for example `/etc/quagga/ospfd.conf`:

```
hostname router1
log file /var/log/quagga/ospfd.log
router ospf
ospf router-id 192.168.110.15
network 192.168.110.0/0 area 0.0.0.0
network 192.168.120.0/0 area 0.0.0.0
access-list localhost permit 127.0.0.1/32
access-list localhost deny any
line vty
access-class localhost
```

# One interface with two IP-addresses routing



```
sergey@Server1:~$ ip addr show dev enp0s8
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel s
roup default qlen 1000
    link/ether 08:00:27:66:83:24 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.1/24 brd 10.0.1.255 scope global noprefixroute enp0s8
        valid_lft forever preferred_lft forever
    inet 10.0.4.1/24 brd 10.0.4.255 scope global noprefixroute enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe66:8324/64 scope link
        valid_lft forever preferred_lft forever
```

```
sergey@Client2:~$ traceroute -n 10.0.1.1
traceroute to 10.0.1.1 (10.0.1.1), 30 hops max, 60 byte packets
 1  10.0.1.1  0.536 ms  0.482 ms  0.462 ms
sergey@Client2:~$ traceroute -n 10.0.4.1
traceroute to 10.0.4.1 (10.0.4.1), 30 hops max, 60 byte packets
 1  10.0.4.1  0.501 ms  0.443 ms  0.423 ms
sergey@Client2:~$ traceroute -n 10.0.1.12
traceroute to 10.0.1.12 (10.0.1.12), 30 hops max, 60 byte packets
 1  10.0.4.1  0.799 ms  0.755 ms  0.741 ms
 2  10.0.1.12  1.053 ms  1.044 ms  1.602 ms
```

---

# General troubleshooting procedures

# Documenting the Network

Network documentation is a complete set of accurate and current network documentation. This documentation includes:

- Configuration files, including network configuration files and end-system configuration files
- Physical and logical topology diagrams
- A baseline performance level

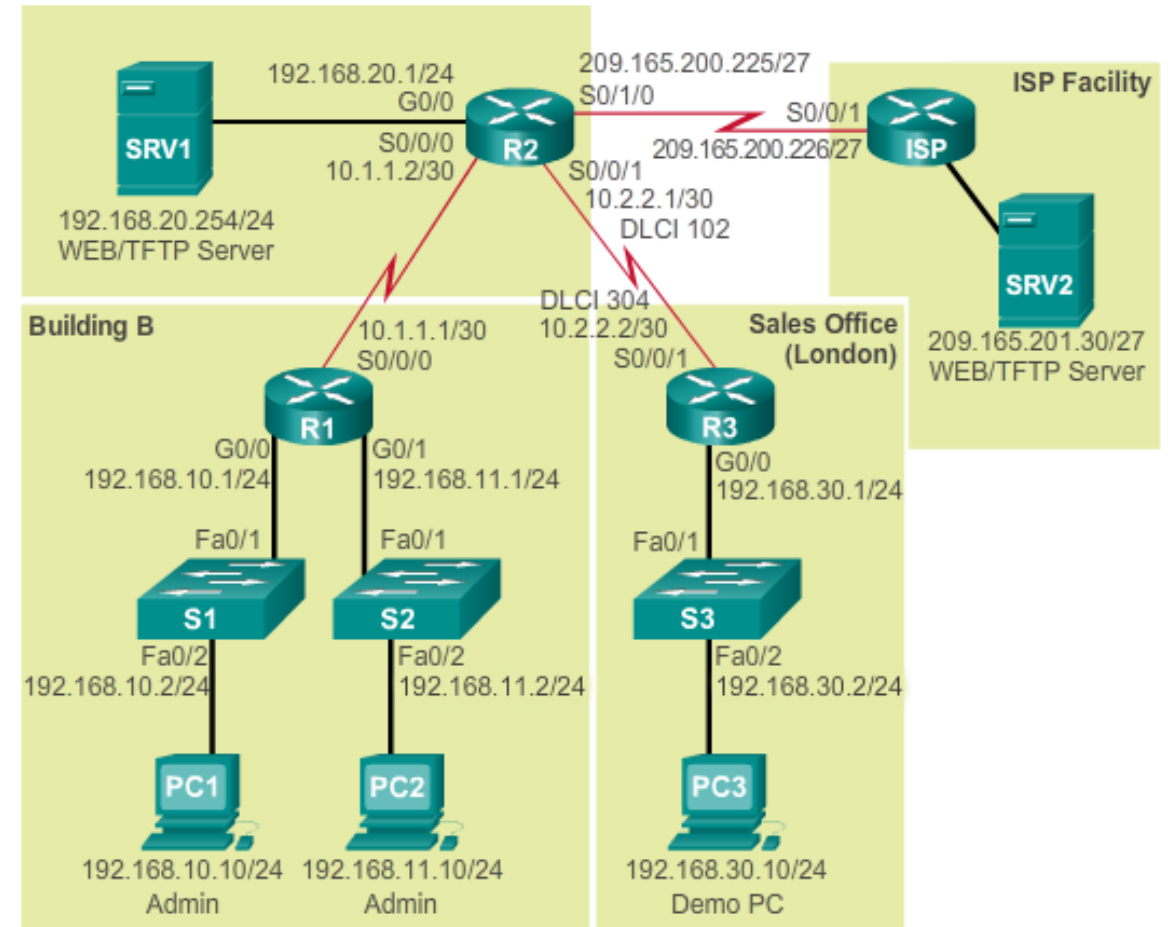
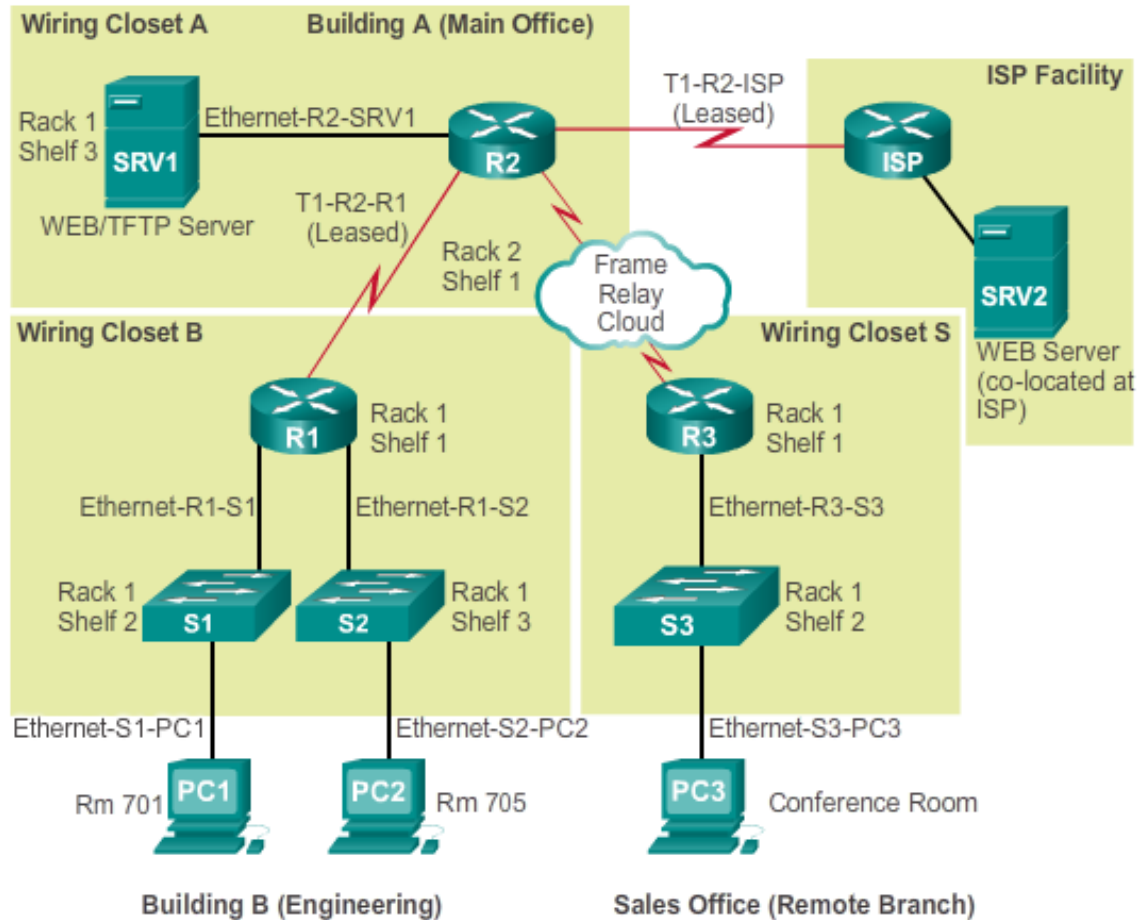
# End-system Configuration Table

End-system configuration files focus on the hardware and software used in end-system devices, such as servers, network management consoles, and user workstations. End-system configuration table includes:

- Device name (purpose)
- Operating system and version
- IPv4 and IPv6 addresses
- Subnet mask and prefix length
- Default gateway, DNS server.
- Any high-bandwidth network applications that the end system runs

Device Name, Purpose	Operating System	MAC Address	IP Address
PC2	Windows 8	5475.D08E.9AD8	192.168.11.10 /24
			2001:DB8:ACAD:11:5075:D0FF:FE8E:9AD8/64
SRV1	Linux	000C.D991.A138	192.168.20.254 /24
			2001:DB8:ACAD:4::100/64

# Physical and Logical Topology





# Network Documentation Instruments

- The **ping** command is used to test connectivity with neighboring devices before logging in to them.
- The **SSH** utility is used to log in remotely to a device for accessing configuration information.
- The **tracert** is a system administrators' utility to trace the route IP packets take from a source system to some destination system.

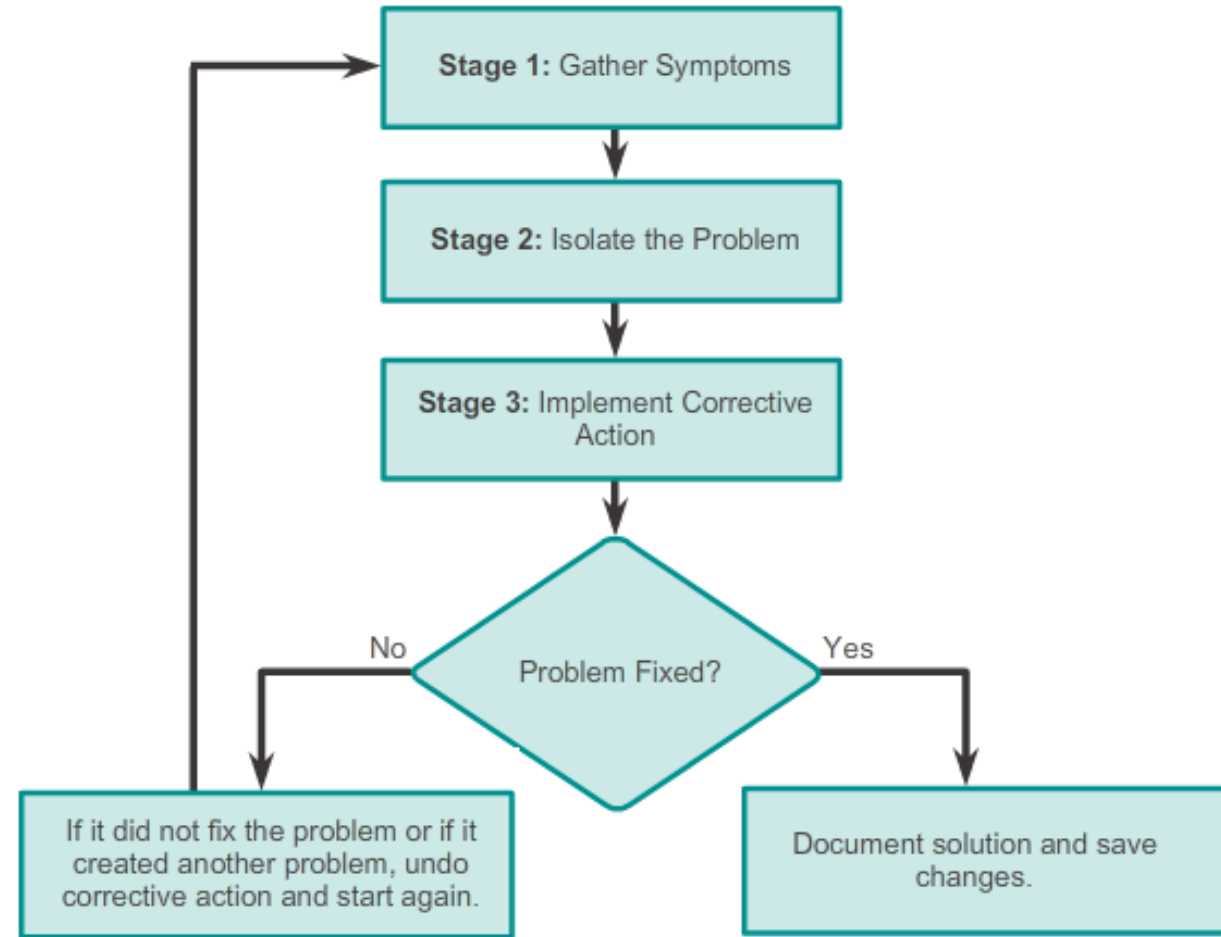
# General Troubleshooting Procedures

- **Stage 1. Gather symptoms -**

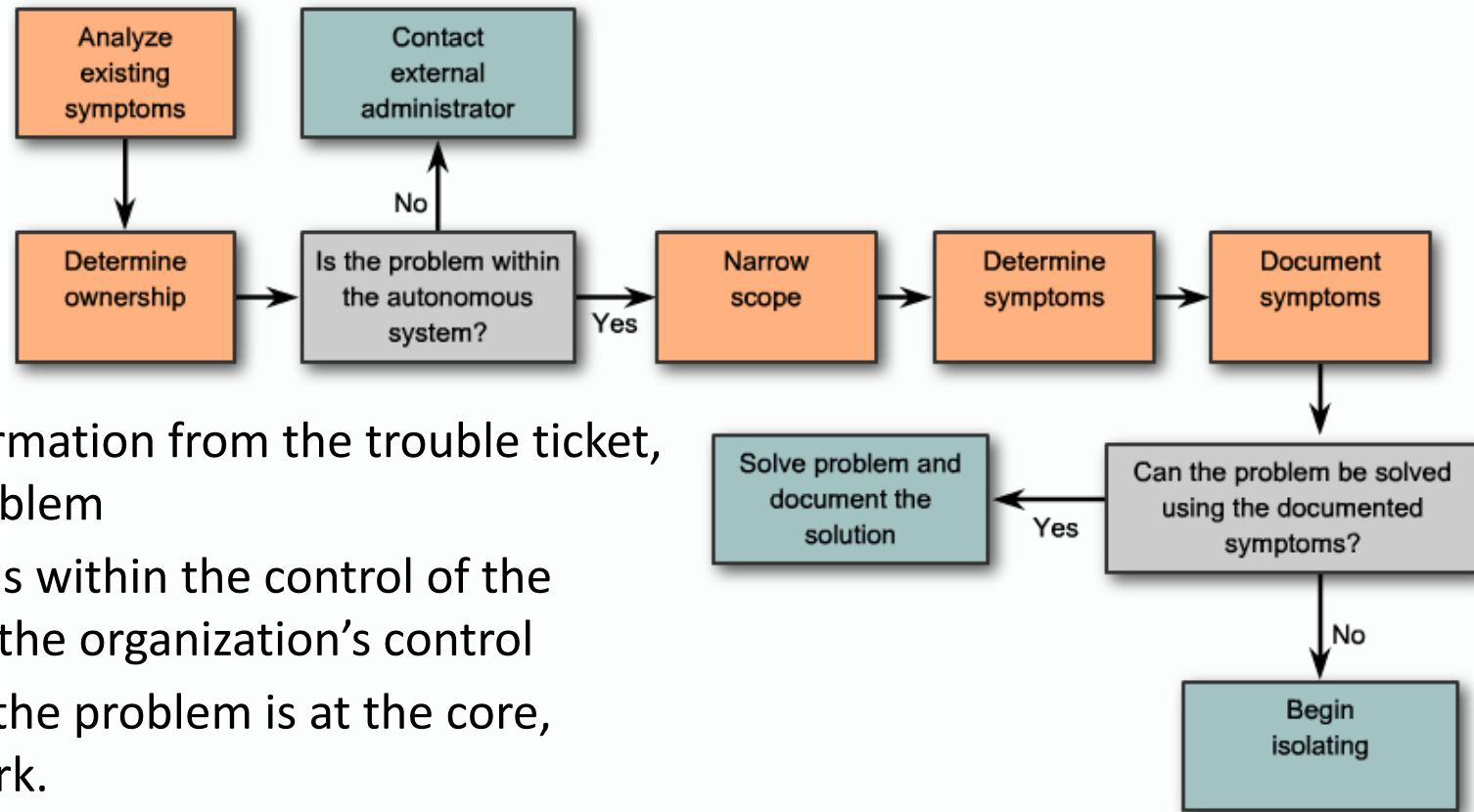
Troubleshooting begins with gathering and documenting symptoms from the network, end systems, and users.

- **Stage 2. Isolate the problem -** Isolating is the process of eliminating variables until a single problem, or a set of related problems has been identified as the cause.

- **Stage 3. Implement corrective action -** Having identified the cause of the problem, the network administrator works to correct the problem by implementing, testing, and documenting possible solutions.



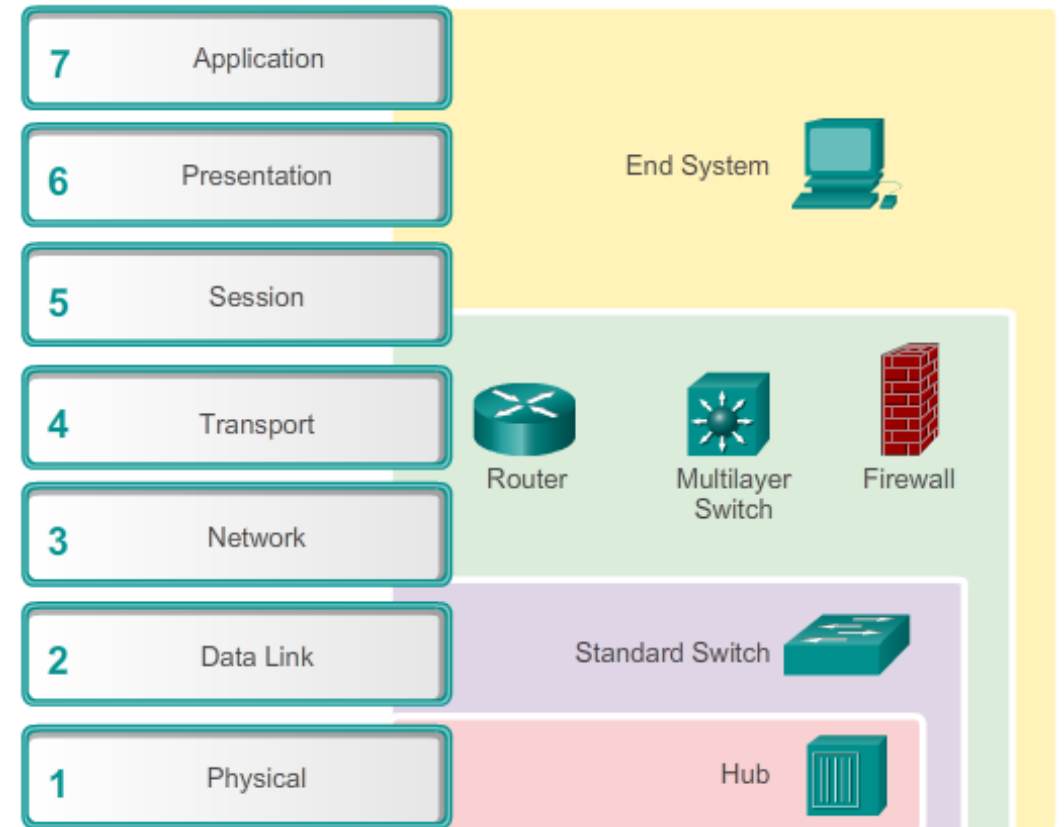
# Gathering Symptoms



- **Step 1. Gather information** - Gather information from the trouble ticket, users, or end systems affected by the problem
- **Step 2. Determine ownership** - problem is within the control of the organization, or outside the boundary of the organization's control
- **Step 3. Narrow the scope** - Determine if the problem is at the core, distribution, or access layer of the network.
- **Step 4. Gather symptoms from suspect devices** - Using a layered troubleshooting approach, gather hardware and software symptoms from the suspect devices.
- **Step 5. Document symptoms** - Sometimes the problem can be solved using the documented symptoms.

# Isolating the Issue Using Layered Models

- After all symptoms are gathered, if no solution is identified, the network administrator compares the characteristics of the problem to the logical layers of the network to isolate and solve the issue.
- Logical networking models, such as the OSI and TCP/IP models, separate network functionality into modular layers. These layered models can be applied to the physical network to isolate network problems when troubleshooting.
- For example, if the symptoms suggest a physical connection problem, the network technician can focus on troubleshooting the circuit that operates at the physical layer. If that circuit functions as expected, the technician looks at areas within another layer that could be causing the problem.

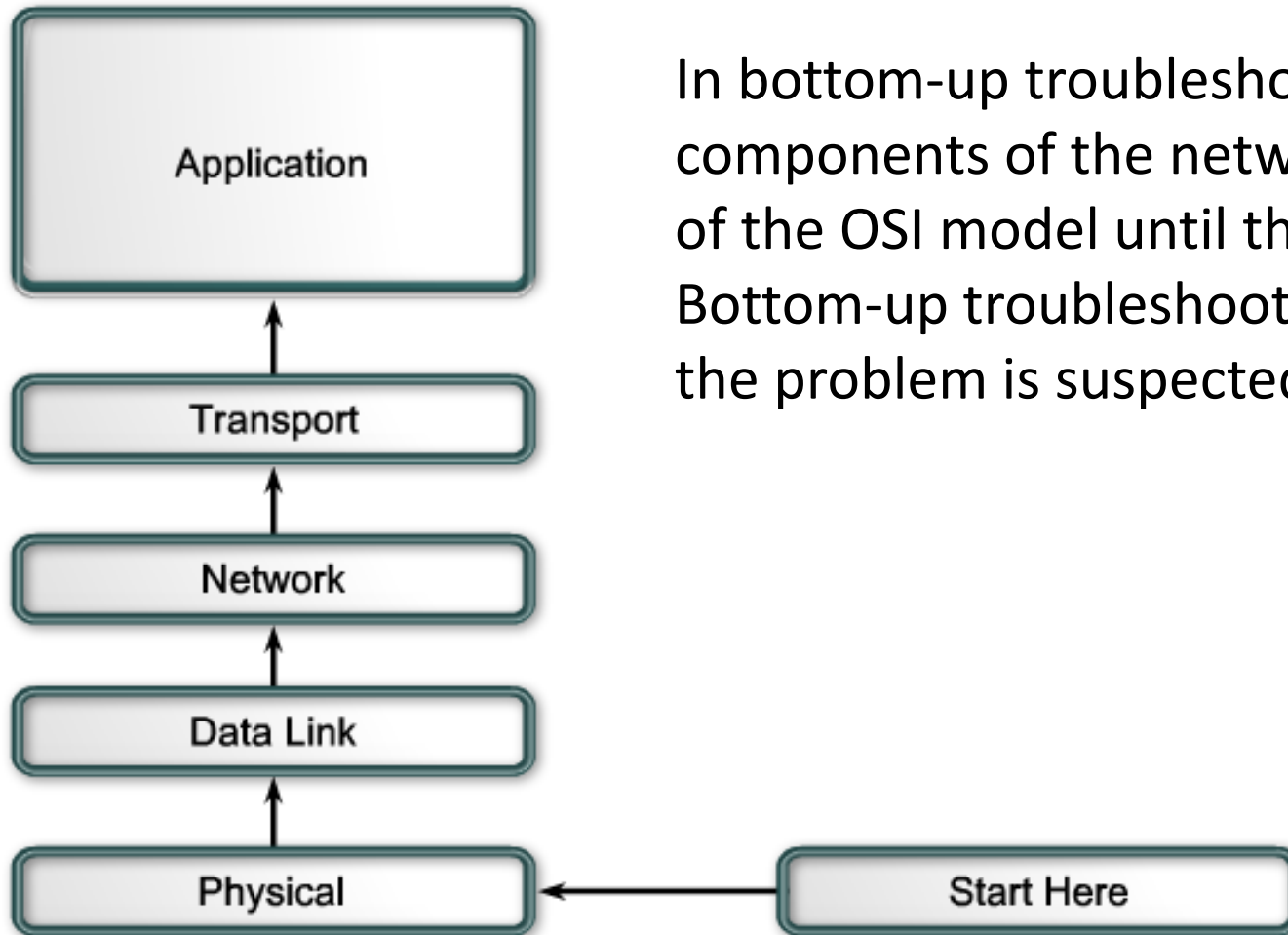


# Troubleshooting Methods

There are three main methods for troubleshooting networks:

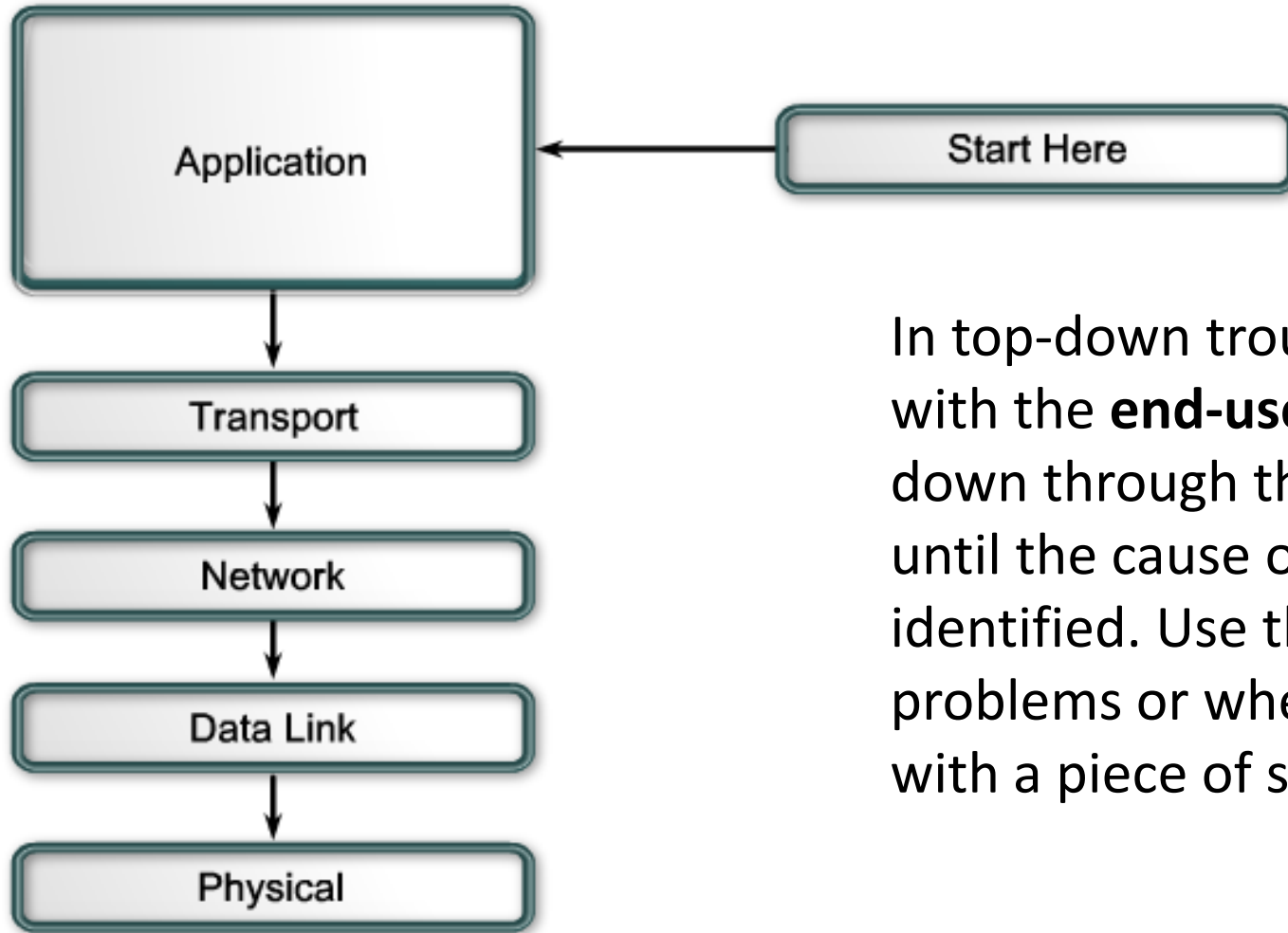
- **Bottom up**
- **Top down**
- **Divide and conquer**

# Bottom-Up Troubleshooting Method



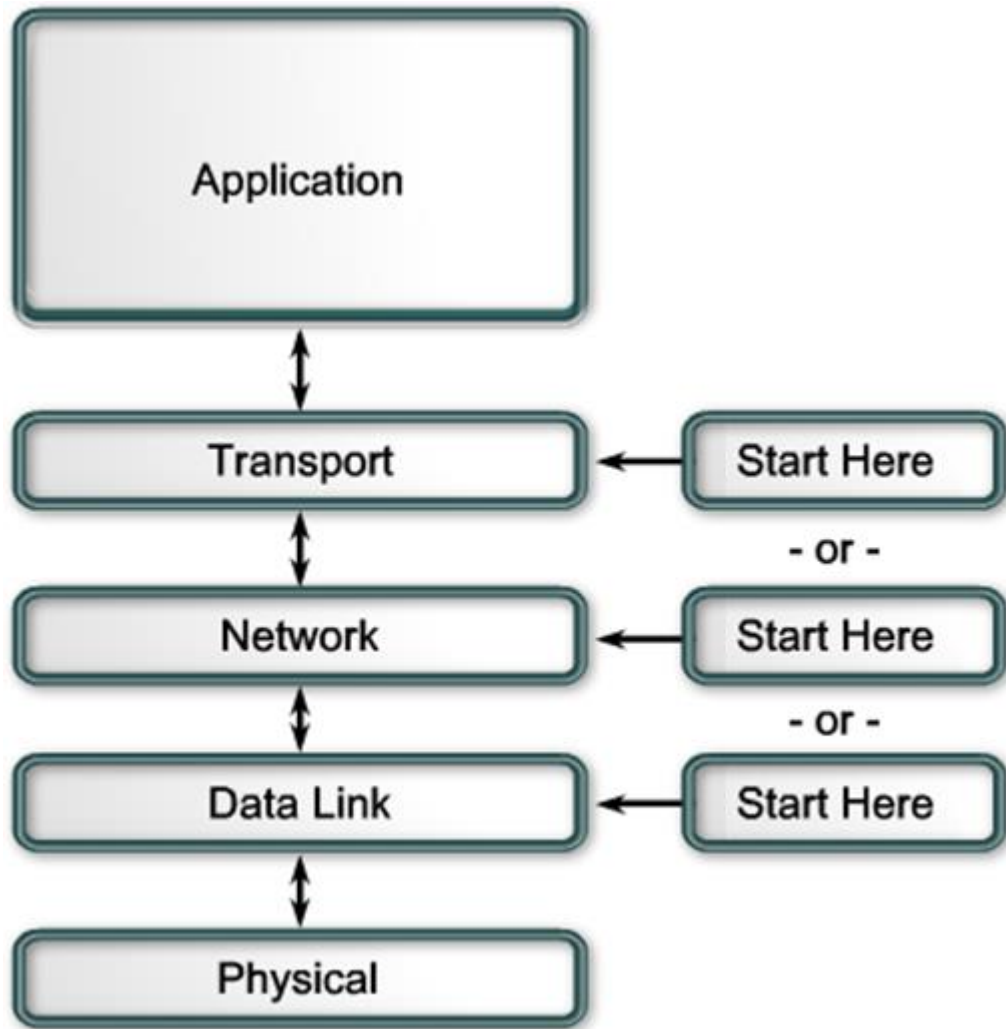
In bottom-up troubleshooting you start with the **physical** components of the network and move up through the layers of the OSI model until the cause of the problem is identified. Bottom-up troubleshooting is a good approach to use when the problem is suspected to be a physical one.

# Top-Down Troubleshooting Method



In top-down troubleshooting you start with the **end-user applications** and move down through the layers of the OSI model until the cause of the problem has been identified. Use this approach for simpler problems or when you think the problem is with a piece of software.

# Divide-and-Conquer Troubleshooting Method



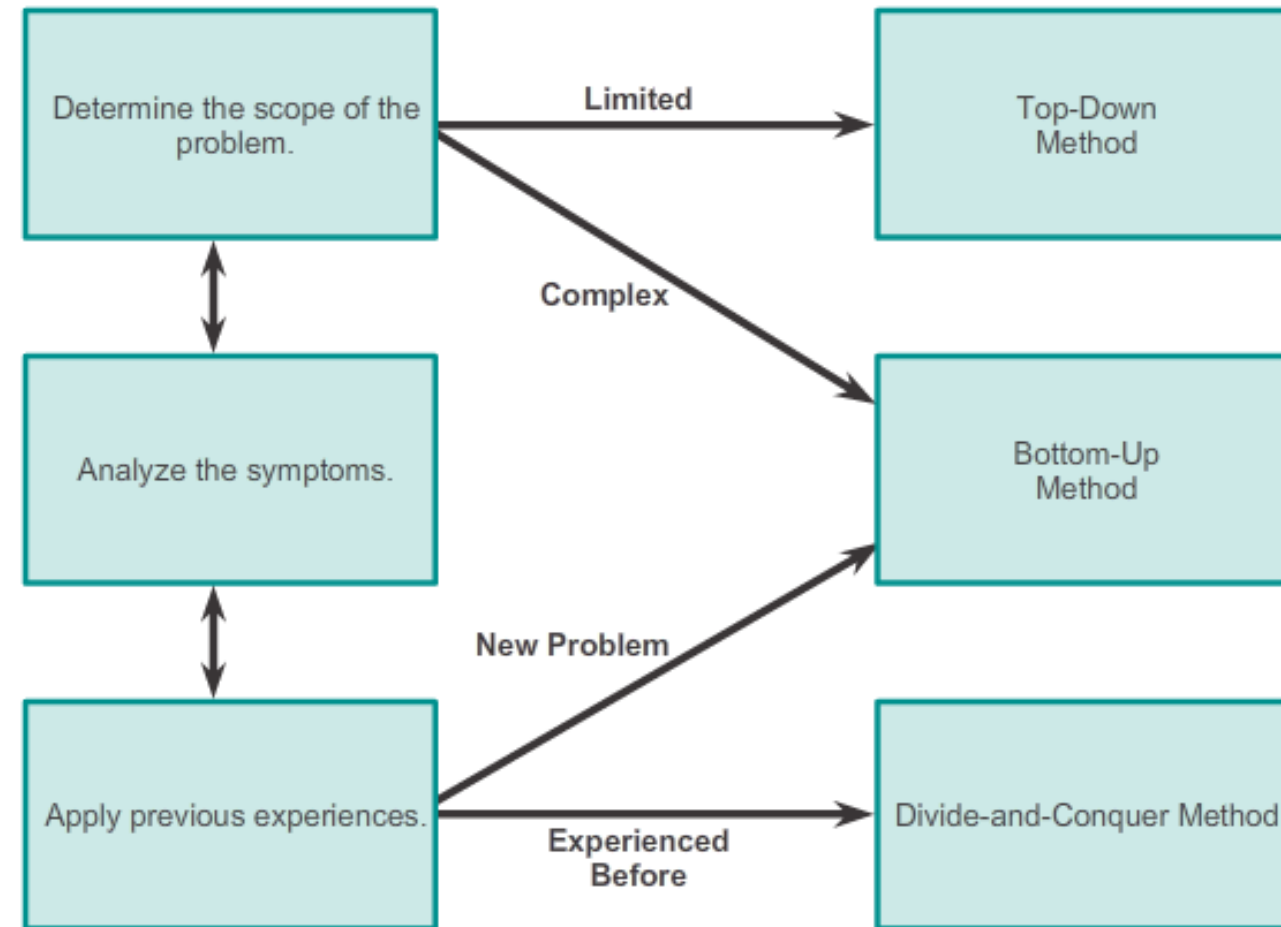
When you apply the divide-and-conquer approach toward troubleshooting a networking problem, you select a layer and test in both directions from the starting layer.



# Troubleshooting Methods

In addition to the systematic, layered approach to troubleshooting, there are also, less-structured troubleshooting approaches:

- One troubleshooting approach is based on an **educated guess** by the network administrator, based on the symptoms of the problem.
- Another approach involves **comparing a working and nonworking situation**, and spotting significant differences, including: Configurations, Software versions, Hardware and other device properties
- **Swapping the problematic device** with a known, working one is a quick way to troubleshoot.



---

# Linux monitoring and troubleshooting tools

# Network monitoring and troubleshooting tools

---

- ping
- traceroute
- mtr
- netstat
- dig
- nmap
- tcpdump

# Ping utility

---

- The Ping utility is an online free tool that help you to verify if a domain/server is operating and network accessible.
- This tool uses the **Internet Control Message Protocol (ICMP)** Echo function as detailed in RFC 792.
- A small packet will be sent through the network to a given IP address (IPv4) or host name. By default, this packet contains **64 bytes**.
- The device that sent the packet then waits and listens for a return packet. If the connections are good and the target domain/server is up, a good return packet will be received.
- Ping can also tell the user the **number of hops** between two targets and the **amount of time** it takes for a packet to make the complete trip.

# Ping utility

```
sergey@Server1:/etc/network$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=18.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=18.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=24.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=18.4 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=18.6 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 17.958/19.621/24.794/2.594 ms
sergey@Server1:/etc/network$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.557 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.799 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.587 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
```

1. **from:** The destination and its IP address. Note that the IP address may be different for a website depending on your geographical location.

2. **icmp\_seq=1:** The sequence number of each ICMP packet. Increases by one for every subsequent echo request.

3. **ttl=118:** The Time to Live value from 1 to 255. It represents the number of network hops a packet can take before a router discards it.

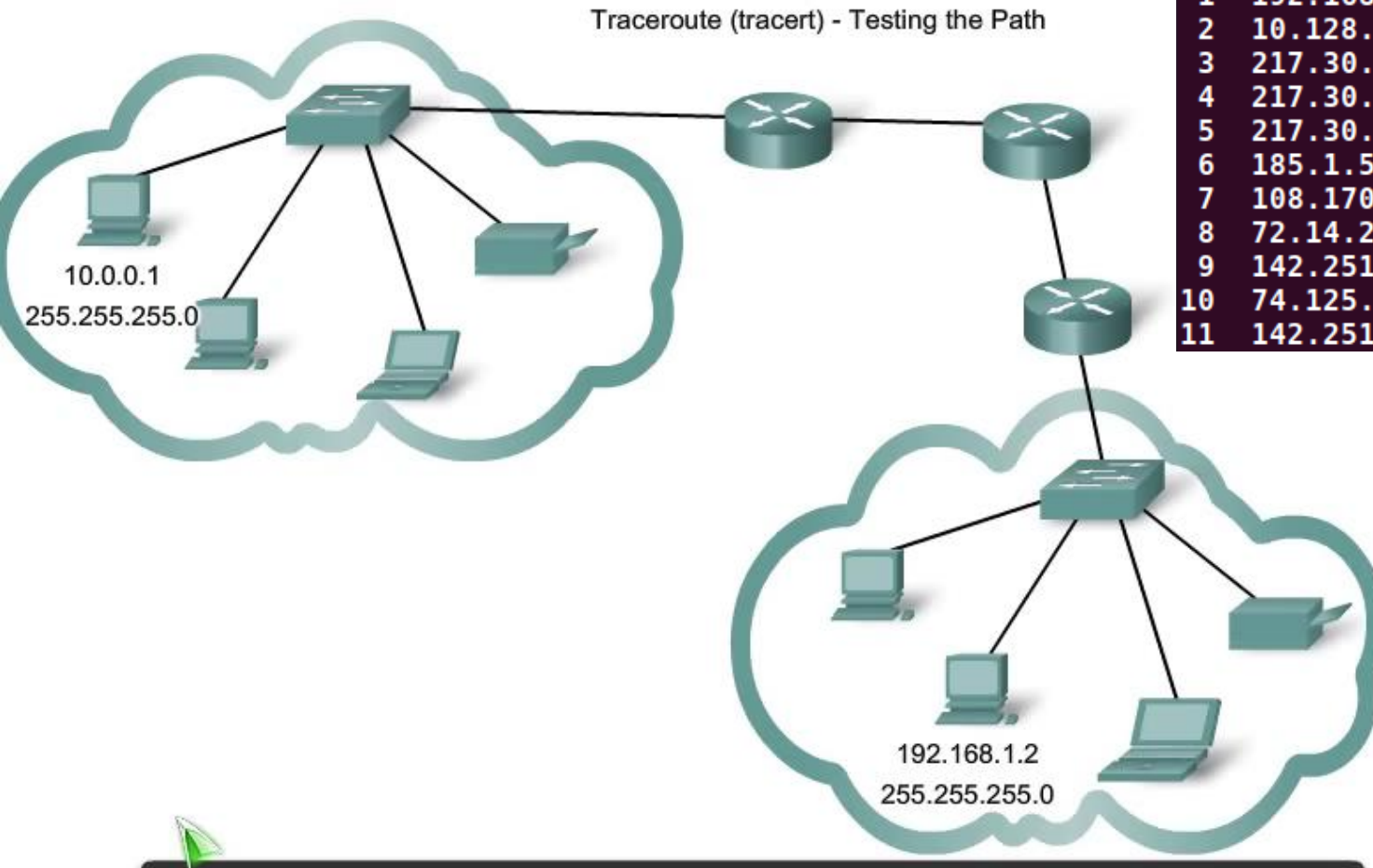
4. **time=7.68 ms:** Round trip time (RTT) - the time it took a packet to reach the destination and come back to the source. Expressed in milliseconds.

# Traceroute utility

---

- Traceroute is a system administrators utility to trace the route IP packets take from a source system to some destination system.
- Traceroute uses the IP TTL (Time To Live) parameter to find the route:
  1. It sends a packet with a TTL value equal to 1.
  2. The first router receives the packet and decreases the TTL.
  3. With a TTL equal to 0, the router sends a timeout back to traceroute, with this packet, traceroute knows about the first router.
  4. Now, traceroute sends another packet with a TTL equal to 2.
  5. The first router decreases the TTL and sends the packet to the second router which decreases it in turn: the TTL is equal to 0...

# Traceroute utility



```
sergey@Server1:/etc/network$ traceroute -n 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  192.168.1.1  0.718 ms  0.660 ms  0.642 ms
 2  10.128.16.1  1.483 ms  1.466 ms  1.840 ms
 3  217.30.200.62 21.617 ms 21.482 ms 21.456 ms
 4  217.30.200.189 2.189 ms 2.050 ms 2.025 ms
 5  217.30.200.218 6.576 ms 6.561 ms 6.544 ms
 6  185.1.50.166 5.286 ms 4.981 ms 5.112 ms
 7  108.170.248.138 5.095 ms 108.170.248.155 5.695 ms 5.662 ms
 8  72.14.239.111 6.542 ms 6.440 ms 6.415 ms
 9  142.251.224.76 18.756 ms 19.009 ms 18.967 ms
10  74.125.242.225 20.217 ms 142.251.228.23 18.933 ms 19.413 ms
11  142.251.228.31 18.110 ms 8.8.8.8 19.361 ms 30.650 ms
```

# mtr utility

- Mtr (my traceroute) is a command line network diagnostic tool that provides the functionality of both the **ping** and **traceroute** commands.
- Just like a traceroute, the mtr command will show the route from a computer to a specified host. mtr provides a lot of statistics about each hop, such as response time and percentage.
- The mtr command output will display the traceroute report in **real time**.

my traceroute [v0.93]

Server1 (192.168.1.105) 2022-01-28T12:20:56+0200

Keys: Help Display mode Restart statistics Order of fields quit

Host	Packets		Pings				
	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1. _gateway	0.0%	52	1.0	1.2	0.6	5.9	0.7
2. 10.128.16.1	0.0%	52	1.4	1.5	0.7	9.7	1.3
3. vl-21.sw-vn-1-1.enet.vn	0.0%	52	16.5	20.5	10.3	111.8	15.0
4. et-0-0-0.boar.enet.vn.u	0.0%	52	5.2	2.1	0.9	13.6	2.2
5. vl-32.sw-kyiv-nt-1.enet	0.0%	52	6.7	6.8	5.9	8.1	0.4
6. google-gw.ix.net.ua	0.0%	52	5.9	6.3	5.0	10.3	1.2
7. 108.170.248.155	0.0%	52	8.9	7.0	5.4	39.0	4.6
8. 72.14.239.111	0.0%	52	6.4	6.8	5.9	10.6	1.0
9. 142.251.224.76	0.0%	51	19.6	21.3	18.6	33.8	4.1
10. 74.125.242.225	0.0%	51	20.3	23.4	19.4	79.9	12.8
11. 209.85.255.243	0.0%	51	20.6	20.4	19.3	33.2	2.1
12. dns.google	0.0%	51	19.3	19.2	18.3	21.4	0.5



# netstat utility

- The Linux netstat command gives you an information about your network connections, the ports that are in use, and the processes using them.

```
sergey@Server1:/etc/dhcp$ netstat -at -n
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp      0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp      0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6     0      0 :::22                   :::*                     LISTEN
tcp6     0      0 :::1:631                 :::*                     LISTEN
```

```
sergey@Server1:/etc/dhcp$ netstat -au -n
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp      0      0 0.0.0.0:5353            0.0.0.0:*               LISTEN
udp      0      0 0.0.0.0:57210           0.0.0.0:*               LISTEN
udp      0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
udp      0      0 0.0.0.0:67              0.0.0.0:*               LISTEN
udp      0      0 0.0.0.0:631             0.0.0.0:*               LISTEN
udp6     0      0 :::5353                  :::*                     LISTEN
udp6     0      0 :::56640                 :::*                     LISTEN
```

# dig and nslookup utility

- Dig, nslookup and host are used for retrieving information about DNS name servers.
- They are basically used for verifying and troubleshooting DNS problems and to perform DNS lookups.
- Installation: In case of Debian/Ubuntu

*\$sudo apt-get install dnsutils*

In case of CentOS/RedHat

*\$sudo yum install bind-utils*

```
sergey@Server1:/etc/network$ host google.com
google.com has address 172.217.20.206
google.com has IPv6 address 2a00:1450:401b:803::200e
google.com mail is handled by 20 alt1.aspmx.l.google.com.
google.com mail is handled by 30 alt2.aspmx.l.google.com.
google.com mail is handled by 40 alt3.aspmx.l.google.com.
google.com mail is handled by 10 aspmx.l.google.com.
google.com mail is handled by 50 alt4.aspmx.l.google.com.
sergey@Server1:/etc/network$
```

```
sergey@Server1:/etc/network$ dig google.com

; <<>> DiG 9.16.1-Ubuntu <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25571
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;google.com.                IN      A

;; ANSWER SECTION:
google.com.                229     IN      A      172.217.20.206

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: nт ciч 28 13:11:54 EET 2022
;; MSG SIZE rcvd: 55
```

```
sergey@Server1:/etc/network$ nslookup google.com
Server:                127.0.0.53
Address:               127.0.0.53#53

Non-authoritative answer:
Name:   google.com
Address: 172.217.20.206
Name:   google.com
Address: 2a00:1450:401b:803::200e
```

# arp utility

- To display the ARP table on a Unix system, just type "*arp -a*" (this same command will show the arp table in the command prompt on a Windows box).
- The output from `arp -a` will list the network interface, target system and physical (MAC) address of each system.

```
sergey@Server1:/etc/network$ arp -a
_gateway (192.168.1.1) at 0c:80:63:eb:1d:7e [ether] on enp0s3
? (10.0.3.10) at 08:00:27:9d:ad:3b [ether] on enp0s9
? (10.0.1.12) at 08:00:27:9e:e9:1a [ether] on enp0s8
sergey@Server1:/etc/network$ arp help
help: Host name lookup failure
sergey@Server1:/etc/network$ arp 10.0.3.10
```

Address	HWtype	HWaddress	Flags	Mask
10.0.3.10	ether	08:00:27:9d:ad:3b	C	
enp0s9				

# Nmap utility

- Nmap, or Network Mapper, is an open source Linux command line tool for network exploration and security auditing.
- With Nmap, server administrators can quickly reveal hosts and services, search for security issues, and scan for open ports.
- The Nmap tool can audit and discover local and remote open ports, as well as network information and hosts.
- To install: `$sudo apt install nmap`
- Help – `nmap -h`
- **Attention - Some countries and companies consider port scanning illegal!**

```
sergey@Client1:~$ nmap 8.8.8.8
Starting Nmap 7.80 ( https://nmap.org ) at 2022-01-29 11:59 EET
Nmap scan report for dns.google (8.8.8.8)
Host is up (0.020s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
53/tcp    open  domain
443/tcp   open  https

Nmap done: 1 IP address (1 host up) scanned in 4.64 seconds
sergey@Client1:~$ nmap cisco.com
Starting Nmap 7.80 ( https://nmap.org ) at 2022-01-29 11:59 EET
Nmap scan report for cisco.com (72.163.4.185)
Host is up (0.17s latency).
Other addresses for cisco.com (not scanned): 2001:420:1101:1::185
rDNS record for 72.163.4.185: redirect-ns.cisco.com
Not shown: 998 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https
```

```
sergey@Client1:~$ nmap -sP 10.0.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2022-01-29 12:09 EET
Nmap scan report for _gateway (10.0.1.1)
Host is up (0.00057s latency).
Nmap scan report for Client1 (10.0.1.12)
Host is up (0.0035s latency).
Nmap done: 256 IP addresses (2 hosts up) scanned in 2.56 seconds
```

# tcpdump utility

- tcpdump is a most powerful and widely used command-line packets **sniffer** or package analyzer tool which is used to capture or filter TCP/IP packets that are received or transferred over a network on a specific interface.
- It is available under most of the Linux/Unix-based operating systems.
- tcpdump also gives us an option to save captured packets in a file for future analysis in a **pcap** format, that can be viewed by tcpdump command or an open-source GUI-based tool called Wireshark

```
sergey@Server1:/etc/network$ ping 10.0.1.12
PING 10.0.1.12 (10.0.1.12) 56(84) bytes of data.
64 bytes from 10.0.1.12: icmp_seq=1 ttl=64 time=0.248 ms
64 bytes from 10.0.1.12: icmp_seq=2 ttl=64 time=0.451 ms
64 bytes from 10.0.1.12: icmp_seq=3 ttl=64 time=0.457 ms
64 bytes from 10.0.1.12: icmp_seq=4 ttl=64 time=0.398 ms
^C
--- 10.0.1.12 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3063ms
rtt min/avg/max/mdev = 0.248/0.388/0.457/0.084 ms
```

```
sergey@Client1:~$ sudo tcpdump icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
12:31:38.655347 IP _gateway > Client1: ICMP echo request, id 5, seq 1, length 64
12:31:38.655383 IP Client1 > _gateway: ICMP echo reply, id 5, seq 1, length 64
12:31:39.670517 IP _gateway > Client1: ICMP echo request, id 5, seq 2, length 64
12:31:39.670549 IP Client1 > _gateway: ICMP echo reply, id 5, seq 2, length 64
12:31:40.694499 IP _gateway > Client1: ICMP echo request, id 5, seq 3, length 64
12:31:40.694527 IP Client1 > _gateway: ICMP echo reply, id 5, seq 3, length 64
12:31:41.718280 IP _gateway > Client1: ICMP echo request, id 5, seq 4, length 64
12:31:41.718304 IP Client1 > _gateway: ICMP echo reply, id 5, seq 4, length 64
^C
8 packets captured
```



# Useful links

---

- <https://phoenixnap.com/kb/linux-ip-command-examples>
- <https://www.geeksforgeeks.org/route-command-in-linux-with-examples/>
- <https://man7.org/linux/man-pages/man8/ip-route.8.html>
- <https://www.linux.com/topic/networking/dynamic-linux-routing-quagga/>
- <https://www.howtogeek.com/657780/how-to-use-the-traceroute-command-on-linux/>
- <https://phoenixnap.com/kb/linux-ping-command-examples>
- <https://www.geeksforgeeks.org/nmap-command-in-linux-with-examples/>

---

# Q&A

A light blue world map is centered on the Atlantic Ocean, showing the continents of North America, South America, Europe, Africa, Asia, and Australia. The map is rendered in a simple, stylized manner with thin lines for coastlines and country borders.

# Thank you!