

Lesson 02 — How Linux Decides Where Packets Go

Every packet that enters or leaves a Linux machine must be routed. The kernel consults a set of rules and routing tables to decide the next hop — or drops the packet if no match exists.

- **Routing table** — the primary data structure: prefix → gateway/interface
- **Longest-prefix match** — most specific route always wins
- **Policy routing** — multiple tables, selected by `ip rule`
- **Netfilter integration** — routing decision splits traffic to INPUT vs FORWARD

Notes: This lesson builds directly on Lesson 01's netfilter flow. By the end students will understand every decision the kernel makes before a packet reaches a socket or hits a forwarding chain.

What Does "Routing" Mean?

For every packet, the kernel asks one question:

"Given this destination IP, which interface do I send out, and who is the next hop?"

- **Locally originated packets** — app writes to socket → kernel routes to outbound interface
- **Forwarded packets** — arrive on interface A, routing decides interface B to send them on
- **Locally destined packets** — routing detects the destination belongs to this host → delivers to a socket
- **No match?** — packet is dropped (or ICMP unreachable is sent)

The kernel resolves this question by walking the **routing table** and applying **longest-prefix-match**.

Notes: Emphasise that routing is a purely kernel-space operation. User-space only configures the tables; the kernel does the per-packet lookup at wire speed.

The Routing Table, Line-by-Line

```
$ ip route show
default          via 192.168.1.1 dev eth0    proto dhcp  metric 100
10.0.5.0/24      via 10.0.5.1   dev eth1    proto static
10.0.0.0/16      via 10.0.1.1   dev eth1    proto static
192.168.1.0/24   dev eth0       proto kernel scope link src 192.168.1.50
```

Each row: **prefix** → **gateway** (via) + **interface** (dev). The `metric` breaks ties between equal-length prefixes. `scope link` means directly reachable — no gateway needed.

Destination 10.0.5.42 arrives — which row wins?

10.0.5.42

default via 192.168.1.1 /0 (least specific)

10.0.0.0/16 via 10.0.1.1 /16

10.0.5.0/24 via 10.0.5.1 /24 (more specific — winner!)

10.0.6.0/24 via 10.0.6.1 /24 (doesn't match)

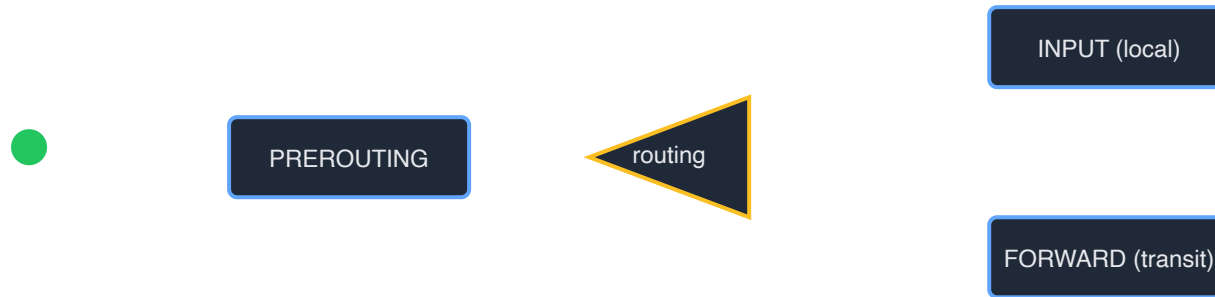
Linux walks the table by specificity (longest prefix first). The first match that covers the destination wins.

Routing in the Netfilter Flow

Recall from Lesson 01: packets traverse netfilter hooks in a fixed order. The **routing decision** sits between PREROUTING and the fork to INPUT or FORWARD.

- **Destination = this host** → routing sends packet UP to `INPUT` chain
- **Destination = elsewhere** → routing sends packet ACROSS to `FORWARD` chain
- DNAT in PREROUTING can change the destination *before* the routing decision

Two packets enter — routing forks them by destination.



Default / Local / Link Routes

The Linux kernel maintains several built-in route types that you will see in every routing table:

Type	Example	Meaning	Added by
link	<code>192.168.1.0/24 dev eth0 scope link</code>	Directly reachable — ARP, no gateway needed	Kernel on <code>ip addr add</code>
local	<code>local 192.168.1.50 dev eth0</code>	Address belongs to this host → deliver to socket	Kernel on <code>ip addr add</code>
default	<code>default via 192.168.1.1 dev eth0</code>	Last-resort gateway; prefix /0 matches everything	DHCP client or admin
static	<code>10.10.0.0/24 via 10.0.0.1 dev eth1</code>	Manually configured persistent route	<code>ip route add</code> / network config
blackhole	<code>blackhole 192.0.2.0/24</code>	Silently discard matching packets	Admin (security / null routing)

Notes: The `local` table is a separate routing table (table 255) that the kernel always consults first. Students can inspect it with `ip route show table local`.

Adding & Deleting Routes

Use `ip route` to manage routes at runtime. Changes take effect immediately but are **not persistent** across reboots unless written to network configuration files.

```
# Add a static route for a subnet via a specific gateway
sudo ip route add 10.0.0.0/24 via 192.168.1.1 dev eth0

# Add with a metric (lower = preferred)
sudo ip route add 10.0.0.0/24 via 192.168.1.1 dev eth0 metric 50

# Add a blackhole route (silently drop)
sudo ip route add blackhole 192.0.2.0/24

# Delete a route
sudo ip route del 10.0.0.0/24

# Replace an existing route atomically
sudo ip route replace 10.0.0.0/24 via 10.0.0.254 dev eth1
```

Persistence tip

On systemd-networkd: add a `[Route]` section to `/etc/systemd/network/*.network`. On NetworkManager: use `nmcli` or the GUI.

Multiple Routing Tables

Linux supports up to **252 user-defined routing tables** (plus built-ins: `local`, `main`, `default`). Named tables are registered in `/etc/iproute2/rt_tables`.

```
# /etc/iproute2/rt_tables
#
# Reserved values
255    local
254    main
253    default
0      unspec
#
# Local
100    isp1
200    isp2
300    vpn
```

- **main** (table 254) — the default table; what `ip route show` displays
- **local** (table 255) — kernel-managed; contains host and broadcast addresses
- **Custom tables** — selected by `ip rule` based on packet attributes

```
# List all routes in a specific table
ip route show table isp1

# Add a route into a named table
sudo ip route add default via 203.0.113.1 table isp1
```

Notes: Multi-table routing is the foundation of ISP dual-homing, VPN split routing, and container networking. The policy routing engine (next slide) is what decides which table to use.

Policy Routing with `ip rule`

Before consulting any routing table, the kernel walks the **rule list** (sorted by priority). The first matching rule names the table to use.

```
# View current rules (lower priority number = checked first)
ip rule show

# Route traffic from subnet 10.10.0.0/24 via table 100
sudo ip rule add from 10.10.0.0/24 lookup 100

# Route by fwmark (set by iptables/nftables MARK target)
sudo ip rule add fwmark 0x10 lookup 200

# Route by destination
sudo ip rule add to 8.8.0.0/16 lookup vpn

# Delete a rule
sudo ip rule del from 10.10.0.0/24 lookup 100
```

Packet has src=10.10.0.5 — walk the rule list, first match selects a table.

src=10.10.0.5

0: from all lookup local

100: from 192.168.0.0/16 lookup 100

200: from 10.10.0.0/24 lookup 200 ← matches!

32766: from all lookup main (default)

32767: from all lookup default

Network Namespaces — Your Free Lab

Network namespaces give each namespace its own routing table, interfaces, and firewall rules. You can build a full multi-router topology with zero hardware.

```
# Create two namespaces
ip netns add red
ip netns add blue

# Create a veth pair (virtual Ethernet cable)
ip link add veth0 type veth peer name veth1

# Move one end into each namespace
ip link set veth0 netns red
ip link set veth1 netns blue

# Assign addresses inside each namespace
ip netns exec red ip addr add 10.0.0.1/24 dev veth0
ip netns exec blue ip addr add 10.0.0.2/24 dev veth1
```

Lab preview

Today's lab uses exactly this pattern to build a two-namespace topology and test policy routing between them.

Diagnostic Toolkit

These three tools answer the most common routing questions in production:

```
# ip route get — show EXACTLY which route the kernel will use for a destination
ip route get 8.8.8.8
# 8.8.8.8 via 192.168.1.1 dev eth0 src 192.168.1.50 uid 1000

# Also test from a specific source (policy routing aware)
ip route get 8.8.8.8 from 10.10.0.5

# traceroute — show each hop to a destination
traceroute 8.8.8.8
# Uses UDP by default; add -T for TCP, -I for ICMP

# mtr — combines traceroute + continuous ping; great for diagnosing flapping
mtr --report 8.8.8.8
mtr --report --tcp --port 443 8.8.8.8
```

ip route get

mtr

Fastest first check — tells you *exactly* what the kernel decides, including which rule matched and which table was consulted.

Real-time hop-by-hop latency and packet loss. Invaluable for spotting asymmetric routing or a bad transit link.

Common Gotchas

- **Asymmetric routing** — packets leave via interface A, replies arrive via interface B. Stateful firewalls and `rp_filter` will silently drop the returning traffic. Fix: ensure return routes mirror outbound paths, or tune `rp_filter=2` (loose mode).
- **Missing return route** — you can reach a remote host but it cannot reply. Classic cause: you added a route on the client but forgot the server's default gateway.
- `rp_filter` (**Reverse Path Filter**) — kernel drops packets whose source IP would not be reachable on the same interface. Check: `sysctl net.ipv4.conf.all.rp_filter`. Set to `0` to disable, `2` for loose validation.
- **Metric collisions** — two routes with the same prefix and metric cause non-deterministic selection (ECMP unless configured). Always set explicit metrics.
- **DNAT vs routing order** — DNAT in PREROUTING fires *before* the routing decision. If you DNAT a packet to a new destination, the routing lookup uses the *new* destination address.
- **IPv6 surprises** — `ip -6 route` is a separate table. A working IPv4 route does not imply a working IPv6 route.

Notes: The `rp_filter` gotcha is extremely common in multi-homed servers and VPN setups. Worth a dedicated

live demo.

Recap & Next Steps

- **Routing decision** — kernel asks "which interface + next hop?" for every packet
- **Longest-prefix match** — most specific prefix always wins; metric breaks ties
- **Routing in netfilter** — sits between PREROUTING and INPUT/FORWARD fork
- **Multiple tables** — `local`, `main`, `default`, plus up to 252 user-defined
- **Policy routing** — `ip rule` selects the table by src IP, dst IP, fwmark, or interface
- **Namespaces** — each has its own isolated routing universe; perfect for lab work
- **Diagnostics** — `ip route get` for instant answer; `traceroute` / `mtr` for path analysis

Coming up: Apply L1 + L2 together in real scenarios. Lesson 03 covers NAT — how MASQUERADE, SNAT, and DNAT interact with the routing table you now understand. The lab combines netfilter rules with policy routing for a practical dual-ISP setup.

