

Lesson 03 — Policy-Based Routing

How Linux picks between multiple routing tables.

Linux Networking

Notes: This lesson builds on Lesson 02's routing table foundations. Students should already know `ip route`, longest-prefix-match, and the netfilter flow. Today we introduce the rule list that sits in front of every routing table lookup.

The routing table is just one table

Lesson 02 showed how the kernel consults the **main** routing table for every packet. But Linux actually ships with three built-in tables:

- **local** (table 255) — kernel-managed; host addresses and broadcasts. Always consulted first.
- **main** (table 254) — what `ip route show` displays. Your static routes live here.
- **default** (table 253) — historically for postrouting; almost never populated today.

Plus up to **252 user-defined tables**, each with its own independent prefix set. The question is: *who decides which table to use?*

Answer: `ip rule`

A sorted list of *rules* maps packet attributes → table IDs, evaluated before any table lookup.

Notes: Quick recap slide — move through it fast if students remember Lesson 02. The key hook is the question at the end: "who decides which table?" That's what this whole lesson answers.

Two lookups, not one

When the kernel needs to route a packet, it performs two distinct lookups in sequence:

Step 1: `ip rule` picks a table

Walk the rule list by priority. The first matching rule names a routing table. If no user rule matches, fall through to `main` (priority 32766).

Step 2: that table is consulted

Perform a normal longest-prefix-match lookup inside the selected table. If the table has no match, the rule list continues to the next rule (not necessarily `main`).

This two-step process lets you route **different traffic classes through entirely different forwarding paths** — without touching the main table at all.

Notes: Emphasise the asymmetry: Step 1 is a linear scan (short list, $O(N)$), Step 2 is a trie lookup (large table, $O(\log N)$). We'll see why this matters in slide 11.

Anatomy of an `ip rule`

Every rule has three parts: a **priority**, a **selector**, and a **target table**.

```
ip rule add pref 100 from 10.10.0.0/24 lookup 200
```

- `pref 100` — priority; lower numbers are evaluated first
- `from 10.10.0.0/24` — selector: match packets whose *source* address falls in this subnet
- `lookup 200` — target: if this rule matches, consult routing table 200

Packet metadata: src=10.10.0.5 fwmark=0x42 iif=eth0

src=10.10.0.5 fwmark=0x42 iif=eth0

pref 0: from all lookup local

pref 50: from 192.168.0.0/16 lookup 50

pref 80: iif wlan0 lookup 80

pref 100: from 10.10.0.0/24 fwmark 0x42 lookup 200 ← match!

pref 32766: from all lookup main

Selectors at a glance

A single rule can combine multiple selectors — all must match for the rule to fire.

Selector	Example	Matches when...
<code>from</code>	<code>from 10.10.0.0/24</code>	Source address is within the given prefix
<code>to</code>	<code>to 8.8.0.0/16</code>	Destination address is within the given prefix
<code>fwmark</code>	<code>fwmark 0x42</code>	Packet carries the given netfilter mark (set by iptables/nftables <code>MARK</code>)
<code>iif</code>	<code>iif eth1</code>	Packet arrived on the named interface
<code>oif</code>	<code>oif eth0</code>	Packet is leaving on the named interface (locally-originated traffic)
<code>tos</code>	<code>tos 0x10</code>	IP Type of Service / DSCP field matches

Notes: `fwmark` is the most powerful — it lets netfilter "tag" packets with arbitrary 32-bit values before the routing decision. This is the key to VPN policy routing, QoS, and split tunnels.

Use case 1 — Source-based routing

A server has two uplinks (eth0 → ISP-A, eth1 → ISP-B). Traffic from subnet `10.10.0.0/24` must always leave via ISP-B, regardless of the destination.

```
# Table 200 gets its own default route via ISP-B
ip route add default via 203.0.113.1 dev eth1 table 200

# Rule: any packet sourced from 10.10.0.0/24 uses table 200
ip rule add from 10.10.0.0/24 lookup 200 pref 100

# All other traffic falls through to table main → ISP-A
ip route add default via 198.51.100.1 dev eth0
```

Why not just change the main table?

Because a single default route can't express "use this gateway *for some sources* and that gateway *for others*." PBR handles this elegantly without NAT or complex iptables rules.

Notes: Classic ISP dual-homing scenario. Works for outbound traffic. Inbound traffic is handled by BGP or DNS-based load balancing — out of scope for this lesson.

Use case 2 — fwmark-driven routing

Netfilter's `MARK` target paints a 32-bit tag onto a packet in the `mangle` table. The `ip rule fwmark` selector then routes by that tag.

```
# Mark packets from VPN clients with 0x42
iptables -t mangle -A PREROUTING -i tun0 -j MARK --set-mark 0x42

# Route all marked packets through table 200
ip rule add fwmark 0x42 lookup 200 pref 100

# Table 200's default: force through the secondary WAN
ip route add default via 10.99.0.1 dev eth1 table 200
```

Netfilter MANGLE paints a mark, ip rule routes by the mark



Persisting marks across a flow — CONNMARK

MARK paints one packet. CONNMARK makes the mark sticky.

CONNMARK saves a mark to the conntrack entry, so every subsequent packet in the flow inherits it — even before the mangle rule fires again.

```
# Three-rule mangle chain for CONNMARK
# 1. Restore saved mark for ESTABLISHED/RELATED packets
iptables -t mangle -A PREROUTING -m conntrack --ctstate ESTABLISHED,RELATED \
-j CONNMARK --restore-mark

# 2. Mark NEW packets
iptables -t mangle -A PREROUTING -m conntrack --ctstate NEW \
-i tun0 -j MARK --set-mark 0x42

# 3. Save the mark back to conntrack for future packets
iptables -t mangle -A PREROUTING -j CONNMARK --save-mark
```

Packet 1 sets the mark and saves it to conntrack. Packets 2-4 inherit it.

MARK --set-mark

--save-mark

--restore-mark

ip rule fwmark

P1 EW

P2 EST

P3 EST

P4 EST

conntrack entry
10.0.0.5 → 1.1.1.1:80

Use case 3 — Multi-WAN with metrics

Two default routes with different metrics provide automatic failover. Lower metric = preferred.

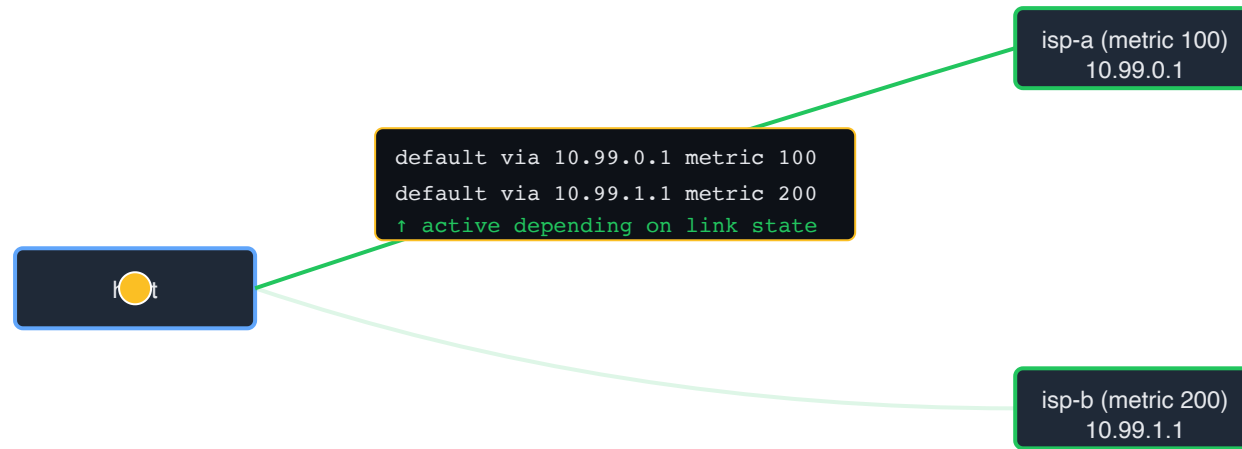
When the primary link goes down, the kernel falls back to the higher-metric route automatically.

```
# Primary default (metric 100 — preferred)
ip route add default via 10.99.0.1 dev eth0 metric 100

# Backup default (metric 200 — used only if eth0 fails)
ip route add default via 10.99.1.1 dev eth1 metric 200

# Optional: source-policy overlay to guarantee return-path symmetry
ip rule add from 10.99.0.0/24 lookup isp-a pref 100
ip rule add from 10.99.1.0/24 lookup isp-b pref 110
```

Primary WAN fails; traffic re-routes to backup.



The `rp_filter` gotcha

When you add policy routing, return packets may arrive on an interface that *doesn't match* the route the kernel would use to reach the sender. The **Reverse Path Filter** will silently drop these packets.

Symptom

You can ping out, but replies never arrive. `ip route get` shows the right table. `conntrack -L` shows the flow. But packets vanish.

```
# Check current rp_filter values
sysctl net.ipv4.conf.all.rp_filter
sysctl net.ipv4.conf.eth0.rp_filter

# 0 = off   |  1 = strict (default) |  2 = loose
# For PBR, use loose mode (2) on the relevant interface
sysctl -w net.ipv4.conf.eth0.rp_filter=2
sysctl -w net.ipv4.conf.all.rp_filter=2
```

```
# Persist in /etc/sysctl.d/  
echo "net.ipv4.conf.all.rp_filter=2" >> /etc/sysctl.d/99-pbr.conf
```

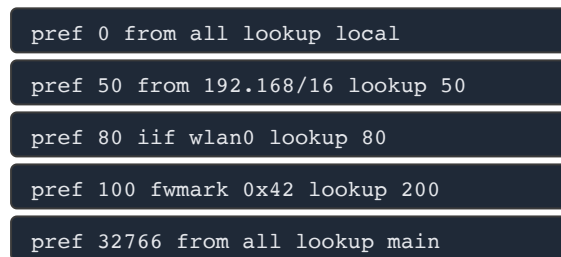
Why loose, not off?

Loose mode (2) accepts packets if *any* route leads back to the source — catching obvious spoofs. Strict mode (1) requires the packet to arrive on the *same* interface the kernel would use to reach the source, which breaks asymmetric PBR setups.

Under the hood — iproute2 internals

Understanding the data structures behind `ip rule` and `ip route` helps you reason about performance and design rule lists correctly.

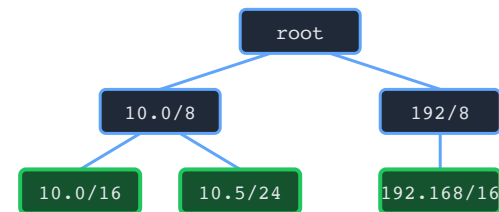
ip rule: sorted linear array



$O(N)$ rules) per packet

walked linearly, top-to-bottom
keep the list short

FIB: LC-trie (longest prefix)



$O(\log N)$ routes) per packet

LC-trie + per-CPU FIB
(per-flow route cache removed
in kernel 3.6, year 2012)

Rules are linear, FIB is a tree.
Order matters in ip rule. Specificity matters in ip route.

Toolkit

Five commands cover 90% of PBR debugging:

```
# 1. Show the rule list (and which table each rule points to)
ip rule show

# 2. Show routes in a specific table
ip route show table 200

# 3. Show all tables at once (filter noise)
ip route show table all | grep -v "^broadcast\|^local\|^multicast"

# 4. Simulate the full two-step lookup for a given source/destination
ip route get 8.8.8.8 from 10.10.0.5

# 5. Watch conntrack marks on live traffic
conntrack -L -o extended | grep mark
```

Notes: `ip route get ... from ...` is the most underused command in Linux networking. It tells you exactly which table the kernel selected and which route it matched, in one call.

Common mistakes

- **Forgetting the return route.** Adding a rule for outbound traffic (src-based) without also ensuring replies come back on the same interface creates asymmetric routing. Always add a matching rule for the source subnet of each WAN gateway.
- **rp_filter=1 in strict mode.** Strictly correct packets dropped silently. Check with `sysctl net.ipv4.conf.all.rp_filter` first before spending hours in tcpdump.
- **Rule list explosion.** Each rule is walked linearly per packet. 50+ rules cause measurable latency on high-pps routers. Consolidate by using fwmark instead of many source-IP ranges.
- **Forgetting CONNMARK for stateful flows.** MARK alone only marks a single packet. If you need all packets in a TCP flow to use the same path, add `--save-mark` / `--restore-mark`.
- **Rules not persistent across reboots.** `ip rule add` is ephemeral like `ip route add`. Write rules to `/etc/iproute2/rt_rules` or your distro's network configuration (NetworkManager, systemd-networkd, ifupdown).

Notes: The first two bullets account for the majority of "PBR isn't working" issues in production. Spend extra time on them during the live session.

Recap

- **Two-step routing** — `ip rule` picks a table; that table is consulted for the longest prefix match.
- **Selectors** — `from`, `to`, `fwmark`, `iif`, `oif`, `tos`; combine freely, lower `pref` wins.
- **CONNMARK** — `--save-mark` / `--restore-mark` makes marks sticky to the entire conntrack flow.
- **Watch out for** `rp_filter` (use loose mode=2), return-route symmetry, and rule list length.

Where next: VRF-lite (`ip link add vrf0 type vrf table 100`), IPv6 policy routing (`ip -6 rule`), and traffic shaping with `tc` using fwmark as a classifier.

