

Linux Networking

Network Discovery Tools

LinuXpert Systems

WhatsApp: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. nmap 1

2. arp-scan 14

3. netdiscover 20

4. fping 27

5. masscan 35

6. avahi-browse 44

7. lldpctl 55

8. traceroute 67

1. nmap

The **nmap** (Network Mapper) command is a powerful Linux network discovery and security auditing utility used to discover hosts, identify open ports, determine services and versions, detect operating systems, and perform network reconnaissance. It is widely used by network administrators, system administrators, cybersecurity professionals, and students for network troubleshooting and security assessment.

Nmap can scan individual hosts, subnets, ranges of IP addresses, and entire networks. It supports TCP, UDP, ICMP, and several specialized scanning techniques.

Syntax

```
nmap [OPTIONS] TARGET
```

- **OPTIONS** - Specify the type of scan or additional scanning parameters.
- **TARGET** - The host, IP address, hostname, subnet, or IP range to scan.

Basic Examples

Scan a single host

```
nmap 192.168.1.10
```

Scans the most common TCP ports on the specified host.

Scan a hostname

```
nmap server.example.com
```

Scan multiple hosts

```
Nmap 192.168.1.10 192.168.1.20 192.168.1.30
```

Scan a range of IP addresses

```
nmap 192.168.1.1-50
Scans IP addresses from 192.168.1.1 through 192.168.1.50.
```

Scan an entire subnet

```
nmap 192.168.1.0/24
```

This scans the 256 addresses represented by the `/24` network.

Common nmap Options

`-v` - Verbose output

```
nmap -v 192.168.1.10
```

Displays additional information while the scan is running.

`-vv` - Very verbose output

```
nmap -vv 192.168.1.10
```

Provides even more detailed progress information.

`-n` - Disable DNS resolution

```
nmap -n 192.168.1.0/24
```

Prevents Nmap from performing reverse DNS lookups, which can make scans faster.

`-R` - Always perform DNS resolution

```
nmap -R 192.168.1.10
```

Requests DNS resolution for discovered hosts.

Host Discovery

Nmap can determine which systems are active before scanning their ports.

`-sn` - Ping scan

```
nmap -sn 192.168.1.0/24
```

Performs host discovery without performing a port scan.

Useful for finding active systems on a LAN.

Example output:

```
Nmap scan report for 192.168.1.1
Host is up.

Nmap scan report for 192.168.1.10
Host is up.
```

`-Pn` - Treat hosts as online

```
nmap -Pn 192.168.1.10
```

Skips host discovery and assumes the target is online.

This is useful when a firewall blocks ICMP/ping requests.

`-PS` - TCP SYN host discovery

```
nmap -PS80,443 192.168.1.10
```

Uses TCP SYN probes against the specified ports.

`-PA` - TCP ACK host discovery

```
nmap -PA80,443 192.168.1.10
```

Uses TCP ACK probes for host discovery.

`-PE` - ICMP Echo discovery

```
nmap -PE 192.168.1.0/24
```

Uses ICMP Echo requests to discover hosts.

Port Scanning

Default scan

```
nmap 192.168.1.10
```

Scans Nmap's default set of commonly used TCP ports.

`-p` - Scan specific ports

```
nmap -p 22 192.168.1.10
```

Scans only port 22.

Multiple ports:

```
nmap -p 22,80,443 192.168.1.10
```

Port range:

```
nmap -p 1-1000 192.168.1.10
```

All TCP ports:

```
nmap -p- 192.168.1.10
```

This scans ports `1-65535`.

Scan common HTTP ports

```
nmap -p 80,443,8080,8443 192.168.1.10
```

TCP Scanning

`-sS` - TCP SYN scan

```
sudo nmap -sS 192.168.1.10
```

The TCP SYN scan is one of Nmap's most commonly used scanning techniques.

It sends TCP SYN packets and analyzes the response without completing the normal TCP connection.

`-sT` - TCP Connect scan

```
nmap -sT 192.168.1.10
```

Uses the operating system's normal TCP connection mechanism.

This is commonly used when raw packet privileges are unavailable.

`-sA` - TCP ACK scan

```
sudo nmap -sA 192.168.1.10
```

Primarily used to determine firewall filtering rather than whether a port is open.

UDP Scanning

`-sU` - UDP scan

```
sudo nmap -sU 192.168.1.10
```

Scans UDP ports.

UDP scanning can be significantly slower than TCP scanning.

Scan selected UDP ports

```
sudo nmap -sU -p 53,67,68,123,161 192.168.1.10
```

For example:

- `53` - DNS
- `67/68` - DHCP
- `123` - NTP
- `161` - SNMP

Service and Version Detection

`-sV` - Detect service versions

```
nmap -sV 192.168.1.10
```

Attempts to determine which services are running and their versions.

Example:

```
PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 8.9
80/tcp  open  http    Apache httpd 2.4.52
```

`--version-intensity`

```
nmap -sV --version-intensity 5 192.168.1.10
```

Controls the intensity of service/version detection.

Values range from `0` to `9`.

Operating System Detection

`-O` - OS detection

```
sudo nmap -O 192.168.1.10
```

Attempts to identify the operating system of the target.

Example:

```
OS details:
Linux 5.x
```

Combine OS and version detection

```
sudo nmap -O -sV 192.168.1.10
```

Aggressive Scan

`-A` - Aggressive scan

```
sudo nmap -A 192.168.1.10
```

Enables several advanced detection features, including:

- OS detection
- Version detection
- Script scanning
- Traceroute

It can generate considerably more network traffic than a basic scan.

Nmap Scripting Engine

Nmap includes the **Nmap Scripting Engine (NSE)**, which provides scripts for service discovery, configuration checks, vulnerability assessment, authentication testing, and many other tasks.

`-sC` - Default NSE scripts

```
nmap -sC 192.168.1.10
```

Runs the default collection of Nmap scripts.

Equivalent to:

```
nmap --script=default 192.168.1.10
```

Run a specific script

```
nmap --script=http-title 192.168.1.10
```

This can retrieve the title of an HTTP website.

Run multiple scripts

```
nmap --script=http-title,http-headers 192.168.1.10
```

List available scripts

On many Linux distributions:

```
ls /usr/share/nmap/scripts/
```

Search for HTTP-related scripts:

```
ls /usr/share/nmap/scripts/ | grep http
```

Firewall and Filtering Detection

Nmap categorizes ports using states such as:

- **open** - An application is actively accepting connections.
- **closed** - The port is reachable but no application is listening.
- **filtered** - A firewall or packet filter prevents Nmap from determining whether the port is open.
- **unfiltered** - The port is accessible but Nmap cannot determine whether it is open or closed.
- **open|filtered** - Nmap cannot determine whether the port is open or filtered.
- **closed|filtered** - Nmap cannot determine whether the port is closed or filtered.

For example:

```
PORT    STATE    SERVICE
22/tcp  open     ssh
80/tcp  open     http
443/tcp filtered https
```

Firewall Detection with ACK Scan

```
sudo nmap -sA 192.168.1.10
```

This can help determine whether ports are filtered by a firewall.

For example:

```
PORT    STATE
22/tcp  unfiltered
80/tcp  filtered
443/tcp filtered
```

Traceroute

`--traceroute`

```
sudo nmap --traceroute 192.168.1.10
```

Attempts to determine the network path between the scanning system and the target.

Scan Faster

`-T4` - Faster timing

```
nmap -T4 192.168.1.0/24
```

Nmap provides timing templates:

```
-T0  Paranoid
-T1  Sneaky
-T2  Polite
-T3  Normal
-T4  Aggressive
-T5  Insane
```

`-T3` is the default. `-T4` is commonly useful on controlled networks.

Scan a Specific Network Interface

If the system has multiple network interfaces, Nmap can be instructed to use a particular interface.

```
nmap -e eth0 192.168.1.10
```

For a wireless interface:

```
sudo nmap -e wlan0 192.168.1.10
```

List interfaces:

```
ip link
```

IPv6 Scanning

`-6`

```
nmap -6 2001:db8::10
```

Enables IPv6 scanning.

Save Scan Results

Normal output

```
nmap -oN scan.txt 192.168.1.10
```

XML output

```
nmap -oX scan.xml 192.168.1.10
```

XML output is useful for importing Nmap results into other tools.

Grepable output

```
nmap -oG scan.txt 192.168.1.10
```

All major formats

```
nmap -oA scan-results 192.168.1.10
```

This creates files such as:

```
scan-results.nmap
scan-results.xml
scan-results.gnmap
```

Scan Multiple Targets

From a file

Create a file:

```
192.168.1.10
192.168.1.20
192.168.1.30
```

Then:

```
nmap -iL targets.txt
```

Exclude a host

```
nmap 192.168.1.0/24 --exclude 192.168.1.10
```

Exclude multiple hosts

```
nmap 192.168.1.0/24 --exclude 192.168.1.10,192.168.1.20
```

Useful Practical Examples

1. Find active computers on a LAN

```
nmap -sn 192.168.1.0/24
```

---

2. Find open SSH ports

```
nmap -p 22 192.168.1.0/24
```

---

3. Find HTTP/HTTPS servers

```
nmap -p 80,443 192.168.1.0/24
```

---

4. Scan all TCP ports

```
sudo nmap -p- 192.168.1.10
```

5. Identify services

```
nmap -sV 192.168.1.10
```

---

6. Identify the operating system

```
sudo nmap -O 192.168.1.10
```

---

7. Perform a comprehensive scan

```
sudo nmap -A 192.168.1.10
```

---

8. Scan TCP and UDP ports

```
sudo nmap -sS -sU 192.168.1.10
```

---

9. Scan a subnet for SSH services and identify versions

```
nmap -p 22 -sV 192.168.1.0/24
```

---

10. Save the complete result

```
sudo nmap -A -oA network-audit 192.168.1.0/24
```

Installing nmap

On Ubuntu/Debian:

```
sudo apt update
sudo apt install nmap
```

Check the installed version:

```
nmap --version
```

Example:

```
Nmap version 7.xx
```

Important Security Note

Nmap is a legitimate network administration and security-auditing tool, but scanning networks or systems **without authorization may violate organizational policies or applicable laws**. For training, students should use their own systems, an isolated laboratory network, or explicitly authorized targets.

Quick Reference

Command	Purpose
`nmap host`	Basic scan
`nmap -sn network`	Discover live hosts
`nmap -Pn host`	Skip host discovery
`nmap -p 80 host`	Scan port 80
`nmap -p 1-1000 host`	Scan port range
`nmap -p- host`	Scan all TCP ports
`nmap -sS host`	TCP SYN scan
`nmap -sT host`	TCP Connect scan
`nmap -sU host`	UDP scan
`nmap -sV host`	Service/version detection
`nmap -O host`	OS detection
`nmap -A host`	Aggressive scan
`nmap -sC host`	Default NSE scripts
`nmap -sA host`	ACK/firewall scan
`nmap --traceroute host`	Network route discovery
`nmap -6 host`	IPv6 scan
`nmap -iL file`	Scan targets from file
`nmap -oN file host`	Save normal output
`nmap -oX file host`	Save XML output
`nmap -oA name host`	Save in multiple formats

2. arp-scan

1. What is `arp-scan` ?

`arp-scan` (Discover Devices on a Local Network) is a Linux network-discovery tool that uses the **ARP** (Address Resolution Protocol) to identify active devices on a local Ethernet or Wi-Fi network. Unlike a conventional ping sweep, ARP discovery can identify devices that may not respond to ICMP ping.

---

2. Install `arp-scan`

Ubuntu / Debian

```
sudo apt update
sudo apt install arp-scan
```

Check the installation:

```
arp-scan --version
```

---

3. Find Your Network Interface

First identify the available interfaces:

```
ip link
```

or:

```
ip addr
```

Example:

```
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP>
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP>
```

Typical interfaces:

Interface	Type
`enp3s0`	Ethernet
`eth0`	Older Ethernet naming
`wlp2s0`	Wi-Fi
`wlan0`	Older Wi-Fi naming

4. Automatically Determine the Local Network

A convenient option is:

```
sudo arp-scan --localnet
```

Example output:

```
Interface: enp3s0, type: EN10MB, MAC: 00:11:22:33:44:55
Starting arp-scan 1.10.0

192.168.1.1     aa:bb:cc:dd:ee:ff    TP-Link
192.168.1.10    10:20:30:40:50:60    Dell Inc.
192.168.1.15    70:80:90:a0:b0:c0    Hewlett-Packard
192.168.1.20    12:34:56:78:9a:bc    Samsung Electronics
4 packets received by filter, 0 packets dropped
```

This is one of the easiest commands for students to start with.

---

5. Scan a Specific Interface

If the computer has multiple network interfaces:

```
sudo arp-scan --interface=enp3s0 --localnet
```

For Wi-Fi:

```
sudo arp-scan --interface=wlp2s0 --localnet
```

This is particularly useful when a system has **Ethernet + Wi-Fi** simultaneously.

---

6. Scan a Specific Network

You can specify a network range:

```
sudo arp-scan 192.168.1.0/24
```

This scans approximately:

```
192.168.1.1
192.168.1.2
...
192.168.1.254
```

You can also specify a smaller range:

```
sudo arp-scan 192.168.1.1-192.168.1.50
```

7. Discover Devices on Wi-Fi

Determine the wireless interface:

```
ip -br addr
```

Example:

```
lo       UNKNOWN  127.0.0.1/8
enp3s0   DOWN
wlp2s0   UP       192.168.1.25/24
```

Then:

```
sudo arp-scan --interface=wlp2s0 --localnet
```

You may see:

```
192.168.1.1     00:11:22:33:44:55    Router
192.168.1.8     aa:bb:cc:dd:ee:ff    Apple
192.168.1.12    11:22:33:44:55:66    Xiaomi
192.168.1.25    77:88:99:aa:bb:cc    Intel
```

---

8. Display Only IP and MAC Information

For learning purposes, students can focus on the three important pieces of information:

```
IP Address → MAC Address → Vendor
```

For example:

```
192.168.1.12    44:65:0d:xx:xx:xx    Raspberry Pi Foundation
```

This helps students understand the relationship between **Layer 3 IP addressing** and **Layer 2 MAC addressing**.

---

9. Scan Using a Different Interface

List interfaces:

```
ip -br link
```

Then explicitly select one:

```
sudo arp-scan -I enp3s0 --localnet
```

`-I` specifies the interface.

Equivalent long form:

```
sudo arp-scan --interface=enp3s0 --localnet
```

---

10. Read the ARP Cache

Linux maintains an ARP/neighbor table.

Display it with:

```
ip neigh
```

Example:

```
192.168.1.1 dev wlp2s0 lladdr aa:bb:cc:dd:ee:ff REACHABLE
192.168.1.10 dev wlp2s0 lladdr 11:22:33:44:55:66 STALE
```

Compare this with:

```
sudo arp-scan --localnet
```

This is an excellent practical exercise for understanding **ARP discovery versus the kernel's neighbor table**.

---

11. Compare `arp-scan` with `ping`

A traditional approach might be:

```
ping 192.168.1.10
```

But a host may block ICMP.

`arp-scan` uses ARP on the local network:

```
sudo arp-scan 192.168.1.0/24
```

Therefore, it can sometimes discover a local device even when:

```
ping 192.168.1.x
```

does not receive a response.

Important:

ARP scanning is primarily useful on the **local Layer-2 network**. It is not a replacement for Nmap when you need remote-network port/service discovery.

12. Useful `arp-scan` Options

Command	Purpose
`arp-scan --localnet`	Scan the local network
`arp-scan -I eth0 --localnet`	Scan through eth0
`arp-scan -I wlan0 --localnet`	Scan through Wi-Fi
`arp-scan 192.168.1.0/24`	Scan a specific subnet
`arp-scan 192.168.1.1-50`	Scan an IP range
`arp-scan --version`	Display version
`arp-scan --help`	Display help

13. Practical Exercise

Exercise 1 - Discover your LAN

Run:

```
sudo arp-scan --localnet
```

Record:

```
IP Address
MAC Address
Vendor
```

Exercise 2 - Identify your router

Run:

```
ip route
```

Look for:

```
default via 192.168.1.1
```

Then:

```
sudo arp-scan 192.168.1.1
```

Compare the MAC address reported by `arp-scan`.

Exercise 3 - Compare Ethernet and Wi-Fi

If both interfaces are available:

```
sudo arp-scan -I enp3s0 --localnet
```

and:

```
sudo arp-scan -I wlp2s0 --localnet
```

Observe how the network/interface affects the discovered devices.

Exercise 4 - Compare with the kernel neighbor table

Run:

```
sudo arp-scan --localnet
```

Then:

```
ip neigh
```

Compare the results.

---

14. `arp-scan` vs `nmap`

Feature	arp-scan	nmap
Local host discovery		
ARP discovery		
MAC address	Yes	Yes
Vendor identification	Yes	Yes
Port scanning	No	Yes
Service detection	No	Yes
OS detection	No	Yes
Remote networks	Limited	Yes
Learning Layer-2 networking	Excellent	Good
Simplicity	Very easy	More advanced

Key learning point

`arp-scan` teaches students how Linux discovers devices at the Ethernet/Wi-Fi Layer-2 level, while **`nmap`** provides much broader network and service reconnaissance.

3. netdiscover

1. What is `netdiscover` ?

`netdiscover` (Local Network Discovery) is a Linux network-discovery tool that uses **ARP requests** to identify active devices on a local network.

It is particularly useful for learning how to:

- Discover computers, phones, routers, and other devices
- Identify **IP addresses and MAC addresses**
- Determine hardware vendors
- Understand local-network topology
- Troubleshoot unknown devices on a LAN

---

2. Install `netdiscover`

Ubuntu / Debian

```
sudo apt update
sudo apt install netdiscover
```

Verify:

```
netdiscover --help
```

---

3. Identify Your Network Interface

Use:

```
ip -br link
```

Example:

```
lo       UNKNOWN
enp3s0   UP
wlp2s0   DOWN
```

For a Wi-Fi connection, you might see:

```
wlp2s0   UP
```

You can also use:

```
ip addr
```

4. Basic Network Discovery

Run:

```
sudo netdiscover
```

`netdiscover` will attempt to determine the local network and begin discovering devices.

Example:

```
Currently scanning: 192.168.1.0/24

IP              MAC Address       Count    Len   MAC Vendor
-------------------------------------------------------------
192.168.1.1     AA:BB:CC:DD:EE:FF  1       60    TP-Link
192.168.1.10    11:22:33:44:55:66  1       60    Dell
192.168.1.15    77:88:99:AA:BB:CC  1       60    Hewlett-Packard
192.168.1.20    12:34:56:78:9A:BC  1       60    Samsung
```

The three most important fields are:

```
IP Address → MAC Address → Vendor
```

---

5. Scan a Specific Network

You can specify the network manually:

```
sudo netdiscover -r 192.168.1.0/24
```

For example, to scan a `/24` network:

```
192.168.1.1
192.168.1.2
...
192.168.1.254
```

This is useful when you know the subnet you want to investigate.

---

6. Scan a Smaller IP Range

You can restrict the scan to a smaller range:

```
sudo netdiscover -r 192.168.1.1-192.168.1.50
```

This can reduce unnecessary traffic when you are investigating a particular portion of the network.

7. Select a Network Interface

If your computer has multiple interfaces, specify which one to use:

```
sudo netdiscover -i enp3s0
```

For Wi-Fi:

```
sudo netdiscover -i wlp2s0
```

This is especially important on systems with both **Ethernet and wireless interfaces**.

---

8. Passive Discovery

One interesting feature of `netdiscover` is passive mode.

```
sudo netdiscover -p
```

In passive mode, **`netdiscover`** listens for network traffic rather than actively sending ARP probes.

This can be useful for teaching the difference between:

Active discovery

```
Linux → ARP Request → Network
```

and:

Passive discovery

```
Network traffic → Linux listens → Device information
```

---

9. Understand ARP Discovery

When a Linux machine wants to communicate with another local IP address, it may need the corresponding MAC address.

For example:

```
Who has 192.168.1.20?
```

The device owning that IP responds with its MAC address:

```
192.168.1.20 is at 12:34:56:78:9A:BC
```

`netdiscover` takes advantage of this ARP mechanism to discover devices on the local network.

10. Compare with `ip neigh`

Linux itself maintains a neighbor/ARP table.

Run:

```
ip neigh
```

Example:

```
192.168.1.1 dev wlp2s0 lladdr aa:bb:cc:dd:ee:ff REACHABLE
192.168.1.10 dev wlp2s0 lladdr 11:22:33:44:55:66 STALE
```

Now run:

```
sudo netdiscover -r 192.168.1.0/24
```

Compare the results.

This is a very good exercise for understanding the relationship between:

```
ARP
↓
MAC Address
↓
Linux Neighbor Table
↓
Network Discovery
```

---

11. Useful `netdiscover` Options

Command	Purpose
`sudo netdiscover`	Automatically discover the local network
`sudo netdiscover -r 192.168.1.0/24`	Scan a specific subnet
`sudo netdiscover -r 192.168.1.1-50`	Scan an IP range
`sudo netdiscover -i eth0`	Use a specific Ethernet interface
`sudo netdiscover -i wlan0`	Use a specific wireless interface
`sudo netdiscover -p`	Passive discovery
`netdiscover -h`	Display help

12. Exercise 1 - Discover Your LAN

Run:

```
sudo netdiscover
```

Record:

IP Address	MAC Address	Vendor
192.168.1.1	...	...
192.168.1.10	...	...
192.168.1.15	...	...

Try to identify:

- Your router
- Your Linux computer
- Other computers
- Mobile phones
- Printers
- IoT devices

Only scan networks you own or are authorized to administer.

---

13. Exercise 2 - Identify Your Router

First determine the default gateway:

```
ip route
```

Example:

```
default via 192.168.1.1 dev wlp2s0
```

Then:

```
sudo netdiscover -r 192.168.1.1/32
```

Observe the router's MAC address and vendor.

14. Exercise 3 - Ethernet vs Wi-Fi

If both interfaces are available:

```
sudo netdiscover -i enp3s0 -r 192.168.1.0/24
```

Then:

```
sudo netdiscover -i wlp2s0 -r 192.168.1.0/24
```

Compare the devices discovered through each interface.

This demonstrates an important networking principle:

Network discovery depends on the Layer-2 network to which the interface is connected.

---

15. Exercise 4 - Active vs Passive Discovery

First run:

```
sudo netdiscover -r 192.168.1.0/24
```

Then:

```
sudo netdiscover -p
```

Observe the difference between **actively probing** the network and **passively observing** network activity.

---

16. `netdiscover` (vs) `arp-scan` (vs) `nmap`

Feature	netdiscover	arp-scan	nmap
Local host discovery			
ARP discovery	Yes	Yes	Yes
MAC address	Yes	Yes	Yes
Vendor identification	Yes	Yes	Yes
Passive mode	**Yes**	No	Limited
Port scanning	No	No	**Yes**
Service detection	No	No	**Yes**
OS detection	No	No	**Yes**
Learning Layer-2 networking	Excellent	Excellent	Good
Ease of use	Very easy	Very easy	Moderate/Advanced

Key learning point

`netdiscover` is excellent for introducing students to local-network discovery and ARP. **`arp-scan`** provides more focused and precise ARP scanning, while **`nmap`** should be introduced later for ports, services, and operating-system discovery.

For **Learning**, a useful progression would be:

```
ip
↓
ip neigh
↓
netdiscover
↓
arp-scan
↓
nmap
↓
Advanced Network Troubleshooting
```

This sequence takes a student from **Linux network configuration → ARP/neighbor concepts → device discovery → comprehensive network scanning**.

4. fping

1. What is `fping` ?

`fping` (Fast Network Host Discovery) is a Linux command-line tool used to **check the availability of multiple network hosts simultaneously**. It is based on ICMP and is particularly useful for quickly identifying which IP addresses are responding on a network.

Unlike **`ping`**, which normally tests one host at a time, `fping` can process **multiple hosts or an entire IP range**.

---

2. Install `fping`

Ubuntu / Debian

```
sudo apt update
sudo apt install fping
```

Check the version:

```
fping --version
```

Display help:

```
fping --help
```

---

3. Test a Single Host

The simplest test is:

```
fping 192.168.1.1
```

Example:

```
192.168.1.1 is alive
```

If the host does not respond:

```
192.168.1.50 is unreachable
```

This provides a quick way to determine whether a particular host is reachable.

4. Test Multiple Hosts

You can specify several IP addresses:

```
fping 192.168.1.1 192.168.1.10 192.168.1.20
```

Example:

```
192.168.1.1 is alive
192.168.1.10 is alive
192.168.1.20 is unreachable
```

This is much more convenient than executing `ping` separately for every host.

---

5. Scan an Entire Subnet

One of the most useful features is subnet discovery:

```
fping -g 192.168.1.0/24
```

`-g` means **generate** a list of addresses from the specified range.

Example:

```
192.168.1.1 is alive
192.168.1.5 is alive
192.168.1.12 is alive
192.168.1.25 is alive
```

Only responding hosts are normally displayed in this form.

---

6. Scan an IP Address Range

You can also specify a starting and ending address:

```
fping -g 192.168.1.1 192.168.1.50
```

This checks:

```
192.168.1.1
192.168.1.2
...
192.168.1.50
```

This is useful when you don't want to scan the entire subnet.

7. Display Unreachable Hosts

Use:

```
fping -g -u 192.168.1.0/24
```

The `-u` option displays **unreachable hosts**.

Example:

```
192.168.1.2
192.168.1.3
192.168.1.7
192.168.1.18
```

This can be useful for identifying unused or unavailable addresses.

---

8. Display Only Alive Hosts

Use:

```
fping -g -a 192.168.1.0/24
```

The `-a` option displays **alive/responding hosts**.

Example:

```
192.168.1.1
192.168.1.5
192.168.1.12
192.168.1.25
```

This is particularly useful when creating a simple **LAN host-discovery exercise**.

---

9. Count the Number of Responses

You can use:

```
fping -g -a 192.168.1.0/24 | wc -l
```

For example:

```
12
```

This means that 12 hosts responded to the test.

10. Read IP Addresses from a File

Create a file:

```
nano hosts.txt
```

Example:

```
192.168.1.1
192.168.1.10
192.168.1.20
192.168.1.30
```

Then:

```
fping < hosts.txt
```

This is useful for checking a predefined list of servers.

---

11. Test Host Availability Repeatedly

You can use:

```
fping -c 4 192.168.1.1
```

`-c 4` sends four probes.

Example:

```
192.168.1.1 : 4 : 4 : 0 : 0.32 : 0.45 : 0.38
```

The output provides information about the number of probes and response times.

---

12. Set a Timeout

You can specify a timeout using:

```
fping -t 500 192.168.1.1
```

Here `500` represents the timeout in milliseconds.

This is useful when testing networks where responses may be slow.

13. Test a Network from a File

You can combine file input with the alive-only option:

```
fping -a < hosts.txt
```

This gives you a simple list of hosts that responded.

For example:

```
192.168.1.1
192.168.1.10
192.168.1.30
```

---

14. Understanding `fping` Output

Consider:

```
192.168.1.1 is alive
```

This tells us:

```
IP Address        → 192.168.1.1
Availability      → Alive
Protocol          → ICMP
Purpose           → Host reachability
```

However, **`fping`** does **not** tell you which ports or services are running.

For example:

```
fping → Is the host responding?

nmap  → Which ports/services are available?
```

---

15. Exercise 1 - Discover Your LAN

First determine your IP address:

```
ip -br addr
```

Suppose you have:

```
192.168.1.25/24
```

Run:

```
fping -g -a 192.168.1.0/24
```

Record all responding hosts.

Create a table:

IP Address	Responding
192.168.1.1	Yes
192.168.1.5	Yes
192.168.1.10	No
192.168.1.25	Yes

16. Exercise 2 - Compare `ping` and `fping`

Test one host:

```
ping -c 4 192.168.1.1
```

Then:

```
fping -c 4 192.168.1.1
```

Observe the difference in output.

The important concept is:

```
ping
  ↓
Normally one host at a time

fping
  ↓
Designed for multiple hosts
```

---

17. Exercise 3 - Compare with `arp-scan`

Run:

```
sudo arp-scan --localnet
```

Then:

```
fping -g -a 192.168.1.0/24
```

Compare the results.

You will notice an important distinction:

`arp-scan`

`IP → MAC → Vendor`

`fping`

`IP → ICMP response`

Thus, they provide complementary information.

---

18. Exercise 4 - Check a List of Servers

Create:

```
nano servers.txt
```

Add:

```
192.168.1.10
192.168.1.20
192.168.1.30
192.168.1.40
```

Then:

```
fping < servers.txt
```

This provides a simple way for a system administrator to check whether several servers are reachable.

---

19. Useful `fping` Commands

Command	Purpose
`fping 192.168.1.1`	Test one host
`fping IP1 IP2 IP3`	Test multiple hosts
`fping -g 192.168.1.0/24`	Generate and test subnet addresses
`fping -g -a 192.168.1.0/24`	Display responding hosts
`fping -g -u 192.168.1.0/24`	Display unreachable hosts
`fping -c 4 192.168.1.1`	Send 4 probes
`fping < hosts.txt`	Test hosts from a file
`fping --help`	Display help

20. `fping` (vs) `arp-scan` (vs) `netdiscover` (vs) `nmap`

Feature	fping	arp-scan	netdiscover	nmap
Host discovery				
ICMP discovery	**Yes**	No	No	Yes
ARP discovery	No	**Yes**	**Yes**	Yes
MAC address	No	**Yes**	**Yes**	Yes
Vendor	No	**Yes**	**Yes**	Yes
Port scanning	No	No	No	**Yes**
Service detection	No	No	No	**Yes**
Fast multiple-host testing	**Excellent**	Excellent	Good	Excellent
Best learning area	Layer 3/ICMP	Layer 2/ARP	Layer 2/ARP	Comprehensive

Recommended Learning Sequence

```
ip
↓
ip neigh
↓
ping
↓
fping
↓
netdiscover
↓
arp-scan
↓
nmap
```

This gives students a logical progression from **basic Linux networking → ICMP host testing → fast network discovery → ARP/LAN discovery → comprehensive network scanning**.

Key concept:

`fping` is not simply a faster **`ping`**; its real educational value is teaching students how to perform **efficient ICMP-based reachability testing across multiple hosts and networks**.

5. masscan

1. What is `masscan` ?

`masscan` (High-Speed Network Port Discovery) is a Linux **high-speed TCP port scanner** designed to discover open ports across large numbers of hosts very quickly.

While tools such as **`fping`, `netdiscover`,** and **`arp-scan`** primarily answer **"Which hosts are active?"**, **`masscan`** answers:

"Which TCP ports are open on these hosts?"

It is therefore a useful next step after basic host discovery.

---

2. Install `masscan`

Ubuntu / Debian

```
sudo apt update
sudo apt install masscan
```

Check the installation:

```
masscan --version
```

Display help:

```
masscan --help
```

---

3. Scan One Host for a Specific Port

For example, to check TCP port 22:

```
sudo masscan 192.168.1.10 -p22
```

Example:

```
Discovered open port 22/tcp on 192.168.1.10
```

This indicates that TCP port 22 is responding as open.

4. Scan Several Ports

You can specify multiple ports:

```
sudo masscan 192.168.1.10 -p22,80,443
```

This checks:

```
22   → SSH
80   → HTTP
443  → HTTPS
```

Example:

```
Discovered open port 22/tcp on 192.168.1.10
Discovered open port 80/tcp on 192.168.1.10
```

---

5. Scan a Port Range

For example:

```
sudo masscan 192.168.1.10 -p1-1000
```

This scans TCP ports **1 through 1000**.

For a small lab network, you can also scan:

```
sudo masscan 192.168.1.0/24 -p80,443
```

This checks HTTP/HTTPS ports across the subnet.

Only scan networks that you own or are explicitly authorized to test.

---

6. Scan All TCP Ports

Masscan can scan the full TCP port range:

```
sudo masscan 192.168.1.10 -p1-65535
```

This is considerably more traffic than scanning a few selected ports, so it should be used carefully.

For a learning lab, start with:

```
sudo masscan 192.168.1.10 -p22,80,443
```

before experimenting with larger ranges.

7. Scan a Local Network

For example:

```
sudo masscan 192.168.1.0/24 -p22,80,443
```

Possible output:

```
Discovered open port 22/tcp on 192.168.1.10
Discovered open port 80/tcp on 192.168.1.15
Discovered open port 443/tcp on 192.168.1.20
```

This gives you both:

```
IP Address → Open TCP Port
```

---

8. Control the Scanning Rate

One of the most important `masscan` options is:

```
--rate
```

For example:

```
sudo masscan 192.168.1.0/24 -p80 --rate 100
```

This limits the scan to approximately **100 packets per second**.

For a classroom/lab environment, using a conservative rate is preferable.

For example:

```
sudo masscan 192.168.1.0/24 -p22,80,443 --rate 100
```

Why rate control matters

Masscan is intentionally designed for speed. An unrestricted or unnecessarily high-rate scan can:

- Generate significant network traffic
- Trigger IDS/IPS alerts
- Overload poorly designed devices
- Cause network administrators to treat the activity as suspicious

Therefore, **rate control is an important part of responsible masscan usage**.

9. Specify the Network Interface

First identify interfaces:

```
ip -br link
```

Example:

```
enp3s0   UP
wlp2s0   UP
```

Masscan can be instructed to use a particular interface:

```
sudo masscan 192.168.1.0/24 -p80 --interface enp3s0
```

This is useful on systems with multiple network interfaces.

---

10. Specify the Source IP

In controlled laboratory environments, you may need to specify the source address:

```
sudo masscan 192.168.1.0/24 -p80 --source-ip 192.168.1.25
```

The source IP should normally belong to the interface being used.

For beginners, this option is usually unnecessary.

---

11. Save Results to a File

Masscan can save scan results.

For example:

```
sudo masscan 192.168.1.0/24 -p22,80,443 -oL results.txt
```

The `-oL` option produces a simple list format.

You can then examine it:

```
cat results.txt
```

Other output formats are available, including XML and JSON.

12. XML Output

For integration with other tools:

```
sudo masscan 192.168.1.0/24 -p80,443 -oX results.xml
```

This is useful when the scan results need to be processed by another program.

---

13. JSON Output

You can also produce JSON:

```
sudo masscan 192.168.1.0/24 -p80,443 -oJ results.json
```

This is particularly useful for automation and scripting.

---

14. Use a Target File

Create a list:

```
nano targets.txt
```

For example:

```
192.168.1.10
192.168.1.20
192.168.1.30
```

Masscan can use a target file:

```
sudo masscan -iL targets.txt -p22,80,443 --rate 100
```

This is useful when an administrator already has a list of systems to test.

---

15. Exercise 1 - Scan Your Linux Server

First identify your IP:

```
ip -br addr
```

Suppose the server is:

```
192.168.1.10
```

Scan common ports:

```
sudo masscan 192.168.1.10 -p22,80,443 --rate 100
```

Record the results:

Host	Port	Status
192.168.1.10	22	Open
192.168.1.10	80	Open/Closed
192.168.1.10	443	Open/Closed

16. Exercise 2 - Discover Web Servers

On your authorized lab network:

```
sudo masscan 192.168.1.0/24 -p80,443 --rate 100
```

The output might identify:

```
192.168.1.5   80/tcp
192.168.1.15  443/tcp
192.168.1.25  80/tcp
```

You have now identified potential web servers.

---

17. Exercise 3 - Find SSH Servers

Run:

```
sudo masscan 192.168.1.0/24 -p22 --rate 100
```

Example:

```
Discovered open port 22/tcp on 192.168.1.10
Discovered open port 22/tcp on 192.168.1.20
```

You can then investigate those hosts with an appropriate administration tool.

18. Masscan and Nmap Together

A particularly useful workflow is:

```
Host Discovery
      ↓
fping / arp-scan / netdiscover
      ↓
Port Discovery
      ↓
masscan
      ↓
Service & Version Detection
      ↓
nmap
```

For example:

Step 1 - Find active hosts

```
fping -g -a 192.168.1.0/24
```

Step 2 - Find open web/SSH ports

```
sudo masscan 192.168.1.0/24 -p22,80,443 --rate 100
```

Step 3 - Investigate a discovered host

```
nmap -sV 192.168.1.10
```

This demonstrates the complementary roles of the three tools.

19. `masscan` (vs) `fping` (vs) `arp-scan` (vs) `nmap`

Feature	fping	arp-scan	masscan	nmap
Host discovery				
ICMP discovery	**Yes**	No	No	Yes
ARP discovery	No	**Yes**	No	Yes
MAC address	No	**Yes**	No	Yes
TCP port discovery	No	No	**Excellent**	**Excellent**
Very high-speed scanning	No	No	**Excellent**	Good
Service detection	No	No	Limited	**Yes**
Version detection	No	No	No	**Yes**
OS detection	No	No	No	**Yes**
Best use	Host reachability	LAN discovery	Fast port discovery	Detailed analysis

20. Important Limitation of `masscan`

Masscan is primarily a **port discovery tool**.

If it reports:

```
192.168.1.10 → 80/tcp open
```

that does not tell you:

- Which web server is running
- Its exact version
- What operating system is running
- Which application is behind the port

For that, follow up with:

```
nmap -sV 192.168.1.10
```

So the conceptual difference is:

`masscan`

```
      ↓
"Is TCP port 80 open?"
```

`nmap`

```
      ↓
"What service is running on TCP port 80?"
```

21. Useful `masscan` Commands

Command	Purpose
`masscan --version`	Display version
`masscan --help`	Display help
`sudo masscan HOST -p22`	Scan SSH
`sudo masscan HOST -p80,443`	Scan web ports
`sudo masscan HOST -p1-1000`	Scan ports 1-1000
`sudo masscan NETWORK -p22,80,443`	Scan selected ports on a network
`sudo masscan NETWORK -p80 --rate 100`	Controlled-rate scan
`sudo masscan -iL targets.txt -p22`	Scan targets from a file
`sudo masscan ... -oL results.txt`	Save list output
`sudo masscan ... -oX results.xml`	Save XML output
`sudo masscan ... -oJ results.json`	Save JSON output

22. Recommended Learning Sequence

Position `masscan` **after host-discovery tools and before advanced Nmap work**:

```
Linux Network Configuration
          ↓
        ping
          ↓
        fping
          ↓
    netdiscover
          ↓
      arp-scan
          ↓
       masscan
          ↓
        nmap
          ↓
Service / Version / OS Analysis
```

Key learning point

`masscan` teaches students the important distinction between discovering hosts and discovering services: **`fping`**, **`netdiscover`**, and **`arp-scan`** help find devices, while **`masscan`** rapidly identifies open TCP ports, and **`nmap`** provides deeper service-level analysis.

6. avahi-browse

1. What is `avahi-browse` ?

`avahi-browse` (Service Discovery on Linux Networks) is a Linux command-line tool for discovering **network services advertised using mDNS / DNS-SD**.

It is useful for finding devices and services such as:

- Network printers
- SSH services
- Web services
- File-sharing services
- Media servers
- Network devices
- Apple Bonjour services
- Other Zeroconf-enabled applications

Unlike **`arp-scan`** or **`netdiscover`**, which primarily discover **IP/MAC addresses**, **`avahi-browse`** discovers **services** being advertised on the local network.

---

2. Install Avahi

Ubuntu / Debian

```
sudo apt update
sudo apt install avahi-daemon avahi-utils
```

Check the installation:

```
avahi-browse --help
```

Check the Avahi service:

```
systemctl status avahi-daemon
```

If it is not running:

```
sudo systemctl start avahi-daemon
```

Enable it at boot:

```
sudo systemctl enable avahi-daemon
```

3. Basic Service Discovery

Run:

```
avahi-browse -a
```

`-a` means **browse all available service types**.

Example output:

```
+ wlp2s0 IPv4 My Laptop            _ssh._tcp       local
+ wlp2s0 IPv4 HP Printer           _ipp._tcp       local
+ wlp2s0 IPv4 Media Server         _http._tcp      local
```

The important information is:

```
Interface → Device/Service → Service Type → Domain
```

---

4. Make the Output Easier to Read

Use:

```
avahi-browse -a -r
```

The `-r` option resolves the discovered services and provides additional information.

Example:

```
= wlp2s0 IPv4 My Laptop
   _ssh._tcp
   hostname = [laptop.local]
   address = [192.168.1.25]
   port = [22]
```

This is particularly useful for students because it connects:

```
Service
  ↓
Hostname
  ↓
IP Address
  ↓
Port
```

5. Discover SSH Services

To look specifically for SSH services:

```
avahi-browse _ssh._tcp
```

For resolved information:

```
avahi-browse -r _ssh._tcp
```

Example:

```
= wlp2s0 IPv4 Linux Server
   _ssh._tcp
   hostname = [server.local]
   address = [192.168.1.20]
   port = [22]
```

This tells us that the device is advertising an SSH service.

---

6. Discover HTTP Services

Search for HTTP services:

```
avahi-browse -r _http._tcp
```

Possible result:

```
= wlp2s0 IPv4 Web Server
   _http._tcp
   hostname = [webserver.local]
   address = [192.168.1.30]
   port = [80]
```

You can then test the service:

```
curl http://192.168.1.30/
```

---

7. Discover Network Printers

Many network printers advertise services using DNS-SD.

For example:

```
avahi-browse -a -r
```

Look for service types such as:

```
_ipp._tcp
_printer._tcp
_ipps._tcp
```

For example:

```
HP Printer
   _ipp._tcp
   address = [192.168.1.40]
   port = [631]
```

This demonstrates how Linux can discover a printer without manually entering its IP address.

---

8. Discover AirPrint-Compatible Services

You may encounter:

```
avahi-browse -r _ipp._tcp
```

or:

```
avahi-browse -r _ipps._tcp
```

These are commonly associated with IPP printing.

---

9. Browse All Services and Resolve Them

A very useful command for practical work is:

```
avahi-browse -a -r
```

It combines:

```
-a → Browse all services
-r → Resolve service information
```

This is probably the best starting command for students learning Avahi.

10. Use a One-Shot Scan

By default, browsing can continue listening for service announcements.

Use:

```
avahi-browse -a -r -t
```

`-t` terminates after the initial browsing operation is complete.

This is convenient for scripts and classroom exercises.

---

11. Display Addresses Numerically

You can use:

```
avahi-browse -a -r -n
```

The `-n` option avoids unnecessary hostname resolution.

This can be useful when you specifically want to work with IP addresses.

---

12. Browse a Specific Network Interface

First identify your interfaces:

```
ip -br link
```

Example:

```
enp3s0   UP
wlp2s0   UP
```

You can select an interface:

```
avahi-browse -a -i wlp2s0
```

For Ethernet:

```
avahi-browse -a -i enp3s0
```

This is useful when a computer has both Ethernet and Wi-Fi connections.

13. Understand mDNS

Avahi is primarily an implementation of **mDNS (Multicast DNS)** and **DNS-SD (DNS Service Discovery)**.

Instead of relying exclusively on a conventional DNS server:

```
Computer → DNS Server → IP address
```

mDNS allows devices on a local network to advertise and resolve names such as:

```
laptop.local
server.local
printer.local
```

DNS-SD goes one step further by advertising services:

```
Server
  ├── SSH
  ├── HTTP
  └── Other services
```

---

14. Test `.local` Name Resolution

If another machine advertises:

```
server.local
```

try:

```
ping server.local
```

or:

```
ssh user@server.local
```

You can also use:

```
getent hosts server.local
```

This demonstrates the relationship between **Avahi, mDNS and local hostname resolution**.

15. Exercise 1 - Discover Local Services

Run:

```
avahi-browse -a -r -t
```

Create a table:

Service	Hostname	IP Address	Port
SSH	server.local	192.168.1.20	22
HTTP	web.local	192.168.1.30	80
IPP	printer.local	192.168.1.40	631

This gives students practical experience with **service discovery**.

---

16. Exercise 2 - Discover SSH

Run:

```
avahi-browse -r _ssh._tcp -t
```

If a server appears, note:

```
Hostname
IP address
Port
```

Then, on a system you are authorized to administer:

```
ssh user@server.local
```

---

17. Exercise 3 - Discover Printers

Run:

```
avahi-browse -a -r -t
```

Search the output for:

```
_ipp._tcp
_ipps._tcp
_printer._tcp
```

Record the printer's:

```
Hostname
IP address
Port
```

This is a useful example of **automatic network-service discovery**.

---

18. Exercise 4 - Compare with `arp-scan`

First:

```
sudo arp-scan --localnet
```

You may get:

```
192.168.1.20    AA:BB:CC:DD:EE:FF    Dell
192.168.1.30    11:22:33:44:55:66    HP
192.168.1.40    77:88:99:AA:BB:CC    Epson
```

Then:

```
avahi-browse -a -r -t
```

You might get:

```
server.local
_ssh._tcp
192.168.1.20
22

printer.local
_ipp._tcp
192.168.1.40
631
```

This demonstrates an important difference:

```
arp-scan
   ↓
"Which devices are on my LAN?"

avahi-browse
   ↓
"What services are these devices advertising?"
```

19. Exercise 5 - Compare with `nmap`

Suppose Avahi discovers:

```
server.local
192.168.1.20
_ssh._tcp
port 22
```

You can investigate the authorized host further with:

```
nmap -sV 192.168.1.20
```

Conceptually:

```
avahi-browse
      ↓
Service discovery
      ↓
SSH on port 22
      ↓
nmap
      ↓
Service/version investigation
```

---

20. Important Limitation

`avahi-browse` is **not a general-purpose network scanner**.

It depends on services being advertised through **mDNS/DNS-SD**.

A device may be physically connected to the network but advertise no Avahi/mDNS services.

Therefore:

```
No Avahi result
       ≠
Device does not exist
```

For comprehensive LAN host discovery, use tools such as:

```
arp-scan
netdiscover
fping
nmap
```

21. Useful `avahi-browse` Commands

Command	Purpose
`avahi-browse -a`	Browse all service types
`avahi-browse -a -r`	Browse and resolve services
`avahi-browse -a -r -t`	One-shot service discovery
`avahi-browse _ssh._tcp`	Find SSH services
`avahi-browse -r _ssh._tcp`	Resolve SSH services
`avahi-browse -r _http._tcp`	Find HTTP services
`avahi-browse -r _ipp._tcp`	Find IPP printers
`avahi-browse -a -i wlp2s0`	Browse through Wi-Fi
`avahi-browse -a -i enp3s0`	Browse through Ethernet
`avahi-browse --help`	Display help

22. `avahi-browse` (vs) Other Discovery Tools

Feature	fping	arp-scan	netdiscover	masscan	avahi-browse	nmap
Host discovery						
ICMP	**Yes**	No	No	No	No	Yes
ARP	No	**Yes**	**Yes**	No	No	Yes
MAC discovery	No	**Yes**	**Yes**	No	No	Yes
Service discovery	No	No	No	Limited	**Excellent**	**Excellent**
Port scanning	No	No	No	**Excellent**	No	**Excellent**
.local discovery	No	No	No	No	**Yes**	Limited
Printer discovery	No	No	No	No	**Yes**	Limited
Best learning area	ICMP	ARP/LAN	ARP/ LAN	TCP ports	**MDNS/ DNS-SD**	Comprehensive

23. Recommended Learning Sequence

```
Linux Network Configuration
          ↓
         ip
          ↓
        ip neigh
          ↓
         ping
          ↓
         fping
          ↓
     netdiscover
          ↓
       arp-scan
          ↓
    avahi-browse
          ↓
       masscan
          ↓
         nmap
```

The progression teaches increasingly higher-level concepts:

IP configuration → Neighbor/ARP → ICMP → LAN discovery → mDNS/DNS-SD service discovery → TCP port discovery → detailed service analysis.

Key learning point

`avahi-browse` teaches students that network discovery is not limited to finding IP addresses-modern Linux networks can also automatically advertise and discover the services running on those devices.

7. lldpctl

1. What is `lldpctl`?

`lldpctl` (Network Neighbor Discovery) is a Linux command-line tool used to display information about **network devices directly connected to a Linux system**.

It works with **LLDP (Link Layer Discovery Protocol)** and can reveal information such as:

- Connected switch
- Switch port
- Device hostname
- Management IP address
- VLAN information
- Link speed
- Duplex mode
- Neighbor device details

Unlike **`arp-scan`**, which discovers hosts across a local network, **LLDP discovers the device directly connected to your network interface**.

---

2. Install LLDP

On Ubuntu/Debian:

```
sudo apt update
sudo apt install lldpd
```

Check the installation:

```
lldpctl --version
```

Display help:

```
lldpctl --help
```

3. Start the LLDP Service

Check the service:

```
systemctl status lldpd
```

If it is not running:

```
sudo systemctl start lldpd
```

Enable it at boot:

```
sudo systemctl enable lldpd
```

You can verify:

```
systemctl is-active lldpd
```

Expected result:

```
active
```

---

4. Display LLDP Neighbor Information

The simplest command is:

```
sudo lldpctl
```

Example:

```
-------------------------------------------------------------------
Interface:    enp3s0, via: LLDP, RID: 1, Time: 0 day, 00:02:15
  Chassis:
    ChassisID:    mac 00:11:22:33:44:55
    SysName:      switch01
    SysDescr:     Cisco IOS Software
    MgmtIP:       192.168.1.2
    Capability:   Bridge, Router
  Port:
    PortID:       ifname GigabitEthernet1/0/10
    PortDescr:    Server-Port
    TTL:          120
```

This is extremely useful for network troubleshooting.

From this output, you can determine:

```
Linux Interface
       ↓
Connected Switch
       ↓
Switch Port
       ↓
Switch Management IP
```

---

5. Understand the Output

Important fields include:

Field	Meaning
`Interface`	Linux network interface
`ChassisID`	Unique identifier of neighbor
`SysName`	Neighbor device name
`SysDescr`	Device description
`MgmtIP`	Management IP address
`PortID`	Port on the neighboring device
`PortDescr`	Description of that port
`Capability`	Device capabilities
`TTL`	LLDP information lifetime

For example:

```
SysName: switch01
```

means the connected device identifies itself as `switch01`.

And:

```
PortID: GigabitEthernet1/0/10
```

means your Linux system is connected to port **10** of that switch.

6. Find Which Switch Port Your Linux Machine Uses

This is one of the most useful applications of **`lldpctl`**.

Run:

```
sudo lldpctl
```

Look for:

```
PortID: ifname GigabitEthernet1/0/10
```

You can then tell the network administrator:

"This Linux server is physically connected to switch01, port GigabitEthernet1/0/10."

This can save considerable time when troubleshooting large networks.

---

7. Display Information for a Specific Interface

First identify your interfaces:

```
ip -br link
```

Example:

```
lo       UNKNOWN
enp3s0   UP
wlp2s0   UP
```

Then:

```
sudo lldpctl enp3s0
```

This focuses on the selected interface.

8. Check All LLDP Neighbors

Use:

```
sudo lldpctl show neighbors
```

Depending on the `lldpd` version, the output may be presented in a compact form.

For a more detailed view:

```
sudo lldpctl show neighbors details
```
This is useful when the Linux system has multiple Ethernet interfaces.

9. Check LLDP on a Specific Interface

For example:

```
sudo lldpctl show neighbors ports enp3s0 details
```

This allows you to inspect the LLDP information received through `enp3s0`.

---

10. Understand LLDP

LLDP stands for:

Link Layer Discovery Protocol

It is standardized as **IEEE 802.1AB**.

The basic concept is:

```
Linux Server
     |
     | LLDP
     |
     v
Network Switch
```

The Linux system and switch exchange information about themselves.

For example:

```
Linux
 ├── Hostname
 ├── Interface
 └── Capabilities

       ↕ LLDP

Switch
 ├── Switch name
 ├── Switch port
 ├── VLAN information
 └── Management address
```

11. LLDP vs ARP

This distinction is important for students.

ARP

ARP answers:

"Who has this IP address?"

Example:

```
192.168.1.1 → MAC address
```

LLDP

LLDP answers:

"What network device am I physically connected to?"

Example:

```
enp3s0
   ↓
switch01
   ↓
GigabitEthernet1/0/10
```

Therefore:

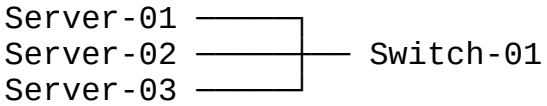
```
ARP
↓
IP ↔ MAC

LLDP
↓
Interface ↔ Neighbor Device ↔ Switch Port
```

---

12. LLDP and Network Topology

Suppose you have three Linux servers:



Running:

```
sudo lldpctl
```

on each server can tell you which switch and switch port each server is connected to.

You can build a basic topology map:

```
Server-01 → Switch-01 → Port 1
Server-02 → Switch-01 → Port 2
Server-03 → Switch-01 → Port 3
```

This makes `lldpctl` particularly valuable for **network documentation**.

---

13. Exercise 1 - Discover the Connected Switch

Run:

```
sudo lldpctl
```

Record:

Information	Result
Linux interface	enp3s0
Switch name	switch01
Switch MAC	...
Switch IP	...
Switch port	...
Port description	...

This is a good introductory LLDP exercise.

---

14. Exercise 2 - Identify the Physical Port

Run:

```
sudo lldpctl show neighbors details
```

Find:

```
PortID
```

For example:

```
PortID: GigabitEthernet1/0/10
```

Then physically trace the network connection if you have access to the switch.

Students can verify that the software-reported port corresponds to the actual network connection.

---

15. Exercise 3 - Multiple Network Interfaces

Check interfaces:

```
ip -br addr
```

Then:

```
sudo lldpctl
```

Determine which interfaces have LLDP neighbors.

For example:

```
enp3s0 → switch01 → port 10
enp4s0 → switch02 → port 24
```

This is particularly useful on **servers with multiple Ethernet connections**.

---

16. Exercise 4 - Compare LLDP with `arp-scan`

Run:

```
sudo arp-scan --localnet
```

Then:

```
sudo lldpctl
```

You will see two very different types of information.

`arp-scan`

```
192.168.1.1
AA:BB:CC:DD:EE:FF
Router vendor
```

`lldpctl`

```
switch01
GigabitEthernet1/0/10
192.168.1.2
```

Therefore:

```
arp-scan
   ↓
Discover devices on the LAN

lldpctl
   ↓
Discover directly connected network infrastructure
```

---

17. Exercise 5 - Compare with `nmap`

`nmap` can discover hosts and services:

```
sudo nmap -sV 192.168.1.0/24
```

But **`lldpctl`** provides information that a normal IP scan cannot easily provide:

```
Physical neighbor
       ↓
Switch
       ↓
Exact switch port
```

This makes LLDP particularly valuable for **physical network troubleshooting**.

---

18. Useful `lldpctl` Commands

Command	Purpose
`sudo lldpctl`	Display LLDP information
`sudo lldpctl enp3s0`	Show information for an interface
`sudo lldpctl show neighbors`	Show LLDP neighbors
`sudo lldpctl show neighbors details`	Detailed neighbor information
`sudo lldpctl show neighbors ports enp3s0 details`	Detailed information for an interface
`lldpctl --help`	Display help
`lldpctl --version`	Display version

19. `lldpctl` (vs) Other Network Discovery Tools

Feature	fping	arp-scan	netdiscover	avahi-browse	lldpctl	masscan	nmap
Host discovery							
ICMP	Yes	No	No	No	No	No	Yes
ARP	No	**Yes**	**Yes**	No	No	No	Yes
MAC discovery	No	**Yes**	**Yes**	No	Limited	No	Yes
Service discovery	No	No	No	**Yes**	No	Limited	**Yes**
TCP port scanning	No	No	No	No	No	**Yes**	**Yes**
Switch discovery	No	No	No	No	**Yes**	No	Limited
Switch-port identification	No	No	No	No	**Yes**	No	No
Network topology	Limited	Limited	Limited	Service-level	**Excellent**	Limited	Good
Best learning area	ICMP	ARP	ARP	mDNS	**LLDP/ physical topology**	TCP ports	Comprehensive

20. Important Limitation

`lldpctl` can only provide useful neighbor information when:

1. The network interface supports LLDP.
2. The LLDP service is running.
3. The neighboring network device supports and transmits LLDP.
4. LLDP is not blocked or disabled on the network.

Therefore, if:

```
sudo lldpctl
```

shows no neighbor, it does **not necessarily mean that no switch is connected**.

The switch may simply have LLDP disabled.

21. Recommended Networking Sequence

With the discovery tools, organize the practical learning path as:

```
Linux Network Configuration
          ↓
         ip
          ↓
       ip neigh
          ↓
        ping
          ↓
        fping
          ↓
    netdiscover
          ↓
      arp-scan
          ↓
    avahi-browse
          ↓
       lldpctl
          ↓
       masscan
          ↓
        nmap
```

This progression covers several different layers:

```
IP configuration
        ↓
ARP / Neighbor Discovery
        ↓
ICMP Host Discovery
        ↓
LAN Device Discovery
        ↓
mDNS Service Discovery
        ↓
LLDP Physical Neighbor Discovery
        ↓
TCP Port Discovery
        ↓
Detailed Network Analysis
```

Key learning point

`lldpctl` teaches Linux students an important networking concept that IP-based discovery tools cannot provide: identifying the directly connected network device and the exact switch port through which the Linux system is connected.

8. traceroute

1. What is `traceroute`?

`traceroute` (Network Path Discovery) is a Linux command-line tool used to **discover the network path between your computer and a destination host**.

It helps students understand:

- Routers between source and destination
- Number of network hops
- IP addresses of intermediate routers
- Network latency at each hop
- Where connectivity problems occur

The basic concept is:

```
Your Linux System
       ↓
   Router 1
       ↓
   Router 2
       ↓
   Router 3
       ↓
   Destination
```

---

2. Install `traceroute`

Ubuntu / Debian

```
sudo apt update
sudo apt install traceroute
```

Check the installation:

```
traceroute --version
```

Display help:

```
traceroute --help
```

3. Basic Traceroute

The simplest command is:

```
traceroute google.com
```

Example:

```
traceroute to google.com (142.250.x.x), 30 hops max

 1  192.168.1.1       1.234 ms
 2  10.10.0.1         5.321 ms
 3  172.16.10.1      12.456 ms
 4  203.x.x.x         18.732 ms
 5  ...               ...
```

Each numbered line represents a **network hop**.

---

4. Trace a Destination by IP Address

You don't have to use a hostname.

For example:

```
traceroute 8.8.8.8
```

This traces the path to Google's public DNS server.

You can also use:

```
traceroute 1.1.1.1
```

---

5. Understand the Output

Consider:

```
 1  192.168.1.1    1.2 ms
 2  10.20.0.1      5.4 ms
 3  172.16.1.1    12.7 ms
 4  203.x.x.x      18.2 ms
```

The columns represent:

```
Hop Number
    ↓
Router/Device IP
    ↓
Round-trip response time
```

For example:

```
2  10.20.0.1  5.4 ms
```

means the second hop responded with approximately **5.4 milliseconds** of round-trip time.

---

6. Why Does Traceroute Find Routers?

Traceroute takes advantage of the **TTL (Time To Live)** field in IP packets.

For example:

```
TTL = 1
```

The first router decreases the TTL to zero and generates a response.

Then traceroute sends:

```
TTL = 2
```

The second router responds.

Then:

```
TTL = 3
```

The third router responds.

This continues until the destination is reached.

Conceptually:

```
TTL 1 → Router 1
TTL 2 → Router 2
TTL 3 → Router 3
TTL 4 → Destination
```

This is the fundamental networking concept behind traceroute.

7. Use Numeric Output

By default, traceroute may attempt reverse DNS lookups.

To display IP addresses without hostname resolution:

```
traceroute -n 8.8.8.8
```

This can make the output faster and easier to study.

Example:

```
 1  192.168.1.1
 2  10.10.0.1
 3  172.16.1.1
 4  8.8.8.8
```

---

8. Limit the Number of Hops

You can specify the maximum number of hops:

```
traceroute -m 10 8.8.8.8
```

Here:

```
-m 10
```

means a maximum of 10 hops.

The default maximum is commonly 30.

---

9. Change the Number of Probes

Traceroute normally sends multiple probes for each hop.

You can specify the number:

```
traceroute -q 1 8.8.8.8
```

This sends one probe per hop.

This can produce a shorter output and is useful for classroom demonstrations.

10. Increase the Timeout

If a network is slow to respond:

```
traceroute -w 5 8.8.8.8
```

`-w 5` specifies a 5-second wait for a response.

---

11. Use ICMP Traceroute

Linux `traceroute` normally uses UDP probes by default.

You can use ICMP:

```
sudo traceroute -I 8.8.8.8
```

Here:

```
-I → Use ICMP Echo requests
```

This can be useful when intermediate devices or firewalls treat UDP and ICMP differently.

---

12. Use TCP Traceroute

You can also use TCP probes:

```
sudo traceroute -T -p 443 google.com
```

Here:

```
-T   → TCP
-p 443 → TCP port 443
```

This can be useful when troubleshooting paths to services that use TCP, particularly HTTPS.

---

13. Trace a Specific Port

For example, HTTPS:

```
sudo traceroute -T -p 443 example.com
```

Or SSH:

```
sudo traceroute -T -p 22 192.168.1.10
```

This can help demonstrate that **routing and service connectivity are related but different concepts**.

14. IPv4 Traceroute

Force IPv4:

```
traceroute -4 8.8.8.8
```

This is useful on systems configured for both IPv4 and IPv6.

---

15. IPv6 Traceroute

For an IPv6 destination:

```
traceroute -6 2001:4860:4860::8888
```

This introduces students to IPv6 routing.

---

16. Understanding * * *

You may see:

```
1  192.168.1.1   1.2 ms
2  * * *
3  172.16.1.1   15.4 ms
```

* means that traceroute did not receive a response to that particular probe within the timeout.

It **does not automatically mean that the router or network is down**.

Possible reasons include:

- Router configured not to respond
- Firewall filtering traceroute packets
- ICMP responses being rate-limited
- Network congestion
- Different routing behavior

This is an important lesson for students.

17. Exercise 1 - Trace Your Internet Path

Run:

```
traceroute -n 8.8.8.8
```

Create a table:

Hop	IP Address	Response Time
1	192.168.1.1	1 ms
2	...	...
3	...	...
4	...	...

Identify:

- Your local gateway
- ISP routers
- Intermediate networks
- Destination

---

18. Exercise 2 - Compare Two Destinations

Run:

```
traceroute -n 8.8.8.8
```

Then:

```
traceroute -n 1.1.1.1
```

Compare:

```
Number of hops
Intermediate routers
Response times
```

Students can observe that two destinations on the Internet can follow **different network paths**.

19. Exercise 3 - Find Your Default Gateway

Run:

```
ip route
```

Example:

```
default via 192.168.1.1 dev wlp2s0
```

Then:

```
traceroute -n 8.8.8.8
```

The first hop should often correspond to the local gateway:

```
1  192.168.1.1
```

This connects basic Linux network configuration with real-world routing.

---

20. Exercise 4 - Compare ICMP and TCP

Run:

```
sudo traceroute -I 8.8.8.8
```

Then:

```
sudo traceroute -T -p 443 google.com
```

Compare the paths and responses.

This demonstrates that different probe types can produce different results because of **firewalls, routing policies, and packet filtering**.

---

21. Exercise 5 - Troubleshoot a Network Path

Suppose a server cannot reach a destination.

First test:

```
ping 8.8.8.8
```

Then:

```
traceroute -n 8.8.8.8
```

You can investigate where responses stop.

For example:

```
1   192.168.1.1       1 ms
2   10.10.0.1         5 ms
3   172.16.1.1       10 ms
4   * * *
5   * * *
```

This doesn't by itself prove that hop 4 is the failure point, but it gives the administrator a useful clue for further investigation.

---

22. `traceroute` (vs) `ping`

Feature	ping	traceroute
Tests reachability	**Yes**	Yes
Shows intermediate routers	No	**Yes**
Shows hop count	No	**Yes**
Measures latency	Yes	**Yes, per hop**
Troubleshoots routing	Limited	**Excellent**
Shows complete path	No	**Yes, when responses are available**

Conceptually:

```
ping
 ↓
"Can I reach the destination?"

traceroute
 ↓
"How does my traffic get there?"
```

---

23. `traceroute` (vs) Other Discovery Tools

Tool	Primary purpose
`fping`	Fast host reachability
`netdiscover`	ARP-based LAN discovery
`arp-scan`	Detailed ARP discovery
`avahi-browse`	mDNS/DNS-SD service discovery
`lldpctl`	Directly connected network-device discovery
`masscan`	High-speed TCP port discovery
`traceroute`	**Network path/router discovery**
`nmap`	Comprehensive host/port/service analysis

This makes **`traceroute`** different from most of the tools in Linux networking series: **it focuses on the path between networks rather than simply discovering hosts or ports.**

24. Useful `traceroute` Commands

Command	Purpose
`traceroute 8.8.8.8`	Basic route tracing
`traceroute -n 8.8.8.8`	Show numeric IP addresses
`traceroute -m 10 8.8.8.8`	Maximum 10 hops
`traceroute -q 1 8.8.8.8`	One probe per hop
`traceroute -w 5 8.8.8.8`	5-second timeout
`sudo traceroute -I 8.8.8.8`	ICMP traceroute
`sudo traceroute -T -p 443 example.com`	TCP/443 traceroute
`traceroute -4 8.8.8.8`	IPv4
`traceroute -6 <IPv6>`	IPv6
`traceroute --help`	Display help

25. Recommended Learning Sequence

Learn **`traceroute`** immediately after the basic host-discovery tools:

```
Linux Network Configuration
          ↓
         ip
          ↓
       ip neigh
          ↓
        ping
          ↓
        fping
          ↓
    netdiscover
          ↓
      arp-scan
          ↓
    avahi-browse
          ↓
       lldpctl
          ↓
     traceroute
          ↓
       masscan
          ↓
        nmap
```

The networking concepts progressively become:

```
IP Configuration
        ↓
ARP / Neighbor Discovery
        ↓
ICMP Host Reachability
        ↓
LAN Device Discovery
        ↓
Service Discovery
        ↓
Physical Network Discovery
        ↓
Routing / Path Discovery
        ↓
TCP Port Discovery
        ↓
Detailed Network Analysis
```

Key learning point

`traceroute` teaches students how packets travel through multiple routers between a source and destination. It is an essential practical tool for understanding IP routing, network topology, latency, and troubleshooting connectivity problems.