

Linux Networking

Troubleshooting and Diagnostics

LinuXpert Systems

WhatsApp: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. ping 1

2. traceroute 8

3. ss 14

4. netstat 21

5. ethtool 28

6. mii-tool 39

7. nc 49

8. tcpdump 59

9. lsof 75

1. ping

The **`ping`** command is a network diagnostic tool used to **test whether a remote host is reachable** over an IP network. It also measures the **round-trip time (RTT)** for packets sent between your computer and the destination.

It works by sending **ICMP (Internet Control Message Protocol) Echo Request** packets to the target host and waiting for **ICMP Echo Reply** packets.

Syntax

```
ping [OPTIONS] destination
```

- **`destination` can be:**
  - Hostname (e.g., `google.com`)
  - IPv4 address (e.g., `8.8.8.8`)
  - IPv6 address (e.g., `2001:4860:4860::8888`)

Basic Examples

1. Ping a hostname

```
ping google.com
```

Example output:

```
PING google.com (142.250.183.14) 56(84) bytes of data.
64 bytes from 142.250.183.14: icmp_seq=1 ttl=117 time=18.4 ms
64 bytes from 142.250.183.14: icmp_seq=2 ttl=117 time=18.2 ms
```

Meaning:

- **icmp_seq** → Packet sequence number
- **ttl** → Time To Live
- **time** → Round-trip time

---

2. Ping an IP address

```
ping 8.8.8.8
```

3. Stop ping

By default, Linux keeps pinging continuously.

Press:

```
Ctrl + C
```

Summary:

```
--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss
```

Common Options

`-c` : Send a fixed number of packets

```
ping -c 4 google.com
```

Output:

```
4 packets transmitted, 4 received
```

Useful for scripts and testing.

`-i` : Specify interval

Send one packet every 2 seconds.

```
ping -i 2 google.com
```

`-W` : Timeout for each reply

Wait only 3 seconds.

```
ping -W 3 google.com
```

`-w` : Overall deadline

Stop after 20 seconds.

```
ping -w 20 google.com
```

`-s` : Packet size

Send a larger packet.

```
ping -s 1000 google.com
```

`-f` : Flood ping (Root only)

```
sudo ping -f google.com
```

Sends packets as fast as possible.

Used for:

- Stress testing
- Performance testing

Use with caution, as it can overload networks.

`-q` : Quiet mode

Only print summary.

```
ping -q -c 5 google.com
```

`-4` : Force IPv4

```
ping -4 google.com
```

`-6` : Force IPv6

```
ping -6 google.com
```

`-D` : Print timestamps

```
ping -D google.com
```

Example:

```
[1722234567.123456] 64 bytes from ...
```

`-I` : Specify network interface

```
ping -I eth0 google.com
```

or

```
ping -I wlan0 google.com
```

`-b` : Ping broadcast address

```
ping -b 192.168.1.255
```

Requires appropriate permissions.

Understanding the Output

Example:

```
64 bytes from 8.8.8.8:
icmp_seq=5 ttl=118 time=20.3 ms
```

Field	Meaning
64 bytes	Size of received packet
from	Sender IP address
icmp_seq	Packet sequence number
ttl	Time To Live (remaining hop count)
time	Round-trip latency

Statistics at the End

Example:

```
5 packets transmitted, 5 received, 0% packet loss
rtt min/avg/max/mdev = 18.2/20.5/24.1/2.1 ms
```

Meaning:

- **Transmitted** → Packets sent
- **Received** → Packets received
- **Packet loss** → Percentage of lost packets

- **min** → Fastest response
- **avg** → Average response
- **max** → Slowest response
- **mdev** → Variation in response time (standard deviation)

Practical Uses

1. Check Internet connectivity

```
ping 8.8.8.8
```

If successful, your network connection is working.

---

2. Check DNS resolution

```
ping google.com
```

If this fails but `ping 8.8.8.8` works, the issue is likely with DNS.

---

3. Test another computer on the LAN

```
ping 192.168.1.10
```

---

4. Measure network latency

```
ping -c 10 server.example.com
```

---

5. Check packet loss

```
ping -c 100 8.8.8.8
```

Look for the packet loss percentage in the summary.

Exit Status

Exit Code	Meaning
0	At least one reply received (success)
1	No replies received
2	Other error (invalid options, permissions, etc.)

Advantages

- Simple and fast network connectivity test
- Measures latency (RTT)
- Detects packet loss
- Helps troubleshoot DNS issues
- Verifies whether a host is reachable

Limitations

- Some hosts block ICMP, so they may not respond even if they are online.
- A successful ping only shows that ICMP traffic works; it does **not** guarantee that services such as HTTP, SSH, or FTP are available.
- Firewalls may filter or rate-limit ping requests.

Frequently Used Commands

```
# Ping continuously
ping google.com

# Send 4 packets
ping -c 4 google.com

# Ping an IP address
ping 8.8.8.8

# Force IPv4
ping -4 google.com

# Force IPv6
ping -6 google.com

# Send larger packets
ping -s 1000 google.com

# Quiet summary only
ping -q -c 10 google.com

# Ping through a specific interface
ping -I eth0 google.com

# Add timestamps
ping -D google.com
```

`ping` is one of the most commonly used networking tools for quickly verifying connectivity, measuring latency, detecting packet loss, and performing basic network troubleshooting.

2. traceroute

The **`traceroute`** command is a network diagnostic tool used to **display the path (route)** that packets take from your computer to a destination host. It also measures the **time taken** to reach each router (hop) along the way.

It is commonly used to:

- Troubleshoot network connectivity issues.
- Identify where network delays occur.
- Detect routing problems.
- Understand the path data takes across the Internet.

Syntax

```
traceroute [options] destination
```

- **destination**: Hostname (e.g., `google.com`) or IP address (e.g., `8.8.8.8`).

How `traceroute` Works

`traceroute` sends packets with increasing **Time-To-Live (TTL)** values.

1. The first packet has TTL = 1.
2. The first router decreases TTL to 0 and returns an **ICMP Time Exceeded** message.
3. `traceroute` records that router.
4. Next packet uses TTL = 2.
5. The second router responds.
6. This continues until the destination is reached.

Each router between your system and the destination is called a **hop**.

Basic Example

```
traceroute google.com
```

Example output:

```
traceroute to google.com (142.250.183.110), 30 hops max

1  192.168.1.1        1.0 ms   0.9 ms   1.1 ms
2  10.10.0.1          7.3 ms   7.5 ms   7.2 ms
3  172.16.5.1        12.2 ms  12.0 ms  12.1 ms
4  203.0.113.5       18.7 ms  19.1 ms  18.9 ms
5  142.250.183.110   25.4 ms  25.1 ms  25.2 ms
```

Understanding the Output

```
1  192.168.1.1   1.0 ms  0.9 ms  1.1 ms
```

Field	Meaning
`1`	Hop number
`192.168.1.1`	Router IP address
`1.0 ms`	First response time
`0.9 ms`	Second response time
`1.1 ms`	Third response time

By default, **`traceroute`** sends **three probes** to each hop.

Common Options

1. Trace to a Host

```
traceroute google.com
```

or

```
traceroute 8.8.8.82. Display Numeric Addresses Only (-n)
```

Avoid DNS lookups for faster results.

```
traceroute -n google.com
```

Output:

```
1 192.168.1.1
2 10.10.0.1
3 172.16.5.1
```

3. Set Maximum Number of Hops (`-m`)

Default is usually **30**.

```
traceroute -m 20 google.com
```

Only checks up to 20 routers.

---

4. Change Number of Probes (`-q`)

Default is **3**.

```
traceroute -q 5 google.com
```

Sends five probes per hop.

---

5. Specify Wait Time (`-w`)

Wait 5 seconds for each reply.

```
traceroute -w 5 google.com
```

---

6. Use ICMP Instead of UDP (`-I`)

```
sudo traceroute -I google.com
```

Uses ICMP Echo Requests (similar to `ping`).

---

7. Use TCP SYN Packets (`-T`)

Useful if UDP or ICMP is blocked.

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

---

8. Specify Destination Port (`-p`)

```
traceroute -p 80 google.com
```

Uses destination port 80.

9. Set Initial TTL (`-f`)

Start tracing from a specific hop.

```
traceroute -f 5 google.com
```

Starts at hop 5.

---

10. Specify Network Interface (`-i`)

```
sudo traceroute -i eth0 google.com
```

Uses interface `eth0`.

Example: Trace to Google's DNS

```
traceroute 8.8.8.8
```

Possible output:

```
1  192.168.1.1
2  10.20.0.1
3  203.0.113.1
4  72.14.212.1
5  8.8.8.8
```

This shows that packets travelled through five routers before reaching Google's DNS server.

Example with Numeric Output

```
traceroute -n 8.8.8.8
```

DNS lookups are skipped, making the command faster.

Understanding `*` in Output

Example:

```
5  *  *  *
```

This means the router did **not respond** to the probes. Possible reasons include:

- Firewall blocking traceroute packets.

- Router configured not to reply.
- Packet loss.
- Network congestion.

A few * entries do not necessarily indicate a broken route if later hops respond successfully.

Common Uses

Check route to a website

```
traceroute google.com
```

Troubleshoot slow connections

```
traceroute example.com
```

Look for a hop with unusually high latency.

Verify routing

```
traceroute server.example.com
```

Shows the path taken to reach a remote server.

Skip DNS resolution

```
traceroute -n 1.1.1.1
```

Installing `traceroute`

Ubuntu/Debian

```
sudo apt install traceroute
```

RHEL/CentOS

```
sudo dnf install traceroute
or
sudo yum install traceroute
```

Arch Linux

```
sudo pacman -S traceroute
```

Exit Status

- **0** – Command completed successfully.
- **Non-zero** – An error occurred (for example, invalid options or the destination could not be reached).

Difference Between '`ping`' and '`traceroute`'

Feature	ping	traceroute
Tests connectivity		
Shows full network path		
Displays each router (hop)		
Measures latency		(per hop)
Useful for routing issues		
Uses ICMP by default		Usually UDP (Linux), with ICMP/TCP options available

Summary

- **Purpose:** Displays the path packets take to a destination and the delay at each hop.
- **Default packet type (Linux):** UDP (ICMP and TCP can also be used with options).
- **Useful for:** Network troubleshooting, identifying routing problems, and locating latency or packet loss.
- **Basic command:**

  ```
  traceroute google.com
  ```

- **Frequently used options:** `-n`, `-m`, `-q`, `-w`, `-I`, `-T`, `-p`, `-f`, `-i`.

3. ss

The **`ss`** (Socket Statistics) command is a Linux utility used to **display network socket information**. It is a modern replacement for the older **`netstat`** command and is much **faster and more efficient**, especially on systems with many network connections.

It allows you to view:

- TCP connections
- UDP connections
- UNIX domain sockets
- Listening ports
- Network statistics
- Process information associated with sockets

Syntax

```
ss [OPTIONS]
```

Common Options

Option	Description
`-t`	Show TCP sockets
`-u`	Show UDP sockets
`-l`	Show listening sockets only
`-a`	Show all sockets (listening and non-listening)
`-n`	Show numeric addresses and ports (don't resolve names)
`-p`	Show the process using each socket
`-s`	Display socket summary statistics
`-4`	Show IPv4 sockets only
`-6`	Show IPv6 sockets only
`-r`	Resolve service names
`-o`	Show timer information
`-m`	Show memory usage
`-H`	Hide the header line

Basic Examples

1. Display all connections

```
ss
```

Example Output

```
Netid State  Recv-Q Send-Q Local Address:Port   Peer Address:Port
tcp   ESTAB  0      0      192.168.1.10:ssh     192.168.1.20:52341
```

Shows active network connections.

---

2. Show all TCP connections

```
ss -t
```

Output

```
State      Recv-Q Send-Q Local Address:Port
ESTAB      0      0      192.168.1.10:ssh
```

---

3. Show all UDP sockets

```
ss -u
```

Output

```
State      Recv-Q Send-Q Local Address:Port
UNCONN     0      0      0.0.0.0:68
```

---

4. Show listening ports

```
ss -l
```

Shows services waiting for incoming connections.

5. Show listening TCP ports

```
ss -lt
```

Example:

```
State  Local Address:Port
LISTEN 0.0.0.0:ssh
LISTEN 127.0.0.1:mysql
```

---

6. Show listening UDP ports

```
ss -lu
```

---

7. Show all sockets

```
ss -a
```

Displays both listening and active connections.

---

8. Show numeric IP addresses

```
ss -n
```

Instead of

```
localhost:ssh
```

You'll see

```
127.0.0.1:22
```

---

9. Show processes using sockets

```
sudo ss -p
```

Example

```
tcp LISTEN 0 128 *:22 *:* users:(("sshd",pid=1050,fd=3))
```

Shows:

- Process name
- PID
- File descriptor

---

10. Show TCP sockets with process names

```
sudo ss -tp
```

---

11. Display socket statistics

```
ss -s
```

Example

```
Total: 450
TCP:   25
UDP:   10
RAW:   1
UNIX:  95
```

Useful for quickly checking overall network activity.

---

12. Show IPv4 connections

```
ss -4
```

---

13. Show IPv6 connections

```
ss -6
```

---

14. Display timer information

```
ss -o
```

Shows TCP timers such as retransmission timers and keepalive timers.

15. Show socket memory usage

```
ss -m
```

Useful for diagnosing memory usage related to sockets.

---

16. Hide the header

```
ss -H
```

Filtering Examples

Find connections on port 22 (SSH)

```
ss -t '( sport = :22 )'
```

Find connections to port 80

```
ss -t '( dport = :80 )'
```

Find established TCP connections

```
ss -t state established
```

Find listening sockets

```
ss -ltn
```

Commonly used by system administrators to see which services are listening.

Check whether a web server is listening on port 80

```
ss -ltn | grep :80
```

Check whether SSH is running

```
ss -ltn | grep :22
```

Display all UNIX sockets

```
ss -x
```

Display UNIX listening sockets

```
ss -lx
```

Important Output Columns

Column	Meaning
`Netid`	Protocol (TCP, UDP, UNIX, etc.)
`State`	Connection state (LISTEN, ESTAB, TIME-WAIT, etc.)
`Recv-Q`	Bytes waiting to be read by the application
`Send-Q`	Bytes waiting to be sent
`Local Address:Port`	Local IP address and port
`Peer Address:Port`	Remote IP address and port

Common TCP States

State	Meaning
`LISTEN`	Waiting for incoming connections
`ESTAB`	Connection established
`SYN-SENT`	Connection request sent
`SYN-RECV`	Connection request received
`FIN-WAIT-1`	Closing connection (first stage)
`FIN-WAIT-2`	Waiting for remote close
`TIME-WAIT`	Waiting to ensure delayed packets are handled
`CLOSE-WAIT`	Waiting for the local application to close
`LAST-ACK`	Waiting for final acknowledgement
`CLOSED`	Connection closed

Practical Uses

- Check which ports are open.
- Verify whether a service (SSH, Apache, Nginx, MySQL, etc.) is listening.
- Find which process is using a specific port.
- Troubleshoot network connectivity issues.

- Monitor active TCP/UDP connections.
- Display socket statistics and memory usage.
- Debug server applications.

Difference Between `ss` and `netstat`

Feature	ss	netstat
Speed	Faster	Slower
Reads kernel data directly	Yes	No
Process information	Yes (-p)	Yes
Socket statistics	Yes	Limited
Default on modern Linux	Yes	No

Most Frequently Used Commands

```
ss
```

Show active connections.

```
ss -tuln
```

Show all listening TCP and UDP ports using numeric addresses.

```
sudo ss -tulpn
```

Show listening TCP and UDP ports along with the processes using them.

```
ss -s
```

Display a summary of socket statistics.

```
ss -tp
```

Show active TCP connections with the associated process information.

`ss` is one of the most useful networking tools on Linux for monitoring and troubleshooting sockets, making it a standard utility for system administrators and network engineers.

4. netstat

The **`netstat`** (**Net**work **Stat**istics) command is a networking utility used to display information about:

- Network connections
- Listening ports
- Routing tables
- Network interface statistics
- Protocol statistics
- Active TCP/UDP connections

It is commonly used for **network troubleshooting**, **server administration**, and **security monitoring**.

Note:

On modern Linux distributions, **`netstat`** is considered **deprecated** and has been replaced by the **`ss`** command (part of the **`iproute2`** package). However, **`netstat`** is still widely used and available through the **`net-tools`** package.

Syntax

```
netstat [options]
```

Common Examples

1. Display all active network connections

```
netstat
```

Example output:

```
Proto Recv-Q Send-Q Local Address     Foreign Address   State
tcp        0      0 localhost:ssh     192.168.1.10:52344 ESTABLISHED
```

Shows currently active TCP connections.

2. Show listening ports

```
netstat -l
```

Output:

```
Proto Local Address      State
tcp   0.0.0.0:ssh        LISTEN
tcp   0.0.0.0:http       LISTEN
```

Displays services waiting for incoming connections.

---

3. Show TCP connections only

```
netstat -t
```

Shows only TCP connections.

---

4. Show UDP connections only

```
netstat -u
```

Shows only UDP sockets.

---

5. Show listening TCP ports

```
netstat -lt
```

---

6. Show listening UDP ports

```
netstat -lu
```

Lists UDP services waiting for packets.

---

7. Display process using each port

```
sudo netstat -tulpn
```

Example:

```
Proto Local Address    State   PID/Program name
tcp   0.0.0.0:22       LISTEN  1002/sshd
```

```
tcp   0.0.0.0:80       LISTEN  567/apache2
udp   0.0.0.0:68               899/dhclient
```

Meaning:

- **PID** = Process ID
- **Program name** = Application using the port

This is one of the most commonly used **`netstat`** commands.

---

8. Show numeric addresses

```
netstat -n
```

Instead of:

```
localhost:ssh
```

You see:

```
127.0.0.1:22
```

This avoids DNS lookups and is faster.

---

9. Show routing table

```
netstat -r
```

Displays the system routing table.

Equivalent modern command:

```
ip route
```

---

10. Show network interfaces

```
netstat -i
```

Example:

```
Iface   MTU RX-OK TX-OK
eth0   1500  1200  1100
```

Shows packet statistics for each interface.

11. Display protocol statistics

```
netstat -s
```

Example:

```
Tcp:
    Active opens: 120
    Passive opens: 54
    Failed connections: 3

Udp:
    Packets received: 500
```

Useful for diagnosing network issues.

---

12. Continuously monitor connections

```
netstat -c
```

Refreshes the output every second.

Frequently Used Options

Option	Description
`-a`	Show all sockets (listening and non-listening)
`-l`	Show only listening sockets
`-t`	Display TCP connections
`-u`	Display UDP connections
`-n`	Show numeric IP addresses and ports
`-p`	Show PID and program name
`-r`	Display routing table
`-i`	Show network interface statistics
`-s`	Show protocol statistics
`-c`	Refresh continuously

Common Option Combinations

Show all TCP connections

```
netstat -at
```

Show all UDP connections

```
netstat -au
```

Show all listening sockets

```
netstat -l
```

Show listening TCP sockets

```
netstat -lt
```

Show listening UDP sockets

```
netstat -lu
```

Show listening ports with process names

```
sudo netstat -tulpn
```

Show all sockets with numeric addresses

```
netstat -an
```

Show protocol statistics

```
netstat -s
```

Show routing table

```
netstat -r
```

Monitor connections continuously

```
netstat -c
```

Understanding Connection States

When viewing TCP connections, you may see these states:

State	Meaning
LISTEN	Waiting for incoming connections
ESTABLISHED	Connection is active
SYN_SENT	Connection request sent
SYN_RECV	Connection request received
FIN_WAIT1	Closing connection (first phase)
FIN_WAIT2	Waiting for peer to close
CLOSE_WAIT	Waiting for local application to close
LAST_ACK	Waiting for final acknowledgment
TIME_WAIT	Waiting before fully closing the connection
CLOSED	Connection has been closed

Installation

On Debian/Ubuntu:

```
sudo apt update
```

```
sudo apt install net-tools
```

On RHEL/CentOS:

```
sudo yum install net-tools
```

Modern Alternatives

`netstat` is largely superseded by newer tools:

Task	netstat	Modern command
Show sockets	`netstat -tulpn`	`ss -tulpn`
Routing table	`netstat -r`	`ip route`
Interfaces	`netstat -i`	`ip -s link`
Interface addresses	`ifconfig`	`ip addr`

When to Use `netstat`

Use `netstat` when you want to:

- Check which ports are open on a system.
- Identify which process is listening on a specific port.
- View active network connections.
- Troubleshoot network connectivity issues.
- Inspect routing tables.
- Monitor network traffic statistics.
- Verify whether a server application (such as SSH, Apache, or MySQL) is listening for incoming connections.

For modern Linux systems such as Ubuntu 22.04 and later, learning the **`ss`** command alongside **`netstat`** is recommended, as **`ss`** provides similar information more quickly and with additional capabilities.

5. ethtool

The **ethtool** command is a Linux networking utility used to **view and configure Ethernet network interface settings**. It is particularly useful for examining the capabilities, link status, speed, duplex mode, auto-negotiation, driver information, statistics, and other hardware-level properties of wired network interfaces.

It is commonly used by Linux system administrators and network engineers for **network configuration, performance tuning, troubleshooting, and diagnosing Ethernet connectivity problems**.

Syntax

```
ethtool [OPTIONS] DEVICE
```

- **DEVICE** – The name of the Ethernet network interface, such as `eth0`, `enp0s3`, or `eno1`.
- **OPTIONS** – Specify the information to display or the network setting to modify.

Common Uses of ethtool

Command	Purpose
`ethtool eth0`	Display Ethernet interface information
`ethtool -i eth0`	Display driver information
`ethtool -k eth0`	Display offloading features
`ethtool -S eth0`	Display interface statistics
`ethtool -g eth0`	Display RX/TX ring buffer sizes
`ethtool -c eth0`	Display interrupt coalescing settings
`ethtool -a eth0`	Display pause parameters
`ethtool -T eth0`	Display timestamping capabilities
`ethtool -p eth0`	Blink the physical interface LED
`ethtool -P eth0`	Display permanent hardware address
`ethtool -h`	Display help

1. Display Ethernet Interface Information

```
ethtool eth0
```

Example:

```
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half
                                10baseT/Full
                                100baseT/Half
                                100baseT/Full
                                1000baseT/Full
        Speed: 1000Mb/s
        Duplex: Full
        Auto-negotiation: on
        Link detected: yes
```

This provides important information about the physical Ethernet connection.

---

2. Check Link Status

```
ethtool eth0
```

Look for:

```
Link detected: yes
```

or:

```
Link detected: no
```

This is useful for determining whether the Ethernet interface has an active physical connection.

---

3. Check Network Speed

```
ethtool eth0 | grep Speed
```

Example:

```
Speed: 1000Mb/s
```

This indicates that the interface has negotiated a **1 Gbps** connection.

Other possible values include:

```
Speed: 100Mb/s
Speed: 2500Mb/s
Speed: 5000Mb/s
Speed: 10000Mb/s
```

---

4. Check Duplex Mode

```
ethtool eth0 | grep Duplex
```

Example:

```
Duplex: Full
```

Possible values include:

```
Duplex: Full
Duplex: Half
```

Modern Ethernet networks normally operate in **full-duplex mode**.

---

5. Check Auto-Negotiation

```
ethtool eth0 | grep Auto-negotiation
```

Example:

```
Auto-negotiation: on
```

Auto-negotiation allows the network interface and switch to automatically negotiate compatible speed and duplex settings.

---

6. Display Ethernet Driver Information

```
ethtool -i eth0
```

Example:

```
driver: e1000e
version: 6.8.0
firmware-version: 0.13-4
bus-info: 0000:00:19.0
```

This is especially useful when troubleshooting driver-related problems.

7. Display Supported Link Modes

```
ethtool eth0
```

Look for:

```
Supported link modes:
        10baseT/Full
        100baseT/Full
        1000baseT/Full
```

This shows the Ethernet speeds and modes supported by the network adapter.

---

8. Display Current Link Modes

```
ethtool eth0
```

For example:

```
Speed: 1000Mb/s
Duplex: Full
```

This represents the **currently negotiated** connection rather than all capabilities of the adapter.

---

9. Display Interface Statistics

```
ethtool -S eth0
```

Example:

```
NIC statistics:
    rx_packets: 125634
    tx_packets: 117823
    rx_bytes: 184563920
    tx_bytes: 163829421
    rx_errors: 0
    tx_errors: 0
    rx_dropped: 12
    tx_dropped: 0
```

This is very useful for diagnosing:

- Packet errors
- Dropped packets
- Transmission problems

- Receive problems
- Hardware issues

The exact statistics depend on the network driver.

---

10. Display Driver and Firmware Information

```
ethtool -i eth0
```

Important fields include:

```
driver:
version:
firmware-version:
bus-info:
```

For example:

```
driver: igb
version: 6.8.0
firmware-version: 1.63, 0x8000e5d9
bus-info: 0000:03:00.0
```

---

11. Display Offloading Features

```
ethtool -k eth0
```

Example:

```
Features for eth0:
rx-checksumming: on
tx-checksumming: on
tcp-segmentation-offload: on
generic-segmentation-offload: on
generic-receive-offload: on
```

Hardware offloading can improve network performance by allowing the network adapter to perform certain operations instead of the CPU.

12. Enable or Disable an Offloading Feature

For example, to disable TCP segmentation offload:

```
sudo ethtool -K eth0 tso off
```

To enable it again:

```
sudo ethtool -K eth0 tso on
```

Similarly:

```
sudo ethtool -K eth0 gro off
```

and:

```
sudo ethtool -K eth0 gro on
```

Note:

Such changes are generally temporary and may be lost after reboot or interface reconfiguration.

---

13. Display Ring Buffer Information

```
ethtool -g eth0
```

Example:

```
Ring parameters for eth0:
Pre-set maximums:
RX:             4096
RX Mini:        0
RX Jumbo:       0
TX:             4096
```

Current hardware settings:

```
RX:             1024
RX Mini:        0
RX Jumbo:       0
TX:             1024
```

Ring buffers are used by the network adapter for storing packets awaiting processing.

14. Change Ring Buffer Size

For example:

```
sudo ethtool -G eth0 rx 2048 tx 2048
```

This changes the receive and transmit ring sizes, provided that the driver supports the requested values.

---

15. Display Pause Parameters

```
ethtool -a eth0
```

Example:

```
Pause parameters for eth0:
        Autonegotiate:  on
        RX:             on
        TX:             on
```

Pause frames can be used for flow control between Ethernet devices.

---

16. Display Interrupt Coalescing Settings

```
ethtool -c eth0
```

This displays interrupt coalescing parameters used by the network adapter.

These settings can affect the balance between:

- Network latency
- CPU utilization
- Network throughput

---

17. Display Hardware Timestamping Capabilities

```
ethtool -T eth0
```

This displays the interface's timestamping capabilities.

Hardware timestamping is important for applications requiring highly accurate network timing, such as:

- Precision Time Protocol (PTP)
- Industrial networking
- Financial networking
- Time-sensitive applications

---

18. Blink the Ethernet Port LED

```
sudo ethtool -p eth0
```

This causes the physical Ethernet port's LED to blink, where supported.

This is particularly useful on servers with many network interfaces because it helps identify **which physical port corresponds to a particular Linux interface**.

---

19. Display Permanent MAC Address

```
ethtool -P eth0
```

Example:

```
Permanent address: 3c:52:82:ab:cd:ef
```

This displays the permanent hardware MAC address programmed into the network adapter, when supported.

---

20. Display Interface Information in a Compact Form

A useful combination is:

```
ethtool eth0 | grep -E 'Speed|Duplex|Auto-negotiation|Link detected'
```

Example:

```
        Speed: 1000Mb/s
        Duplex: Full
        Auto-negotiation: on
        Link detected: yes
```

This provides a quick summary of the Ethernet link.

21. Change Ethernet Speed and Duplex

For example:

```
sudo ethtool -s eth0 speed 100 duplex full autoneg off
```

This attempts to configure:

- Speed: **100 Mbps**
- Duplex: **Full**
- Auto-negotiation: **Disabled**

To restore auto-negotiation:

```
sudo ethtool -s eth0 autoneg on
```

Caution:

Manually forcing speed/duplex can cause connectivity problems if the switch is configured differently. Auto-negotiation is normally recommended for modern Ethernet networks.

---

22. Display All Supported Commands and Options

```
ethtool --help
```

or:

```
man ethtool
```

The manual page provides detailed information about the many options supported by the installed version of **`ethtool`**.

Important Options

Option	Description
`-i`	Display driver information
`-k`	Display offloading features
`-K`	Change offloading features
`-S`	Display NIC statistics
`-g`	Display ring parameters
`-G`	Change ring parameters
`-c`	Display coalescing settings
`-C`	Change coalescing settings
`-a`	Display pause parameters
`-A`	Change pause parameters
`-T`	Display timestamping capabilities
`-p`	Blink port LED
`-P`	Display permanent MAC address
`-s`	Change link settings
`-h`	Display help

Practical Troubleshooting Examples

Check whether the cable/link is detected:

```
ethtool eth0 | grep "Link detected"
```

Check negotiated speed:

```
ethtool eth0 | grep Speed
```

Check duplex:

```
ethtool eth0 | grep Duplex
```

Check driver:

```
ethtool -i eth0
```

Check hardware errors:

```
ethtool -S eth0
```

Identify the physical Ethernet port:

```
sudo ethtool -p eth0
```

Summary

`ethtool` operates primarily at the **Ethernet hardware/driver level**, making it complementary to commands such as **`ip`, `nmcli`**, and **`ifconfig`.**

A useful way to distinguish them is:

Command	Primary Purpose
`ip`	Configure and manage network interfaces, IP addresses and routes
`ifconfig`	Older interface configuration and status utility
`route`	Older routing-table management utility
`nmcli`	Configure and manage NetworkManager connections
`ethtool`	Examine and configure Ethernet hardware/link properties

For Linux networking, `ethtool` **is particularly important because it bridges the gap between IP/network configuration and the underlying Ethernet hardware and driver**.

6. mii-tool

The **`mii-tool`** command is a Linux networking utility used to **view and manipulate the Media Independent Interface (MII)** status of Ethernet network interfaces. It can be used to check whether an Ethernet link is available and to display information such as negotiated speed, duplex mode, and link status.

`mii-tool` is part of the traditional **net-tools** package. However, it is now considered a **legacy/obsolete utility**, and **`ethtool`** is recommended for modern Ethernet hardware.

Syntax

```
mii-tool [OPTIONS] interface
```

or

```
mii-tool [OPTIONS] interface1 interface2 ...
```

Common options

Option	Description
`-v`	Display verbose MII information
`-V`	Display version information
`-R`	Reset the MII interface
`-r`	Restart autonegotiation
`-w`	Continuously watch link status
`-l`	Log link-status changes while using -w
`-A`	Specify media types to advertise
`-F`	Force a particular media type
`-p`	Specify the PHY address

The **`mii-tool`** is a legacy tool and **`ethtool`** is a recommend replacement.

1. Display the status of all Ethernet interfaces

```
mii-tool
```

Example:

```
eth0: negotiated 100baseTx-FD, link ok
eth1: no link
```

This provides a quick overview of the Ethernet interfaces that support MII.

The output generally indicates:

- Interface name
- Negotiated Ethernet speed
- Duplex mode
- Link status

For example:

```
eth0: negotiated 100baseTx-FD, link ok
```

means that **`eth0`** has negotiated **100 Mbps, full-duplex** operation and that the physical link is available.

---

2. Check a specific interface

```
mii-tool eth0
```

Example:

```
eth0: negotiated 100baseTx-FD, link ok
```

This is useful when a system has several network interfaces and you want to check only one of them.

---

3. Check whether the Ethernet cable is connected

```
mii-tool eth0
```

If a link is available:

```
eth0: negotiated 100baseTx-FD, link ok
```

If no physical link is detected:

```
eth0: no link
```

This makes **`mii-tool`** useful for basic troubleshooting of wired Ethernet connectivity.

However, a **`no link`** result indicates a problem at the physical/link layer; it does **not** necessarily mean that IP networking is incorrectly configured.

---

4. Display verbose information

```
mii-tool -v eth0
```

The `-v` option provides more detailed information about the MII status.

Example:

```
eth0: negotiated 100baseTx-FD, link ok
  product info: vendor ...
  basic mode:   autonegotiation enabled
  basic status: link ok
```

The exact output varies according to the Ethernet adapter and driver.

---

5. Display version information

```
mii-tool -V
```

This displays the version of **`mii-tool`**.

Example:

```
mii-tool.c 1.60 2009/02/04
```

The exact version information depends on the installed `net-tools` package.

---

6. Restart Ethernet autonegotiation

```
sudo mii-tool -r eth0
```

The `-r` option restarts the **autonegotiation process**.

Autonegotiation allows the two ends of an Ethernet connection to determine an appropriate combination of:

- Speed
- Duplex mode
- Other supported link characteristics

This can sometimes be useful when an Ethernet interface appears to have a link problem after connecting or reconnecting a cable.

---

7. Watch Ethernet link status

```
mii-tool -w eth0
```

The `-w` option continuously monitors the interface and reports changes in link status.

For example:

```
eth0: negotiated 100baseTx-FD, link ok
eth0: no link
eth0: negotiated 100baseTx-FD, link ok
```

This can be useful when troubleshooting an unstable Ethernet connection.

The interface is polled approximately once per second when using the watch mode. (Linux Die)

Press:

```
Ctrl+C
```

to stop monitoring.

---

8. Watch and log link-status changes

```
sudo mii-tool -w -l eth0
```

The `-l` option is used together with `-w` to record link-status changes in the system log rather than simply displaying them on the terminal.

---

9. Reset the MII interface

```
sudo mii-tool -R eth0
```

The `-R` option resets the MII interface to its default configuration.

Use this option carefully because changing the state of a network interface can temporarily interrupt network connectivity.

---

10. Force a specific Ethernet mode

The `-F` option can be used to disable autonegotiation and force a particular supported media mode.

For example:

```
sudo mii-tool -F 100baseTx-FD eth0
```

This attempts to force:

```
100 Mbps
Full Duplex
```

Other traditional modes include:

```
100baseTx-FD
100baseTx-HD
10baseT-FD
10baseT-HD
```

The documented legacy **`mii-tool`** interface supports these media types.

Important

For modern Ethernet adapters, especially Gigabit Ethernet and faster interfaces, **`ethtool`** should normally be used instead:

```
sudo ethtool -s eth0 speed 100 duplex full autoneg off
```

The capabilities and syntax supported by **`ethtool`** are substantially broader than those of **`mii-tool`**.

---

11. Specify advertised media types

The `-A` option specifies which media types should be advertised during autonegotiation.

For example:

```
sudo mii-tool -A 100baseTx-FD,100baseTx-HD eth0
```

This configures the interface to advertise the specified media types.

`-A` and `-F` are mutually exclusive because one controls autonegotiation advertising while the other forces a fixed mode.

12. Specify a PHY address

The `-p` option can be used to specify the PHY address:

```
sudo mii-tool -p 1 eth0
```

This is mainly useful for systems with particular Ethernet hardware or multiple PHY devices.

Practical Troubleshooting Examples

Example 1: Check whether the link is up

```
mii-tool eth0
```

If you get:

```
eth0: negotiated 100baseTx-FD, link ok
```

the physical Ethernet link is operational.

Example 2: Check for a disconnected cable

```
mii-tool eth0
```

Output:

```
eth0: no link
```

Possible causes include:

- Ethernet cable disconnected
- Faulty cable
- Switch port disabled
- Switch problem
- NIC/PHY problem
- Unsupported MII operation

Example 3: Monitor an unstable connection

```
mii-tool -w eth0
```

If the output repeatedly changes between:

```
eth0: link ok
```

and:

```
eth0: no link
```

the connection may be experiencing a physical-layer problem.

Check:

1. Ethernet cable
2. Switch port
3. NIC
4. Network adapter driver
5. Switch configuration

Example 4: Restart autonegotiation

```
sudo mii-tool -r eth0
```

Then check the result:

```
mii-tool eth0
```

Common Errors

`No such device`

Example:

```
SIOCGMIIPHY on 'eth0' failed: No such device
```

This means that the specified network interface does not exist or is not known to the kernel.

First determine the actual interface name:

```
ip link
```

You might find:

```
2: enp3s0:
```

Then use:

```
mii-tool enp3s0
```

`Operation not supported`

Example:

```
SIOCGMIIPHY on 'eth0' failed: Operation not supported
```

This generally means that the interface does not support the MII operations required by **`mii-tool`**. Modern Ethernet hardware may use mechanisms better supported by **`ethtool`**.

Try:

```
sudo ethtool eth0
```

---

mii-tool (vs) ethtool

Feature	mii-tool	ethtool
Check link status	Yes	Yes
Display speed	Yes	Yes
Display duplex	Yes	Yes
Autonegotiation	Basic	Extensive
Modern Ethernet support	Limited	Extensive
Driver information	No	Yes
NIC statistics	Limited/No	Yes
Offload configuration	No	Yes
Ring-buffer configuration	No	Yes
Wake-on-LAN	Limited	Yes
EEPROM access	No	Yes
Hardware diagnostics	No	Yes
Current recommended tool	**No**	**Yes**

`ethtool` is specifically designed to query and control network-driver and hardware settings and provides considerably more functionality.

Important Note

`mii-tool` is worth learning because it is historically important in Linux networking and is still encountered on older systems and in older documentation. However, it should be presented as a **legacy Ethernet troubleshooting command**, not as the preferred modern tool.

A good progression for learning wired networking is:

```
ifconfig
   ↓
mii-tool
   ↓
ethtool
   ↓
ip
   ↓
nmcli
```

In particular, **`mii-tool`** is useful for understanding **Ethernet link, speed, duplex, PHY and autonegotiation**, while **`ethtool`** provides the modern and much more comprehensive interface for Ethernet hardware management.

Quick Reference

```
# Display all interface MII status
mii-tool

# Check eth0
mii-tool eth0

# Verbose information
mii-tool -v eth0

# Display version
mii-tool -V

# Restart autonegotiation
sudo mii-tool -r eth0
```

```
# Watch link status
mii-tool -w eth0

# Watch and log link changes
sudo mii-tool -w -l eth0

# Reset MII
sudo mii-tool -R eth0

# Force 100 Mbps full duplex
sudo mii-tool -F 100baseTx-FD eth0

# Specify advertised modes
sudo mii-tool -A 100baseTx-FD,100baseTx-HD eth0

# Specify PHY address
sudo mii-tool -p 1 eth0
```

7. nc

1. What is `nc` ?

`nc` (Netcat: Network Connection and Troubleshooting Tool) is a simple but powerful Linux networking utility for **testing TCP/UDP connectivity, checking whether ports are open, creating temporary network connections, transferring data, and troubleshooting network services**.

It is often described as the **“Swiss Army knife of networking.”**

---

2. Why learn `nc` ?

`nc` is particularly useful because tools such as **`ping`** and **`dig`** only test specific layers of networking.

For example:

```
ping 192.168.1.10
```

tests ICMP connectivity, but it doesn't tell you whether a web server's TCP port 443 is accessible.

With `nc`:

```
nc -vz 192.168.1.10 443
```

you can directly test whether **TCP port 443 is reachable**.

This makes `nc` extremely useful for:

- Network troubleshooting
- TCP port testing
- Firewall diagnosis
- Testing server availability
- Checking listening services
- Testing DNS-independent connectivity
- Testing UDP services
- Simple client/server communication
- Troubleshooting web, SSH, mail and database services

3. Check whether nc is installed

```
which nc
```

or:

```
nc -h
```

On Ubuntu 22.04, install it with:

```
sudo apt install netcat-openbsd
```

Check the version:

```
nc -h
```

Ubuntu normally provides the **OpenBSD implementation of netcat**.

---

4. Basic syntax

```
nc [options] host port
```

For example:

```
nc google.com 80
```

This attempts to establish a TCP connection to port 80.

---

5. Test whether a TCP port is open

This is one of the most useful applications of nc.

```
nc -vz google.com 443
```

Typical successful output:

```
Connection to google.com 443 port [tcp/https] succeeded!
```

Here:

- `-v` = verbose
- `-z` = zero-I/O mode; scan without sending data

You can test SSH:

```
nc -vz server.example.com 22
```

HTTP:

```
nc -vz server.example.com 80
```

HTTPS:

```
nc -vz server.example.com 443
```

SMTP:

```
nc -vz mail.example.com 25
```

---

6. Test an IP address

```
nc -vz 192.168.1.100 22
```

This is particularly useful when diagnosing DNS problems.

For example:

```
dig web.arattai.in
```

returns:

```
74.188.224.19
```

You can then bypass DNS:

```
nc -vz 74.188.224.19 443
```

This helps determine whether the problem is **DNS or TCP connectivity**.

---

7. Test a range of ports

You can test multiple ports:

```
nc -vz 192.168.1.100 20-25
```

For example:

```
22
23
24
25
```

This can help identify which services are reachable.

However, for comprehensive network/port scanning, **Nmap** is much more appropriate.

8. Set a timeout

A connection can otherwise wait for a considerable amount of time.

Use:

```
nc -vz -w 5 192.168.1.100 443
```

Here:

```
-w 5
```

sets a **5-second timeout**.

This is very useful in scripts and troubleshooting.

For example:

```
nc -vz -w 3 10.10.10.20 443
```

---

9. Check a web server

You can use **nc** to manually communicate with an HTTP server.

```
nc example.com 80
```

Then type:

```
GET / HTTP/1.1
Host: example.com
```

Notice the **blank line after the Host header**.

The server should return an HTTP response.

This is a great way to understand what happens underneath commands such as:

```
curl http://example.com
```

---

10. Check an SMTP server

For example:

```
nc mail.example.com 25
```

You may receive something like:

```
220 mail.example.com ESMTP
```

You can then type:

```
EHLO example.com
```

The server will respond with its supported SMTP capabilities.

This makes `nc` useful for diagnosing mail-server connectivity.

---

11. Create a simple TCP server

nc can also listen for connections.

On one machine:

```
nc -l 5000
```

This creates a TCP listener on port 5000.

On another machine:

```
nc 192.168.1.100 5000
```

Now type text on either side.

For example:

```
Hello from client
```

The other machine will receive it.

This is a very simple way to test:

Client → Network → Server

without involving an actual application server.

---

12. Find whether a local service is listening

Suppose you expect SSH to be running.

First:

```
ss -lntp
```

Then test it locally:

```
nc -vz localhost 22
```

For HTTP:

```
nc -vz localhost 80
```

For HTTPS:

```
nc -vz localhost 443
```

This is useful for distinguishing:

```
Application not running
        ↓
Port not listening
        ↓
Firewall
        ↓
Network problem
```

---

13. Test UDP

TCP is the default.

For UDP use:

```
nc -vzu 192.168.1.100 53
```

where:

```
-u = UDP
-z = zero-I/O
-v = verbose
```

For example:

```
nc -vzu 8.8.8.8 53
```

However, UDP testing has an important limitation: **a successful UDP nc result does not necessarily prove that an application is responding**, because UDP is connectionless.

14. Test DNS connectivity

You can test whether UDP port 53 is reachable:

```
nc -vzu 8.8.8.8 53
```

But don't confuse this with actually testing DNS resolution.

For DNS resolution use:

```
dig google.com
```

For DNS port connectivity use:

```
nc -vzu 8.8.8.8 53
```

So:

Tool	What it tests
`ping`	ICMP
`nc`	TCP/UDP port connectivity
`dig`	DNS
`curl`	HTTP/HTTPS
`ssh`	SSH protocol
`traceroute`	Network path

---

15. Very useful troubleshooting combination

Suppose:

```
ping example.com
```

works, but the website doesn't open.

Start with:

```
dig example.com
```

Then:

```
nc -vz example.com 80
nc -vz example.com 443
```

Then:

```
curl -4 -I https://example.com
```

This gives a layered diagnosis:

```
DNS
↓
IP address
↓
TCP 443
↓
TLS
↓
HTTP
```

---

16. Your `web.arattai.in` example

For the problem you were investigating, you already obtained:

```
web.arattai.in
        ↓
CNAME
        ↓
zs-in1arattai-lc1.arattai.in
        ↓
74.188.224.19
```

Now use:

```
nc -vz -w 5 74.188.224.19 443
```

If you get:

```
Connection ... succeeded!
```

then:

DNS works + TCP/443 works.

Next:

```
curl -4 -I https://web.arattai.in
```

If that fails, the problem is probably **above the basic TCP layer**, such as TLS, HTTP, server configuration, proxying, or application-level behavior.

If nc fails:

```
timed out
```

or:

```
Connection refused
```

then we have a different diagnosis.

`Connection refused`

Usually means the host is reachable but nothing is accepting that port, or a firewall is actively rejecting it.

`Connection timed out`

Usually indicates packet filtering, routing, firewall, or a server/network that is silently dropping the connection.

`No route to host`

Usually indicates a routing or network-layer problem.

---

17. Useful `nc` options

Option	Meaning
`-v`	Verbose
`-z`	Scan without sending data
`-w`	Connection timeout
`-u`	UDP
`-l`	Listen
`-n`	Don't perform DNS lookup
`-4`	IPv4
`-6`	IPv6

Examples:

```
nc -vz example.com 443
```

```
nc -vz -w 5 example.com 443
```

```
nc -vzu example.com 53
```

```
nc -4 -vz example.com 443

nc -6 -vz example.com 443
```

---

18. nc (vs) nmap

nc is excellent for **quick, targeted connectivity testing**:

```
nc -vz server 443
```

nmap is better when you want systematic port/service discovery:

```
nmap server
```

A Linux networking student should ideally learn both.

A good learning progression

```
ping
↓
ip
↓
ss
↓
dig
↓
nc
↓
traceroute
↓
curl
↓
nmap
↓
tcpdump
```

This progression moves from basic connectivity toward **real network troubleshooting and packet-level diagnostics**.

8. tcpdump

1. What is `tcpdump` ?

`tcpdump` (Network Packet Capture and Analysis) is a command-line **packet capture and network traffic analysis tool** for Linux.

It allows you to see packets travelling through a network interface in real time.

For example:

```
sudo tcpdump -i eno1
```

can show traffic such as:

```
16:55:21.123456 IP 192.168.1.20.54321 > 142.250.183.14.443: Flags
[S], seq 12345, win 64240, length 0
```

Unlike **`ping`**, **`dig`**, **`nc`**, or **`curl`**, **`tcpdump`** lets you observe **what is actually happening on the network**.

---

2. Why learn `tcpdump` ?

`tcpdump` is one of the most important tools for Linux network troubleshooting.

It can help diagnose:

- DNS problems
- TCP connection failures
- HTTP/HTTPS connectivity
- Packet loss
- Retransmissions
- Network latency
- ARP problems
- DHCP problems
- Routing problems
- Firewall filtering
- IPv4/IPv6 problems

- Unexpected network traffic
- Server/client communication problems

It is particularly valuable because it allows you to answer:

Did the packet actually leave my machine, and did a response come back ?

---

3. Check whether `tcpdump` is installed

```
which tcpdump
```

Check the version:

```
tcpdump --version
```

On Ubuntu 22.04:

```
sudo apt install tcpdump
```

---

4. Basic syntax

```
sudo tcpdump [options] [filter]
```

The most basic command is:

```
sudo tcpdump
```

However, this may capture traffic from the default interface and can produce a huge amount of output.

It is better to specify the interface.

---

5. List network interfaces

```
sudo tcpdump -D
```

Example:

```
1. eno1
2. wlxf0090d40fad3
3. virbr0
4. any
5. lo
```

You can also use:

```
ip link
```

In your system, for example, `eno1` appears to be your main Ethernet interface.

---

6. Capture packets on a specific interface

```
sudo tcpdump -i eno1
```

This captures packets travelling through `eno1`.

Stop the capture with:

```
Ctrl+C
```

You'll see a summary such as:

```
125 packets captured
130 packets received by filter
0 packets dropped by kernel
```

---

7. Capture packets from all interfaces

Linux provides a convenient pseudo-interface:

```
sudo tcpdump -i any
```

This is extremely useful on servers with multiple interfaces.

For example, your system has:

```
eno1
wlxf0090d40fad3
virbr0
```

You can monitor everything with:

```
sudo tcpdump -i any
```

8. Capture only a specific number of packets

Use `-c`:

```
sudo tcpdump -i eno1 -c 20
```

This captures 20 packets and exits automatically.

This is very useful when creating demonstrations or scripts.

---

9. Capture DNS traffic

DNS normally uses UDP port 53.

```
sudo tcpdump -i eno1 port 53
```

Then in another terminal:

```
dig google.com
```

You should see DNS packets.

A more specific filter:

```
sudo tcpdump -i eno1 udp port 53
```

For TCP DNS:

```
sudo tcpdump -i eno1 tcp port 53
```

---

10. Capture traffic from a particular host

```
sudo tcpdump -i eno1 host 8.8.8.8
```

This captures traffic to or from `8.8.8.8`.

For a particular IP:

```
sudo tcpdump -i eno1 host 192.168.1.100
```

---

11. Capture traffic to a particular host

```
sudo tcpdump -i eno1 dst host 8.8.8.8
```

Only packets going **towards** `8.8.8.8` are captured.

12. Capture traffic from a particular host

```
sudo tcpdump -i eno1 src host 192.168.1.100
```

Only packets originating from `192.168.1.100` are captured.

---

13. Capture TCP traffic

```
sudo tcpdump -i eno1 tcp
```

Capture TCP port 443:

```
sudo tcpdump -i eno1 tcp port 443
```

Capture SSH:

```
sudo tcpdump -i eno1 tcp port 22
```

HTTP:

```
sudo tcpdump -i eno1 tcp port 80
```

---

14. Capture UDP traffic

```
sudo tcpdump -i eno1 udp
```

DNS:

```
sudo tcpdump -i eno1 udp port 53
```

DHCP:

```
sudo tcpdump -i eno1 udp port 67 or udp port 68
```

---

15. Capture ICMP / ping traffic

Run:

```
sudo tcpdump -i eno1 icmp
```

Then from another terminal:

```
ping -c 3 google.com
```

You will see ICMP echo requests and replies.

This is a very good demonstration for students:

```
ping
  ↓
ICMP Echo Request
  ↓
Network
  ↓
ICMP Echo Reply
```

---

16. Observe TCP connection establishment

One of the most important things to learn with **`tcpdump`** is the **TCP three-way handshake**.

Run:

```
sudo tcpdump -i eno1 tcp port 443
```

Then:

```
curl https://google.com
```

You may see:

```
Flags [S]
Flags [S.]
Flags [.]
```

These represent:

```
Client                    Server
  |                         |
  | ---- SYN ------------>  |
  | <--- SYN + ACK ------- |
  | ---- ACK ------------>  |
  |                         |
```

This is the TCP three-way handshake.

17. Understanding TCP flags

Some important TCP flags are:

Flag	Meaning
S	SYN
.	ACK
F	FIN
R	RST
P	PSH
U	URG

For example:

```
Flags [S]
```

means SYN.

```
Flags [S.]
```

means SYN + ACK.

```
Flags [R.]
```

means RST + ACK.

---

18. Diagnose a connection timeout

Suppose:

```
nc -vz 74.188.224.19 443
```

does not connect.

Run:

```
sudo tcpdump -i eno1 host 74.188.224.19 and port 443
```

Then try:

```
nc -vz -w 5 74.188.224.19 443
```

Case 1 - SYN leaves but nothing returns

You may see:

```
Client > Server: Flags [S]
Client > Server: Flags [S]
Client > Server: Flags [S]
```

but no response.

This suggests something is **dropping the packets or responses**.

Potential causes:

- Firewall
- Routing problem
- Remote firewall
- Network filtering
- Server unavailable

Case 2 - SYN followed by RST

```
Client > Server: Flags [S]
Server > Client: Flags [R.]
```

The remote host is reachable, but the connection is being actively rejected.

Case 3 - SYN → SYN/ACK → ACK

```
[S]
[S.]
[.]
```

TCP connectivity is working.

This makes `tcpdump` extremely useful alongside `nc`.

19. Diagnose Arattai problem

Discovered earlier:

```
web.arattai.in
        ↓
74.188.224.19
```

We can now watch exactly what happens.

Start:

```
sudo tcpdump -i eno1 -nn host 74.188.224.19
```

Then, in another terminal:

```
curl -4 -v https://web.arattai.in
```

The `-nn` option is useful because it prevents tcpdump from trying to resolve IP addresses and port numbers into names.

For an even more focused capture:

```
sudo tcpdump -i eno1 -nn host 74.188.224.19 and tcp port 443
```

This is an excellent troubleshooting technique.

---

20. Why `-nn` is useful

Normally tcpdump may display:

```
google.com.https
```

instead of:

```
142.250.x.x.443
```

Use:

```
sudo tcpdump -nn -i eno1
```

to disable:

- Hostname resolution
- Port/service-name resolution

This makes packet captures **faster and easier to interpret**.

Use `-nn` during troubleshooting.

21. Display packet contents

Use:

```
sudo tcpdump -i eno1 -A
```

`-A` displays packet contents as ASCII where possible.

For example, with unencrypted HTTP traffic you may see:

```
GET / HTTP/1.1
Host: example.com
```

This is useful for learning HTTP.

HTTPS traffic, however, is encrypted, so you won't see the HTTP contents this way.

---

22. Display packet contents in hexadecimal

```
sudo tcpdump -i eno1 -X
```

This displays packet data in hexadecimal and ASCII.

Useful for lower-level protocol analysis.

---

23. Capture packets without resolving names

```
sudo tcpdump -nn -i eno1
```

This is one of the commands I recommend students learn early.

---

24. Save packets to a file

Instead of displaying packets:

```
sudo tcpdump -i eno1 -w capture.pcap
```

Stop with:

```
Ctrl+C
```

The resulting file:

```
capture.pcap
```

can later be analyzed using Wireshark or tcpdump.

25. Read a captured file

```
tcpdump -r capture.pcap
```

With numeric addresses:

```
tcpdump -nn -r capture.pcap
```

You can filter the saved capture:

```
tcpdump -nn -r capture.pcap tcp port 443
```

This is very useful for classroom exercises.

---

26. Capture DNS into a file

```
sudo tcpdump -i eno1 -nn -w dns.pcap port 53
```

Then run:

```
dig google.com
```

Stop tcpdump.

Read it:

```
tcpdump -nn -r dns.pcap
```

Students can therefore capture **real DNS packets** and study them later.

---

27. Capture only traffic from your computer to a server

For example:

```
sudo tcpdump -i eno1 -nn host 74.188.224.19
```

Or specifically HTTPS:

```
sudo tcpdump -i eno1 -nn host 74.188.224.19 and tcp port 443
```

28. Combine filters

`tcpdump` uses Boolean operators.

AND

```
sudo tcpdump -i eno1 host 192.168.1.100 and port 443
```

OR

```
sudo tcpdump -i eno1 port 80 or port 443
```

NOT

```
sudo tcpdump -i eno1 not port 22
```

For example:

```
sudo tcpdump -i eno1 -nn 'host 74.188.224.19 and tcp port 443'
```

Quoting complex filters is a good habit.

---

29. Monitor ARP traffic

```
sudo tcpdump -i eno1 arp
```

Then:

```
ip neigh
```

ARP is responsible for mapping:

```
IPv4 address → MAC address
```

For example:

```
Who has 192.168.1.1?
192.168.1.1 is at xx:xx:xx:xx:xx:xx
```

This is useful for diagnosing local-network problems.

30. Monitor DHCP

```
sudo tcpdump -i eno1 -nn 'udp port 67 or udp port 68'
```

This allows students to observe DHCP traffic when obtaining an IP address.

The basic process is:

```
DHCP Discover
      ↓
DHCP Offer
      ↓
DHCP Request
      ↓
DHCP ACK
```

---

31. Monitor IPv6

```
sudo tcpdump -i eno1 ip6
```

For IPv4:

```
sudo tcpdump -i eno1 ip
```

This is particularly useful when investigating:

```
IPv4 works

IPv6 doesn't work
```

---

32. Monitor a specific port

For example, SSH:

```
sudo tcpdump -i eno1 -nn tcp port 22
```

HTTPS:

```
sudo tcpdump -i eno1 -nn tcp port 443
```

DNS:

```
sudo tcpdump -i eno1 -nn port 53
```

NTP:

```
sudo tcpdump -i eno1 -nn port 123
```

---

33. Useful `tcpdump` options

Option	Purpose
`-i`	Select interface
`-D`	List interfaces
`-n`	Don't resolve hostnames
`-nn`	Don't resolve hostnames or ports
`-v`	Verbose
`-vv`	More verbose
`-c`	Capture specified number of packets
`-w`	Write capture to file
`-r`	Read capture file
`-A`	ASCII packet contents
`-X`	Hex + ASCII contents
`-s`	Snapshot length
`-t`	Don't print timestamp
`-q`	Less verbose output

34. A very useful troubleshooting workflow

For a website that doesn't work:

Step 1 - DNS

```
dig web.arattai.in
```

Step 2 - TCP

```
nc -vz -w 5 74.188.224.19 443
```

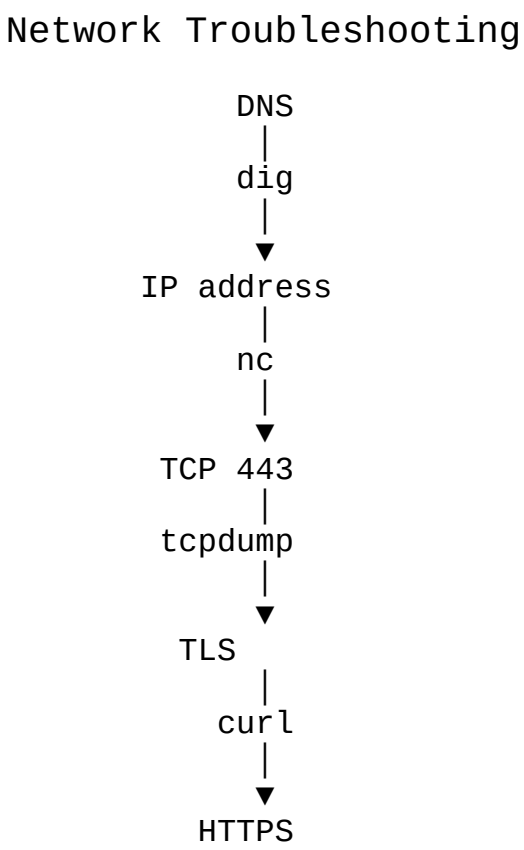
Step 3 - Packet capture

```
sudo tcpdump -i eno1 -nn host 74.188.224.19 and tcp port 443
```

Step 4 - HTTPS

```
curl -4 -v https://web.arattai.in
```

Troubleshooting progressively:



This is a **good practical methodology for Linux network troubleshooting.**

35. `tcpdump` (vs) other networking tools

Tool	Primary purpose
`ip`	Network configuration
`ping`	Basic IP/ICMP connectivity
`traceroute`	Network path
`dig`	DNS troubleshooting
`ss`	Socket/listening-port information
`nc`	TCP/UDP connectivity testing
`curl`	HTTP/HTTPS testing
`tcpdump`	**Packet capture and analysis**
`nmap`	Port/service discovery
`ethtool`	Ethernet interface diagnostics
`iw`	Wi-Fi diagnostics

The important distinction is:

`nc` tells you whether a connection can be established; **`tcpdump`** lets you see what happens to the packets while you try.

36. Five commands students should learn first

```
sudo tcpdump -D
sudo tcpdump -i eno1 -nn
sudo tcpdump -i eno1 -nn port 53
sudo tcpdump -i eno1 -nn tcp port 443
sudo tcpdump -i eno1 -nn -w capture.pcap
```

Once they understand these, move to **BPF filters, TCP flags, retransmissions, ARP, DHCP, DNS packet analysis, and Wireshark**.

The key lesson

`tcpdump` changes network troubleshooting from:

“The website is not working.”

to:

“DNS resolved correctly, the TCP SYN left my machine, no SYN-ACK came back, therefore I need to investigate the network path/firewall/server.”

9. lsof

What is `lsof` ?

`lsof` means **List Open Files and Network Connections**.

In Linux, almost everything is treated as a file-including **network sockets**. Therefore, **`lsof`** is a powerful troubleshooting tool for finding **which process is using a file, device, port, or network connection**.

It is particularly useful when a service is occupying a port, a file cannot be unmounted, or you need to identify the process behind a network connection.

---

1. Check whether `lsof` is installed

```
lsof -v
```

On Ubuntu/Debian:

```
sudo apt install lsof
```

---

2. List all open files

```
lsof
```

This can produce a large amount of output.

Useful columns include:

```
COMMAND   PID   USER   FD   TYPE   DEVICE   SIZE/OFF   NODE   NAME
```

- **COMMAND** - program name
- **PID** - process ID
- **USER** - user running the process
- **FD** - file descriptor
- **TYPE** - type of file
- **NAME** - file/device/socket name

3. Find files opened by a particular process

```
lsof -p 1234
```

Replace `1234` with the actual PID.

First find the PID:

```
pgrep nginx
```

Then:

```
sudo lsof -p $(pgrep nginx)
```

---

4. Find which process is using a port

This is one of the most useful networking applications of `lsof`.

```
sudo lsof -i :80
```

For HTTPS:

```
sudo lsof -i :443
```

Example:

```
COMMAND   PID   USER   FD   TYPE  DEVICE  NODE  NAME
nginx    1523   root    6u  IPv4  ...     TCP   *:http (LISTEN)
```

This tells you that **nginx is listening on port 80**.

---

5. Find all listening TCP ports

```
sudo lsof -iTCP -sTCP:LISTEN
```

This is useful when troubleshooting:

"Which services are listening on my system?"

6. Find all network connections

```
sudo lsof -i
```

To display only TCP connections:

```
sudo lsof -iTCP
```

Only UDP:

```
sudo lsof -iUDP
```

---

7. Find network connections belonging to a user

```
sudo lsof -i -u username
```

Replace `username` with the required Linux user.

---

8. Find which process has a file open

```
sudo lsof /var/log/syslog
```

This can be useful when investigating:

- Log files
- Configuration files
- Database files
- Mounted filesystems

---

9. Find processes using a filesystem

```
sudo lsof /mnt/data
```

This is particularly useful when you receive an error such as:

```
umount: target is busy
```

You can investigate which processes are still using the filesystem.

Practical Networking Exercise

Situation

Suppose you want to know:

Which application is using TCP port 8080?

Run:

```
sudo lsof -i :8080
```

You may see:

```
COMMAND   PID   USER   FD   TYPE  DEVICE  NODE  NAME
python3  2451  user    3u  IPv4  ...     TCP   *:8080 (LISTEN)
```

Now you know:

```
Port 8080
   ↓
TCP
   ↓
python3
   ↓
PID 2451
```

You can investigate the process further:

```
ps -fp 2451
```

`lsof` (vs) `ss`

Both are useful, but they answer slightly different questions.

Tool	Best for
`ss`	Examining sockets and network connections
`lsof`	Connecting **processes** to files, ports and sockets
`netstat`	Older/general network statistics tool

Remember

`ss` - asks: "What network connections exist?"

`lsof` - asks: "Which process is responsible for this connection?"

Quick Reference

```
lsof
```

List open files

```
lsof -p PID
```

Files opened by a process

```
sudo lsof -i
```

Network connections

```
sudo lsof -i :80
```

Who is using port 80

```
sudo lsof -iTCP -sTCP:LISTEN
```

TCP listening services

```
sudo lsof -iUDP
```

UDP activity

```
sudo lsof /path/to/file
```

Processes using a file

```
sudo lsof /mount/point
```

Processes using a filesystem

`lsof` is a powerful Linux troubleshooting tool for connecting files, processes, devices, ports and network sockets together.