



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Multiprotocol Label Switching (MPLS)

Phạm Huy Hoàng - SoICT/HUST
hoangph@soict.hust.edu.vn

MPLS Plan

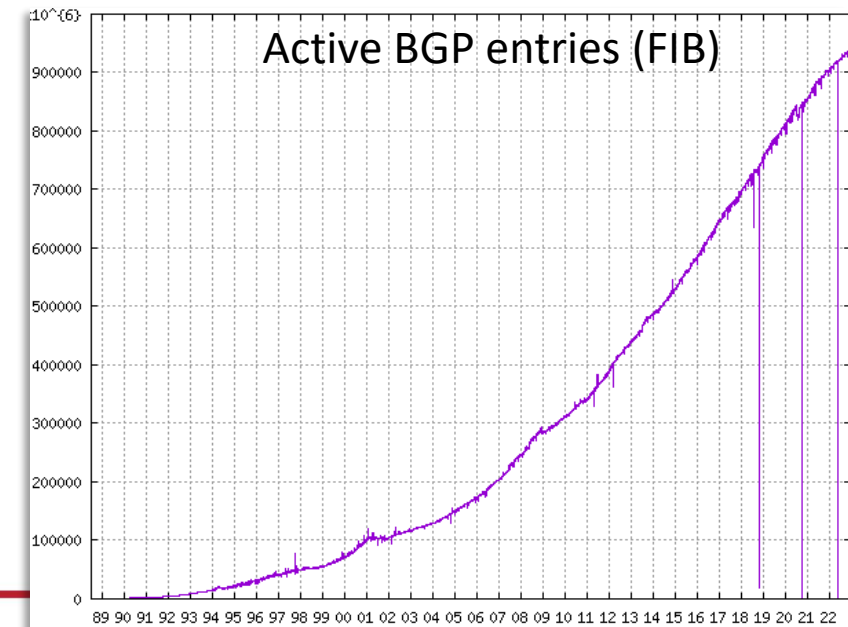
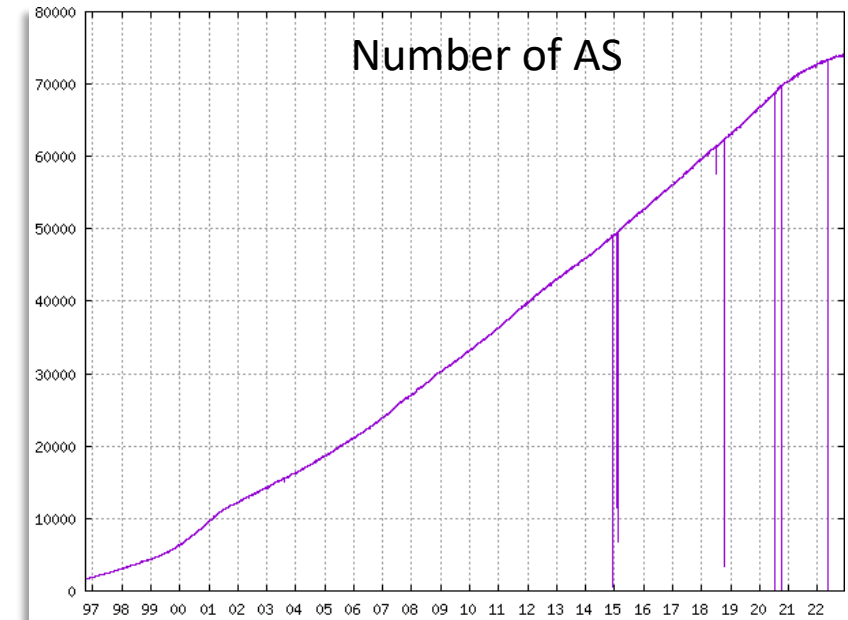
- Quick start
- MPLS Architecture
- Label Distribution Protocol (LDP)
- Appendix: Linux Traffic Control

MPLS Quick start

Some Internet backbone facts

- BGP remind: when an ISP (or AS) acquires a block of globally IP addresses (called a prefix), it advertises it to the global Internet routing table → all other ASes in the world to find out that the new prefix is now alive, and also how and where to send any IP packets destined for it.
- The number of prefixes in the global Internet routing table is increasing at an alarming rate.
- Most organisations have dealt with this problem by throwing money at it. They go to vendors such as Cisco Systems or Juniper Networks and buy their routers. Those are purpose-built to be able to perform high-speed routing forwarding of network traffic even if the routing table should grow past a million routes

Source: CIDR report (<https://www.cidr-report.org/as2.0>)



MPLS Motivation

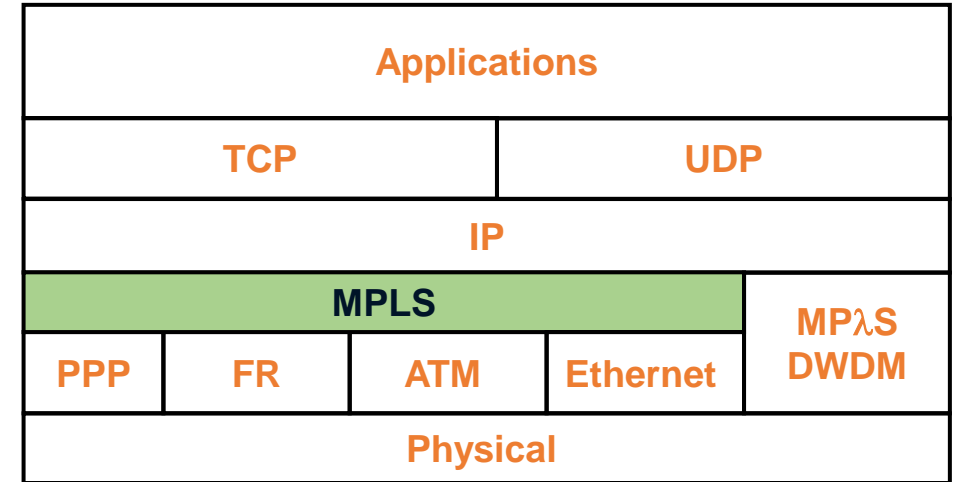
- Growth and evolution of the Internet
 - The need to evolve routing algorithm
 - The need for advanced forwarding algorithm
- Routing vs. Switching
 - routing: flexibility (in term of range – IGP/EGP)
 - switching: price/performance (fixed length label lookup faster than longest match used by IP routing)
- Can we forward/switch IP packets?
 - Allow speed of L2 switching at L3
 - Router makes L3 forwarding decision based on a single field:
 - no routing algorithm
 - similar to L2 forwarding
 - → Sppppppeeaaaa!!!
- IETF MPLS
 - <https://datatracker.ietf.org/wg/mpls/about>
 - RFC3031 (2001): MPLS Architecture
 - RFC3036/3037 (2001), RFC5036 (2007), RFC7552 (2015): MPLS LDP
 - MPLS (Active WG): 2010 ~ 2021 (<https://tools.ietf.org/wg/mpls>)

Routing performance:

- List tất cả các mạng đích của Internet vào routing table
- Tra danh sách prefix (network) được các AS export (xem số lượng bao nhiêu): <https://hackertarget.com/as-ip-lookup>
- No index matching: áp dụng lần lượt từng netmask lên địa chỉ IP đích cho mỗi gói tin IP đi đến
- Vết cạn: do yêu cầu classless routing → phải thử hết bảng routing để tìm netmask dài nhất

MPLS is a layer 2.5 protocol

- Experience & proposal
 - IP over ATM
 - IP Switching by Ipsilon
 - Cell Switching Router (CSR) by Toshiba
 - Tag switching by Cisco
 - Aggregate Route-based IP Switching (IBM)
- IETF: MPLS layer
 - When a layer is added, no modification is needed on the existing layers
 - However, traditional IP implementation does not understand MPLS header (forwarded from lower level – Ethernet, PPP, etc...)
 - Need MPLS implementation either in each layer-2 devices, or in **router OS**

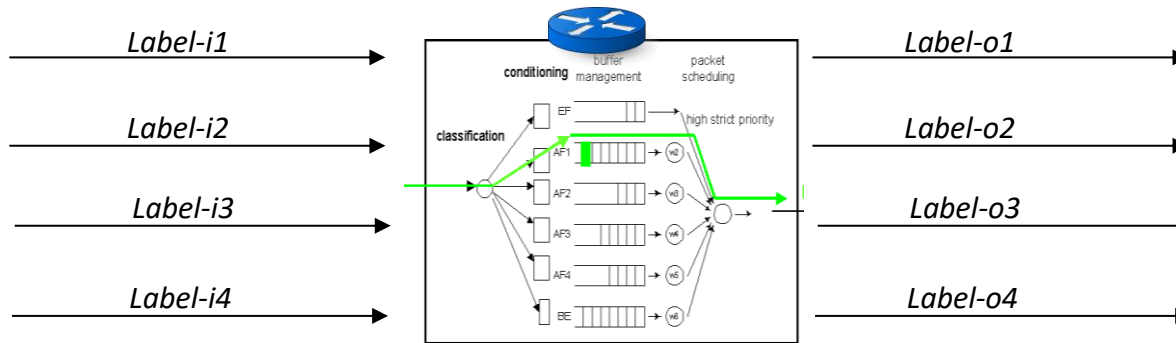


```

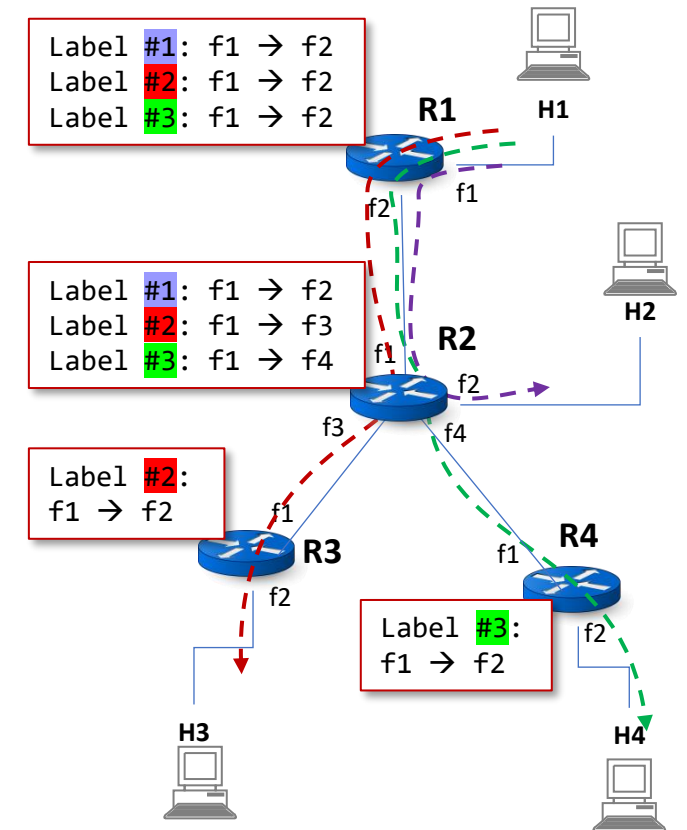
~$ sudo tcpdump -i enp0s9 -env
tcpdump: listening on enp0s9, link-type EN10MB (Ethernet), capture
size 262144 bytes
11:02:25.161445 08:00:27:1c:c2:15 > 08:00:27:61:de:17,
ethertype MPLS unicast (0x8847), length 102: MPLS (label
123, exp 0, [S], ttl 255)
(tos 0x0, ttl 64, id 37895, offset 0, flags [DF],
proto ICMP (1), length 84)
192.168.1.120 > 192.168.2.123:
ICMP echo request, id 16, seq 1085, length 64
    
```

How MPLS works

- Instead of routing based on IP destination address, routers switch packets by a label in packet header (much quicker!!!)
- Routing table → pre-defined label information (switching rules):
 - input label (upstream interface) → output label (downstream interface)

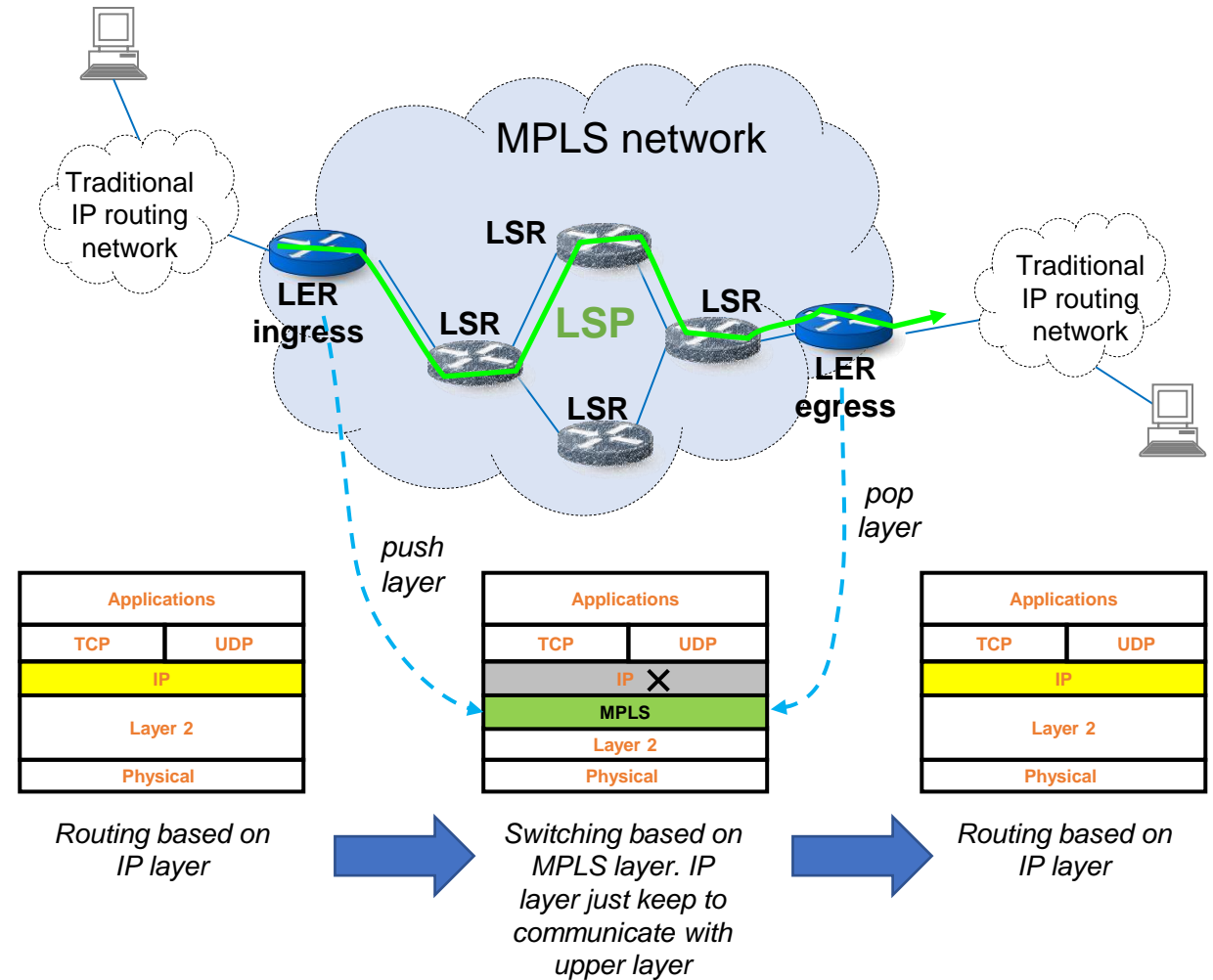


- Label Switch Path (LSP):
 - Fix the path in MPLS network
 - Pre-defined by network administrator in all participated routers



MPLS deployment IP world

- Inside a MPLS network, routers (called Label Switch Router – LSR) make use of MPLS layer with label in each packet to switch packet from upstream to downstream interface
- MPLS network can inter-network with traditional IP routing networks. LSR is the router inside MPLS network, Label Edge Router (LER) is the router to connect with traditional routing network
- At the incoming, a MPLS layer is pushed into the packets with a “label” by ingress LER.
- Packets are forwarded along a LSP where each LSR makes forwarding decisions based solely on the contents of the MPLS layer (no use of IP routing anymore)
- LSPs are established by network operators for a variety of purposes (such as QoS). In many ways, LSPs are no different than circuit-switched paths, or connection-oriented links.
- At the outgoing, egress LER pop the MPLS layer from packet and forward to traditional IP network



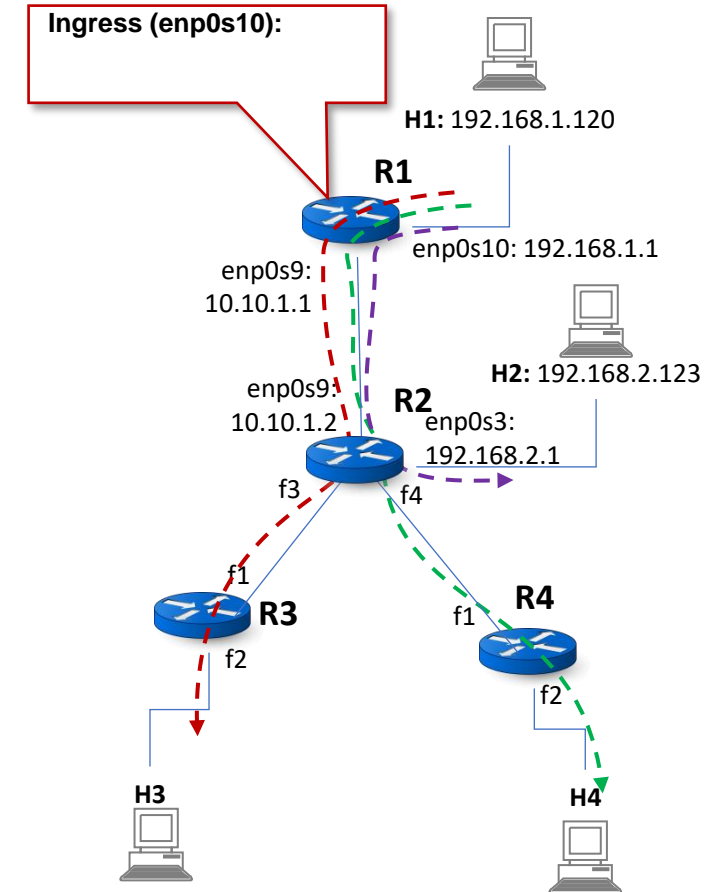
MPLS Demo

- MPLS network gồm 4 router.
- Cần thiết lập 3 LSP. R1 là ingress LER. Tùy vào mỗi LSP mà router cuối cùng có vai trò là egress LER:
 - H1 → R1 → R2 → H2
 - H1 → R1 → R2 → R3 → H3
 - H1 → R1 → R2 → R4 → H4
- Trên R1, mặc định có các qdisc *fq_codel* và không có filter:

```
R1:~$ sudo tc qdisc show
qdisc fq_codel 0: dev enp0s9 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms
interval 100.0ms memory_limit 32Mb ecn
qdisc fq_codel 0: dev enp0s10 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms
interval 100.0ms memory_limit 32Mb ecn
R1:~$ tc filter show dev enp0s10
```

- Thiết lập qdisc *ingress* cho R1:

```
R1:~$ sudo tc qdisc add dev enp0s10 handle ffff: ingress
R1:~$ sudo tc qdisc show dev enp0s10
qdisc fq_codel 0: dev enp0s10 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms
interval 100.0ms memory_limit 32Mb ecn
...
```



MPLS Demo (2)

- Thiết lập filter cho ingress LER R1: gói IP gửi đến H2 thì chèn MPLS vào với label #123.

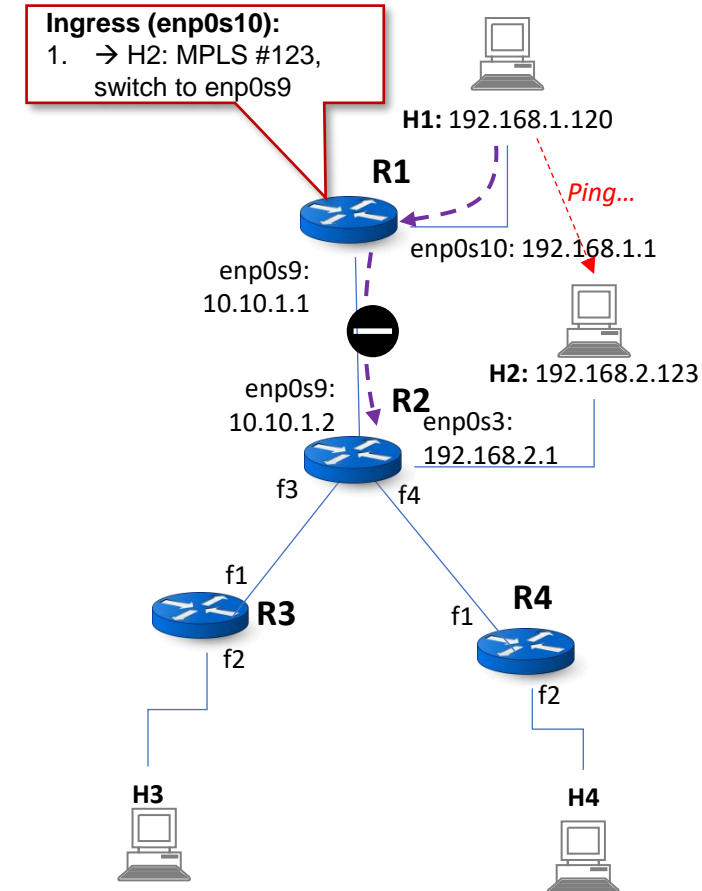
```
R1:~$ sudo tc filter add dev enp0s10 protocol ip parent ffff: \
    flower dst_ip 192.168.2.0/24 \
    action mpls push protocol mpls_uc label 123 \
    action mirrored egress redirect dev enp0s9
```

- Ping từ H1 đến H2 và bắt gói tin tại các điểm trên đường truyền. R1 đã thực hiện vai trò ingress LER (chèn layer MPLS vào gói ICMP với label #123)

```
R1:~$ sudo tcpdump -i enp0s9 -env
tcpdump: listening on enp0s9, link-type EN10MB (Ethernet), capture size 262144 bytes
11:02:25.161445 08:00:27:1c:c2:15 > xx:xx:xx:xx:xx:xx, ethertype MPLS unicast (0x8847),
length 102: MPLS (label 123, exp 0, [S], ttl 255)
(tos 0x0, ttl 64, id 37895, offset 0, flags [DF], proto ICMP (1), length 84)
192.168.1.120 > 192.168.2.123: ICMP echo request, id 16, seq 1085, length 64
```

- Bắt gói tin trên R2: chưa nhận được gói tin MPLS chuyển từ R1. Lý do là dest MAC khi R1 switch từ enp0s10 sang enp0s9 vẫn giữ nguyên các địa chỉ MAC → R2 không nhận được gói tin (Ethernet frame)

```
R2:~$ sudo tcpdump -i enp0s9 -env
```



MPLS Demo (3)

- Bổ sung thêm luật sửa dest MAC khi switch từ enp0s10 sang enp0s9:

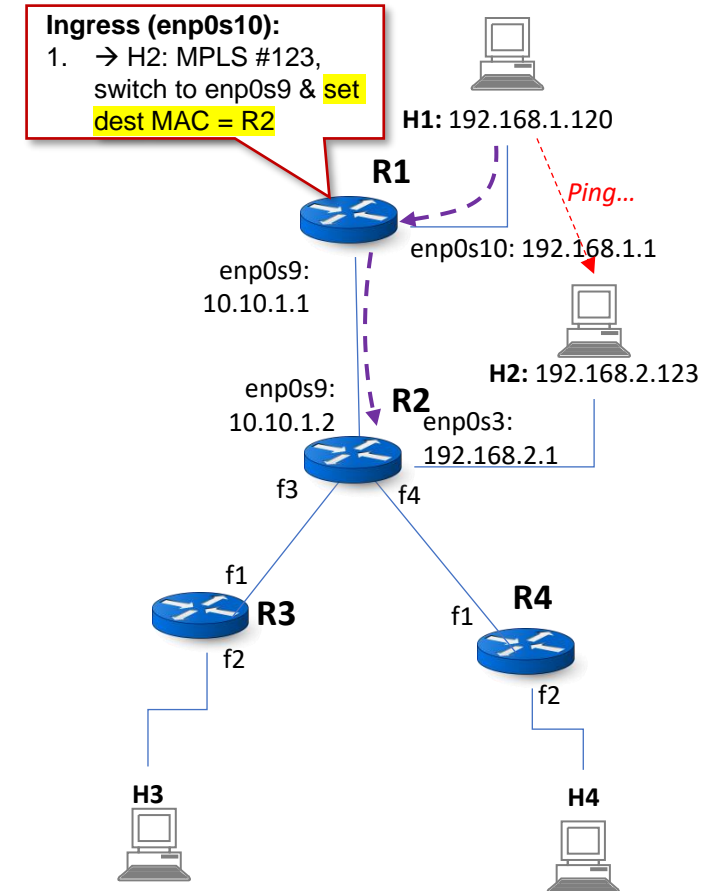
```
R1:~$ sudo tc filter add dev enp0s10 protocol ip parent ffff: \
    flower dst_ip 192.168.2.0/24 \
    action mpls push protocol mpls_uc label 123 \
    action skbmod set dmac 08:00:27:61:de:17 \
    action mirred egress redirect dev enp0s9
```

- Bắt gói tin trên R2: đã nhận được gói tin MPLS chuyển từ R1.

```
R2:~$ sudo tcpdump -i enp0s9 -env
tcpdump: listening on enp0s9, link-type EN10MB (Ethernet), capture size 262144 bytes
11:02:25.161445 08:00:27:1c:c2:15 > 08:00:27:61:de:17, ethertype MPLS unicast (0x8847),
length 102: MPLS (label 123, exp 0, [S], ttl 255)
(tos 0x0, ttl 64, id 37895, offset 0, flags [DF], proto ICMP (1), length 84)
192.168.1.120 > 192.168.2.123: ICMP echo request, id 16, seq 1085, length 64
```

- Tuy nhiên gói tin chưa được switch sang enp0s3 để gửi đến H2 như routing truyền thống. Lý do là R2 chưa xử lý được gói tin MPLS từ R1

```
R2:~$ sudo tcpdump -i enp0s3 -env
```



MPLS Demo (4)

- Thiết lập xử lý MPLS cho R2: kiểm tra MPLS label #123 thì switch từ *enp0s9* sang *enp0s3* và set dest MAC là H2:

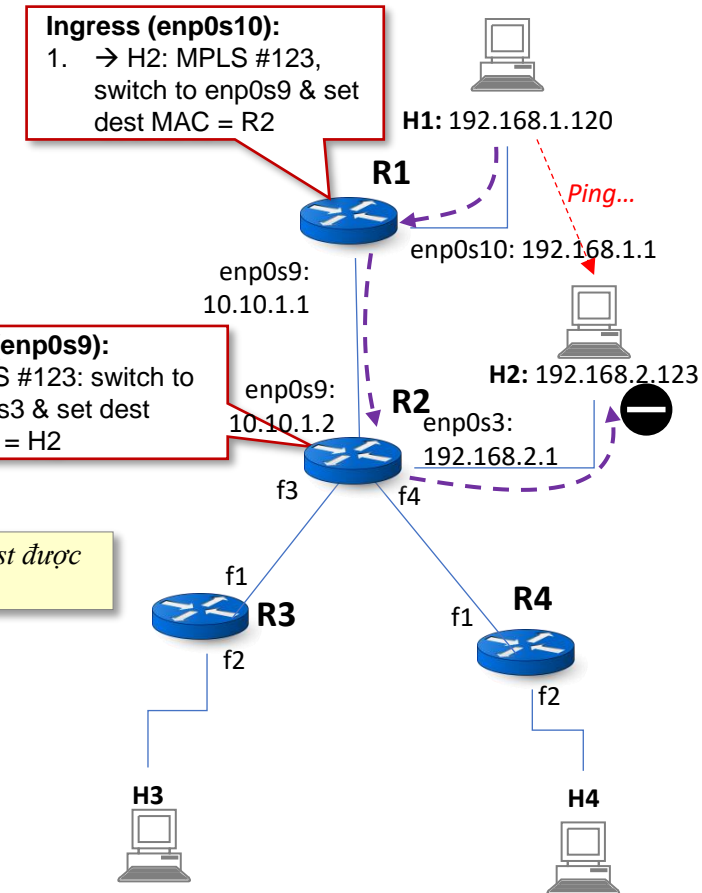
```
R2:~$ sudo tc qdisc add dev enp0s9 handle ffff: ingress
R2:~$ sudo tc filter add dev enp0s9 ingress \
    protocol mpls_uc flower mpls_label 123 \
    action skbmod set dmac 08:00:27:75:25:d1 \
    action mirrored egress redirect dev enp0s3
```

- Kiểm tra thấy *ping* vẫn chưa thành công. Tuy nhiên gói tin đã đi qua được R2 và đến H2.
- Kiểm tra các gói tin đi đến H2 tại kết nối với R2 thấy đã nhận được ICMP Echo Request nhưng không thấy ICMP Echo Reply gửi về:

```
H2:~$ sudo tcpdump -i enp0s10 -env
tcpdump: listening on enp0s10, link-type EN10MB (Ethernet), capture size 262144 bytes
08:49:05.740279 08:00:27:1c:c2:15 > 08:00:27:75:25:d1, ethertype MPLS unicast (0x8847),
length 102: MPLS (label 123, exp 0, [S], ttl 255)
(tos 0x0, ttl 64, id 57302, offset 0, flags [DF], proto ICMP (1), length 84)
192.168.1.120 > 192.168.2.123: ICMP echo request, id 2, seq 153, length 64
```

ICMP Echo Request được gói trong MPLS

- Lý do là H2 nhận được gói tin từ tầng 2 gửi lên không phải là gói tin IP (có chèn layer MPLS vào) nên nó không xử lý được. Cần phải bỏ layer này đi.



MPLS Demo (5)

- Bổ sung xử lý egress MPLS cho R2: khi switch gói tin sang enp0s3 thì bỏ MPLS layer đi:

```
R2:~$ sudo tc qdisc add dev enp0s9 handle ffff: ingress
R2:~$ sudo tc filter add dev enp0s9 ingress \
    protocol mpls_uc flower mpls_label 123 \
    action mpls pop protocol ipv4 \
    action skbmod set dmac 08:00:27:75:25:d1 \
    action mirred egress redirect dev enp0s3
```

- Kiểm tra thấy ping đã thành công

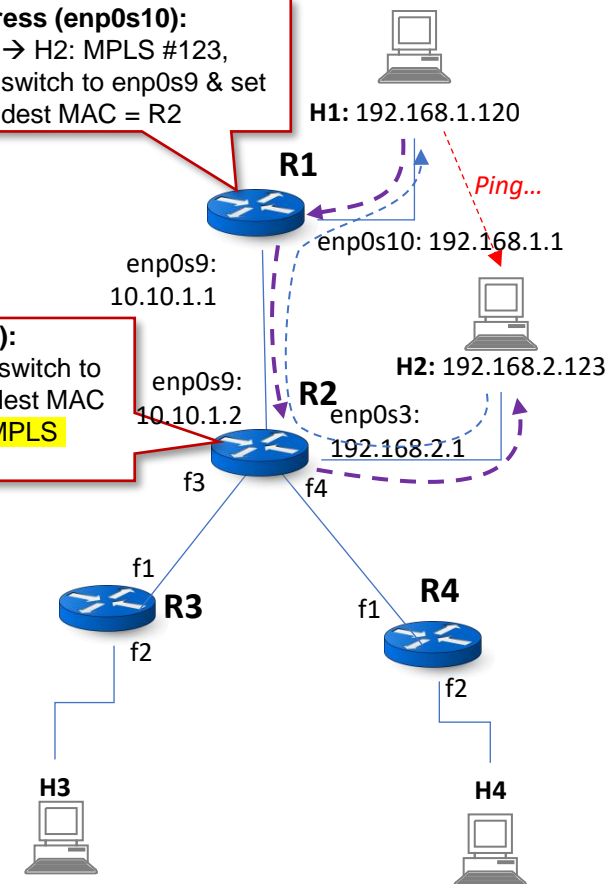
```
H1:~$ ping 192.168.2.123
64 bytes from 192.168.2.123: icmp_seq=1 ttl=62 time=2.26 ms
64 bytes from 192.168.2.123: icmp_seq=2 ttl=62 time=2.03 ms
```

- Kiểm tra các gói tin giữa R1 và R2: H1 → H2: MPLS, H2 → H1: Ethernet

```
R2:~$ sudo tcpdump -i enp0s9 -env
11:34:58.748397 08:00:27:1c:c2:15 > 08:00:27:61:de:17, ethertype MPLS unicast (0x8847),
length 102: MPLS (label 123, exp 0, [S], ttl 255)
(tos 0x0, ttl 64, id 28062, offset 0, flags [DF], proto ICMP (1), length 64)
192.168.1.120 > 192.168.2.123: ICMP echo request, id 18, seq 61, length 64
11:34:58.749055 08:00:27:61:de:17 > 08:00:27:4f:d1:59, ethertype IPv4 (0x0800), length 98:
(tos 0x0, ttl 63, id 62230, offset 0, flags [none], proto ICMP (1), length 84)
192.168.2.123 > 192.168.1.120: ICMP echo reply, id 18, seq 61, length 64
```

Ingress (enp0s9):
1. MPLS #123: switch to enp0s3, set dest MAC = H2 & pop MPLS layer

Ingress (enp0s10):
1. → H2: MPLS #123, switch to enp0s9 & set dest MAC = R2

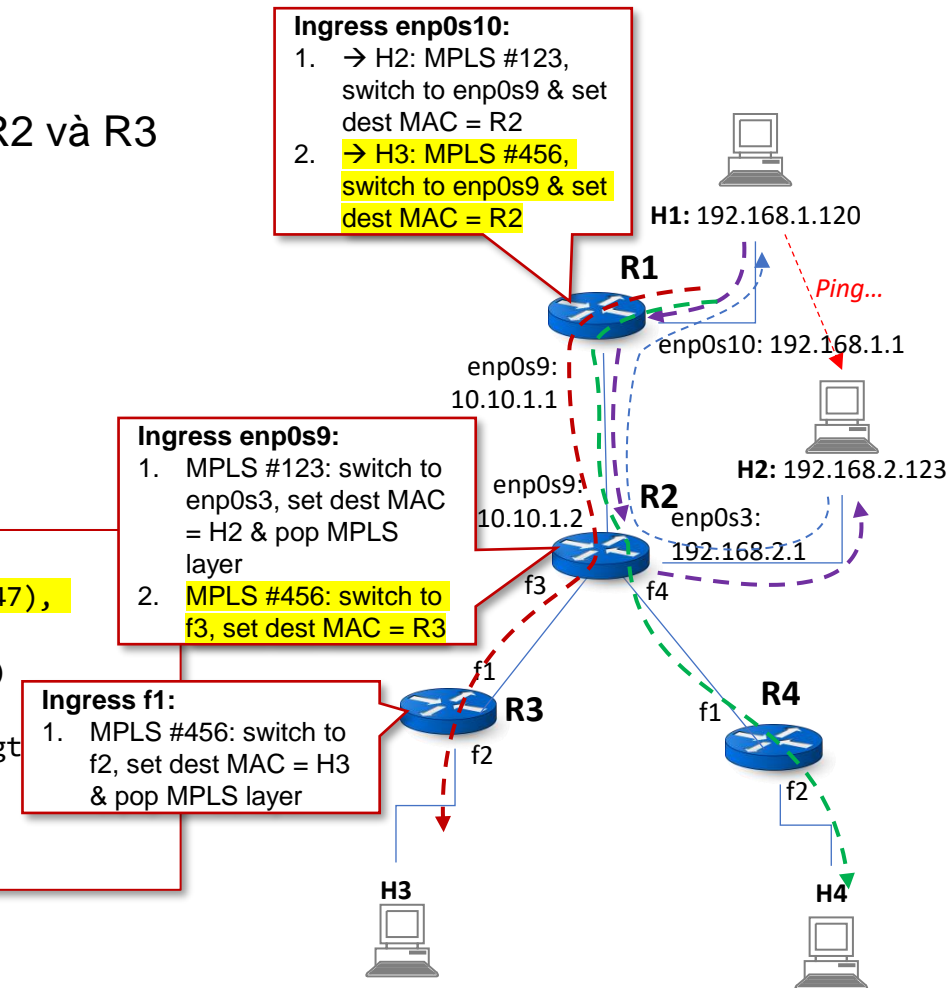


MPLS Demo (6)

- Thiết lập thêm filter để tạo LSP H1 → ... → H3 với label #456 trên R1, R2 và R3
- Tại R2 chỉ forward gói tin MPLS #456 sang kết nối với R3
- Tại R3, forward gói tin MPLS #456 và bỏ MPLS trong gói tin đi
- LSP H1 → R1 → R2 → R3 → H3 được thiết lập
- Gói tin ICMP Echo Request từ H1 đến H3 bằng MPLS
- Gói tin ICMP Echo Reply từ H3 về H1 bằng IP routing

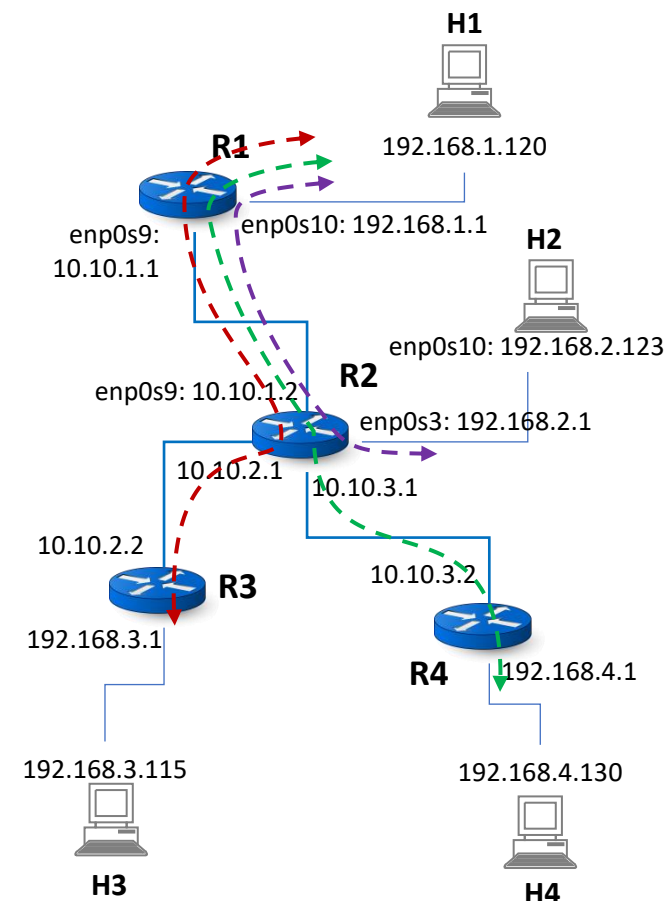
```
R2:~$ sudo tcpdump -i enp0s9 -env
16:26:22.431816 08:00:27:1c:c2:15 > 08:00:27:61:de:17, ethertype MPLS unicast (0x8847),
length 102: MPLS (label 456, exp 0, [S], ttl 255)
(tos 0x0, ttl 64, id 32117, offset 0, flags [DF], proto ICMP (1), length 84)
192.168.1.120 > 192.168.3.115: ICMP echo request, id 6, seq 1, length 64
16:26:22.433124 08:00:27:61:de:17 > 08:00:27:4f:d1:59, ethertype IPv4 (0x0800), length 84
(tos 0x0, ttl 62, id 38947, offset 0, flags [none], proto ICMP (1), length 84)
192.168.3.115 > 192.168.1.120: ICMP echo reply, id 6, seq 1, length 64
```

- Tương tự với LSP H1 → R1 → R2 → R4 → H4



Thực hành MPLS với Linux Traffic Control

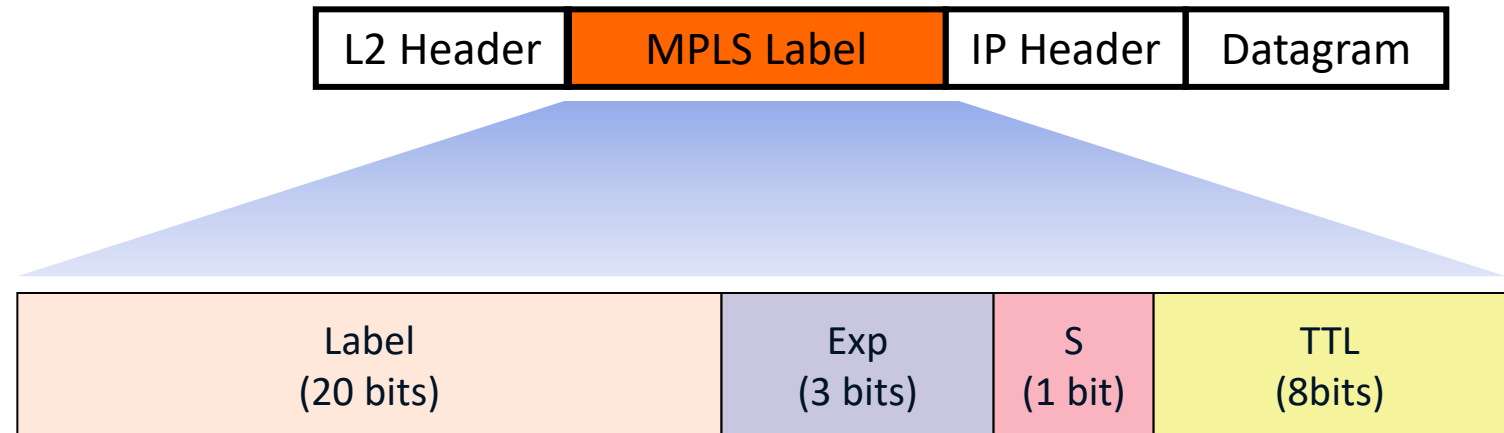
- Thiết lập hệ thống kết nối các router và host
- Tạo LSP H1 → R1 → R2 → H2
 - Khởi tạo ingress LSR trên R1
 - Thiết lập luật chuyển tiếp (filter) trên R1 và R2
 - Xử lý pop MPLS layer trên egress LSR R2
 - Kiểm tra kết nối ping H1 → H2
 - Bắt gói tin ICMP Echo Request và Echo Reply, phân tích chế độ chuyển tiếp các gói tin này bằng MPLS và IP routing
- Tạo LSP H1 → R1 → R2 → R3 → H3
- Tạo LSP H1 → R1 → R2 → R4 → H4



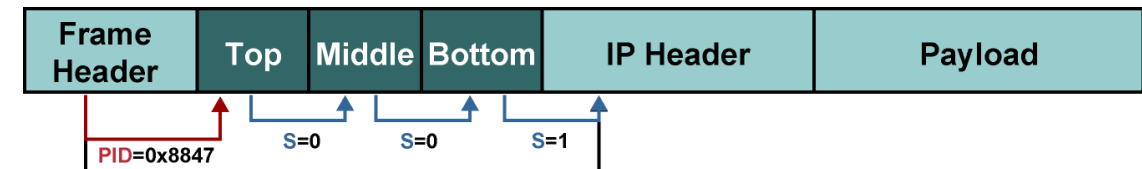
MPLS Architecture

MPLS Encapsulation

- Label: Label value (0 to 15 are reserved for special use)
- Exp: Experimental Use
 - Can use to define the QoS treatment (PHB)
 - *Cisco IOS copies the three most significant bits of the DSCP to the EXP*
- S: Bottom of Stack
 - Set to 1 for the last entry in the label
 - MPLS label can be encapsulated as part of the Layer 2 header, or part of IPv6 network layer header
- TTL: Time To Live
 - Same to IP TTL
 - -1 by each LSR
 - expire when =0 & ICMP "Time Exceeded" to source



```
R2:~$ sudo tcpdump -i enp0s9 -env
16:26:22.431816 08:00:27:1c:c2:15 > 08:00:27:61:de:17,
    ethertype MPLS unicast (0x8847), length 102: MPLS (label 456, exp 0, [S], ttl 255)
        (tos 0x0, ttl 64, id 32117, offset 0, flags [DF], proto ICMP (1), length 84)
        192.168.1.120 > 192.168.3.115:
            ICMP echo request, id 6, seq 1, length 64
```



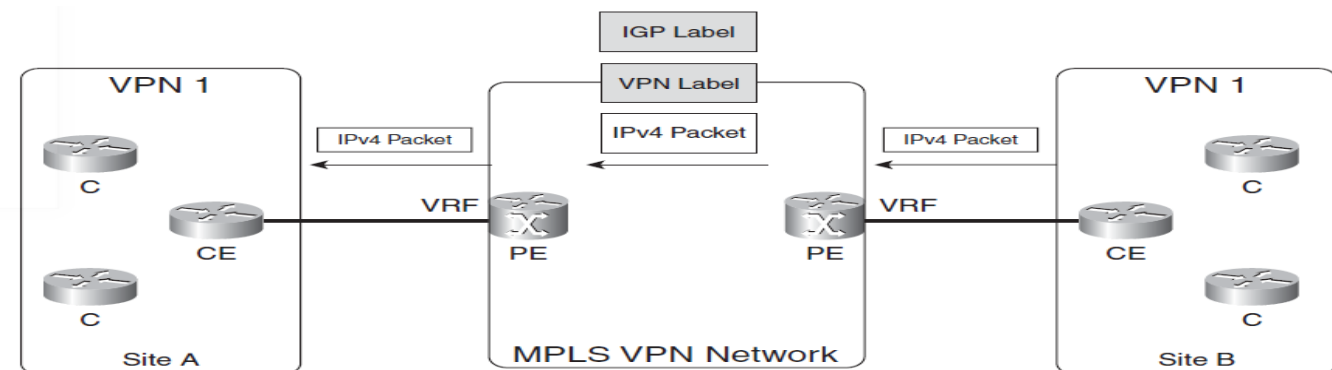
MPLS Multi Label Stack

- Usually only one MPLS label is assigned to a packet, but multiple labels are supported.
- Field “S” to indicate label at bottom of stack
- LRS routers switch packets by using label at the top of stack
- Some MPLS applications need more than one label in the label stack to forward the labeled packets. Examples of such MPLS applications is MPLS VPN:
 - VPN Provider run MPLS VPN network
 - VPN Customers have CE device to access PE device in VPN MPLS provider network
 - Top label (IGP label) points to the egress router (PE)
 - The second label (VPN label) identifies the VPN.

Top of stack

Label #1	Exp	S=0	TTL
Label #2	Exp	S=0	TTL
Label #3	Exp	S=1	TTL

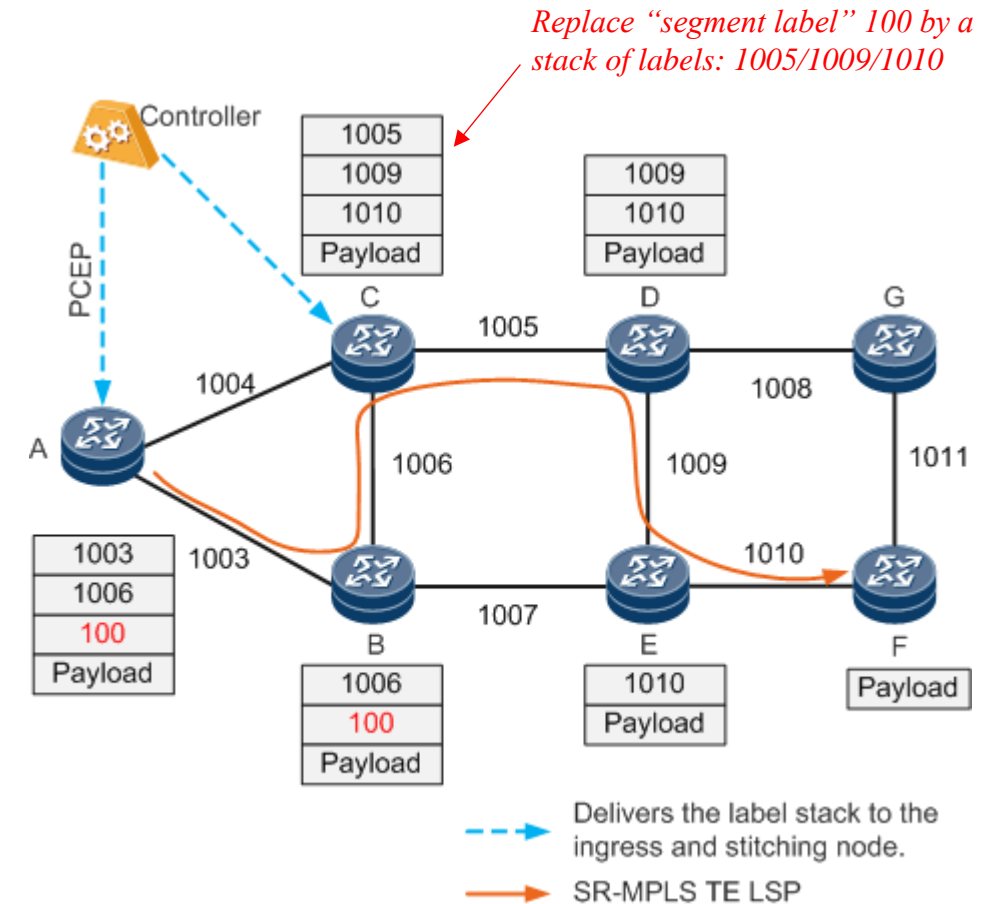
```
~$ sudo tcpdump -i enp0s10 -env
09:05:27.766094 08:00:27:84:bd:f8 > 08:00:27:10:11:15,
  ethertype MPLS unicast (0x8847), length 106:
  MPLS (label 106, exp 0, ttl 61)
  (label 203, exp 0, [S], ttl 64)
  (tos 0x0, ttl 64, proto ICMP (1))
  192.168.4.130 > 192.168.3.115: ICMP echo request
```



source: MPLS Fundamentals (Cisco press 2007)

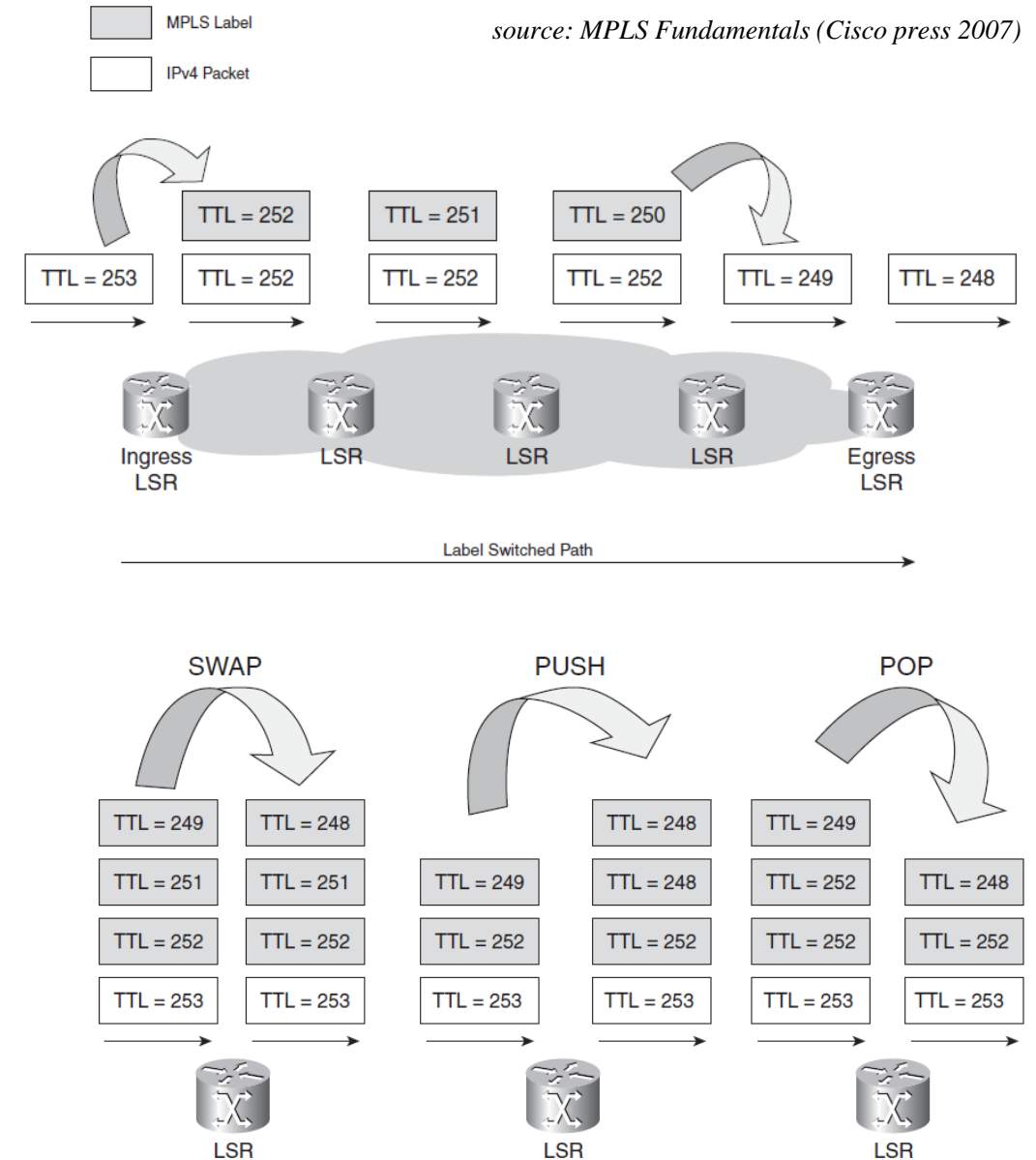
MPLS Multi Label Stack example: SR-MPLS

- Segment Routing:
 - Forwarding packets on the network based on the source routing paradigm.
 - The source chooses a path and encodes it in the packet header as an ordered list of segments.
- SR-MPLS:
 - Label stack pushed at source to decide the route
 - One “segment label” can represent a “stack of routing labels”
 - LSR router when receive packet with “segment label” can replace with a stack of labels and forward the packet.



TTL Behavior

- Time To Live (TTL) is a well-known mechanism thanks to IP, signifies the time that a packet still has before its life ends and is dropped.
- When an IP packet is sent, its TTL is usually 255 and is then decremented by 1 at each hop.
- In MPLS, labels are added to IP packets in which the TTL is propagated from the IP header into the label stack and vice versa.
- In principle, TTL of IP packets coming in/out a MPLS network should be manipulated as the same result as they are routed in IP network:
 - At ingress LSR: IP TTL is copied (after being decremented by 1) to the MPLS TTL values of the pushed label(s).
 - At egress LSR, the label is removed. The IP TTL is copied from the MPLS TTL (top label after decrementing it by 1).
 - At a LSR, swap/push/pop label also affect the TTL in the stack



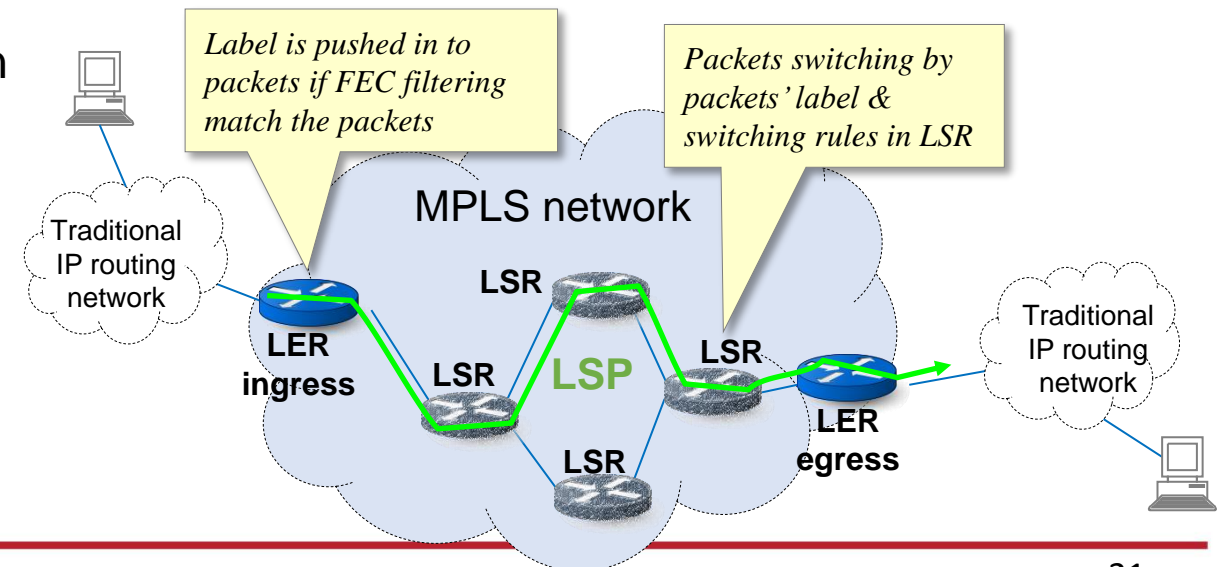
Forward Equivalent Class (FEC)

- FEC = A group of packets that are treated the same way by a router
- A packet can be mapped to a particular FEC based on the following criteria:
 - destination IP address,
 - source IP address,
 - TCP/UDP port,
 - class of service (CoS) or type of service (ToS),
 - application used,
 - ...
 - any combination of the previous criteria.
- In IP routing, FEC is implemented by ToS filed in IP packet
- Inside MPLS, FEC is implemented by LS Path which consisted of labels bound to “network prefix” in each LSR router
- FEC filtering to match the packets is only done once at the ingress LSR

Ingress Label	FEC	Egress Label
-	192.168.2.0/24	123

```
R1:~$ sudo tc filter add dev enp0s10 protocol ip parent ffff: \
    flower dst_ip 192.168.2.0/24 \
    action mpls push protocol mpls_uc label 123 \
    action mpls egress redirect dev enp0s9
```

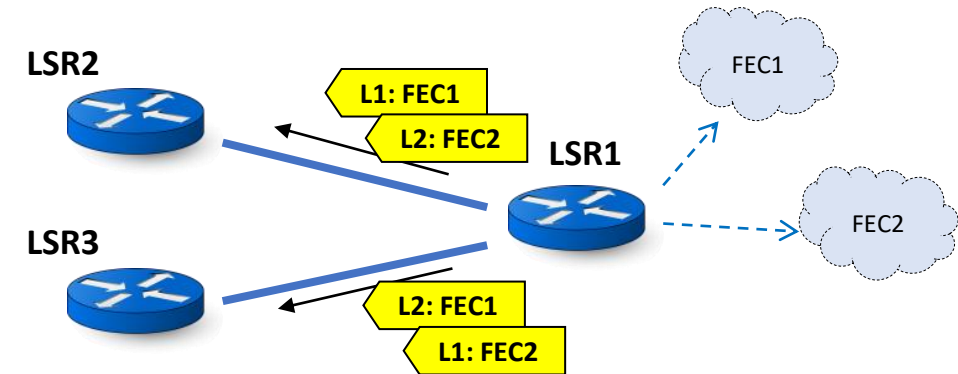
Có thể bổ sung filter lọc gói tin theo nhiều trường thông tin kết hợp: địa chỉ IP nguồn/đích, cổng TCP nguồn/đích, ToS, v.v...



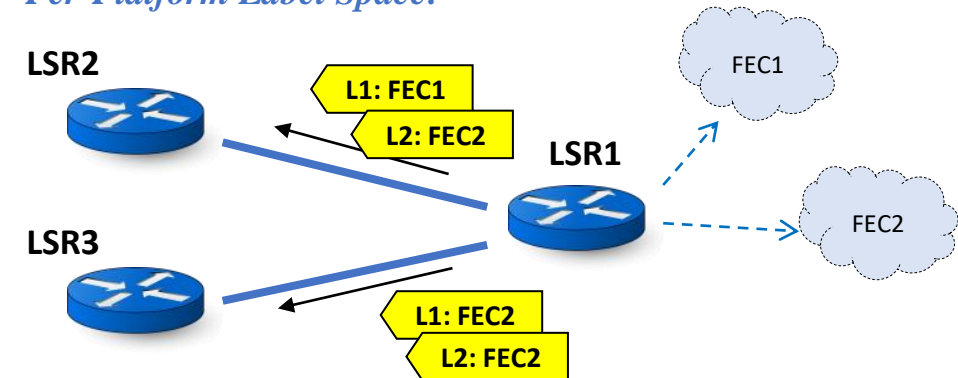
MPLS Label Spaces

- As LSR1 is the next hop for network prefix (FEC1, FEC2), we can create LS path from LSR2/LSR3 to these FEC through LSR1
- For each “adjoint LSR”, LSR1 can from LS paths independently with other “adjoint LSR”. Label should be unique in each network interface → per-interface label space.
- If per-interface label space is used, the packet is not forwarded solely based on the label, but based on both the incoming interface and the label.
- The other possibility is that the label is not unique per interface, but over the LSR assigning the label → per-platform label space.
- If per-platform label space is used, the packet is forwarded solely based on the label, independently from the incoming interface.

Per-Interface Label Space:

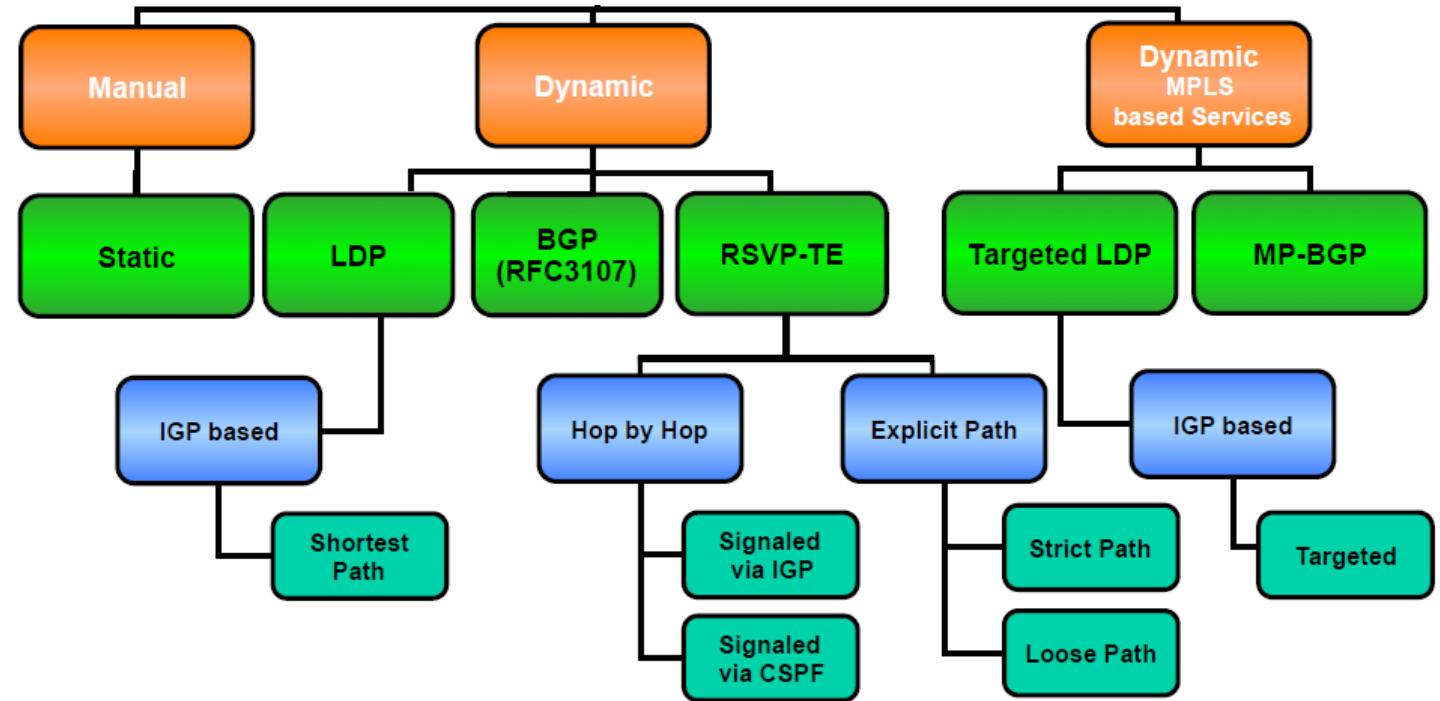


Per-Platform Label Space:



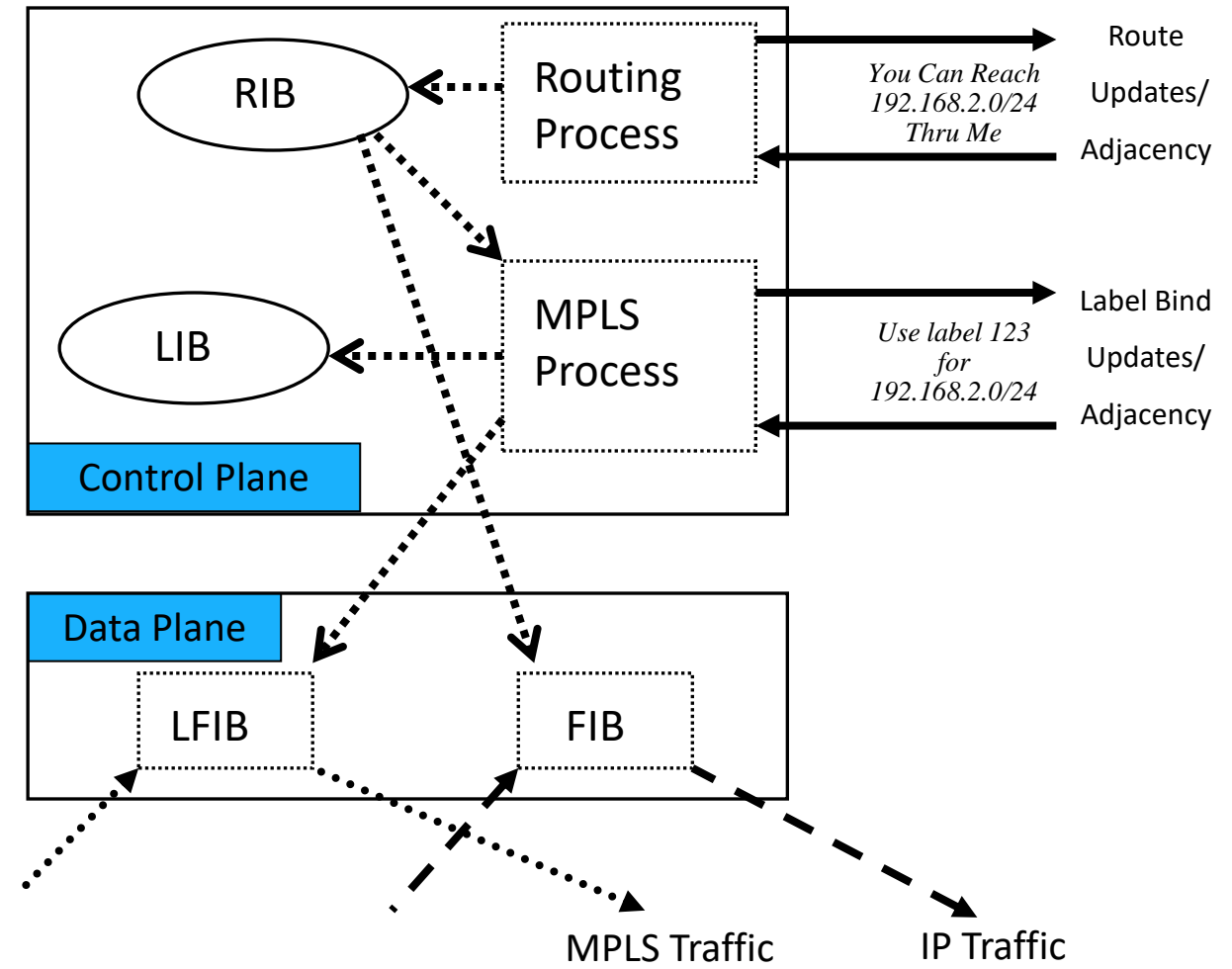
MPLS Label Distribution

- How to distribute labels among LSR to build LS paths?
- Static:
 - Label are assigned manually by the operator.
 - Configured manually at each router (ingress, transit, egress).
 - No dependence on IGP or local forwarding table.
 - No failure detection and no re-routing capabilities.
- Dynamic:
 - Using protocols.
 - Allows labels to be assigned automatically from ingress router to egress router.
 - Dependent on IGP and local forwarding table.



LSR Insight: Control & (Data) Forward plane

- Control plan:
 - Preparing phase
 - Building rules
 - Works with routing protocols
- Data forward plane
 - Acting phase
 - Matching rules with packets to forward to the next hope
- IP routing:
 - RIB: Routing Information Base
 - FIB: Forwarding Information Base
- MPLS switching:
 - LIB: Label Information Base
 - LFIB: Label Forward Information Base



Label Distribution Protocol (LDP)

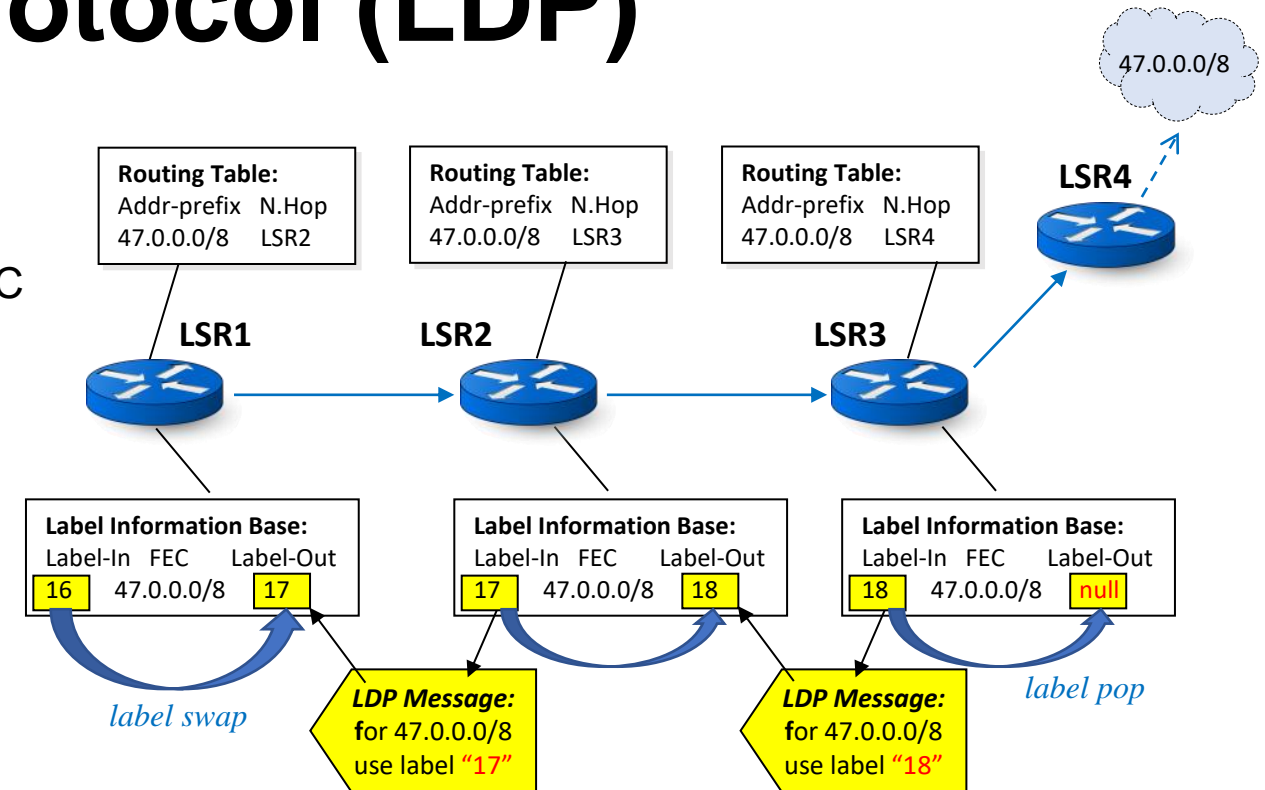
Label Distribution Protocol (LDP)

- Label binding among all LSR routers:
 - Assign a label to a given FEC in each LSR
 - Toward all routers have a “common view” of FEC
→ LS path establishment
 - Label push (ingress LSR), swap & pop (egress LSR) respecting the granularity

→ How???

- LDP is the solution:
 - RFC3036/3037 (2001), RFC5036 (2007) replace RFC3036, RFC7552 (2015) – for IPv6
 - Used to distribute labels in a MPLS network
- Transport layer used:
 - UDP port 646 for LDP discovery
 - TCP port 646 for LDP session
- LDP is Not a routing protocol (no route finding, neither best route selection)

Based on routing table → Work with IGP



LSR3# show mpls ldp binding

Destination	Nexthop	Local Label	Remote Label
47.0.0.0/8	LSR4	18	imp-null

LSR2# show mpls ldp binding

Destination	Nexthop	Local Label	Remote Label
47.0.0.0/8	LSR3	17	18

LSR1# show mpls ldp binding

Destination	Nexthop	Local Label	Remote Label
47.0.0.0/8	LSR2	16	17

LDP initialization: Adjacency

- LDP discovery:
 - Enables an LSR to discover LDP peers (neighbor)
 - Basic Discovery:
 - to discover LSR neighbors directly connected at the link level.
 - periodically sends LDP Link Hellos: UDP multicast 224.0.0.2:646
 - Extended Discovery:
 - to locate LSRs that are not directly connected at the link level.
 - periodically sends LDP Targeted Hellos to a specific address (UDP packets to the specific address)
- LDP peers & LDP session:
 - Two LSRs (LDP peer) use LDP to exchange label/FEC mapping info
 - LDP session: unicast TCP connection at port 646
 - Network prefix (Forward Equivalent Class - FEC) is exchanged in LDP session with MPLS label

```
~$ sudo tcpdump -i enp0s9 -env udp port 646
14:52:08.553043 08:00:27:4f:d1:59 > 01:00:5e:00:00:02,
ethertype IPv4 (0x0800), length 84: (tos 0xc0, ttl 1, id
51824, offset 0, flags [DF], proto UDP (17), length 70)
10.10.1.1.646 > 224.0.0.2.646:
LDP, Label-Space-ID: 1.1.1.1:0, pdu-length: 38
Hello Message (0x0100),
Common Hello Parameters TLV (0x0400),
Hold Time: 15s, Flags: [Link Hello]
IPv4 Transport Address TLV (0x0401),
IPv4 Transport Address: 10.10.1.1
```

```
~$ sudo tcpdump -i enp0s9 -env tcp port 646
15:10:43.597230 08:00:27:61:de:17 > 08:00:27:4f:d1:59,
ethertype IPv4 (0x0800), length 103: (tos 0x0, ttl 255, id
36599, offset 0, flags [DF], proto TCP (6), length 89)
10.10.1.2.39835 > 10.10.1.1.646: Flags [P.], ....
LDP, Label-Space-ID: 2.2.2.2:0, pdu-length: 33
Label Mapping Message (0x0400), ....
FEC TLV (0x0100), length: 7, Flags:
Prefix FEC (0x02): IPv4 prefix 192.168.1.0/24
Generic Label TLV (0x0200), length: 4, Flags:
Label: 16
```

LDP Messages

- Discovery Messages

- Discovery messages announce and maintain the presence of a router in a network.
- Routers indicate their presence in a network by sending hello messages periodically.
- Hello messages are transmitted as UDP packets to the LDP port at the group multicast address for all routers on the subnet.

- Session Messages

- Session messages establish, maintain, and terminate sessions between LDP peers.
- When a router establishes a session, it uses the LDP initialization procedure over TCP transport.
- When the initialization procedure is completed successfully, the two routers are LDP peers and can exchange advertisement messages.

- Advertisement Messages

- Advertisement messages create, change, and delete label mappings for FECs.
- In general, the router requests a label mapping from a neighboring router when it needs one and advertises a label mapping to a neighboring router when it wants the neighbor to use a label.

- Notification Messages

- Notification messages provide advisory information and signal error information.
- LDP sends notification messages to report errors and other events of interest.

LDP Procedures & Working modes

- The principal objective of LDP is to establish “common view” of FEC for all LSR routers.
- Procedures & working modes are standardized for LSR to collaborate in LDP sessions.
- Label advertisement (also called Label distribution)
 - An LSR binds a label to a specific FEC and notifies its upstream LSRs of the binding.
 - Downstream unsolicited (DU) mode & Downstream on demand (DoD) mode
 - The label advertisement modes on upstream and downstream LSRs must be the same.
- Label distribution control
 - The label distribution control mode defines how an LSR distributes labels.
 - Independent mode & Ordered mode
- Label retention
 - The label retention mode defines how the LSR handles label mapping from non-preferred next hop.
 - The label mapping that an LSR receives may or may not originate at the next hop.
 - Liberal mode & Conservative mode
- LSR can support the combinations of procedures working modes

FRR (version 7.2.1) implement LDP by combination: Downstream Unsolicited + Independent Control + Liberal Label Retention

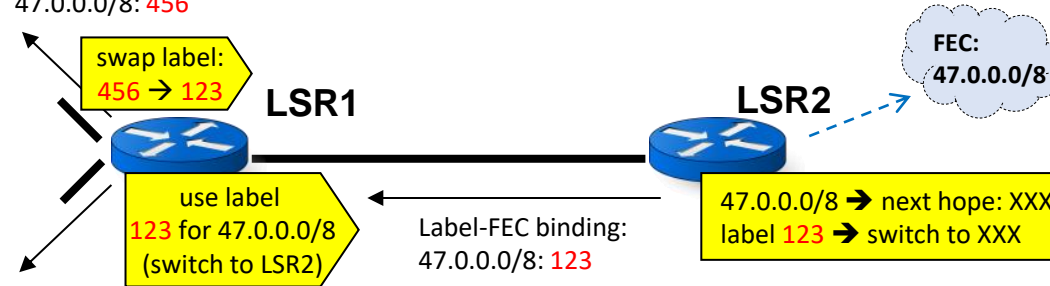
Label Distribution modes

LSR binds a label to each specific FEC and notifies its upstream LSRs of the binding. This process is called label advertisement (or distribution).

Downstream-Unsolicited: actively by downstream LSR

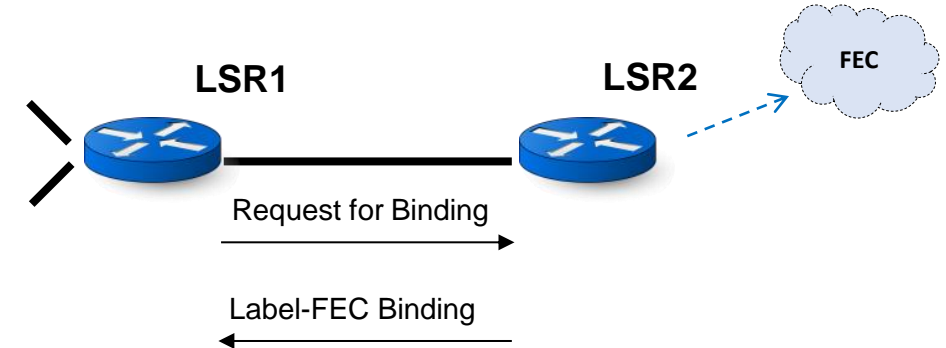
Label-FEC binding:

47.0.0.0/8: 456



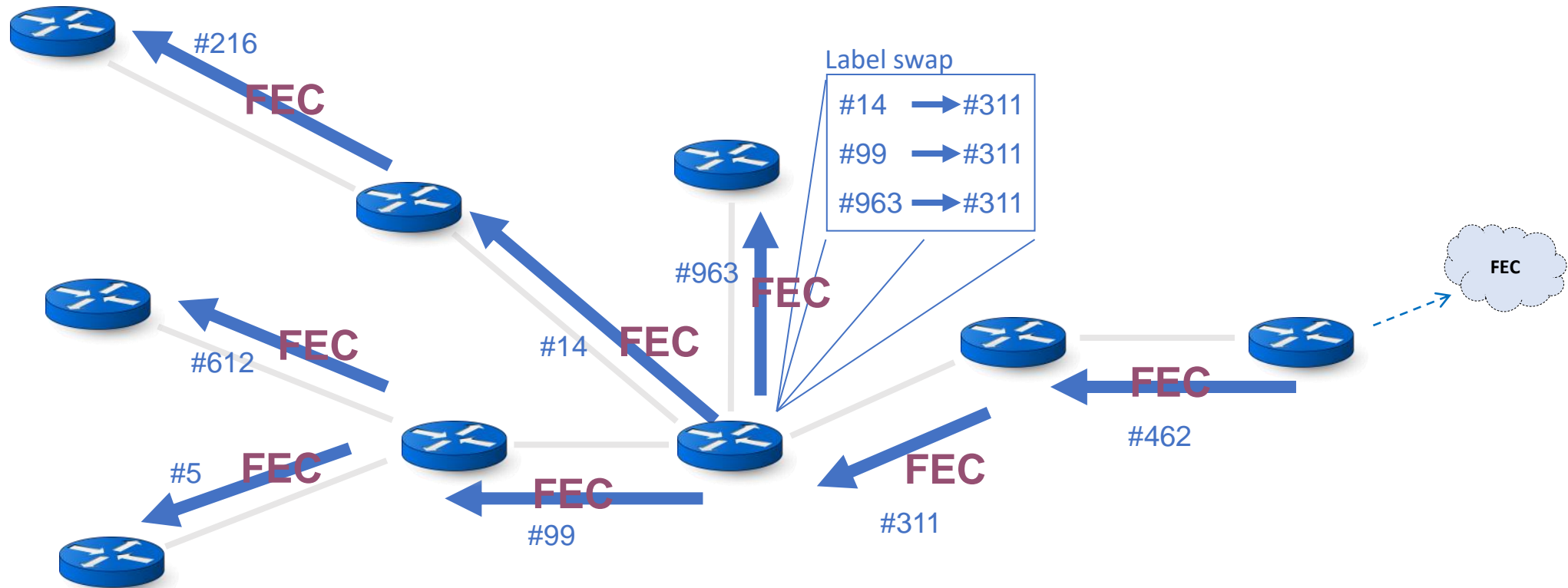
- LSR2 and LSR1 are said to have an “LDP adjacency” (LSR2 being the downstream LSR)
- LSR2 discovers a ‘next hop’ for a particular FEC
- LSR2 generates a label for the FEC and communicates the binding to LSR1
- LSR1 inserts the binding into its forwarding tables
- If LSR2 appear to be the next hop for the FEC, LSR1 can (recursively) use that label to continue the distribution

Downstream-on-Demand: by request from upstream LSR

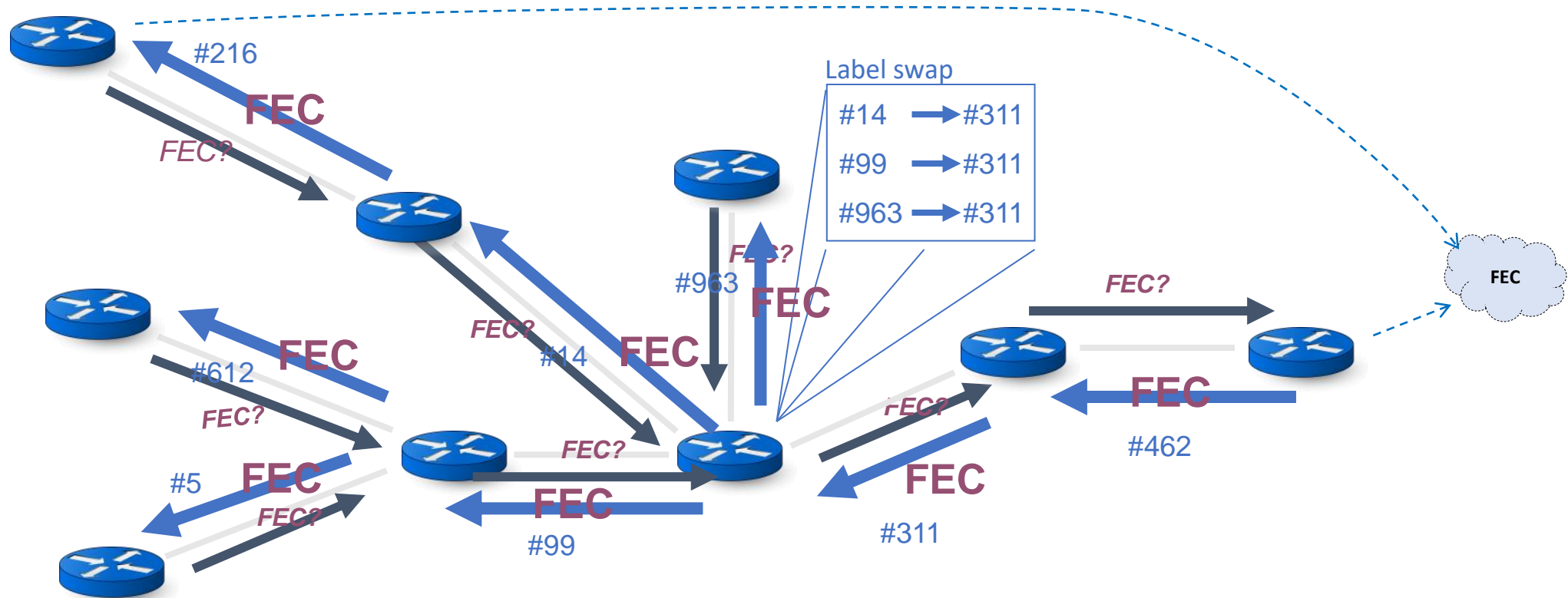


- LSR1 recognizes LSR2 as its next-hop for an FEC
- A request is made to LSR2 for a binding between the FEC and a label
- If LSR2 recognizes the FEC and has a next hop for it, it creates a binding and replies to LSR1
- Both LSRs then have a common understanding

SPF tree building (downstream unsolicited)



SPF tree building (downstream on demand)



Distribution Control modes

The label distribution control defines how an LSR distributes labels.

- Independent label distribution control:

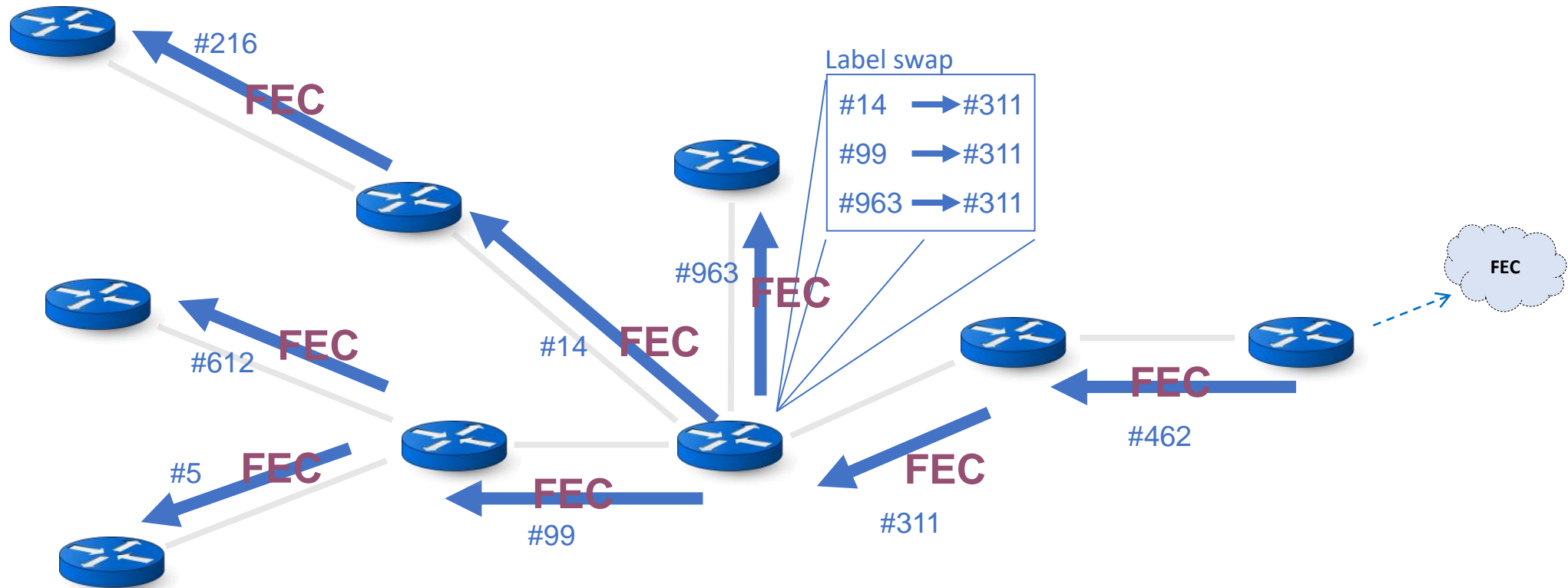
- LSR makes independent decision on when to generate labels and communicate them to upstream peers
- In principal, label-FEC binding to peers once next-hop has been recognized
- LSP is formed as incoming and outgoing labels are spliced together

- Ordered label distribution control:

- LSR sends a Label Mapping message to an upstream LSR only when:
 - LSR is the 'egress' LSR to particular FEC
 - label binding has been received from upstream LSR
- LSP formation 'flows' from egress to ingress

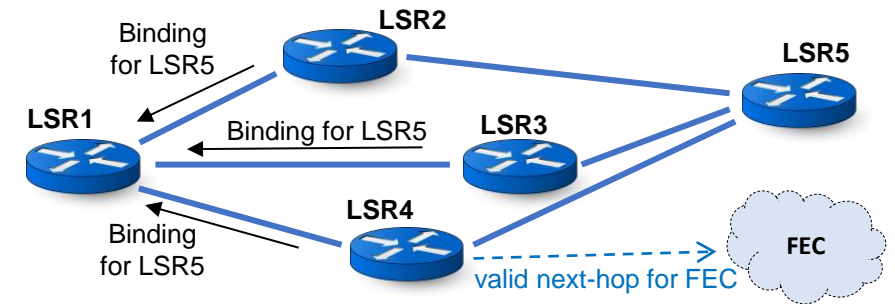
Comparison	
<ul style="list-style-type: none">• Labels can be exchanged with less delay• Does not depend on availability of egress node• Granularity may not be consistent across the nodes at the start• May require separate loop detection/mitigation method	<ul style="list-style-type: none">• Requires more delay before packets can be forwarded along the LSP• Depends on availability of egress node• Mechanism for consistent granularity and freedom from loops• Used for explicit routing and multicast

Independent label distribution control



Label Retention modes

- An LSR may receive label bindings from multiple LSRs
- Some bindings may come from LSRs that are not the valid next-hop for that FEC



• Liberal Label Retention

Label Bindings
for LSR5

LSR4's Label

LSR3's Label

LSR2's Label

- LSR maintains bindings received from LSRs other than the valid next hop
- If the next-hop changes, it may begin using these bindings immediately
- May allow more rapid adaptation to routing changes
- Requires an LSR to maintain many more labels

• Conservative Label Retention

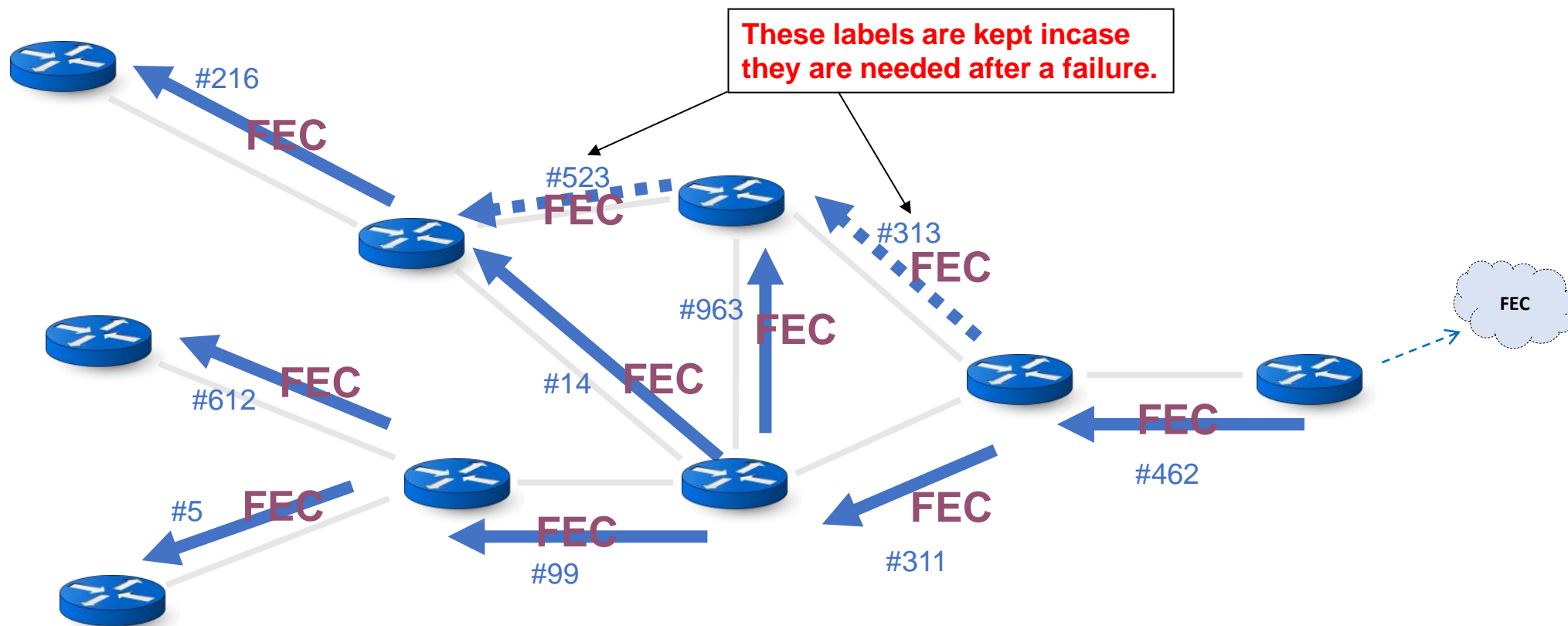
Label Bindings
for LSR5

LSR4's Label

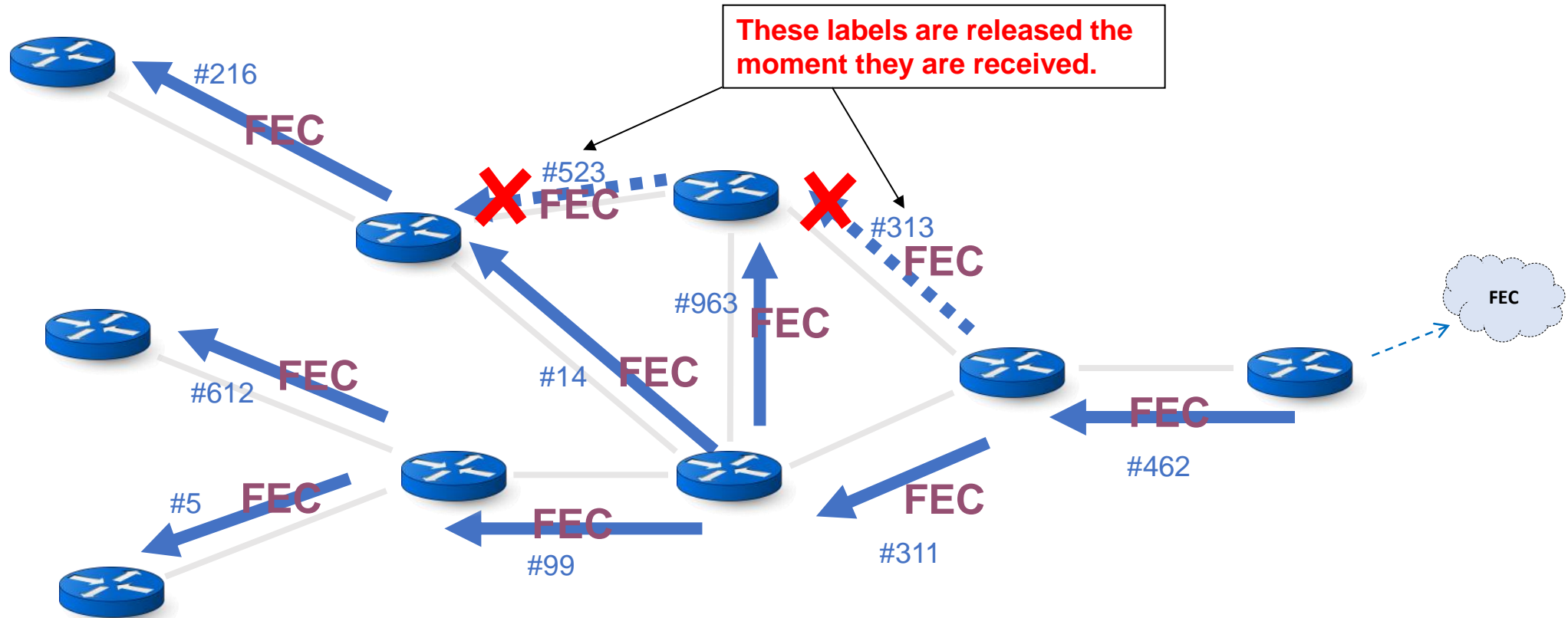


- LSR only maintains bindings received from valid next hop
- If the next-hop changes, binding must be requested from new next hop
- Restricts adaptation to changes in routing
- Fewer labels must be maintained by LSR

Liberal Retention mode



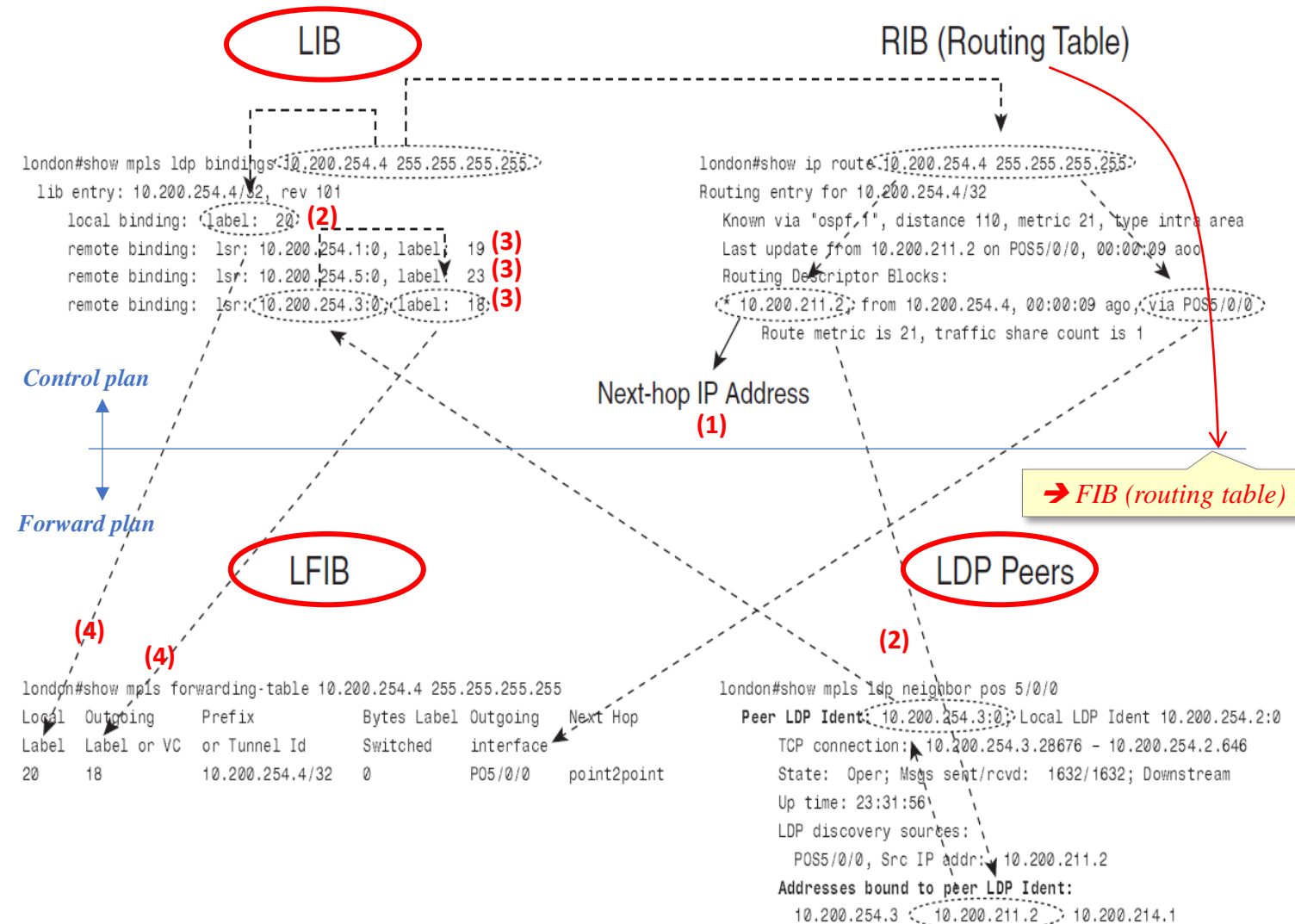
Conservative Retention mode



LDP update flow: RIB → LIB → LFIB

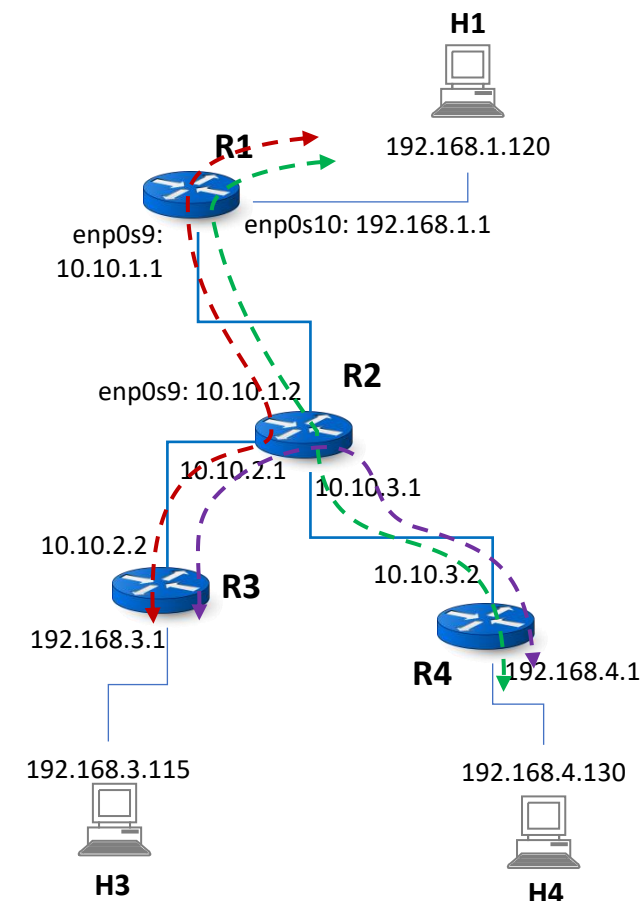
source: MPLS Fundamentals (Cisco press 2007)

- IP routing:
 - IGP → RIB → (best route) → FIB
 - FIB (routing table): rules to forward IP packet
- MPLS switching:
 - LDP → LIB → (best label) → LFIB
 - LFIB: rules to switch MPLS packet
- LDP update flow in Peer:
 1. Detect network prefix with next hop from RIB (by synchronize with IGP)
 2. Assign label for each network prefix in RIB and advertise to upstream LSR (DU mode) or request remote label to downstream LSR (DoD mode)
 3. Update network prefix with label to LIB (in term of FEC). Many labels may bound to one FEC (Liberal mode)
 4. Select “best label” to implement in LFIB



Thực hành LDP

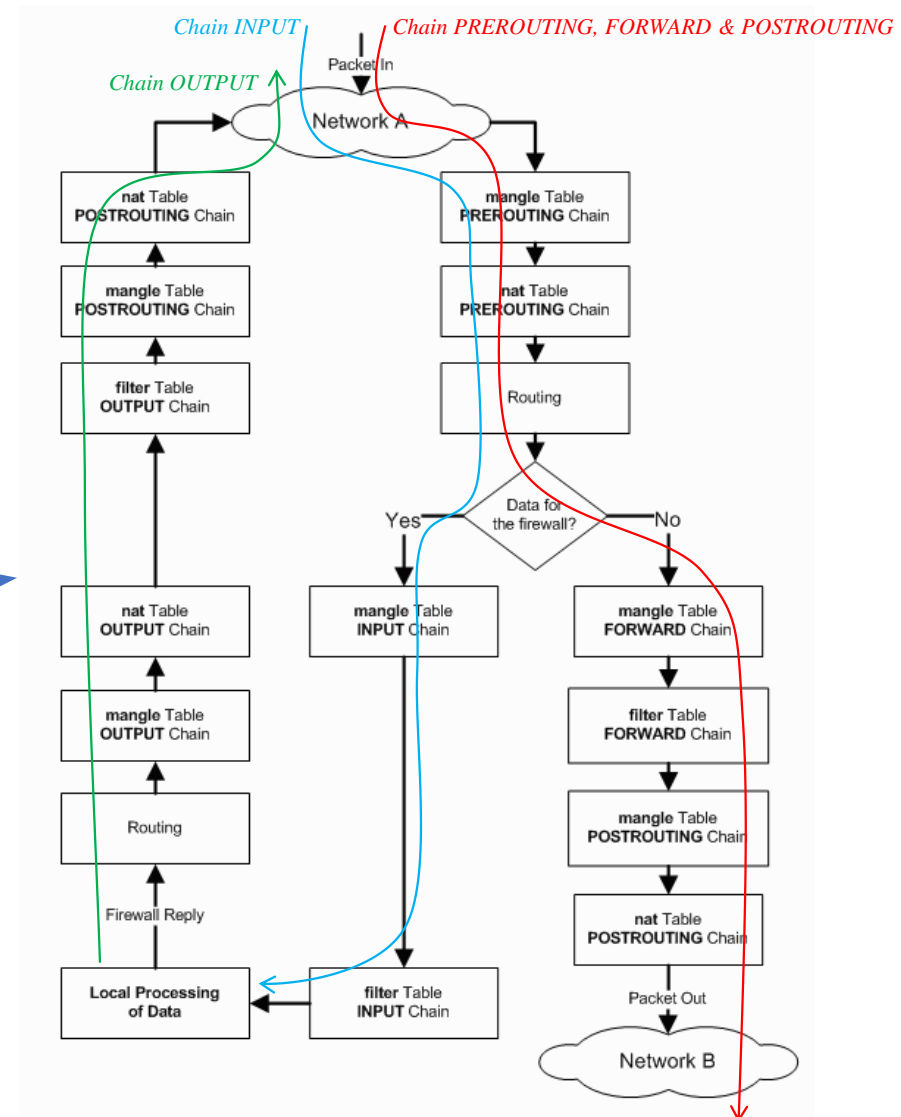
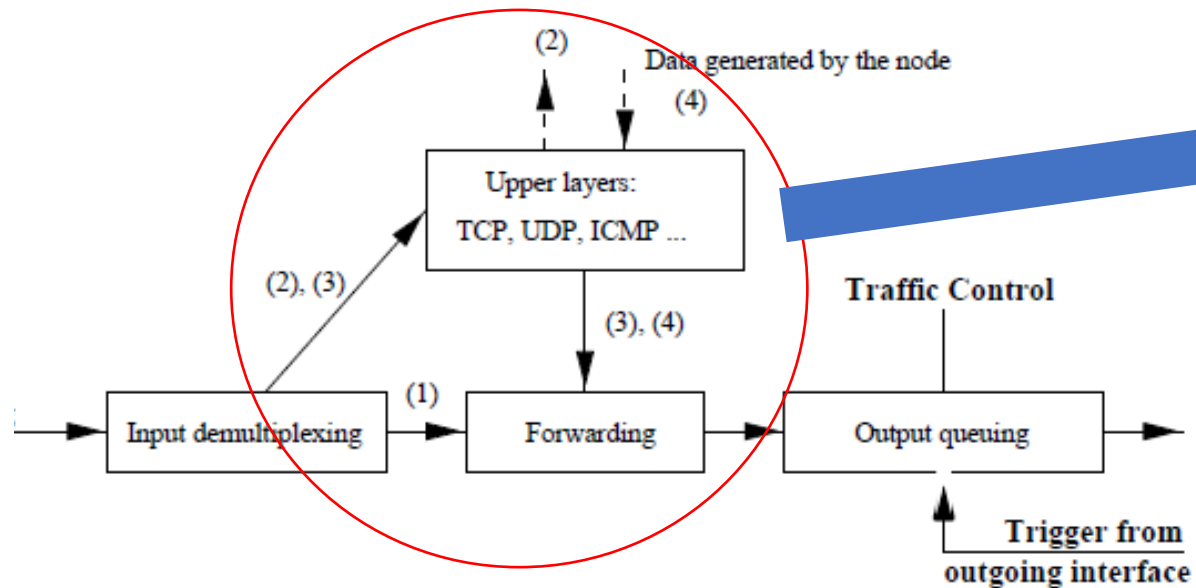
- Thiết lập hệ thống kết nối các router và host
- Bật chức năng IGP (OSPF) và LDP trong các router bằng FRR
- Kiểm tra các gói tin LDP Hello khi router tìm kiếm peer
- Kiểm tra label được tự động cập nhật cho các FEC tại các router
- Kiểm tra các LS path được tự động thiết lập giữa H1, H3, H4
- Tạo ingress LSR bằng phương pháp manual
- Test hệ thống bằng ping giữa H1, H3, H4



Appendix: Linux Traffic Control

Traffic Control in Linux kernel

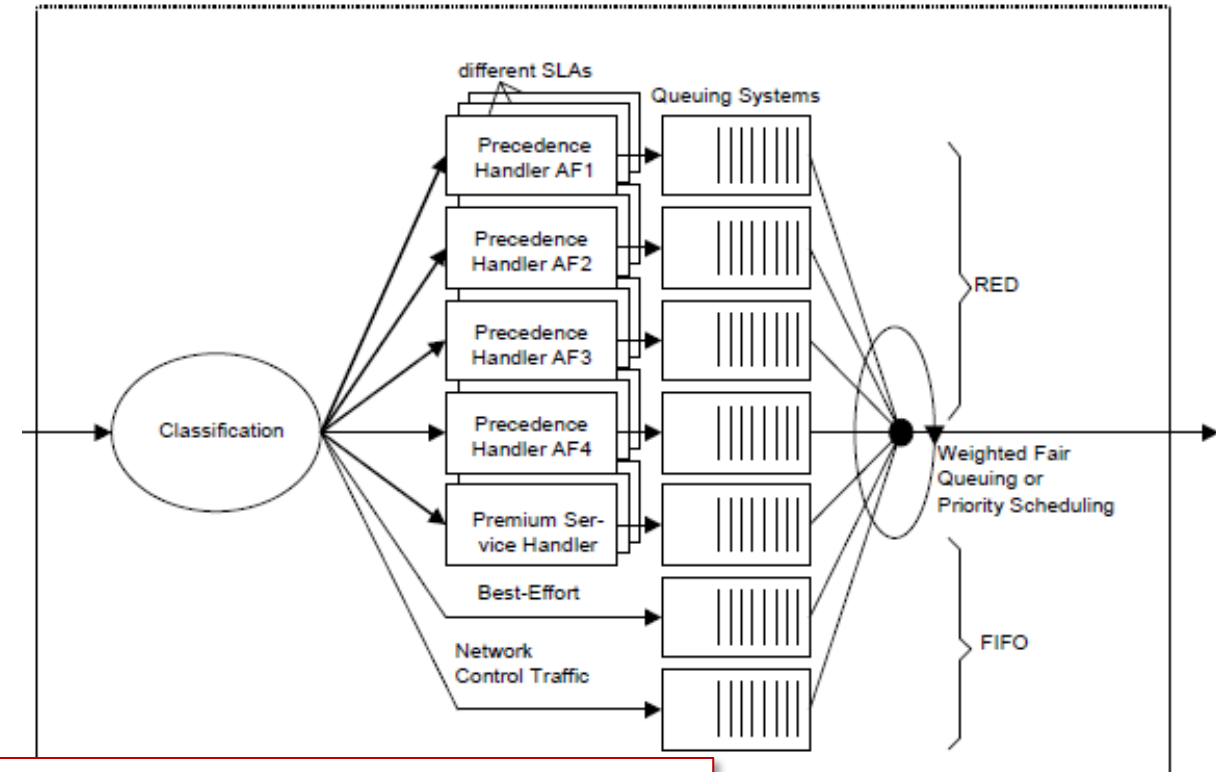
- Remember the packet flow in *iptables*
- Packet incoming and demultiplexing (1). In case of end system (server, workstation, etc.), the packets is passed to higher layers in protocol stack (2).
- Traffic Control takes place right after forwarding module.



DiffServ implementation in Linux

Source: A Linux Implementation of a Differentiated Services Router
- Lecture Notes in Computer Science (2000)

- Sử dụng Traffic Control (TC) trong Linux kernel.
- Một số modul DiffServ có sẵn trong Linux kernel kiểu được đặt option khi dịch kernel hoặc được khai báo gắn vào kernel trong lúc run-time.
- Queuing Systems là các hệ thống xử lý gói tin theo từng lớp (class) đã được phân loại. TC gọi là các *qdisc* (queueing discipline)
- *qdisc code1* là qdisc được cài đặt sẵn trong linux kernel với thuật toán Controlled-Delay Active Queue Management (AQM), và mặc định gắn với mỗi kết nối mạng
- Ngoài *qdisc pfifo*, có nhiều *qdisc* khác tiếp tục được thêm vào linux kernel để xử lý hàng đợi theo thuật toán mới



```
~$ tc qdisc show
```

```
qdisc noqueue 0: dev lo root refcnt 2
```

```
qdisc fq_code1 0: dev enp0s3 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval 100.0ms memory_limit 32Mb ecn
```

```
qdisc fq_code1 0: dev enp0s8 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval 100.0ms memory_limit 32Mb ecn
```

```
qdisc fq_code1 0: dev enp0s9 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval 100.0ms memory_limit 32Mb ecn
```

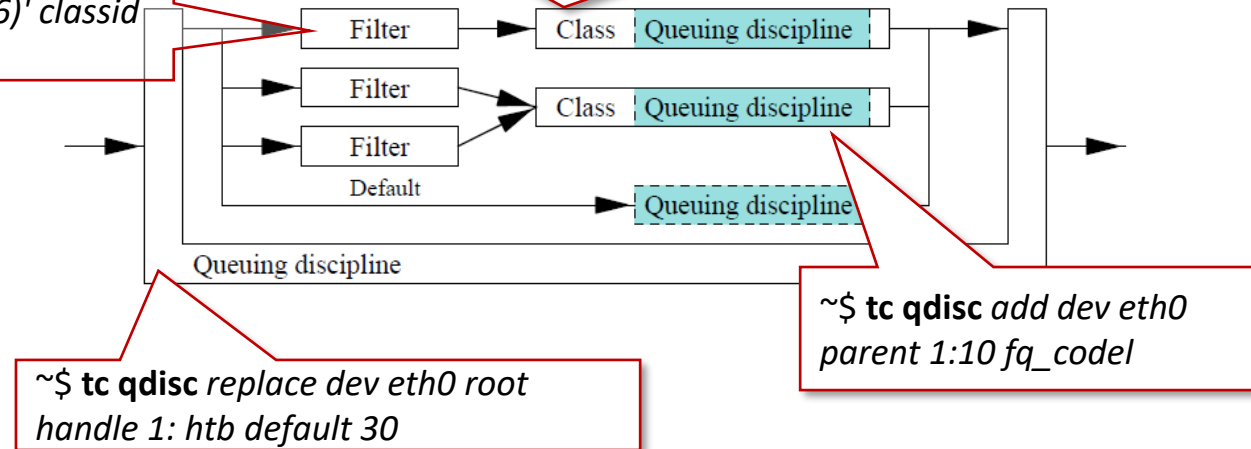
```
qdisc pfifo_fast 8001: dev enp0s10 root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
```

TC: filter, class & queue discipline

- Packets over the same interface, may desire different treatment.
- For an enqueued packet, one filter is checked after the other, until there is a match with a class
→ applying queue discipline
- No filter (& class) matched
→ default queueing discipline (Best Effort).
- Each class defines some specific values in processing packets: min/max rate, priority (in comparing with other classes), etc...
- Recursive queueing discipline: inside a *qdisc*, several classes are defined by other *qdisc*
- Control the queue not overload by policy: when new packet to be enqueued, policing component can decide to drop the currently processed packet or it can refuse the enqueueing of the new one.
- Each network device has an associated queueing discipline, in which the packets are stored in the order they have been enqueued. The packets are taken from the queue as fast as the device can transmit them

```
~$ tc filter add dev eth0  
parent 1: basic match  
'meta(priority eq 6)' classid  
1:10
```

```
~$ tc class classid 1:10 htb rate 1mbit ceil 20mbit prio 1
```



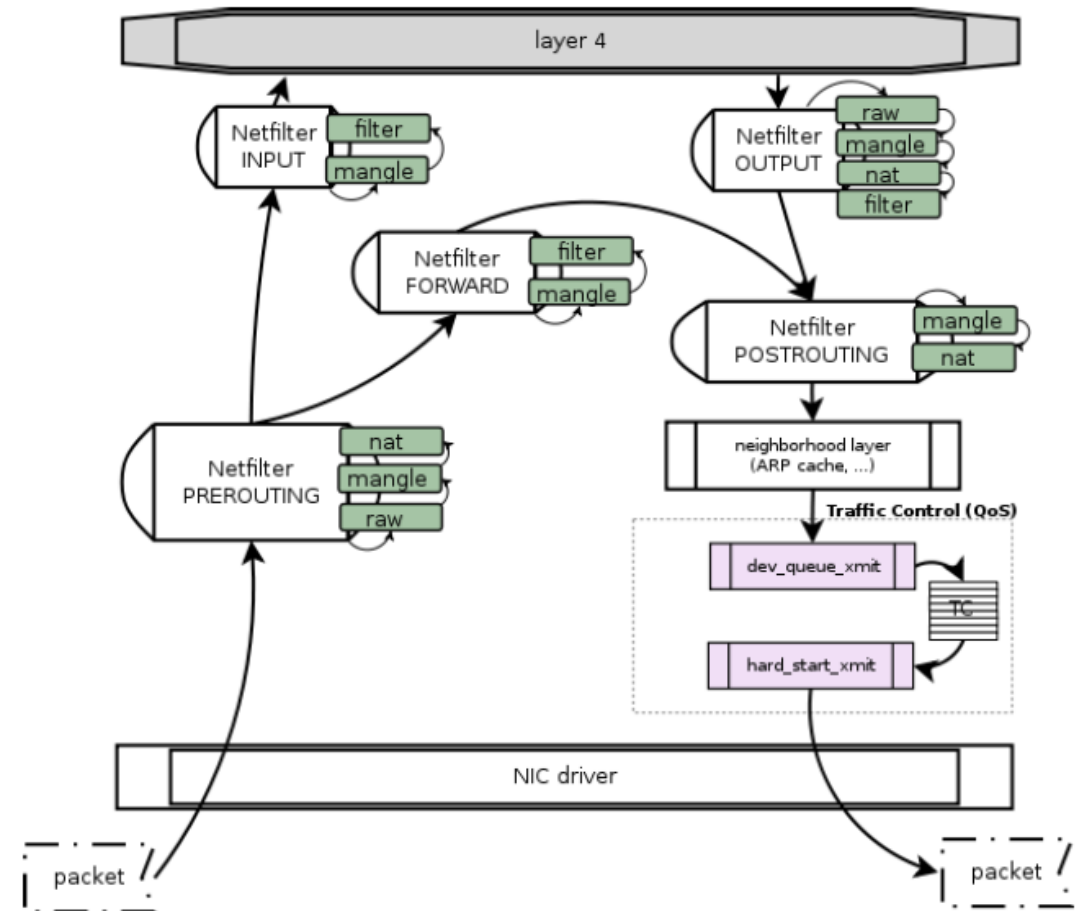
```
~$ tc qdisc replace dev eth0 root  
handle 1: htb default 30
```

```
~$ tc qdisc add dev eth0  
parent 1:10 fq_codel
```


Linux kernel debug



1. Traffic Control (TC) bắt đầu xử lý gói tin sau điểm POSTROUTING trong linux kernel
2. Linux kernel định nghĩa cấu trúc `sk_buff` để đại diện cho một gói tin đang được xử lý cùng với thông tin kết nối mạng sẽ được sử dụng (trường `skb` → `dev`).
3. Hầu hết các công việc của TC được cài đặt trong hàm `dev_queue_xmit()` (file nguồn `net/core/dev.c`), với tham số là `sk_buff`
4. `dev_queue_xmit()` kiểm tra nếu gói tin sẵn sàng để được gửi đi (tương thích MTU, checksum OK, v.v.) → đưa `sk_buff` vào **queue phù hợp** của kết nối mạng `skb` → `dev`
5. Nếu queue bị đầy, `dev_queue_xmit()` không đưa gói tin vào hàng đợi nữa mà xử lý theo policy đã được khai báo (thông thường là drop gói tin và thông báo cho TCP flow control)
6. Triệu gọi hàm `qdisc_run()` để xử lý gói tin vừa được queue
7. Cuối cùng, `hard_start_xmit()` được gọi để chuyển gói tin ra kết nối mạng vật lý

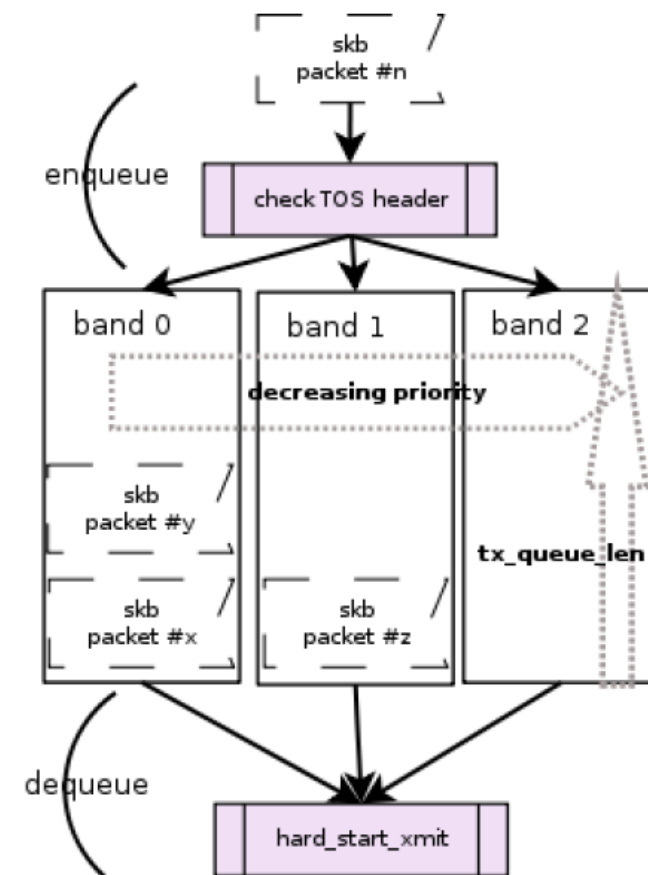


Source: Journey to the Center of the Linux Kernel

Queueing discipline (algorithm): pfifo_fast



- Giải thuật được cài đặt trong file kernel *net/sched/sch_generic.c* (schedule generic)
- Sử dụng thuật toán đơn giản fifo (first in first out)
- Các hàng đợi (queue band) được thiết lập mức ưu tiên: hàm *hard_start_xmit()* xử lý dequeue các hàng đợi theo mức ưu tiên (ví dụ hết band 2, sang band 1 rồi mới đến band 0)
- Hàng đợi cũng được thiết lập độ dài, là số lượng gói tin có thể đưa vào queue để chờ xử lý
- Dựa vào filter (ToS byte → DS codepoint), xác định gói tin cần đưa vào hàng đợi nào.
- Khi số gói tin trong hàng đợi đã đạt đến mức giới hạn, TC thực hiện drop các gói tin nhận được cho queue này
- *Nhắc lại:* TCP flow control kiểm soát không làm tràn queue bằng cách trao đổi với trạm truyền để thiết lập kích thước cửa sổ trượt phù hợp (sliding window)



Source: Journey to the Center of the Linux Kernel

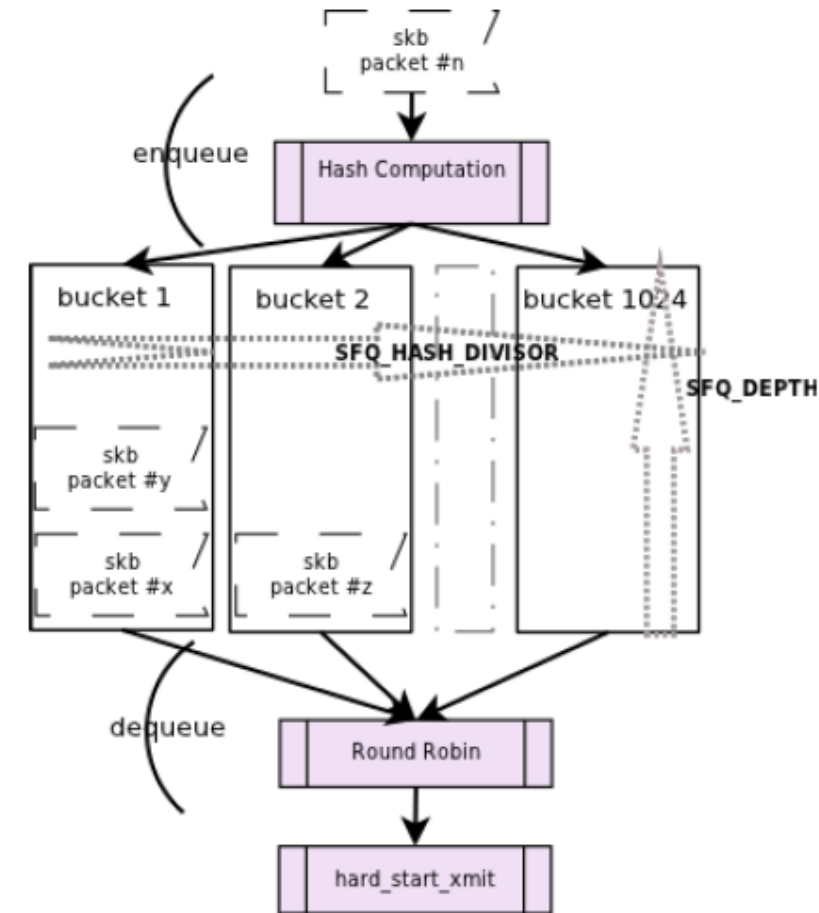
Stochastic Fairness Queuing (SFQ)

- SFQ: thuật toán hàng đợi công bằng ngẫu nhiên
- Chia sẻ băng thông giữa các ứng dụng mà không đưa ra đặc quyền cho bất cứ ai
- Ý tưởng tạo hàm băm (hash) trên các header của gói tin và sử dụng kết quả hàm băm này để đưa gói tin vào 1 trong 1024 thùng chứa (bucket) có sẵn. Các thùng chứa này được xử lý gửi gói tin đi (cho đến hết) theo qui tắc xoay vòng (round-robin).

```
$ tc qdisc add sfq help
```

```
Usage: ... sfq [ limit NUMBER ] [ perturb SECS ] [ quantum BYTES ]
```

- limit: kích thước của thùng chứa
- perturb: tần suất cập nhật hàm băm
- quantum: số lượng tối đa mà *dequeue* được phép lấy gói tin khỏi thùng chứa để truyền đi theo qui tắc round-robin



Và các qdisc khác trong Linux kernel

<https://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html>

- \$ tc

tc - show / manipulate traffic control settings

- CLASSLESS QDISCS

- choke: CHOOSE and Keep for responsive flows
- coDel: Adaptive "no-knobs" active queue management algorithm (AQM)
- [p|b]fifo: Simplest usable qdisc, pure First In, First Out behaviour
- fq: Fair Queue Scheduler
- fq_codel: Fair Queuing Controlled Delay
- gred: Generalized Random Early Detection
- hhf: Heavy-Hitter Filter
- ingress: Special qdisc applies to incoming traffic
- mqprio: The Multiqueue Priority Qdisc
- multiq: qdisc optimized for devices with multiple Tx queues
- netem: Network Emulator
- pfifo_fast: Standard qdisc for 'Advanced Router' enabled kernels
- sfb: Stochastic Fair Blue
- sfq: Stochastic Fairness Queueing
- tbfb: The Token Bucket Filter

- CLASSFUL QDISCS

- ATM: Map flows to virtual circuits
- CBQ: Class Based Queueing
- DRR: The Deficit Round Robin Scheduler
- DSMARK: Classify packets based on TOS field
- HFSC: Hierarchical Fair Service Curve
- HTB: The Hierarchy Token Bucket
- PRIO: Non-shaping container for a configurable number of classes
- QFQ: Quick Fair Queueing

PHB Demo with Linux Traffic Control

- Mặc định ban đầu: *qdisc tq_codel*

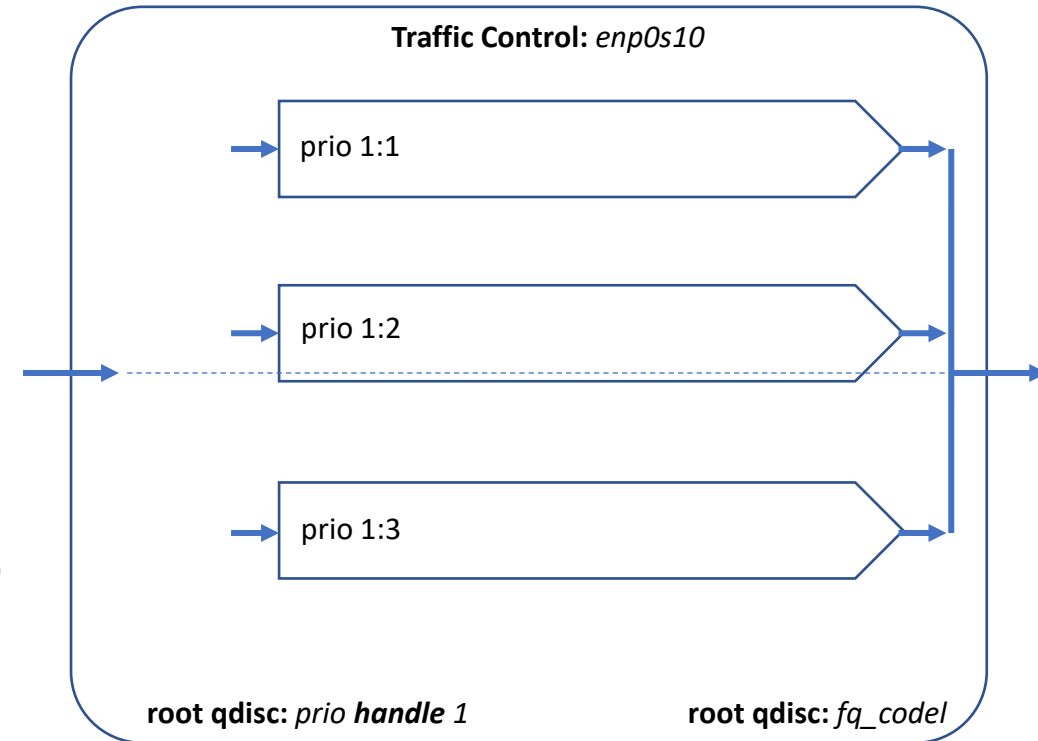
```
$ tc qdisc show dev enp0s10
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024
```

- Thiết lập qdisc gốc: *prio*

```
$ tc qdisc add dev enp0s10 root handle 1: prio
$ tc qdisc show dev enp0s10
qdisc prio 1: root refcnt 2 bands 3 priomap
```

- Qdisc *prio* cài đặt tự động 3 classes để xử lý hàng đợi cho các gói IP với mức ưu tiên khác nhau:

```
$ sudo tc class show dev enp0s10
class prio 1:1 parent 1:
class prio 1:2 parent 1:
class prio 1:3 parent 1:
```



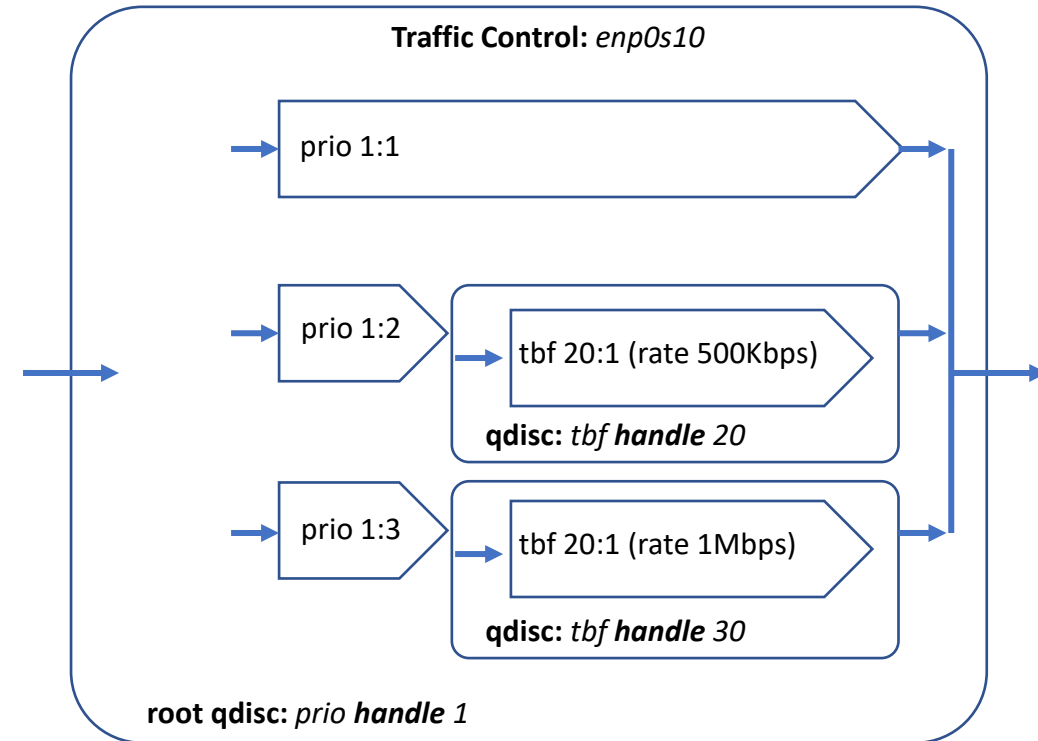
PHB Demo with Linux Traffic Control (2)

- Giữ nguyên class *prio 1:1* để phục vụ kênh truyền ưu tiên mức cao nhất, thiết lập các hạn chế bằng thông cho 2 kênh truyền 1:2 và 1:3
- Gán qdisc *tbf* cho class 1:2 & băng thông 500Kbps

```
$ tc qdisc add dev enp0s10 parent 1:2 \
    handle 20: tbf \
    rate 500kbit burst 3000 limit 5000
```
- Gán qdisc *tbf* cho class 1:3 & băng thông 1Mbps

```
$ tc qdisc add dev enp0s10 parent 1:3 \
    handle 30: tbf \
    rate 1mbit burst 3000 limit 9000
```
- Các class được qdisc *tbf* tự động thiết lập bên trong nó, giống như khi *prio* tự động thiết lập 3 class *prio*:

```
$ tc class show dev enp0s10
class prio 1:1 parent 1:
class prio 1:2 parent 1: leaf 20:
class prio 1:3 parent 1: leaf 30:
class tbf 30:1 parent 30:
class tbf 20:1 parent 20:
```



PHB Demo with Linux Traffic Control (3)

- Thiết lập filter để nhận gói tin từ nguồn 192.168.2.123 vào xử lý trong class 1:1

```
$ tc filter add dev enp0s10 \
  parent 1: protocol ip u32 match \
  ip src 192.168.2.123 flowid 1:1
```

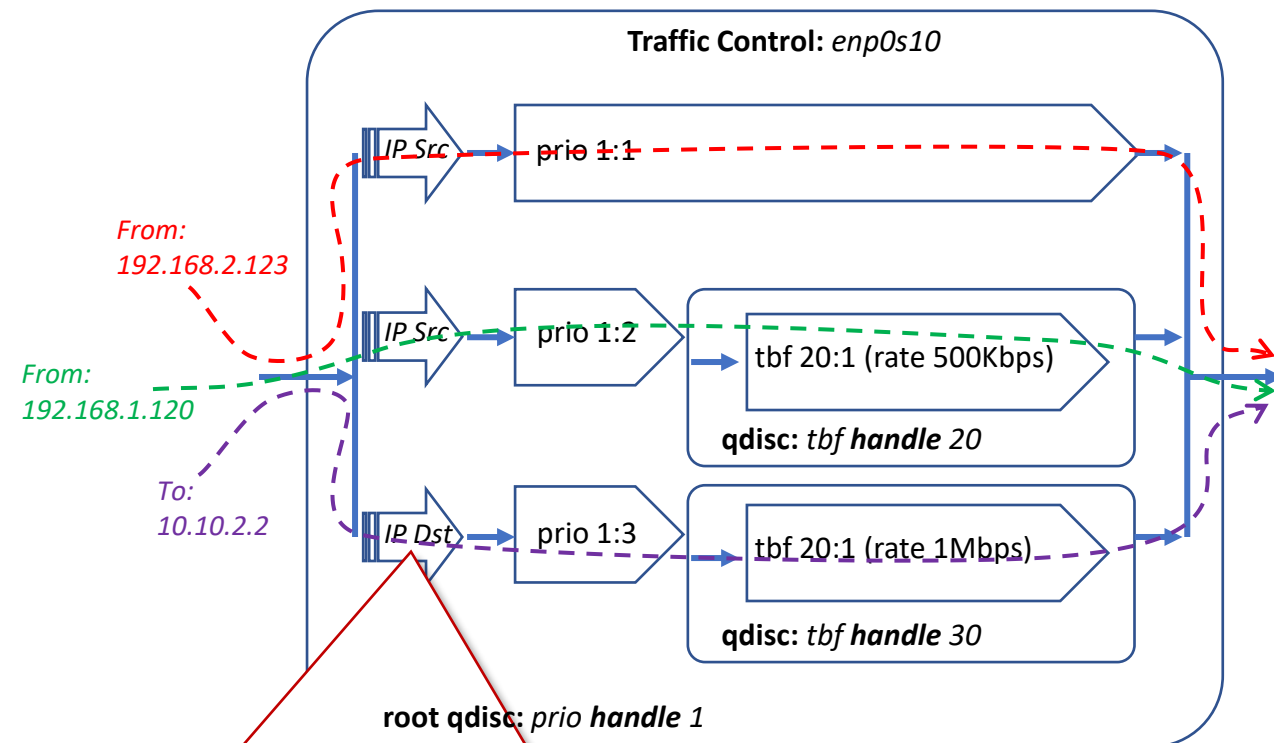
- Thiết lập filter để nhận gói tin từ nguồn 192.168.1.120 vào xử lý trong class 1:2

```
$ tc tc filter add dev enp0s10 \
  parent 1: protocol ip u32 match \
  ip src 192.168.1.120 flowid 1:2
```

- Thiết lập filter để nhận gói tin đến đích 10.10.2.2 vào xử lý trong class 1:3

```
$ tc filter add dev enp0s10 \
  parent 1: protocol ip u32 match \
  ip dst 10.10.2.2 flowid 1:3
```

- Các dòng gói tin đi qua được kiểm tra theo các filter và xử lý trong các class với qdiscs tương ứng



Ngoài địa chỉ IP nguồn & đích, filter có thể được thiết lập theo các trường dữ liệu khác trong gói tin IP, bao gồm ToS → triển khai được PHB theo DSCP codepoint:

```
$tc filter add dev enp0s10 \
  parent 1: protocol ip u32 match \
  ip tos 0x12 0xff flowid 1:1
```