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

IP QoS

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

Plan

- IP QoS Overview
- IntServ
- RSVP
- DiffServ
- Linux Traffic Control
- MPLS
- MPLS Applications & QoS Support

IP QoS Overview

History & Now

- Mạng TCP/IP (Internet) ban đầu được thiết kế cho Bộ QP Mỹ, với một chất lượng dịch vụ: Best Effort.
  - Không đảm bảo thời gian truyền cũng như khả năng thành công
  - Có một số thiết kế giao thức định hướng QoS, nhưng không được sử dụng
- QoS bắt đầu được quan tâm trong thập kỷ 1990
  - Các mạng riêng (private networks) sử dụng Internet làm công cụ truyền dữ liệu
  - Người dùng trả tiền để sử dụng Internet
  - Dịch vụ mới trên Internet ➔ công nghệ mới trên Internet (streaming over the Internet, etc.) ➔ cạnh tranh business ➔ trải nghiệm khách hang ➔ QoS!!!
- Integrated Service (1994) & RSVP (1997)
- Differentiated Service (1998) & PHB (AF: 1999, EF: 2003)
- MPLS (2001), MPLS + DiffServ (2002), LDP (2007), EXP (CoS: 2009)
- MPLS (Active WG): 2010 ~ 2021 (https://tools.ietf.org/wg/mpls)
- Ubuntu server 18.04 starts to support MPLS, 21.04 enhancement supports: http://manpages.ubuntu.com/manpages/impish/en/man8/tc-mpls.8.html

Need a "new Internet" with QoS

- Network flexibility is becoming central to enterprise strategy
  - Rapidly-changing business functions no longer carried out in stable ways, in unchanging locations, or for long time-periods
  - Network-enabled applications often crucial for meeting new market opportunities, but there's no time to custom-build a network
- Many recent applications, such as interactive voice or video applications, have serious demands on bandwidth and latency
- Applications requiring QoS: a traffic contract guarantees for a network ability of
  - Performance
  - Throughput
  - Latency
- Examples of applications that require such guarantee
  - Streaming multimedia may require guaranteed throughput
  - IP telephony may require strict limits on jitter and delay
  - Dedicated link emulation requires both guaranteed throughput and imposes limits on maximum delay
  - Safety-critical application, such as remote surgery may require a guaranteed level of availability (this is also called hard QoS).

How is QoS measured

- Three measurements are used to determine the quality of service
- Dropped packets
  - Percentage of packets lost as they move from end to end.
- Jitter
  - Unpredictable variable in delay caused by congestion.

```
:~$ iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.3.115 port 5001 connected with 192.168.1.120 port 54613
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total
Datagrams
[  3]  0.0- 1.0 sec  61.7 KBytes   506 Kbits/sec   3.459 ms   42/   85 (49%)
[  3]  1.0- 2.0 sec  58.9 KBytes   482 Kbits/sec   3.130 ms   47/   88 (53%)
[  3]  2.0- 3.0 sec  60.3 KBytes   494 Kbits/sec   3.125 ms   49/   91 (54%)
[  3]  0.0- 3.1 sec   184 KBytes   490 Kbits/sec   2.964 ms  140/  268 (52%)
```

- Latency
  - Time it takes a signal to move through a unit in test.
  - Low latency must be designed into a network from the start and it can not be changed later.

➔ Up to now, applications cannot require network to provide (guarantee such measures)

QoS problem: the internetworking

- When the Internet was created, there was no need for a QoS.
- Internetworking mechanism ran on a “best effort” system.
- Internetworking was never intended to support real time applications.
- Type of service (ToS) exists in IPv4 and IPv6, but has not been utilized by networks.
- IP transport is unreliable because as packets travel from origin to destination they can be
  - Dropped
    - Routers buffer was full when packet arrived.
    - Depends on state of a network.
  - Delayed
    - Routers in middle had long queues.
    - Packet took a longer router to avoid congestion.
  - Out of order
    - Packets took different routes with different delays.
  - Error
    - packets are misdirected, or combined together, or corrupted, while in route.

	Bits 0–7	Bits 8–15	Bits 16–23	Bits 24–31
Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags and offset*	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (*optional*)	Payload Data			

Hang on, Internet myth about Bandwidth!!!

- It is more cost effective to "buy" 200% more bandwidth than a network requires than it is to worry about QoS.
  - Standards are being developed that will change this.
  - Internet is still growing and bandwidth alone can’t provide solutions needed due to its growth.
- Over designing a network and throwing bandwidth at the QoS problem is only a temporary fix -- not a solution.
- The “if you build it they will come” phenomenon. The faster the network, the more user traffic it will have.

Thực tế là 50 năm lịch sử Internet đã vận hành theo cách này, và đến nay vẫn đang vận hành khá tốt như vậy!!!

QoS Definitions

- Quality of Service (QoS) classifies network traffic and then ensures that some of it receives special handling.
  - May track each individual dataflow (sender:receiver) separately.
  - May include attempts to provide better error rates, lower network transit time (latency), and decreased latency variation (jitter).
- Differentiated Class of Service (CoS) is a simpler alternative to QoS.
  - Doesn't try to distinguish among individual dataflows; instead, uses simpler methods to classify packets into one of a few categories.
  - All packets within a particular category are then handled in the same way, with the same quality parameters.
- Policy-Based Networking provides end-to-end control.
  - The rules for access and for management of network resources are stored as policies and are managed by a policy server.

Obtaining QoS

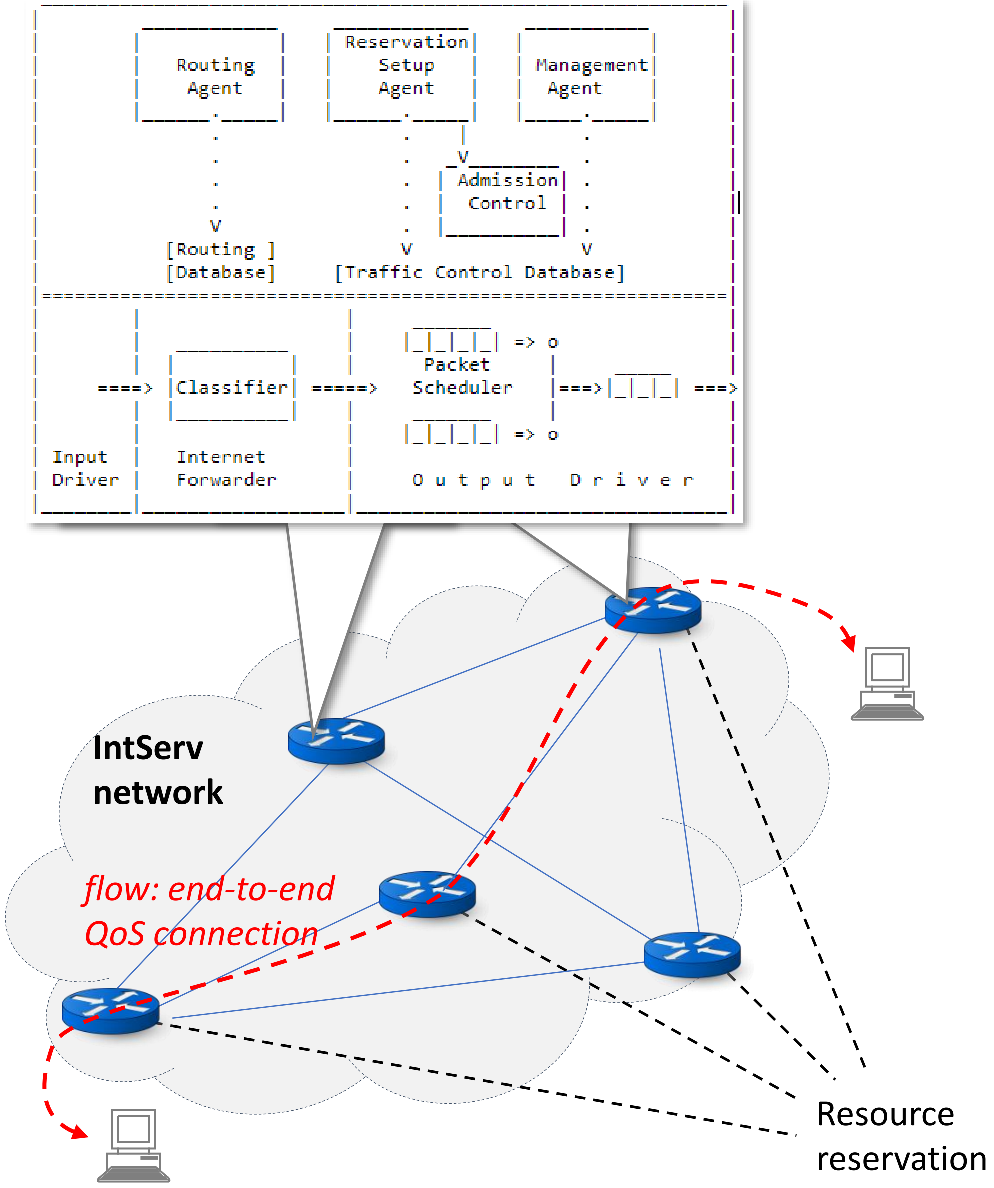
- Integrated Services(IntServ)
  - Application that requires some kind of guarantees has to make an individual reservation.
- Differentiated Services(DiffServ)
  - Categorizes traffic into different classes, also called class of service (CoS), and applies QoS parameters to those classes.
- Multiprotocol Label Switching(MPLS)
  - Tagging each packet to determine priority.
- 802.11e
  - Packets carry their priority code.
  - For wireless last mile.

Integrated Services (IntServ)

IntServ Overview

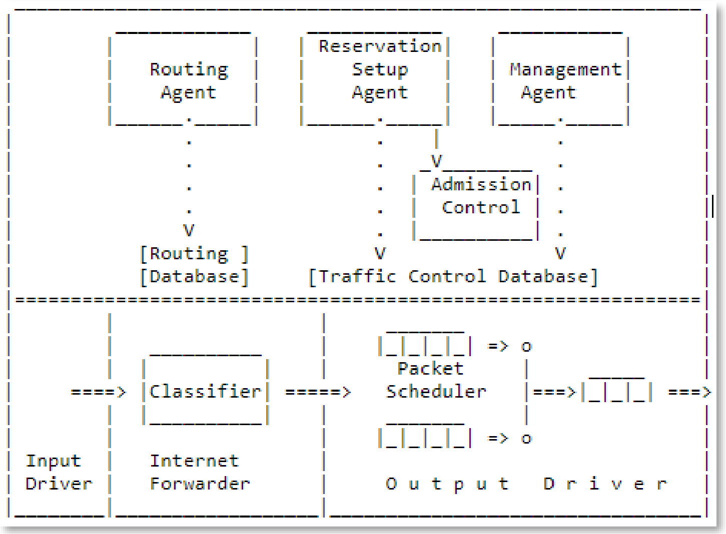
- RFC1633 (1994)
- Not a technical specification, but a model that attempts to guarantee QoS on networks by reserving resources
- Every router in the system implements IntServ components (reference model for router)
- Every application that requires some kind of guarantees has to make an individual reservation
  - per-flow reservations
  - flows provide traffic characterization
  - “heavy” state: per-flow

Implementation Reference Model for Routers (RFC1633)



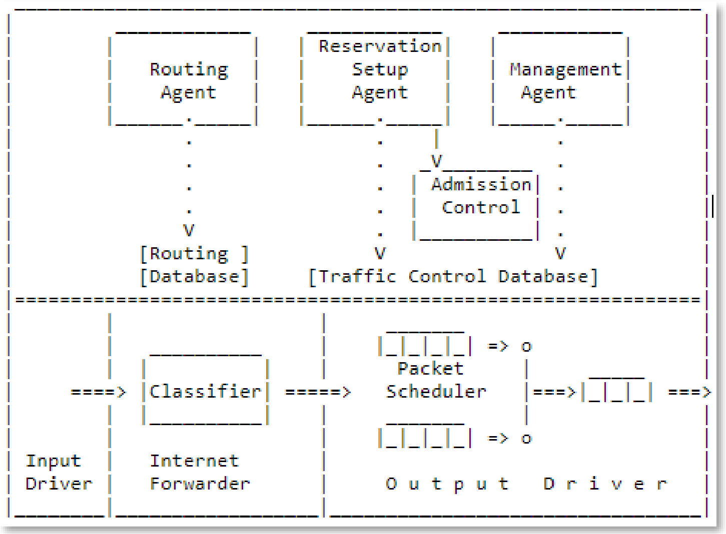
IntServ Components

- All components implemented at all routers!
- Packet Scheduler
  - Manages forwarding of different streams
  - Required resources: sets of queues, timers
  - Example: Implementation of Weighted Fair Queuing (WFQ)
- Classifier
  - Maps packets to a class
  - Packets in same class treated similarly
  - Examples:
    - per-flow class
    - video-packet class



IntServ Components (2)

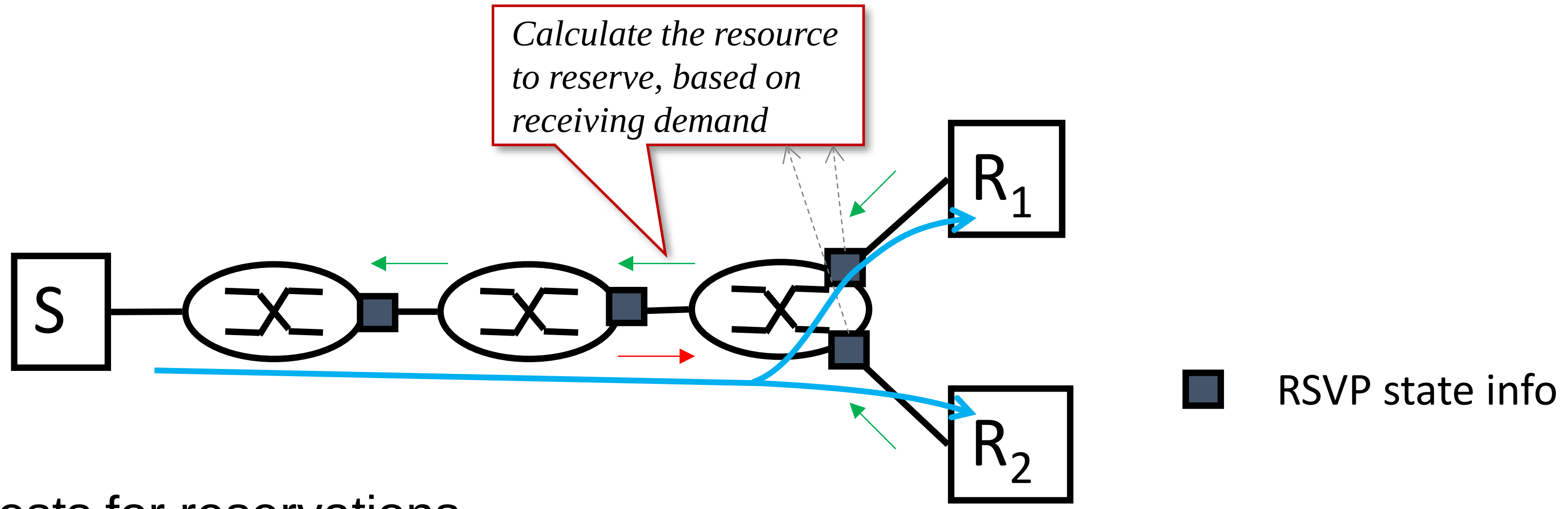
- Admission Controller
  - Determines whether or not to admit a new flow
  - Q: why would a flow be rejected?
  - Requirements:
    - Knowledge of available resources at router
    - (conservative) model of flow’s resource consumption
    - e.g., leaky bucket
  - The hard part: getting apps to characterize their flows
- Reservation Setup Protocol
  - Sets up and maintains (distributed) flows’ network resource usage
    - “negotiates” between admission controllers at routers
    - establishes active classifiers at routers
  - RSVP protocol



ReSerVation protocol (RSVP)

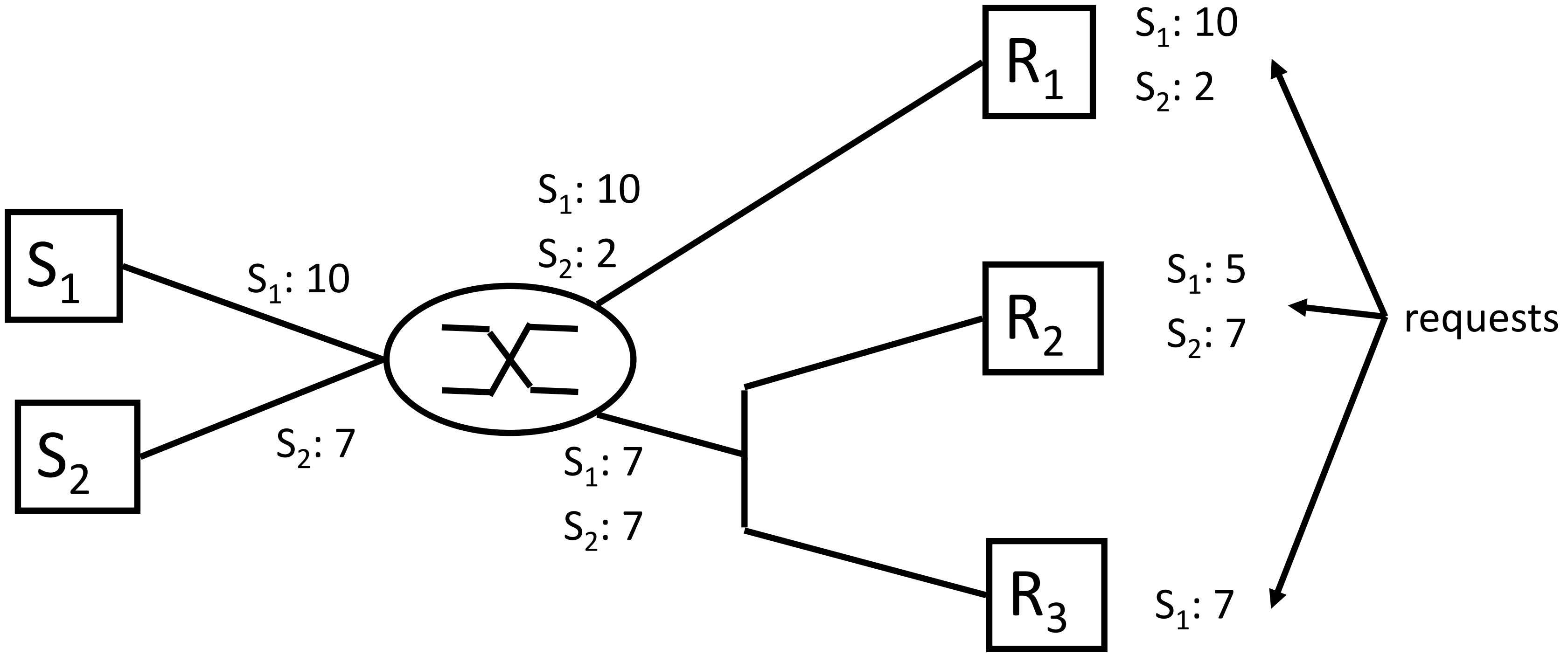
- RSVP is used by routers to deliver QoS requests to all nodes along the path of the flow and to establish and maintain state to provide the requested service (connection-oriented in IP level).
- Designed mainly for multicast sessions (unicast is a special case)
- One way reservation: from receiver, upstream until sender
- To make reservations, the RSVP daemon communicates with two local decision modules:
  - Admission control - determines whether the node has sufficient available resources to supply the requested QoS.
  - Policy control - determines whether the user has administrative permission to make the reservation.
- If either check fails, the RSVP program returns an error notification to the application.
- If both checks succeed, the RSVP daemon sets parameters in a packet classifier and packet scheduler to obtain the desired QoS.
  - Packet classifier - determines the QoS class for each packet.
  - Packet scheduler - orders packet transmission to achieve the promised QoS for each stream.

RSVP Messaging



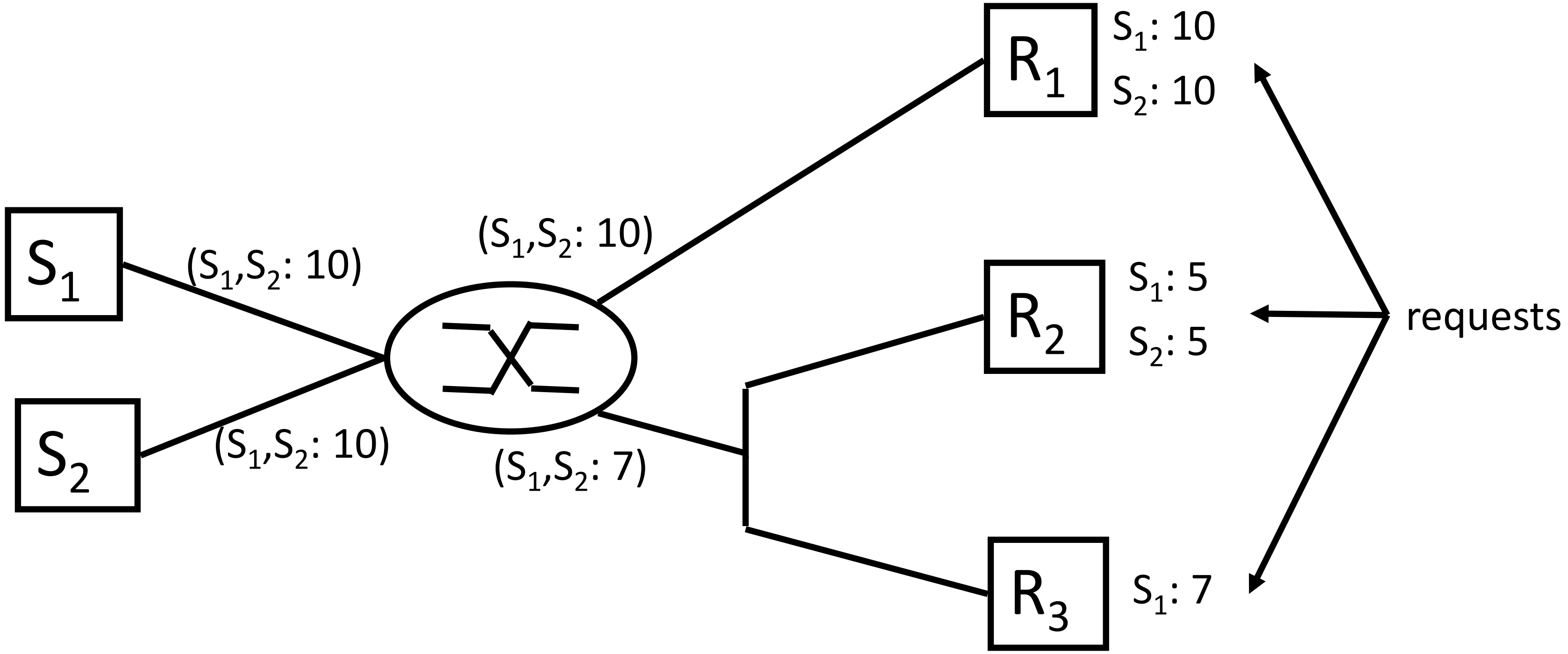
- Receivers make requests for reservations.
- Sufficient resources: Router reserves per outgoing interface (i.e., link) and forwards request upstream
- Insufficient: send ResvError message downstream
- Path messages: from sender toward rcvr so that routers know where to forward receiver requests.
- Problem of resource calculation for reserving

Resource calculation for reservation



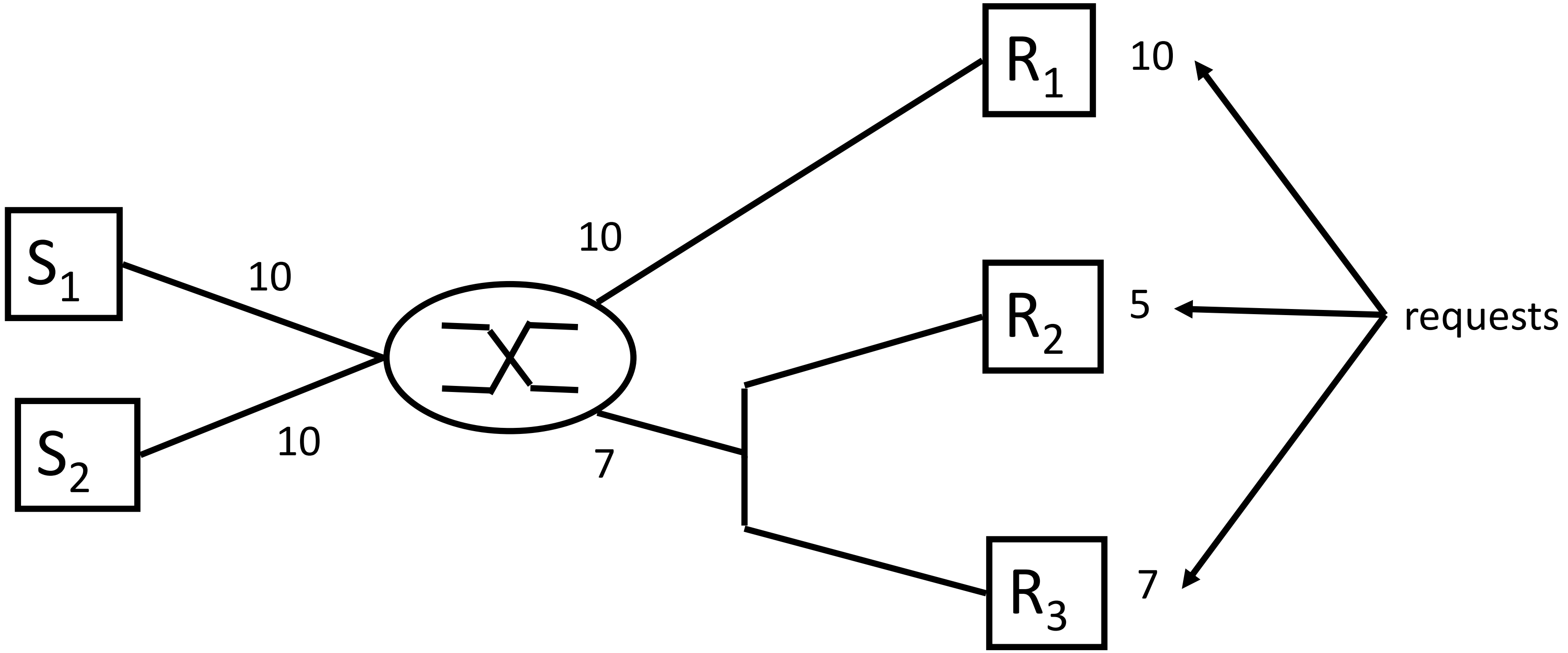
- Fixed-Filter: reservation per sender
  - Senders don't "share" bandwidth
  - Dynamic event: receivers wants to change a sender allocation
- Allocation per sender indicated
  - Sample application: multimedia (e.g., send audio (S1) and video (S2) at same time)

Shared-Explicit Reservation



- Shared-Explicit: reservation per list-of-senders
  - Fixed set of senders “share” bandwidth
  - Dynamic event: receivers wants to add/remove sender or change group allocation
- Allocation shared by list of senders
  - Sample application: multimedia (e.g., debate w/ 2 speakers)

Wildcard Reservation



- Wildcard-Filter: no sender specified w/ reservation
  - Any sender can “share” bandwidth
  - Dynamic event: new sender begins transmitting, rcvr wants to increase its receiving allocation
- Allocation shared by all senders
  - Sample application: town meeting (one sender, but not clear who the speakers might be)

IntServ Problems

- Reservation protocols and structure complicated
  - lots of message passing
  - coordination problems
- All routers maintain state
  - state maintenance requires additional processing / memory resources
  - Lots of flows traverse core (backbone) routers
    - Lots of state: need more memory
    - Lots of RSVP messages to process: slows transfer speeds
    - Scheduler and Classifier have too much to deal with
- Why did IntServ fail?
  - Economic factors
    - Deployment cost vs Benefit
  - Is reservation, the right approach?
    - Multicast centric view
  - Is per-flow state maintenance an issue?
  - More about QoS in general …

Differentiated Services(DiffServ)

IP QoS: Type of Service – Starting Point

- IP packets have a field called Type of Service (also known as the TOS byte).
- Original idea of TOS byte was that we could specify a priority and request a route for high throughput, low delay and high reliable service
- TOS byte was defined RFC791 (1981) but the way we use it has changed throughout the years.
- ToS byte:
  - 8 bit
  - Precedence: 3 bit. The higher the value, the more important the IP packet is, in case of congestion the router would drop the low priority packets first.
  - ToS: 5 bit. To assign what kind of delay, throughput and reliability we want

	Bits 0–7	Bits 8–15	Bits 16–23	Bits 24–31
Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (*optional*)	Payload Data			

IP QoS(2): Precedence & ToS

- RFC791 (1981):

Precedence (3 bit):

000	Routine	011	Flash	110	Internetwork Control
001	Priority	100	Flash Override	111	Network Control
010	Immediate	101	Critic/Critical		

Type of Service (5 bit):

Bit 3: 0 = normal delay 1 = low delay
Bit 4: 0 = normal throughput 1 = high throughput
Bit 5: 0 = normal reliability 1 = high reliability
Bit 6-7: Reserved for future use

- RFC1349 (1992):
  - Must Be Zero (MBZ)

ToS Byte:

0	1	2	3	4	5	6	7
Precedence			Type of Service				MBZ

Type of Service (4 bit):

1000 minimize delay
0100 maximize throughput
0010 maximize reliability
0001 minimize monetary cost
0000 normal service

IP QoS(3): DSCP for DiffServ

- RFC2474 (1998) with new ToS Byte format

ToS Byte (RFC2474 – 1998):

0	1	2	3	4	5	6	7
DSCP						CU *(currently unused)*	

- DS CodePoint (DSCP): affect the PHB (Per Hop Behavior)
  - Similar to *Precedence* that used in the TOS byte
  - Each class specifies a buffer and bandwidth.
  - Precedence priority: if class #2 has to be dropped, routers start dropping AF23 first then AF22, and AF21

DifServ codepoint (6 bit):

	Class 1	Class 2	Class 3	Class 4
Low Drop Precedence	*(AF11: 10)* 001010	*(AF21: 18)* 010010	*(AF31: 26)* 011010	*(AF41: 34)* 100010
Medium Drop Precedence	*(AF12: 12)* 001100	*(AF22: 20)* 010100	*(AF32: 28)* 011100	*(AF42: 36)* 100100
High Drop Precedence	*(AF13: 14)* 001110	*(AF23: 22)* 010110	*(AF33: 30)* 011110	*(AF43: 38)* 100110

IP QoS(4): RFC4594 recommendations

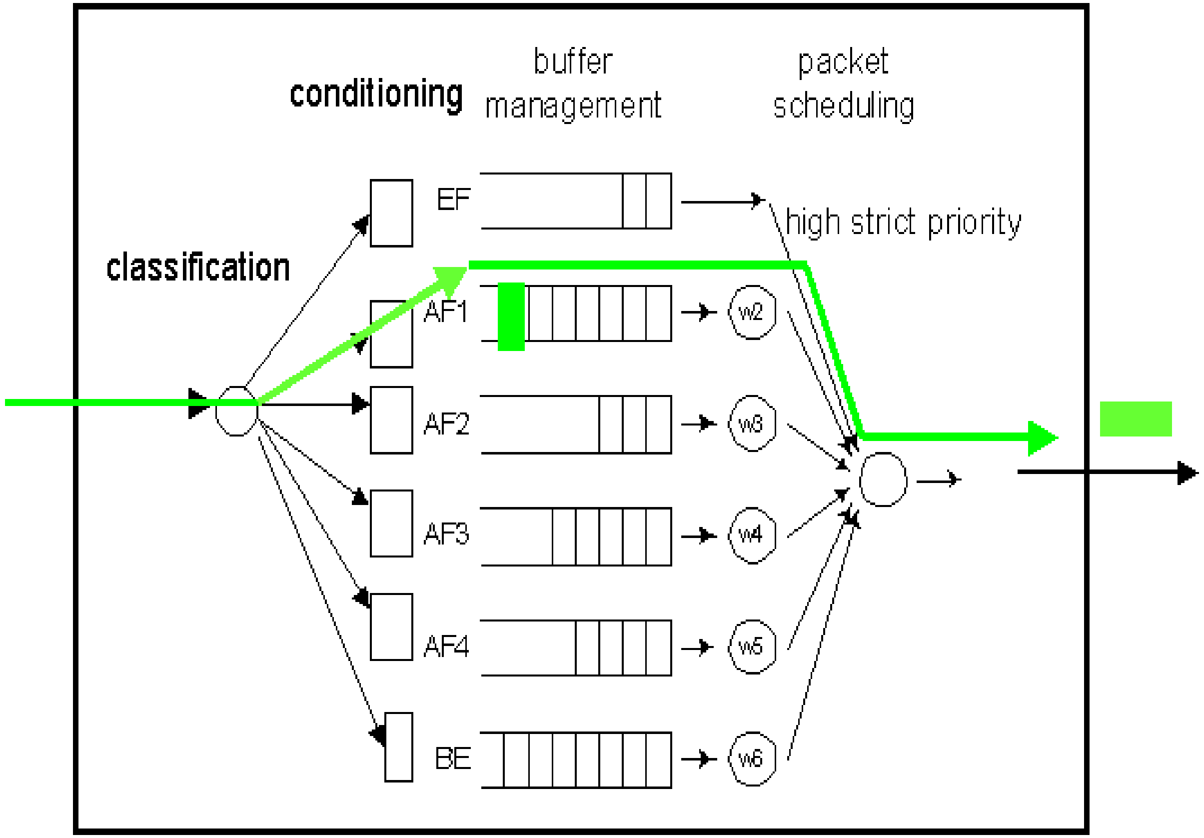
- Configuration Guidelines for DiffServ Service Classes (2006)
- Extend RFC2474 (1998)
- Recommendations for the use of code points
- DS codepoint: user services (Assured Forwarding - AF)
- CS#x: signaling
- DF: default forwarding
- EF: expedited forwarding
- Queuing: rate or priority
- AQM: Controlled-Delay Active Queue Management algorithm

Service class	DSCP Name	DSCP Value	PHB	Queuing	AQM
Network control	CS6	48	RFC 2474	Rate	Yes
Telephony	EF	46	RFC 3246	Priority	No
Signaling	CS5	40	RFC 2474	Rate	No
Multimedia conferencing (class 4)	AF41, AF42, AF43	34, 36, 38	RFC 2597	Rate	Yes per DSCP
Real-time interactive	CS4	32	RFC 2474	Rate	No
Multimedia streaming (class 3)	AF31, AF32, AF33	26, 28, 30	RFC 2597	Rate	Yes per DSCP
Broadcast video	CS3	24	RFC 2474	Rate	No
Low-latency data (class 2)	AF21, AF22, AF23	18, 20, 22	RFC 2597	Rate	Yes per DSCP
OAM	CS2	16	RFC 2474	Rate	Yes
High-throughput data (class 1)	AF11, AF12, AF13	10, 12, 14	RFC 2597	Rate	Yes per DSCP
Standard	DF	0	RFC 2474	Rate	Yes
Low-priority data	CS1	8	RFC 3662	Rate	Yes

Priority: xuất hiện làn xe ưu tiên, các làn xe khác phải dừng lại nhường đường
Rate: đèn giao thông tại các điểm giao cắt (nhịp xanh đỏ có ưu tiên tuyến đường - rate)

PHB: DiffServ processing in router

- Per Hop Behavior
- Default:
  - Best Effort (BE)
  - Default DS code: 000000
  - BE buffering & queue
- Top priority:
  - Expedited Forwarding (EF)
  - Component of the integrated services model
  - Provides a guaranteed bandwidth service
- Packet classification by DS class (ToS)
  ➔ buffer management
  (& drop by precedence priority)
- Bandwidth by DS class: packet scheduling



2 Competing PHBs

- Expedited Forwarding (EF) [RFC 2598]
  - Router must support classes' configured rates
  - EF class allocated fixed portion of router processing per unit time, e.g.,
    - Class-based queueing (CBQ) with priority to EF queue
    - Weighted Fair Queuing

- Assured Forwarding (AF) [RFC 2597]
  - N classes (current standard: N=4)
  - M possible drop preferences within class (current standard: M=3)
  - Each classes' traffic handled separately
  - Packet drop "likelihood" increases drop preference

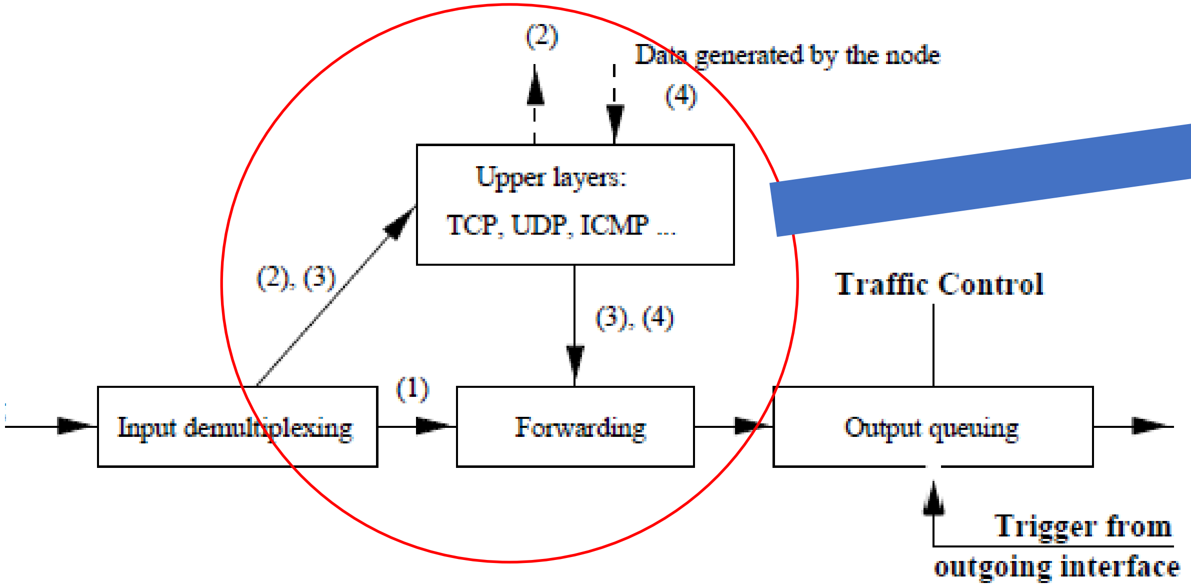
DifServ codepoint (6 bit):

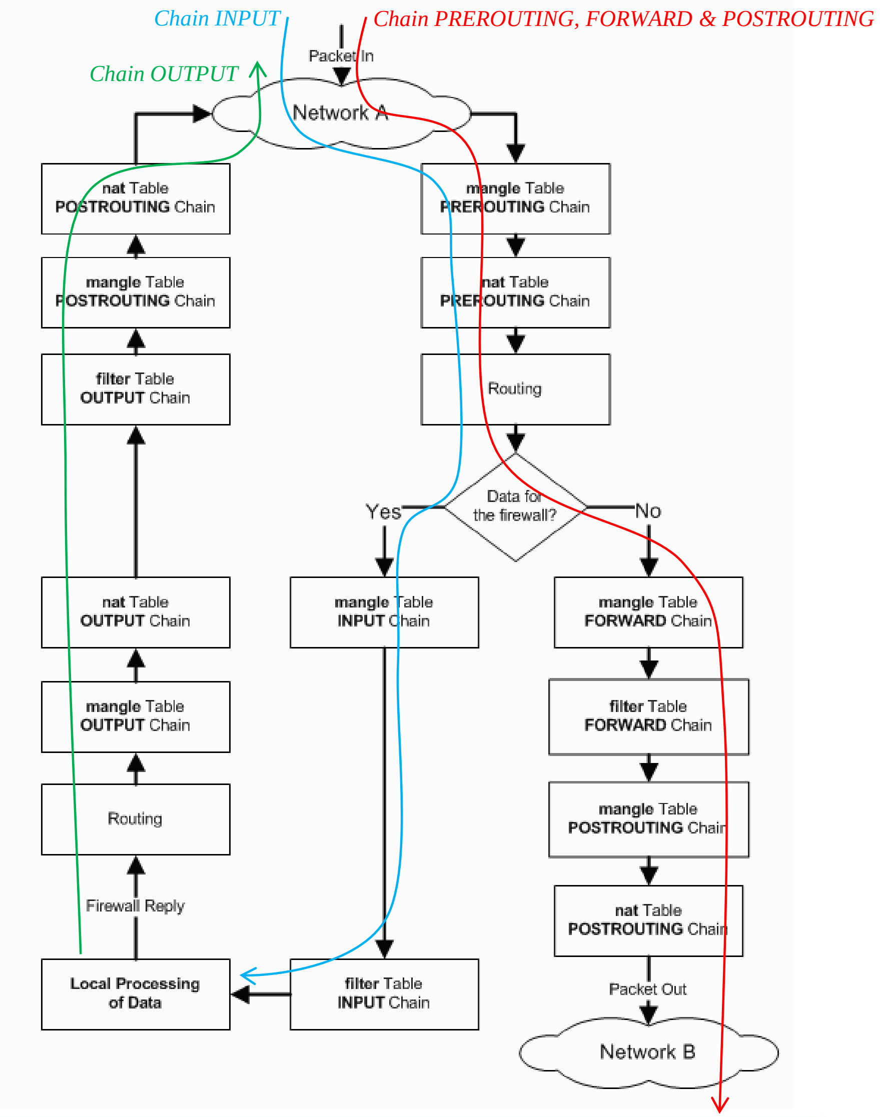
	Class 1	Class 2	Class 3	Class 4
Low Drop Precedence	*(AF11: 10)* 001010	*(AF21: 18)* 010010	*(AF31: 26)* 011010	*(AF41: 34)* 100010
Medium Drop Precedence	*(AF12: 12)* 001100	*(AF22: 20)* 010100	*(AF32: 28)* 011100	*(AF42: 36)* 100100
High Drop Precedence	*(AF13: 14)* 001110	*(AF23: 22)* 010110	*(AF33: 30)* 011110	*(AF43: 38)* 100110

Linux Traffic Control

Traffic Control in Linux kernel

- Remeber the packet flow in *iptables*
- Packet incomming 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.

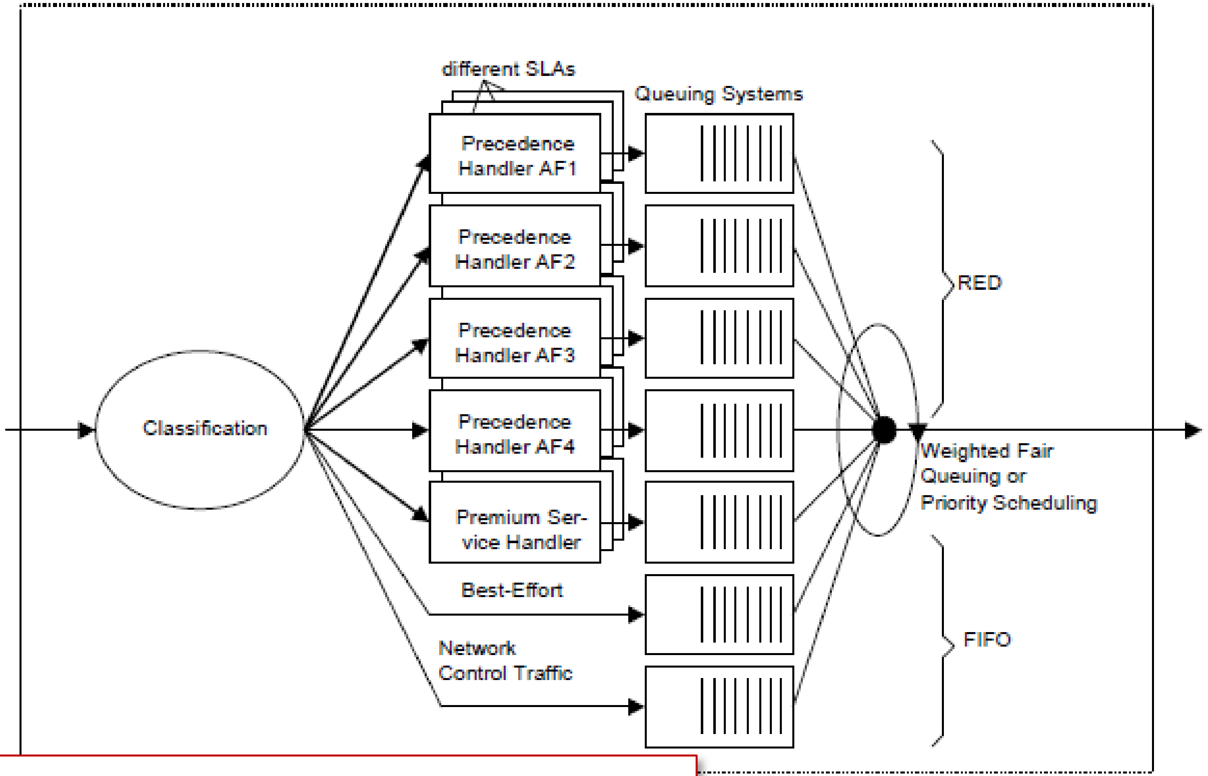




DiffServ implementation in Linux

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 codel* 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



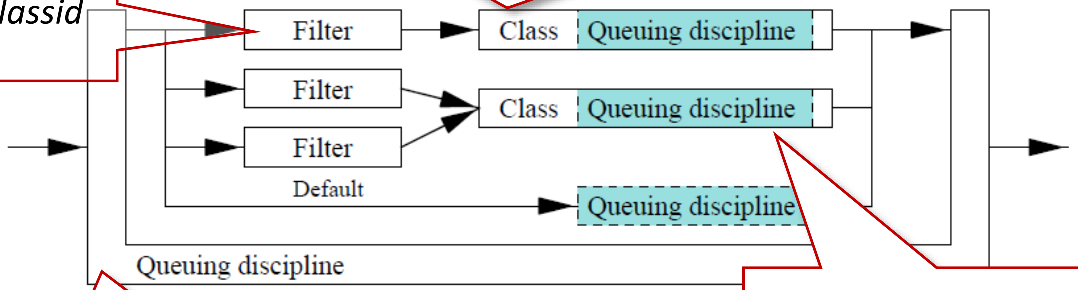

```
~$ tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc fq_codel 0: dev enp0s3 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval 100.0ms memory_limit 32Mb ecn
qdisc fq_codel 0: dev enp0s8 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval 100.0ms memory_limit 32Mb ecn
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 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 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) mached
  ➔ default queuing discipline (Best Effort).
- Each class defines some specific values in processing packets: min/max rate, priority (in comparing with other classes), etc…
- Recursive queuing 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 enqueuing of the new one.
- Each network device has an associated queuing 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*

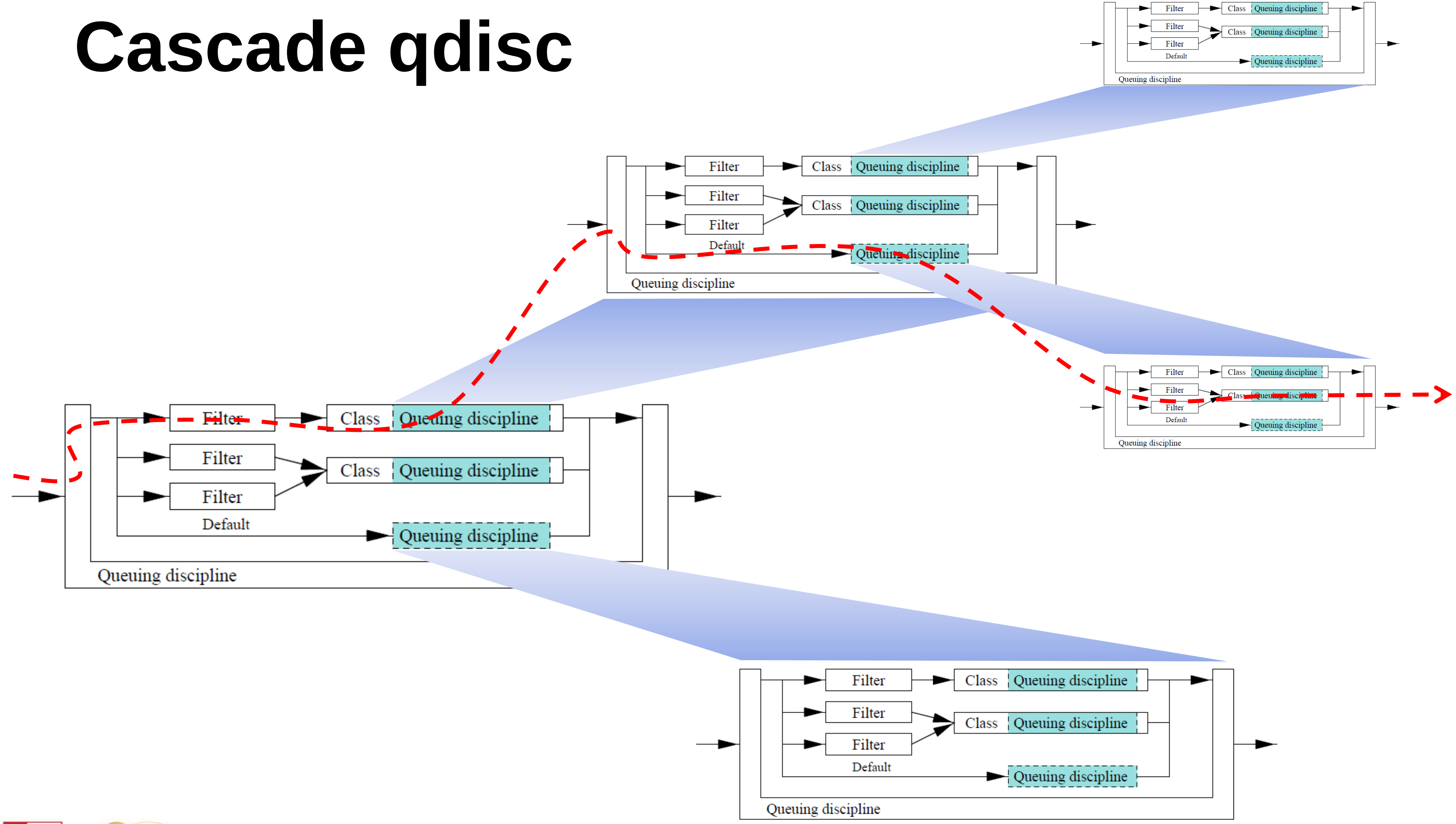
~$ **tclass** *classid 1:10 htb rate 1mbit ceil 20mbit prio 1*



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

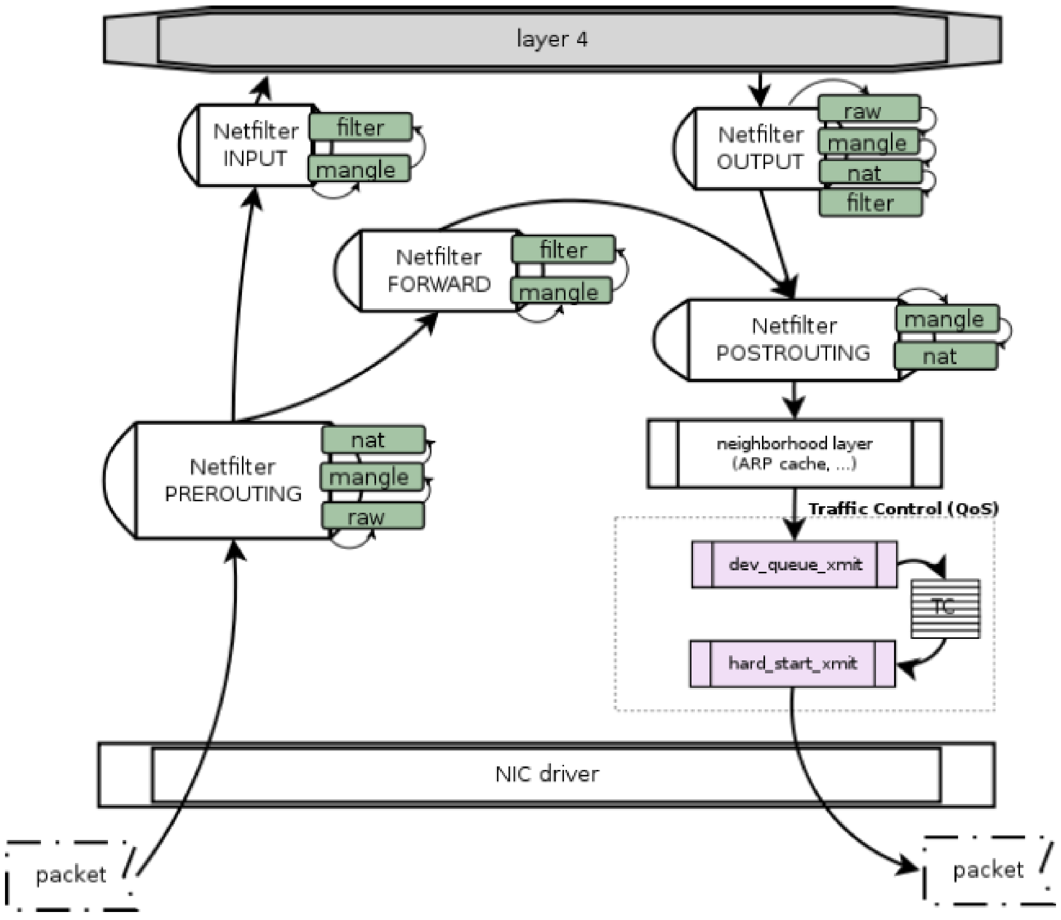
~$ **tc qdisc** *replace dev eth0 root handle 1: htb default 30*

Cascade qdisc



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

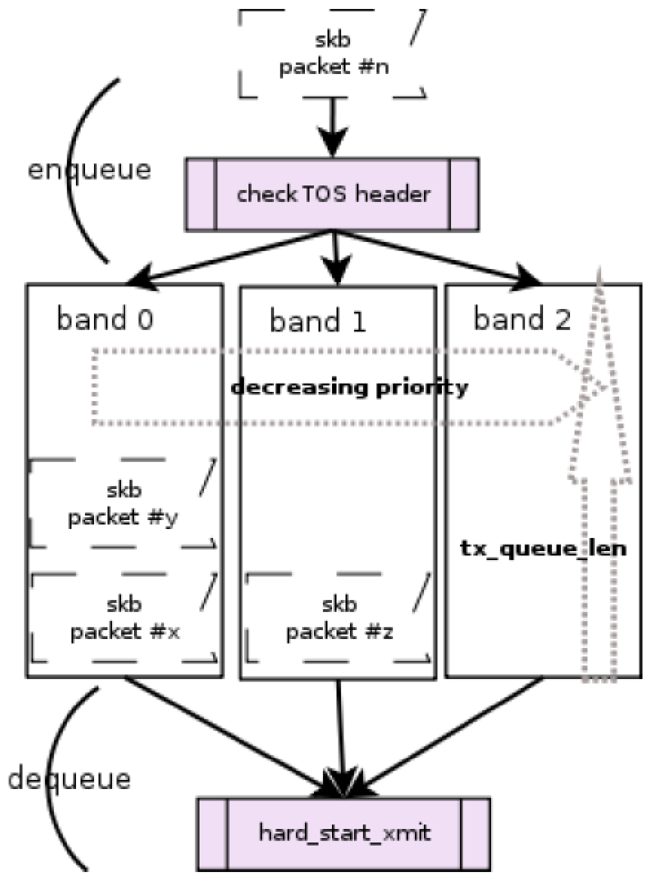


Source: *Journey to the Center of the Linux Kernel*

Queueing discipline (algorithm): pfifo_fast

Self tudy

- 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 cùng được thiết lập độ dài, là số lượng gói tin có để 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*

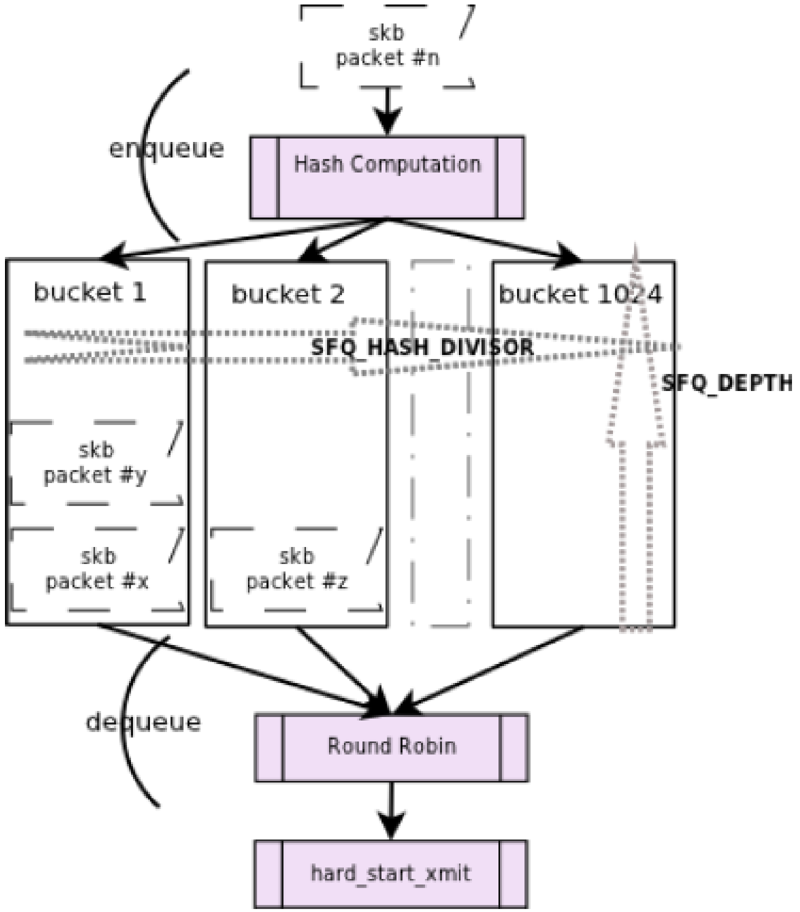
Stochastic Fairness Queuing (SFQ)

Self-study

- 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
  - tbf: 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

**các qdisc sẽ được sử dụng trong demo/thực hành*

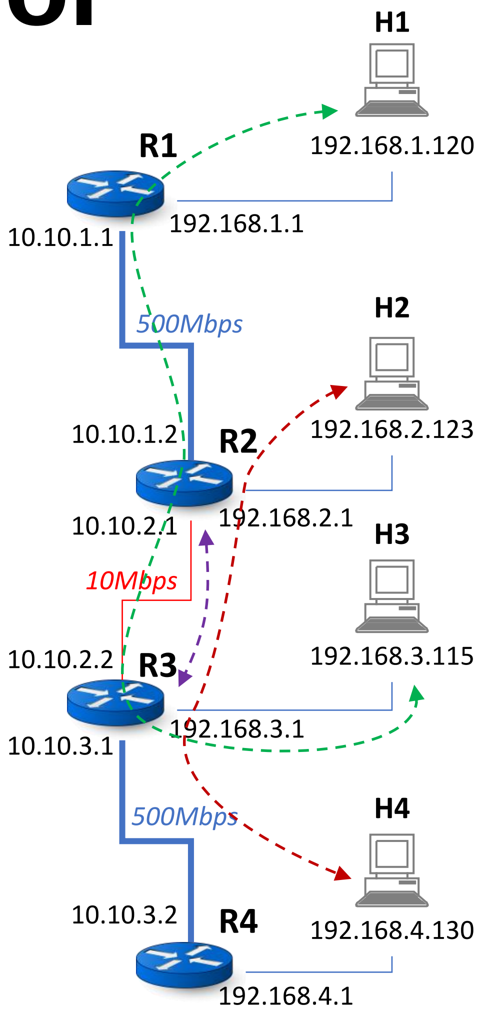
PHB Demo with Linux Traffic Control

- Kênh truyền vật lý R2-R3 là 10Mbps
- Dòng dữ liệu *iperf* H2-H4: 15Mbps
- Thêm dòng dữ liệu *iperf* H1-H3.
  ➔ H2-H4 giảm còn 1Mbps
- Thêm dòng dữ liệu *iperf* R2-R3.
  ➔ H2-H4 giảm còn 100Kbps
- Lý do: xử lý gói tin của các luồng cạnh tranh trên router R2
- Bài toán: đảm bảo kênh truyền H2-H4 để không bị ảnh hưởng khi xuất hiện các luồng cạnh tranh

```
H2:~$ iperf -c 192.168.4.130 -i 1 -t 30
------------------------------------------------------------
Client connecting to 192.168.4.130, TCP port 5001
------------------------------------------------------------
[  3] local 192.168.2.123 port 51806 connected with
192.168.4.130 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 1.0 sec  1.19 MBytes  9.96 Mbits/sec
[  4]  1.0- 2.0 sec  1.18 MBytes  9.92 Mbits/sec
[  4]  2.0- 3.0 sec  1.17 MBytes  9.82 Mbits/sec
. . .
```

```
[  3]  1.0- 2.0 sec   156 KBytes  1.27 Mbits/sec
[  3]  2.0- 3.0 sec  0.00 Bytes  0.00 bits/sec
[  3]  3.0- 4.0 sec  63.6 KBytes   521 Kbits/sec
. . .
```

```
[  4]  3.0- 4.0 sec  11.3 KBytes  92.7 Kbits/sec
[  4]  4.0- 5.0 sec  19.8 KBytes   162 Kbits/sec
[  4]  5.0- 6.0 sec  12.7 KBytes   104 Kbits/sec
```



PHB Demo with Linux Traffic Control

- Mặc định ban đầu: *qdisc fq_codel*

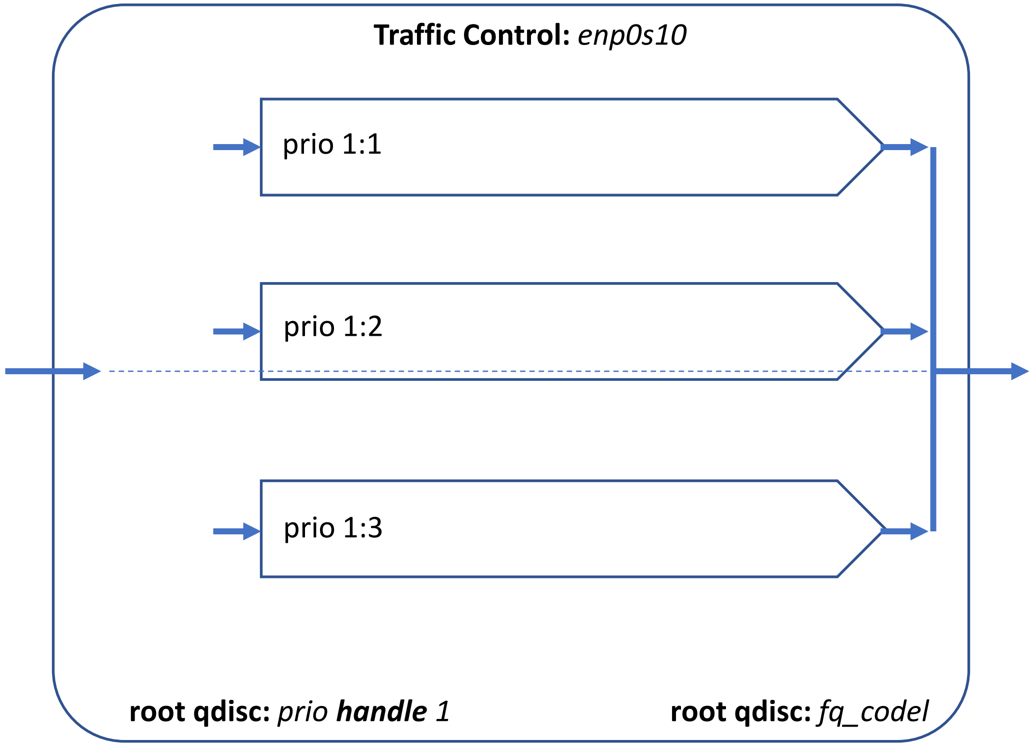
```
$ tc qdisc show dev enp0s10
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024
```

- 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:
```



qdisc fq_codel: https://man7.org/linux/man-pages/man8/tc-fq_codel.8.html
qdisc prio: https://man7.org/linux/man-pages/man8/tc-prio.8.html

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

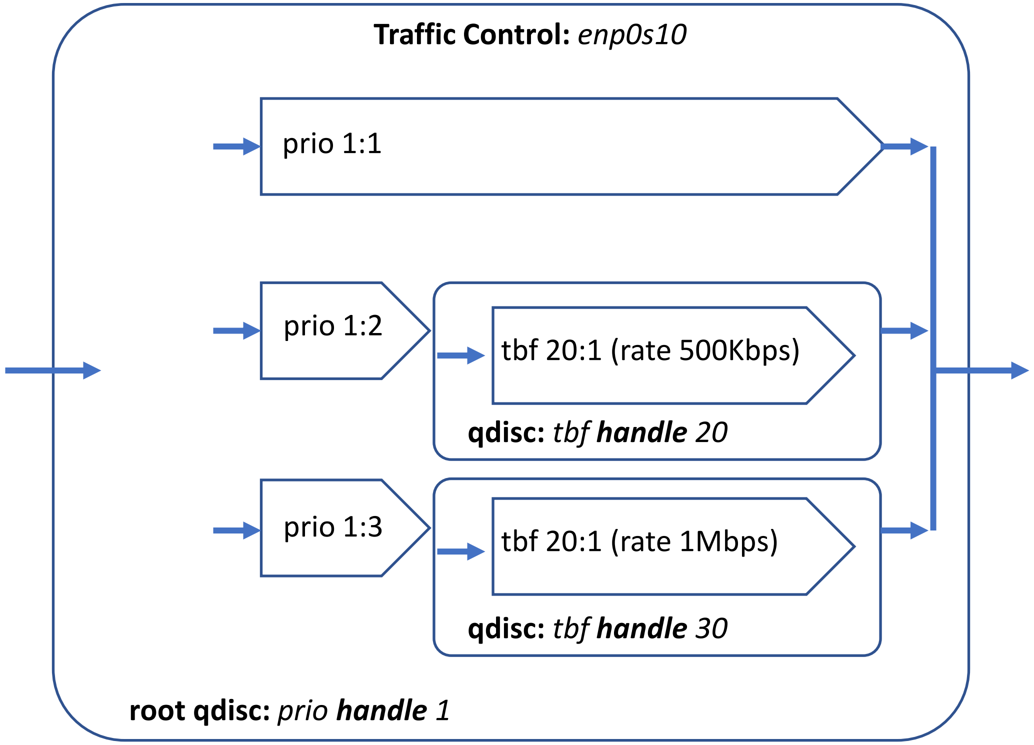
```
$ tc qdisc add dev enp0s10 parent 1:2 \
      handle 20: tbf \
      rate 500kbit burst 3000 limit 5000
```

- 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 thiếp 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:
```



qdisc fq_codel: https://man7.org/linux/man-pages/man8/tc-fq_codel.8.html
qdisc prio: https://man7.org/linux/man-pages/man8/tc-prio.8.html
qdisc tbf: https://man7.org/linux/man-pages/man8/tc-tbf.8.html

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
```

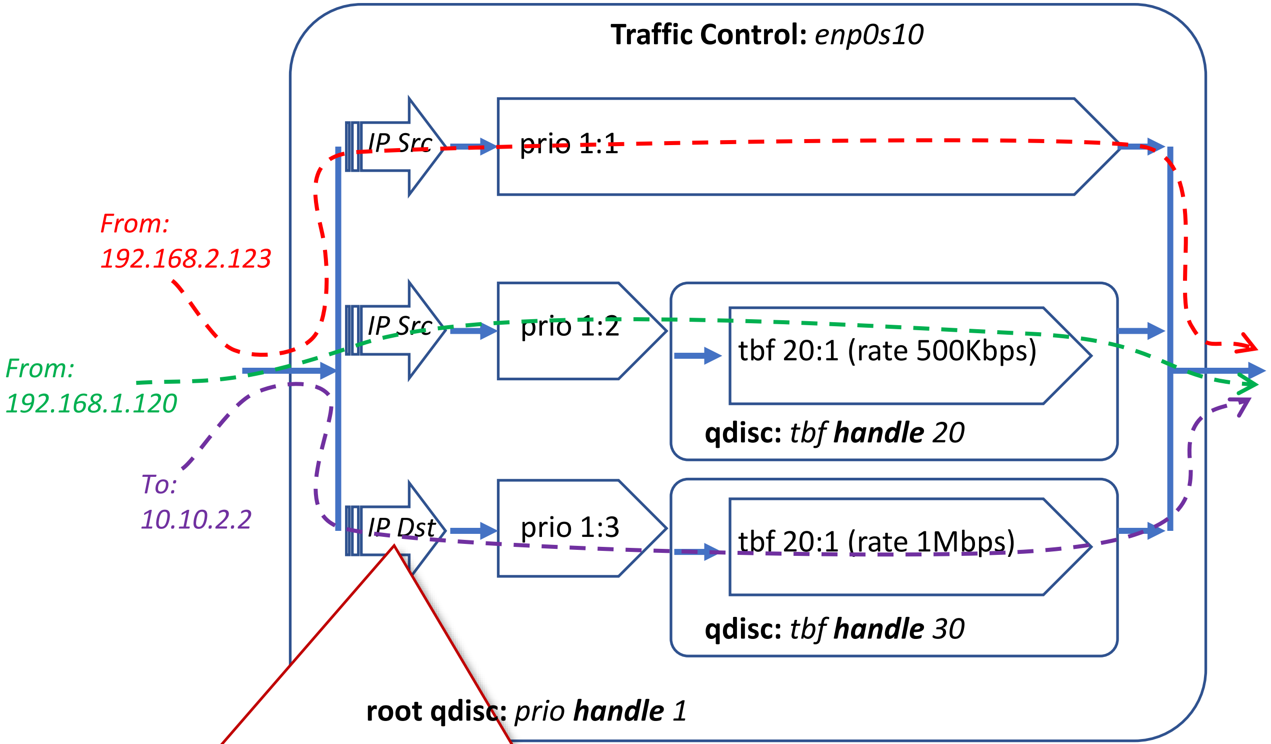
- Thiết lập filter để nhận gói tin từ nguồn 192.168.1.120 vào xử lý trong class 1:2

```
$ 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 qdics 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
```

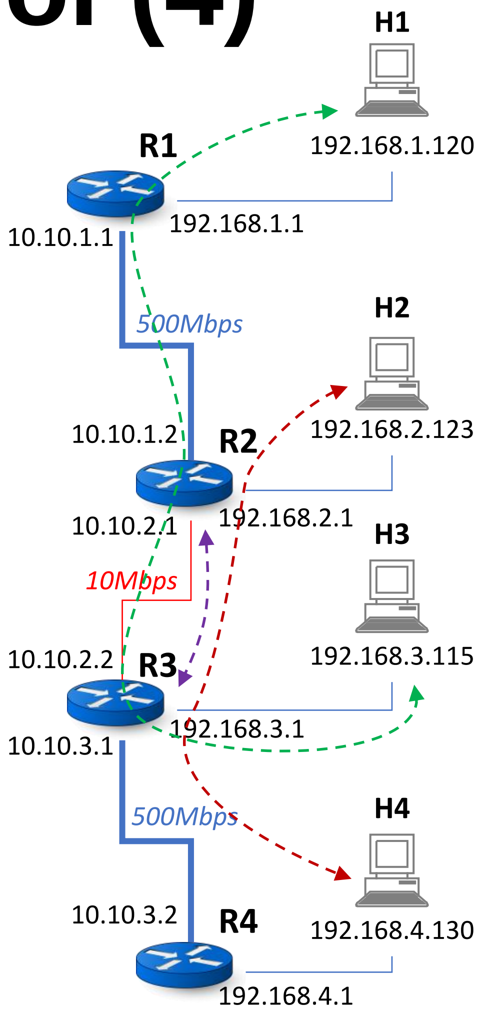
PHB Demo with Linux Traffic Control (4)

- Khởi động cả 3 dòng dữ liệu cạnh tranh *iperf* H2-H4, H1-H3, R2-R3.
- Luồng H1-H3 hoạt động theo mức băng thông giới hạn 1Mbps
- Luồng R2-R3 hoạt động theo mức băng thông giới hạn 500Kbps
- Luồng H2-H4 không bị ảnh hưởng, hoạt động ở mức trên 9Mbps

```
H3:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
------------------------------------------------------------
[  4] local 192.168.3.115 port 5001 connected with
192.168.1.120 port 54754
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-301.4 sec  16.6 MBytes   462 Kbits/sec
. . .
```

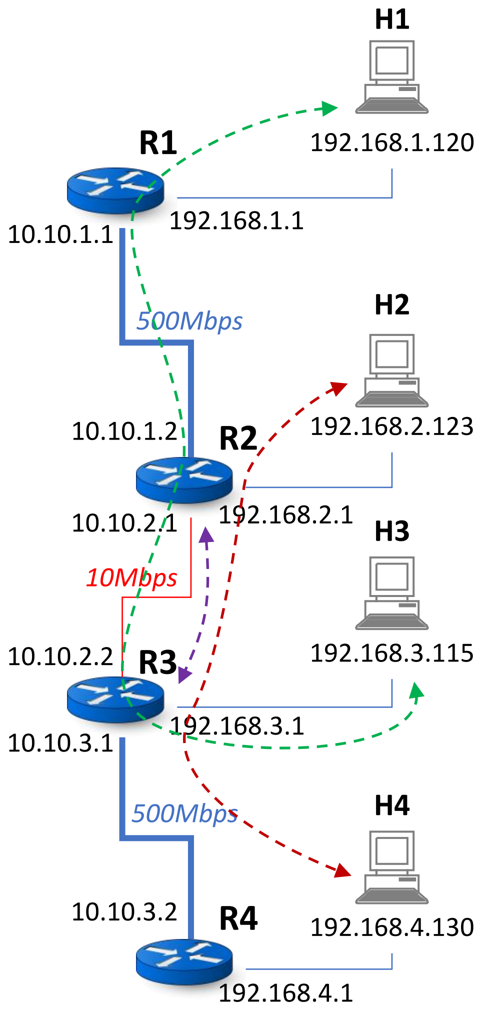
```
R3:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
------------------------------------------------------------
[  4] local 10.10.2.2 port 5001 connected with
10.10.2.1 port 53494
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-301.5 sec  33.1 MBytes   921 Kbits/sec
. . .
```

```
H2:~$ iperf -c 192.168.4.130 -i 1 -t 30
------------------------------------------------------------
Client connecting to 192.168.4.130, TCP port 5001
------------------------------------------------------------
[  3] local 192.168.2.123 port 51806 connected with
192.168.4.130 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 1.0 sec  1.07 MBytes  9.00 Mbits/sec
[  4]  1.0- 2.0 sec  1.12 MBytes  9.36 Mbits/sec
[  4]  2.0- 3.0 sec  1.12 MBytes  9.42 Mbits/sec
. . .
```



Thực hành DiffServ

- Thiết lập kết nối mạng 3 router và 3 host
- Thiết lập hạn chế tốc độ đường truyền R2 – R3 là 10Mbps, các đường khác mặc định là 500 Mbps
- Tạo các luồng dữ liệu cạnh tranh H1 – H3, H2 – H4 và R2 – R3 thấy tốc độ cũng như độ mất mát gói tin bị ảnh hưởng lẫn nhau
- Sử dụng *qdisc prio* trên R2 để thiếp lập PHB, với 3 class ưu tiên khác nhau & gán cho 3 luồng cạnh tranh theo địa chỉ IP
- Áp dụng kịch bản luồng cạnh tranh thấy luồng được ưu tiên không bị ảnh hưởng bởi các luồng cạnh tranh
- Thay đổi filter xử lý PHB trong R2 từ địa chỉ IP thành DSCP codepoint và chạy lại các kịch bản luồng cạnh tranh
- Thiết lập DiffServ network gồm R1, R2, R3 trong đó R1 là ingress, R2 là core và R3 là egress. R1 sử dụng *iptables* để gán DSCP codepoint vào trường ToS của gói tin IP theo yêu cầu



Multiprotocol Label Switching Label Switching (MPLS)

Overview

Architecture

LDP

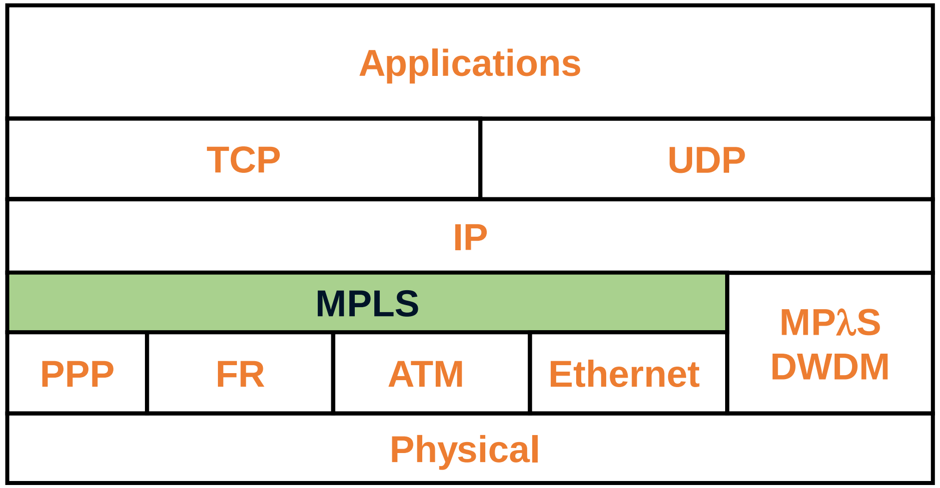
MPLS Motivation

- 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
    - ➔ Sppppppeeeeed
- 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*
- *No index matching: áp dụng lần lượt từng netmask lên địa chỉ IP đích cho mỗi gói tin IP đi đến*
- *Vét cạn: do yêu cầu classless routing ➔ phải thử hết bảng routing để tìm netmask dài nhất*

MPLS is a layer 2.5 protocol

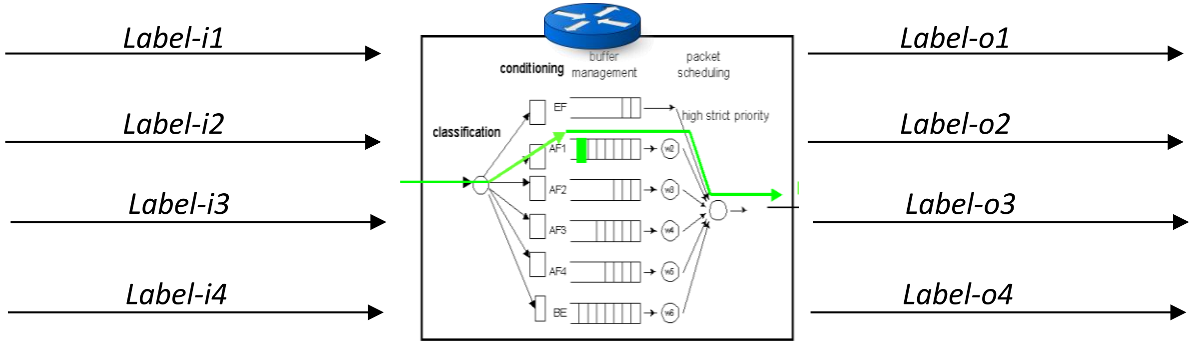
- Experience & proposal
  - IP over ATM
  - IP Switching by Ipsilon
  - Cell Switching Router (CSR) by Toshiba
  - Tag switching by Cisco
  - Aggregate Route-based IP Switching (IBM)
- IETF: MPLS layer
  - When a layer is added, no modification is needed on the existing layers
  - However, traditional 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

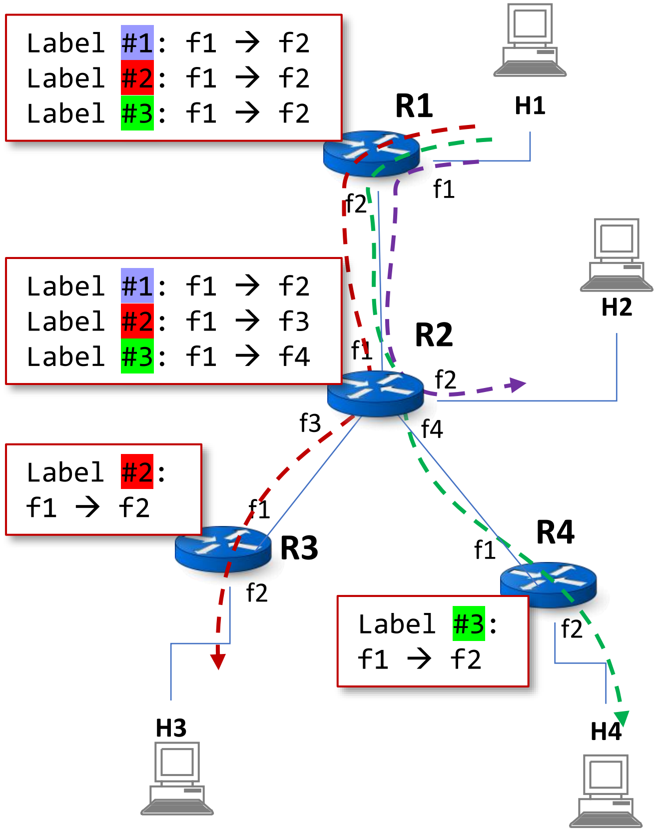



```
:~$ sudo tcpdump -i enp0s9 -env
tcpdump: listening on enp0s9, link-type EN10MB (Ethernet), capture
size 262144 bytes
11:02:25.161445 08:00:27:1c:c2:15 > 08:00:27:61:de:17,
ethertype MPLS unicast (0x8847), length 102: MPLS (label
123, exp 0, [S], ttl 255)
      (tos 0x0, ttl 64, id 37895, offset 0,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

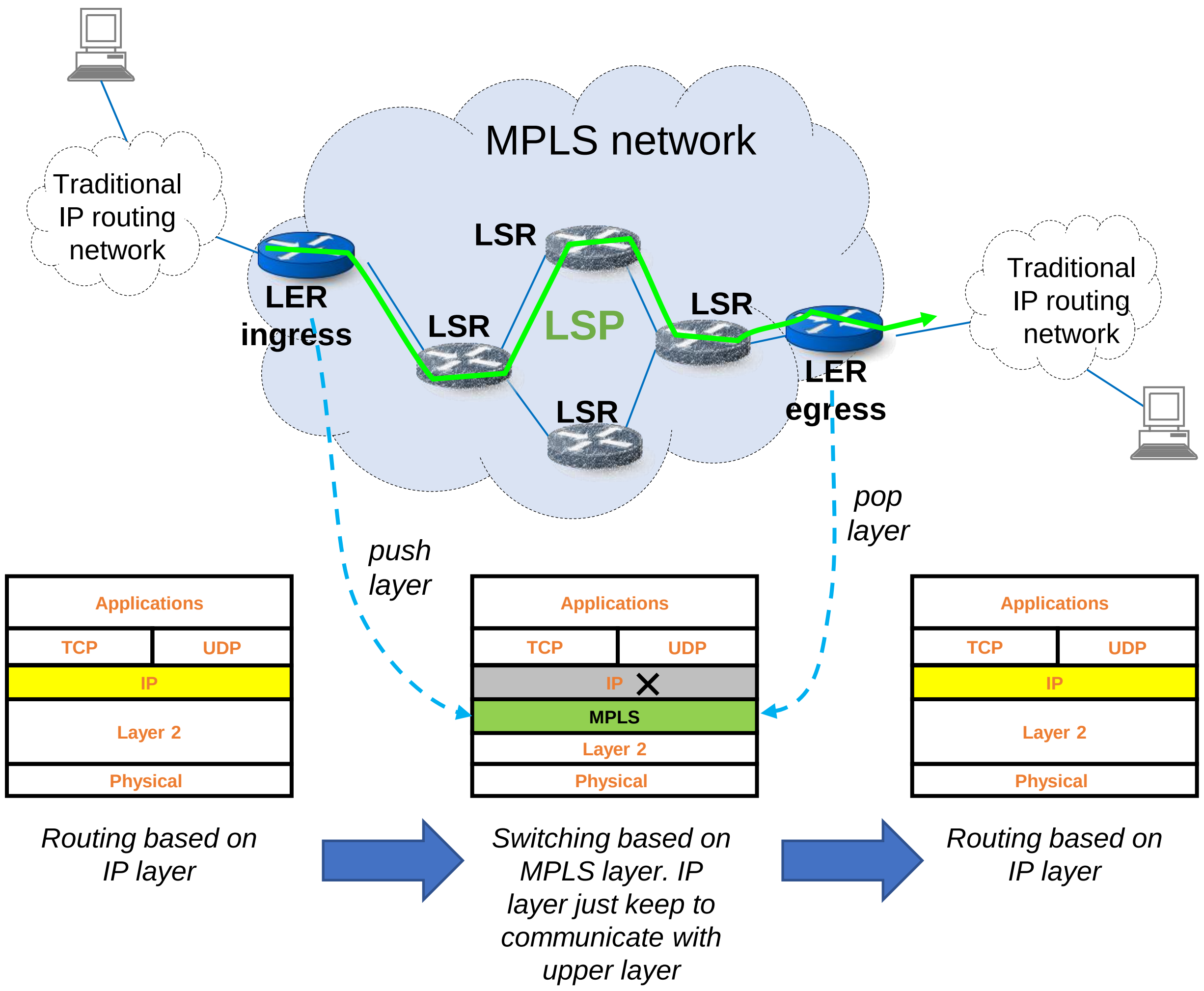




- Label Switch Path (LSP):
  - Fix the path in MPLS network
  - Pre-defined by network administrator in all participated routers

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 outcoming, egress LER pop the MPLS layer from packet and forward to traditional IP network



MPLS Insight

- MPLS network gồm 4 router.
- Cần thiếp 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:

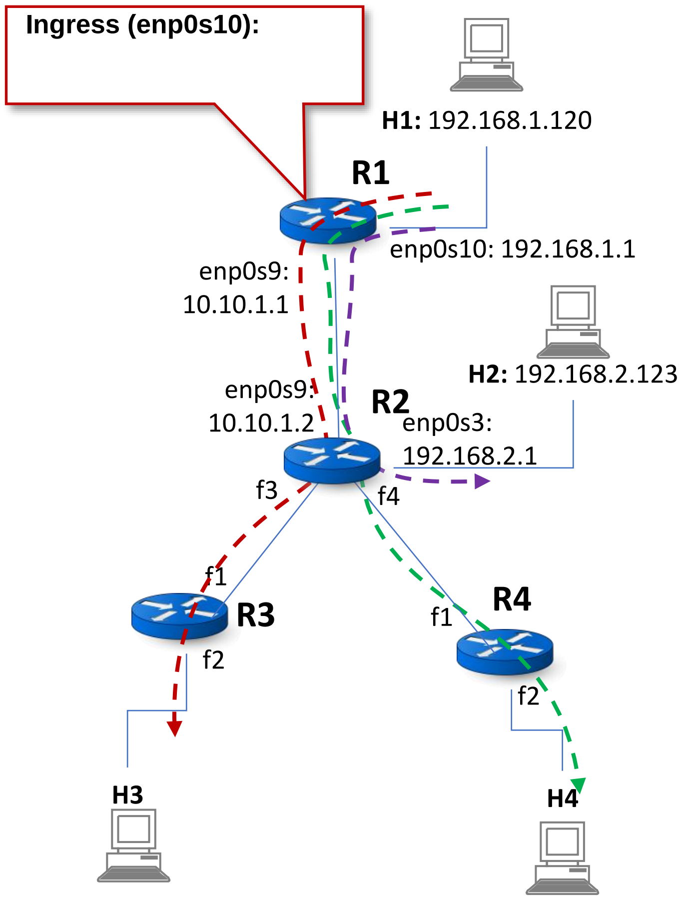
```
R1:~$ sudo tc qdisc show
qdisc fq_codel 0: dev enp0s9 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms
interval 100.0ms memory_limit 32Mb ecn
qdisc fq_codel 0: dev enp0s10 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms
interval 100.0ms memory_limit 32Mb ecn

R1:~$ tc filter show dev enp0s10
```

- Thiếp 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 Insight (2)

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

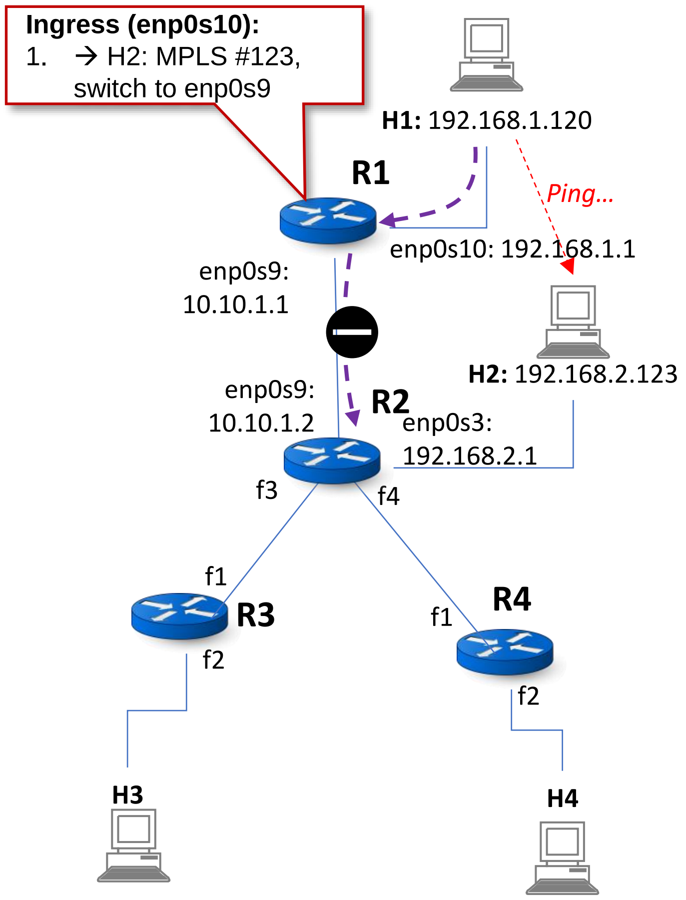
```
R1:~$ sudo tc filter add dev enp0s10 protocol ip parent ffff: \
      flower dst_ip 192.168.2.0/24 \
            action mpls push protocol mpls_uc label 123 \
            action mirred egress redirect dev enp0s9
```

- 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 ICPM 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 Insight (3)

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

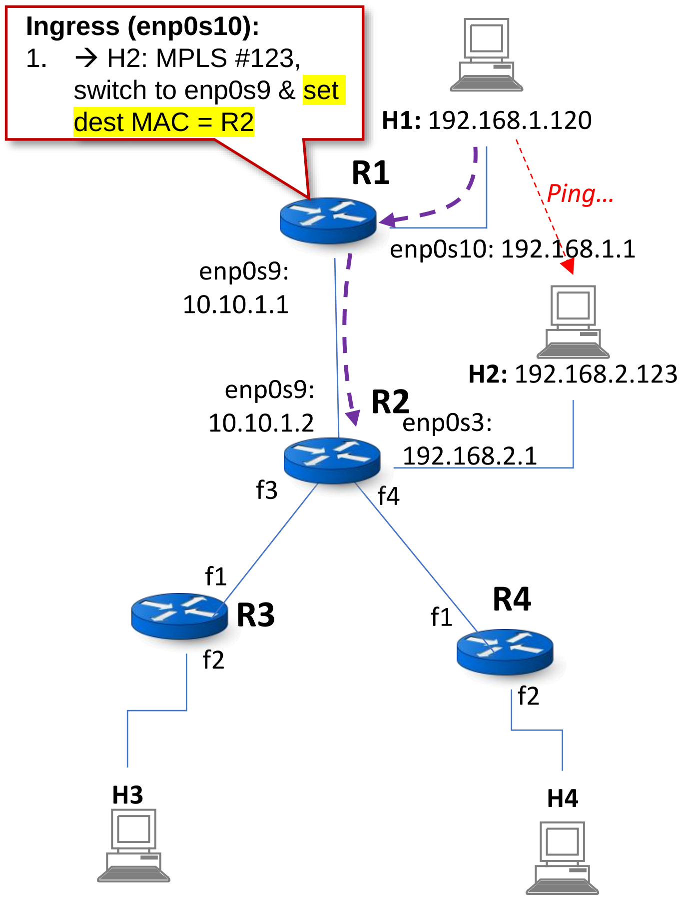
```
R1:~$ sudo tc filter add dev enp0s10 protocol ip parent ffff: \
      flower dst_ip 192.168.2.0/24 \
            action mpls push protocol mpls_uc label 123 \
            action skbmod set dmac 08:00:27:61:de:17 \
            action mirred egress redirect dev enp0s9
```

- 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 Insight (4)

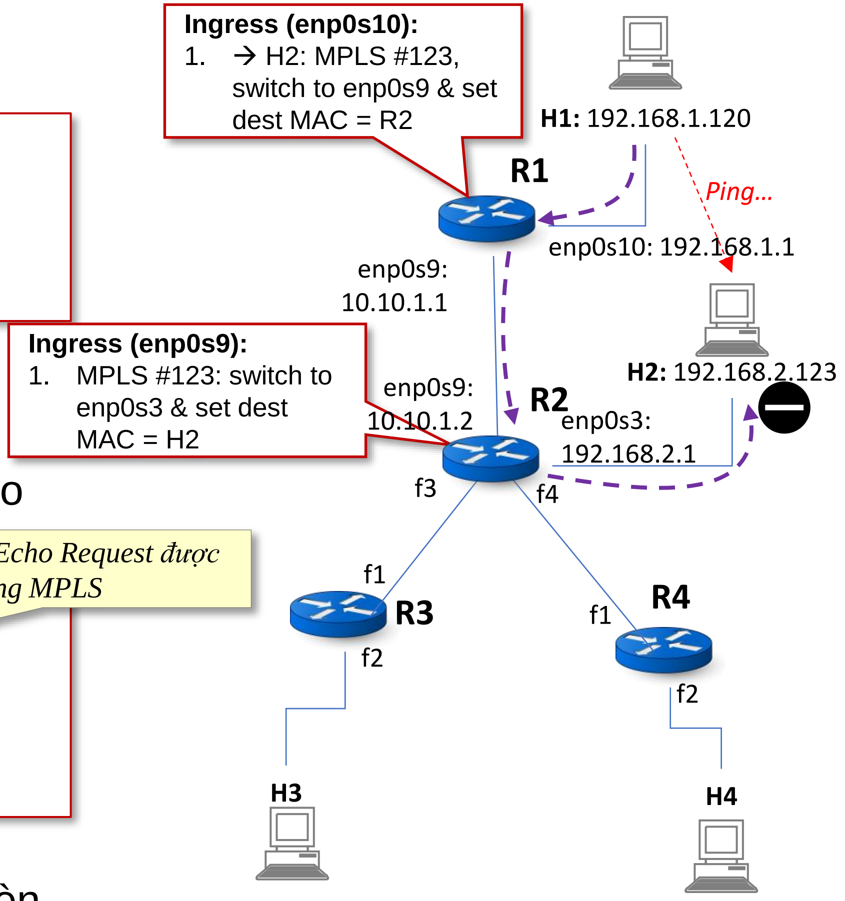
- Thiếp lập xử lý MPLS cho R2: kiểm tra MPLS label #123 thì switch từ *enp0s9* sang *enp0s3* và set dest MAC là H2:

```
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 mirred 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 laywer MPLS vào) nên nó không xử lý được. Cần phải bỏ layer này đi.

MPLS Insight (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

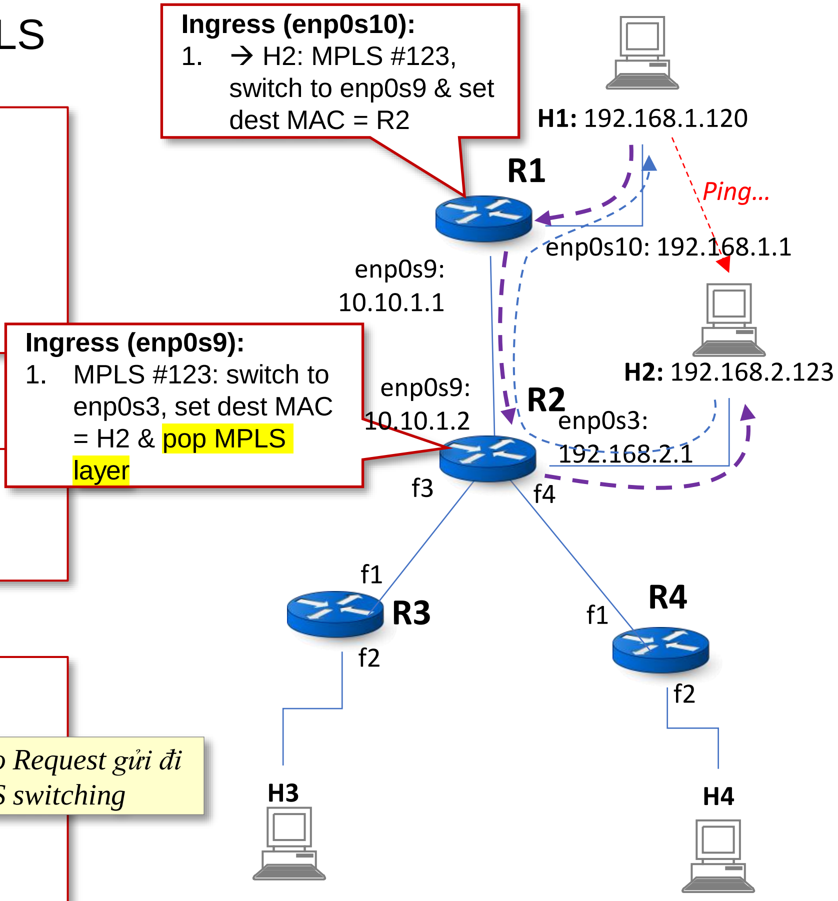
```
H1:~$ ping 192.168.2.123
64 bytes from 192.168.2.123: icmp_seq=1 ttl=62 time=2.26 ms
64 bytes from 192.168.2.123: icmp_seq=2 ttl=62 time=2.03 ms
```

- 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
    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
```

ICMP Echo Request gửi đi bằng MPLS switching

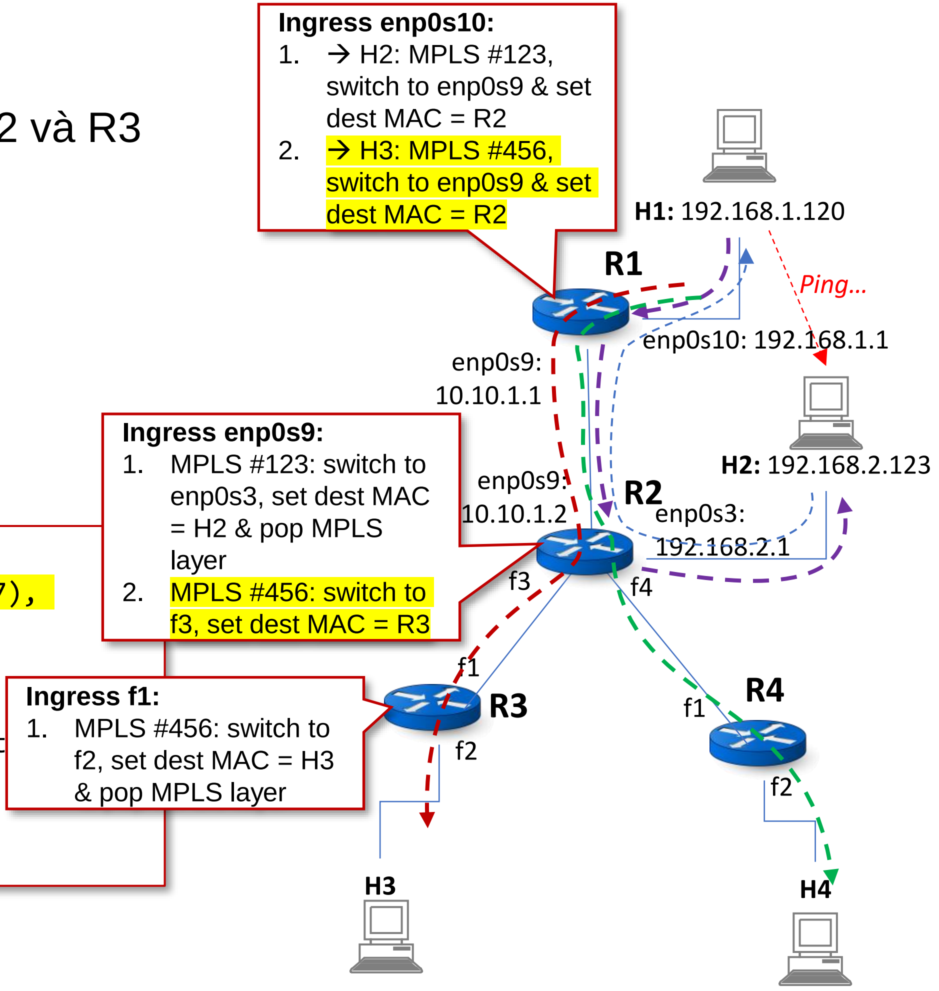
ICMP Echo Reply gửi về bằng IP routing



MPLS Insight (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

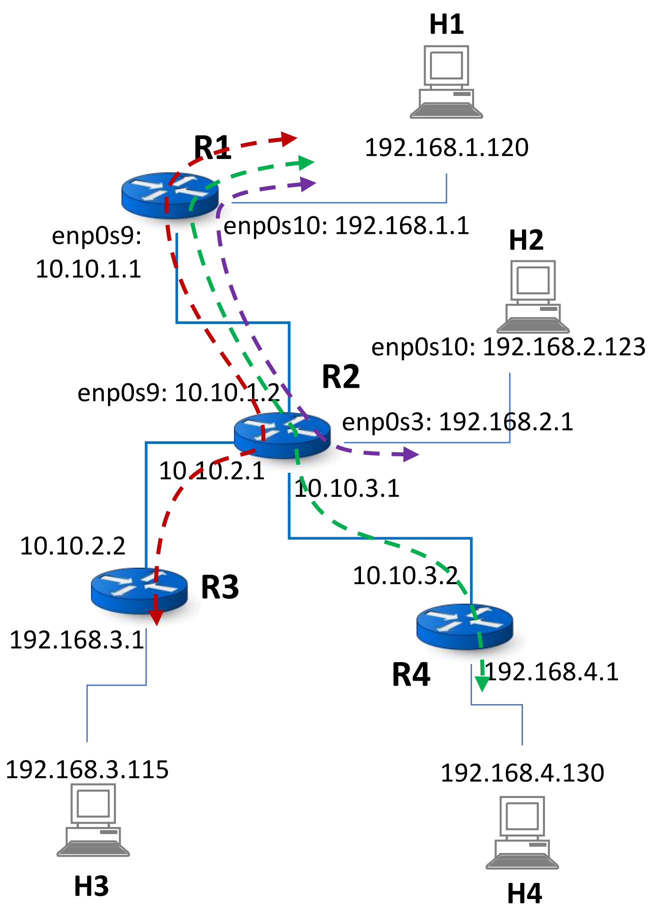
```
R2:~$ sudo tcpdump -i enp0s9 -env
16:26:22.431816 08:00:27:1c:c2:15 > 08:00:27:61:de:17, ethertype MPLS unicast (0x8847),
length 102: MPLS (label 456, exp 0, [S], ttl 255)
        (tos 0x0, ttl 64, id 32117, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.1.120 > 192.168.3.115: ICMP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

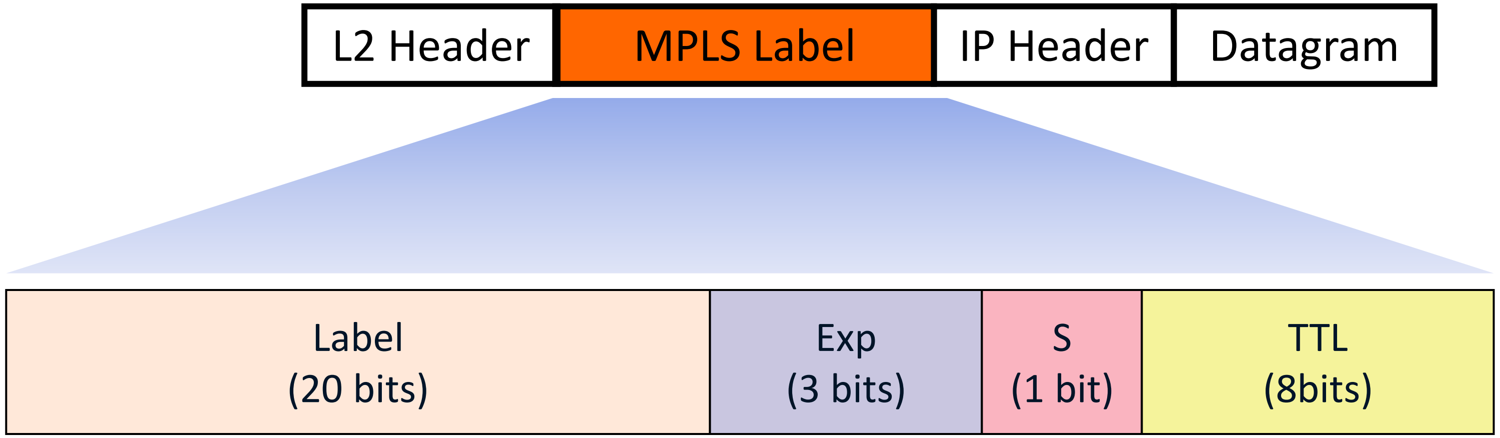
- Sử dụng Linux Trafic Control để xây dựng LSR
- Thiết lập hệ thống kết nối các router và host
- Tạo LSP H1 → R1 → R2 → H2
  - Khởi tạo ingress LSR trên R1
  - Thiết lập luật chuyển tiếp (filter) trên R1 và R2
  - Xử lý pop MPLS layer trên egress LSR R2
  - Kiểm tra kết nối ping H1 → H2
  - Bắt gói tin ICPM 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



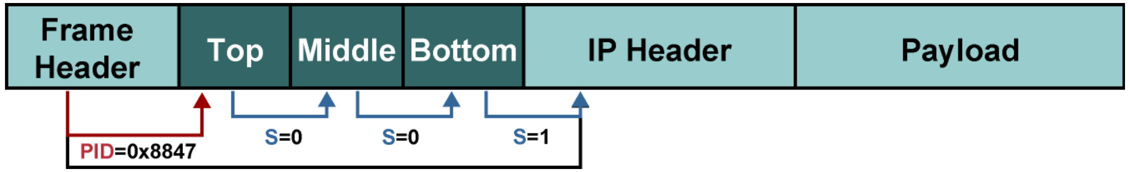
MPLS Architecture

MPLS Encapsolution

- 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

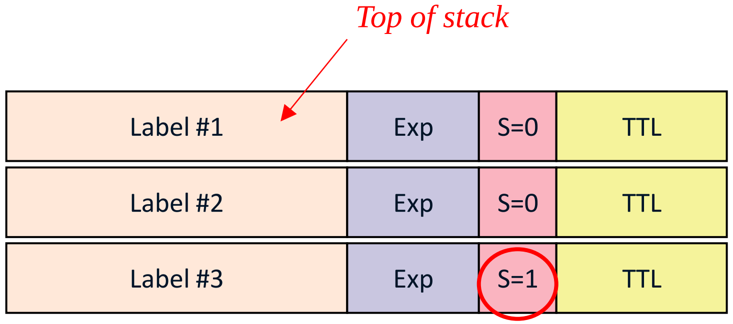



```
R2:~$ sudo tcpdump -i enp0s9 -env
16:26:22.431816 08:00:27:1c:c2:15 > 08:00:27:61:de:17,
        ethertype MPLS unicast (0x8847), length 102: MPLS (label 456, exp 0, [S], ttl 255)
                (tos 0x0, ttl 64, id 32117, offset 0, flags [DF], proto ICMP (1), length 84)
                    192.168.1.120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

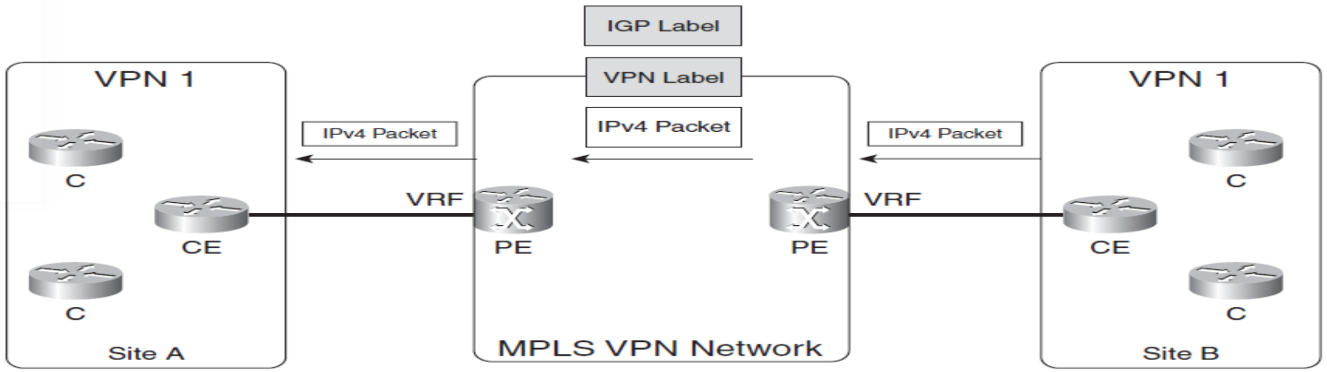


MPLS Multi Label Stack

- 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.




```
~$ 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
```

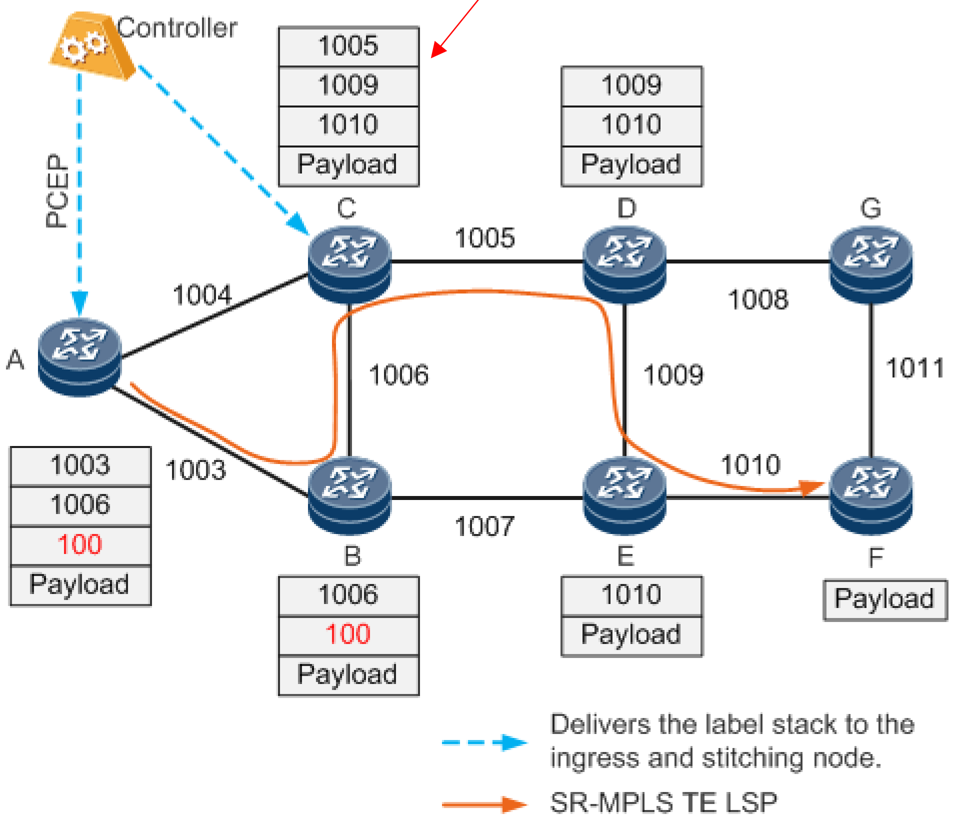


source: MPLS Fundamentals (Cisco press 2007)

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.

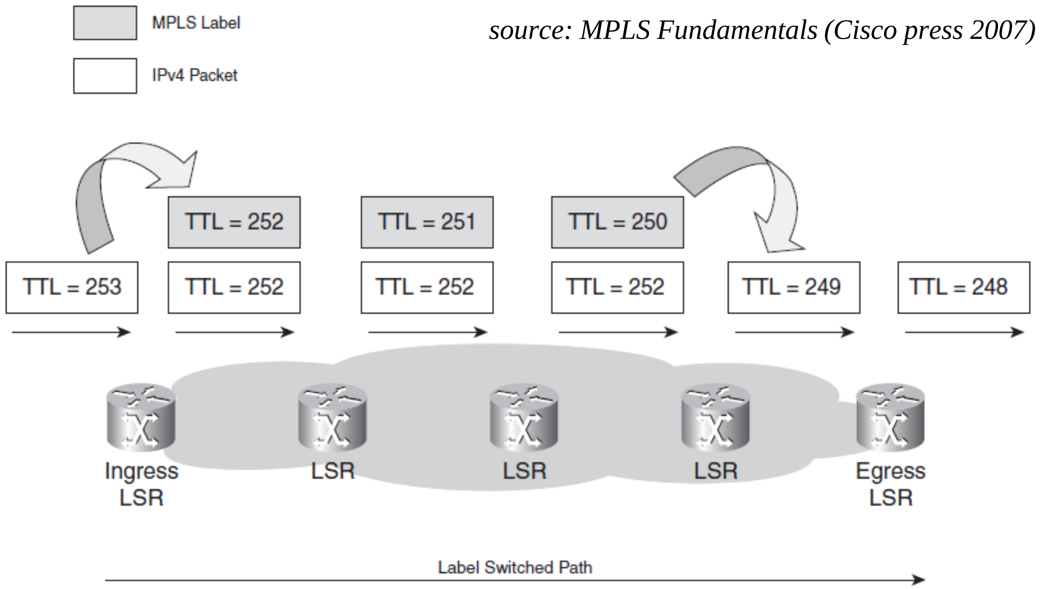
Replace “segment label” 100 by a stack of labels: 1005/1009/1010

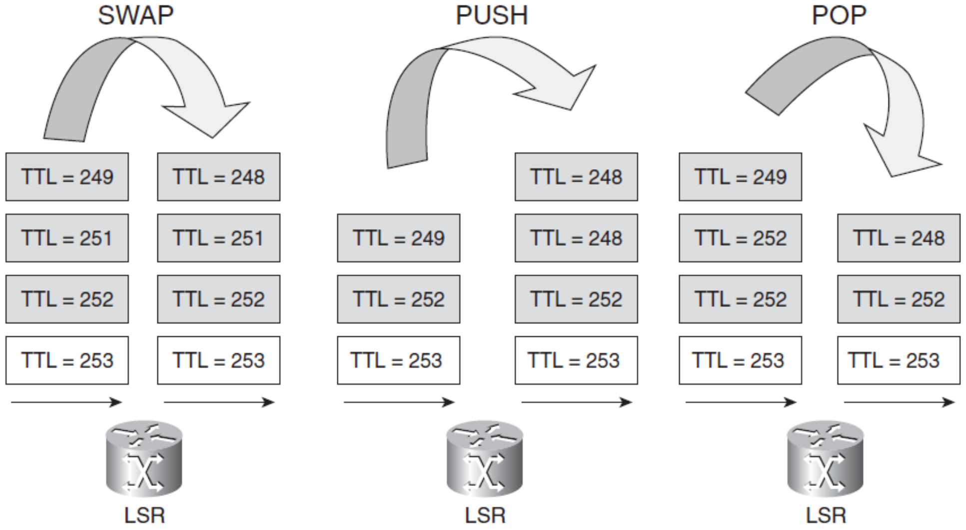


TTL Behavior

- 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 metwork 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

source: MPLS Fundamentals (Cisco press 2007)





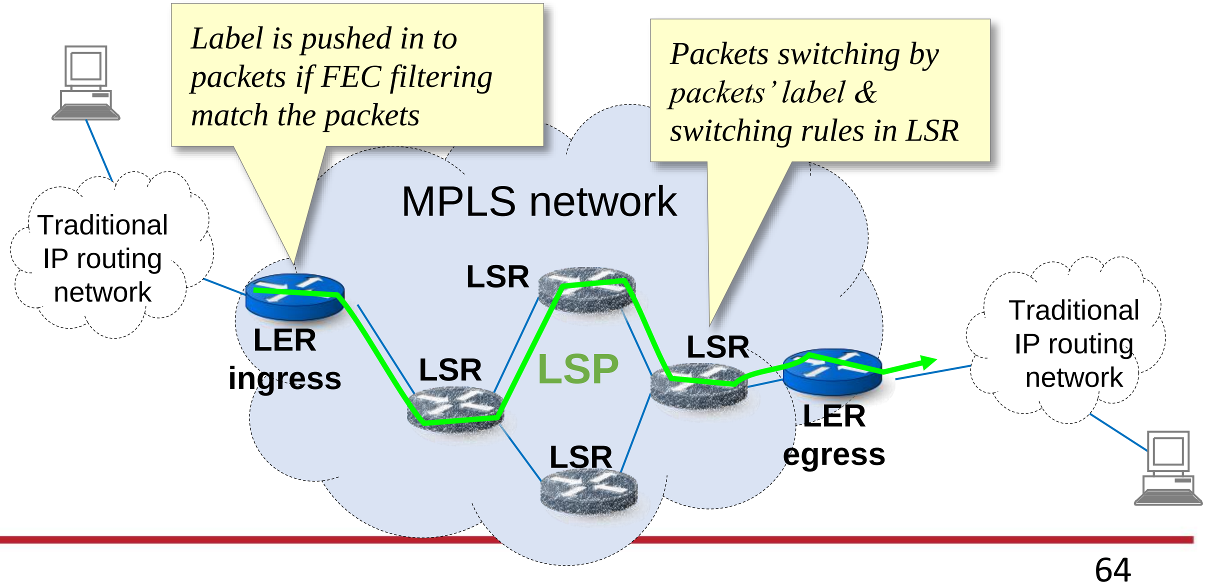
Forward Equivalent Class (FEC)

- 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 r  ls push protocol mpls_uc label 123 \
            action      d egress redirect dev enp0s9
```

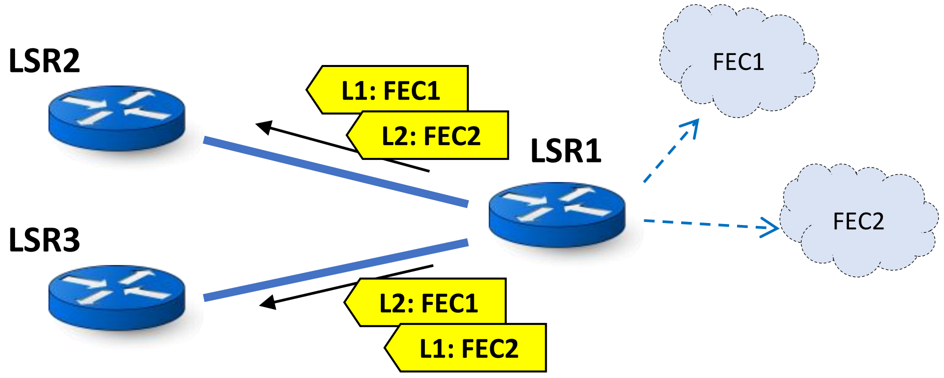
Có thể bổ sung filter lọc gói tin theo nhiều trường thông tin kết hợp: địa chỉ IP nguồn/đích, cổng TCP nguồn/đích, ToS, v.v…



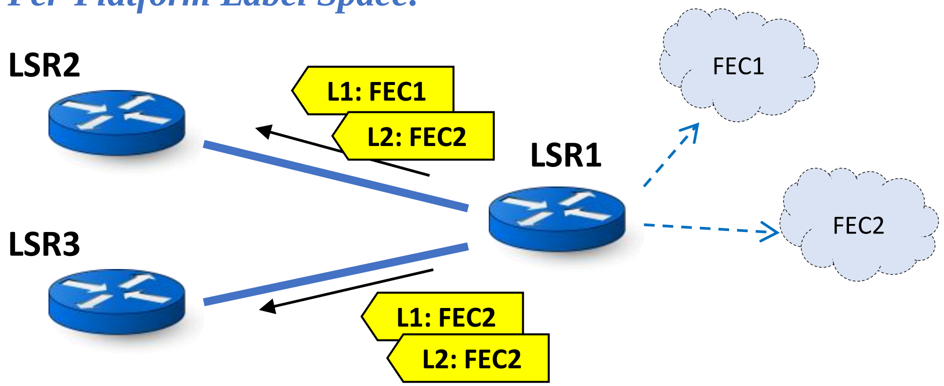
MPLS Label Spaces

- 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:

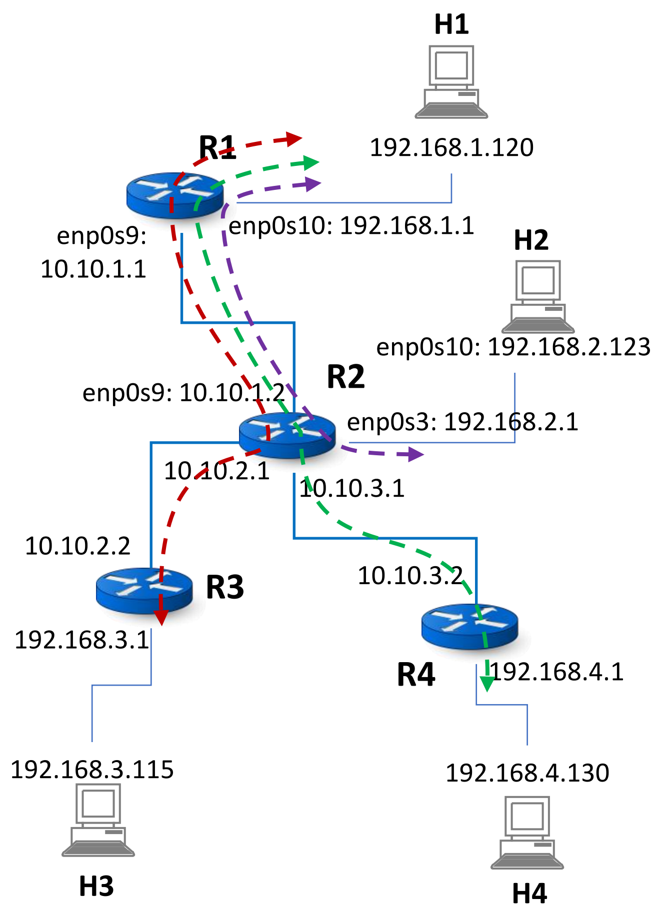


Per-Platform Label Space:



Thực hành MPLS Static Labeling

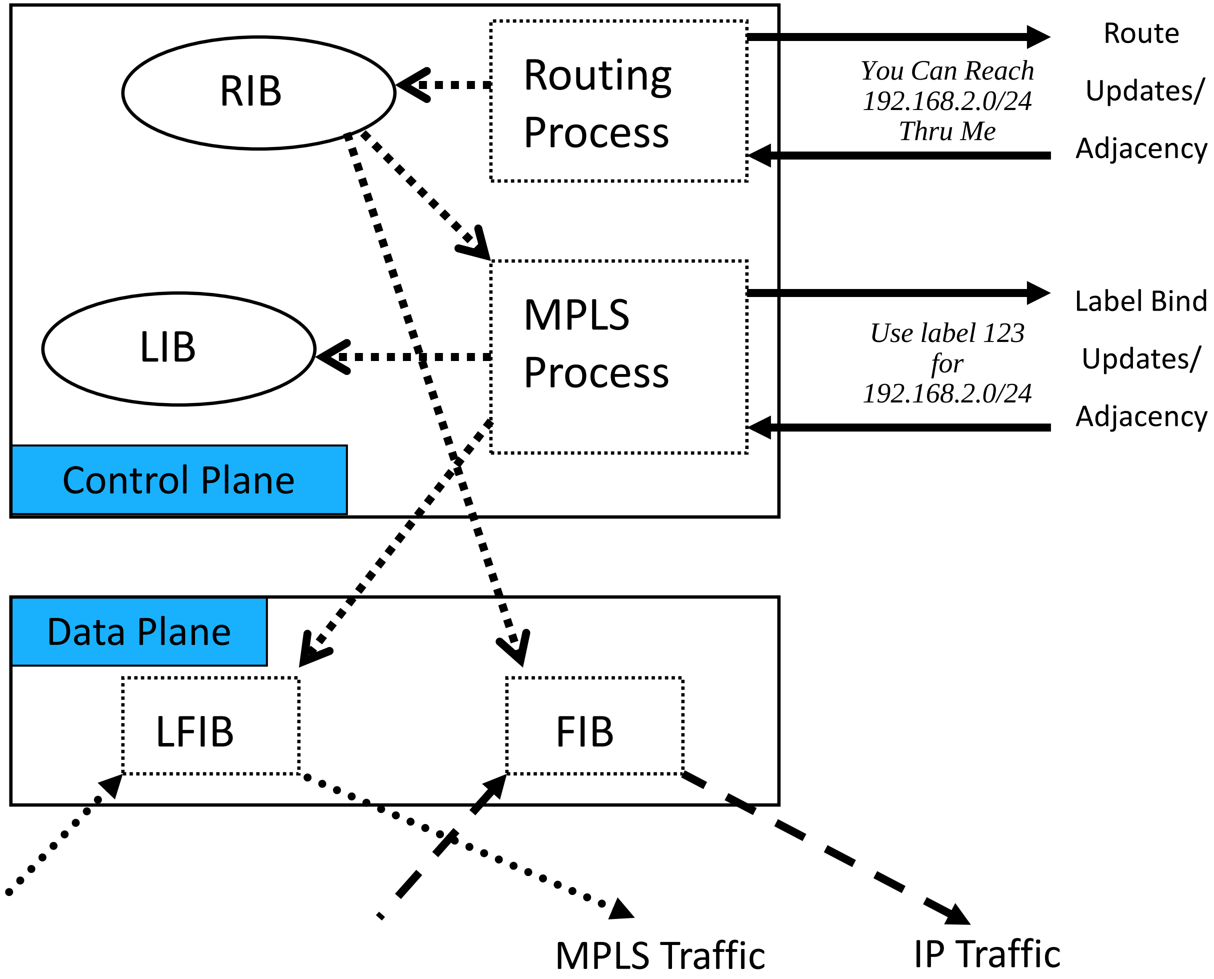
- Sử dụng kernel modul MPLS để xây dựng LSR
- Thiết lập hệ thống kết nối các router và host
- Tạo LSP H1 → R1 → R2 → H2
  - Khởi tạo ingress LSR trên R1
  - Thiết lập luật chuyển tiếp (filter) trên R1 và R2
  - Xử lý pop MPLS layer trên egress LSR R2
  - Kiểm tra kết nối ping H1 → H2
  - Bắt gói tin ICPM 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



Label Distribution Protocol (LDP)

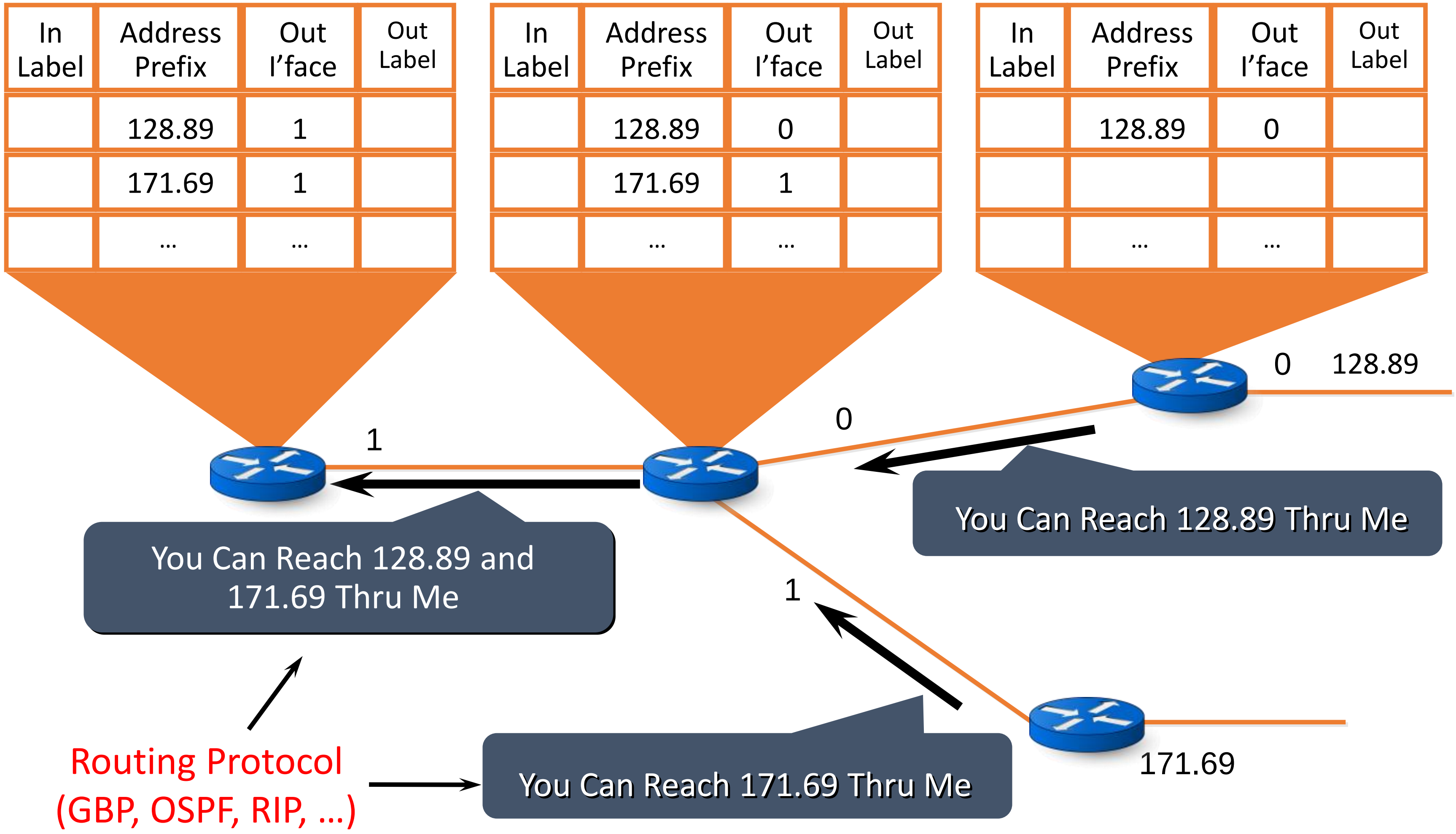
LSR Insight: Control & (Data) Forward plane

- 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



Revised IP routing

- Dynamic update RIB
  - Hey!!! neighbour!!!
  - You can reach… thought me
- Decide FIB from RIB
  - Many way to select the best route
  - Backup the others
  - Activate a backup if current route down
- Forward IP packets following the "matched" FIB rule



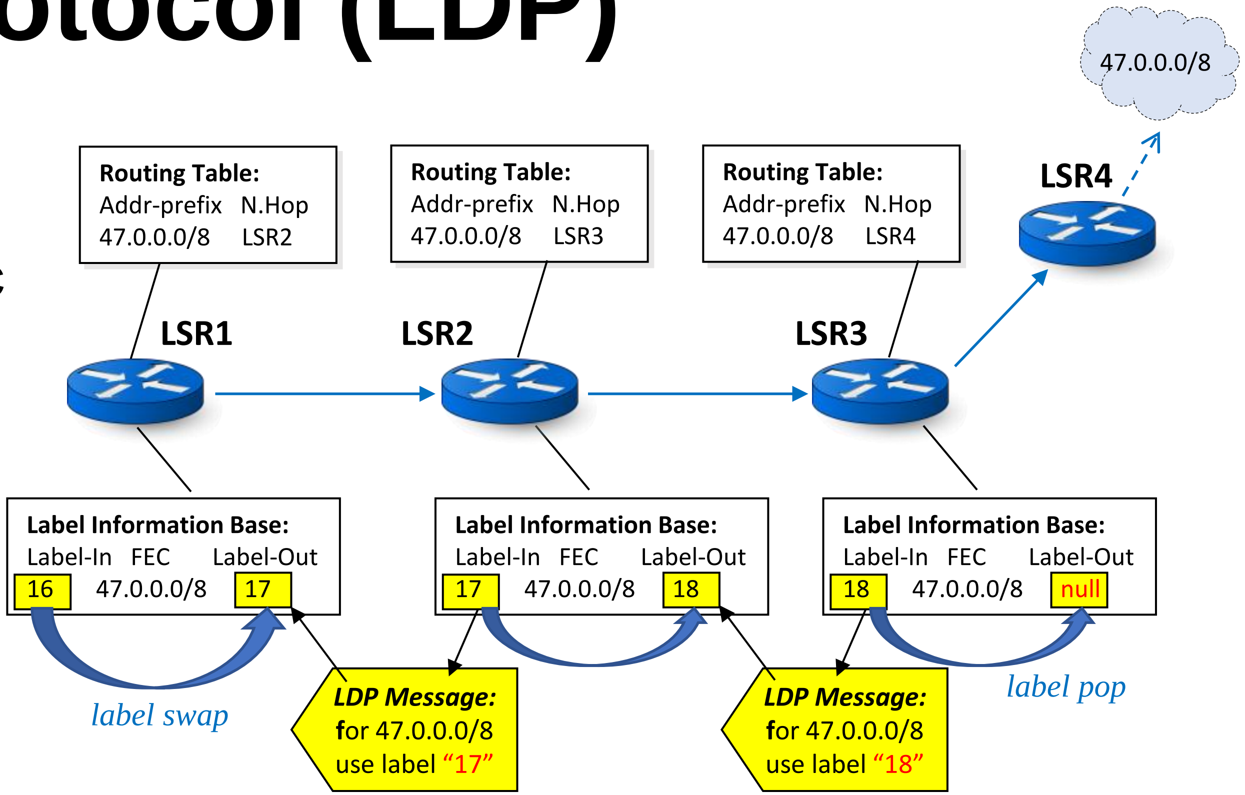
Label Distribution Protocol (LDP)

- 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

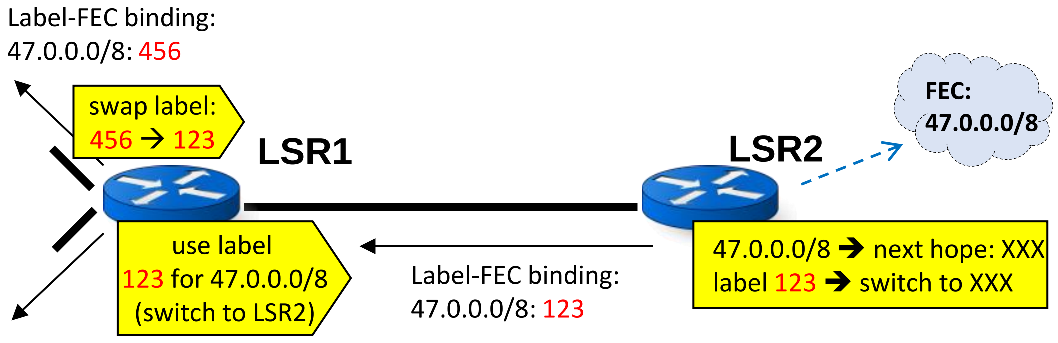
- The principal objective of LDP is to establish “common view” of FEC for all LSR routers.
- Procedures & working modes are standardized for LSR to collaborate in LDP sessions.
- Label advertisement (also called Label distribution)
  - An LSR binds a label to a specific FEC and notifies its upstream 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 woring modes

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

Label Distribution modes

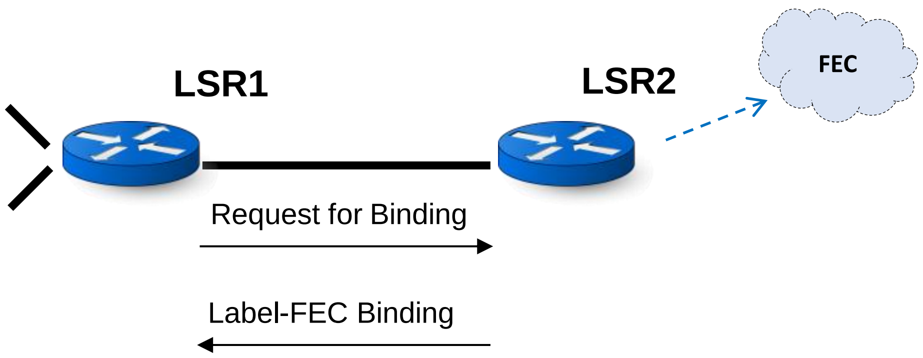
LSR binds a label to each specific FEC and notifies its upstream LSRs of the binding. This process is called label advertisement (or distribution).

Downstream-Unsolicited: *actively by downstream LSR*



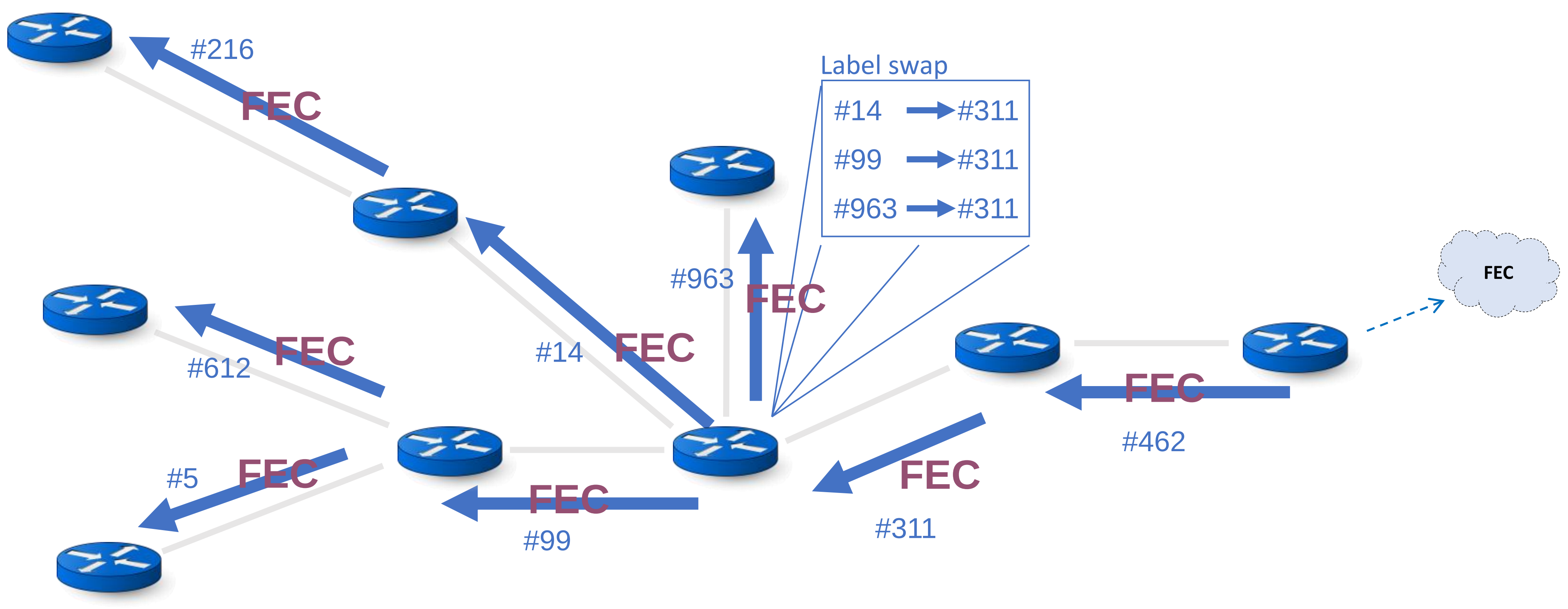
- LSR2 and LSR1 are said to have an "LDP adjacency" (LSR2 being the downstream LSR)
- LSR2 discovers a 'next hop' for a particular FEC
- LSR2 generates a label for the FEC and communicates the binding to LSR1
- LSR1 inserts the binding into its forwarding tables
- If LSR2 appear to be the next hop for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

Downstream-on-Demand: *by request from upstream LSR*

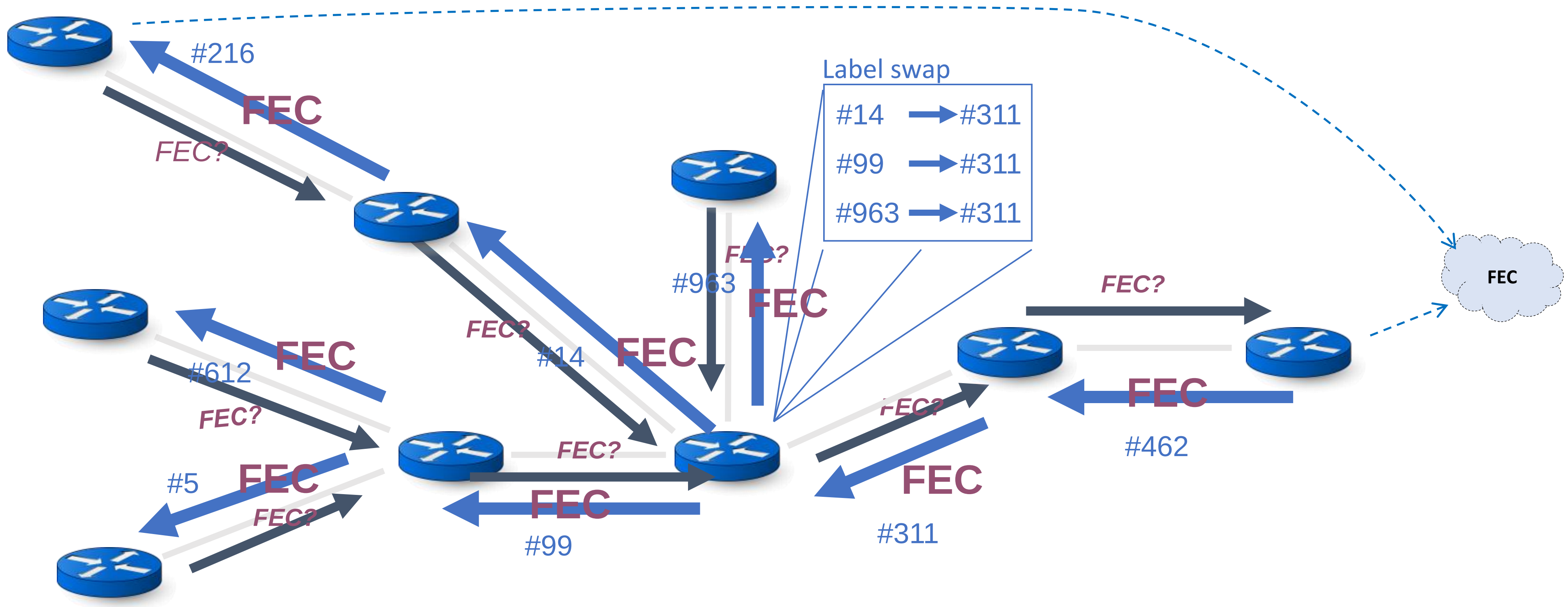


- LSR1 recognizes LSR2 as its next-hop for an FEC
- A request is made to LSR2 for a binding between the FEC and a label
- If LSR2 recognizes the FEC and has a next hop for it, it creates a binding and replies to LSR1
- Both LSRs then have a common understanding

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

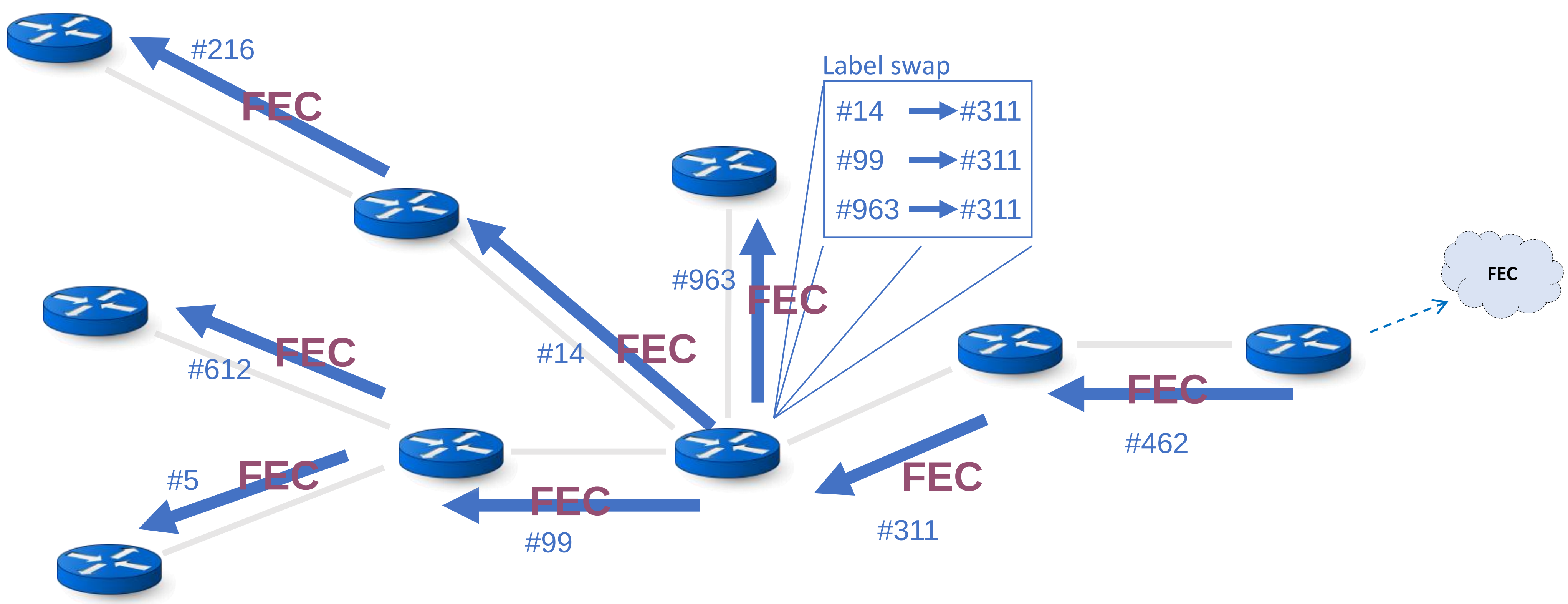
Independent:

- 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

Ordered:

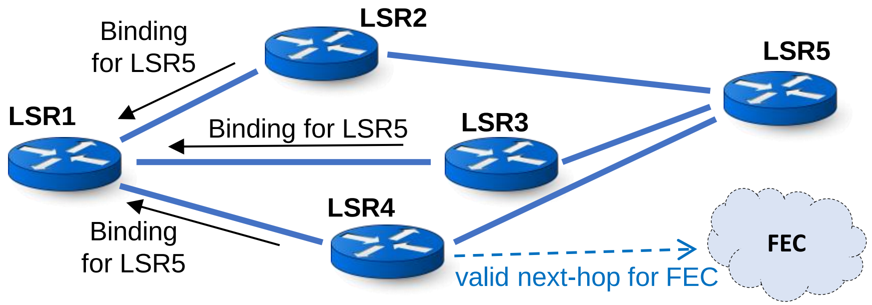
- Requires more delay before packets can be forwarded along the LSP
- Depends on availability of egress node
- Mechanism for consistent granularity and freedom from loops
- Used for explicit routing and multicast

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

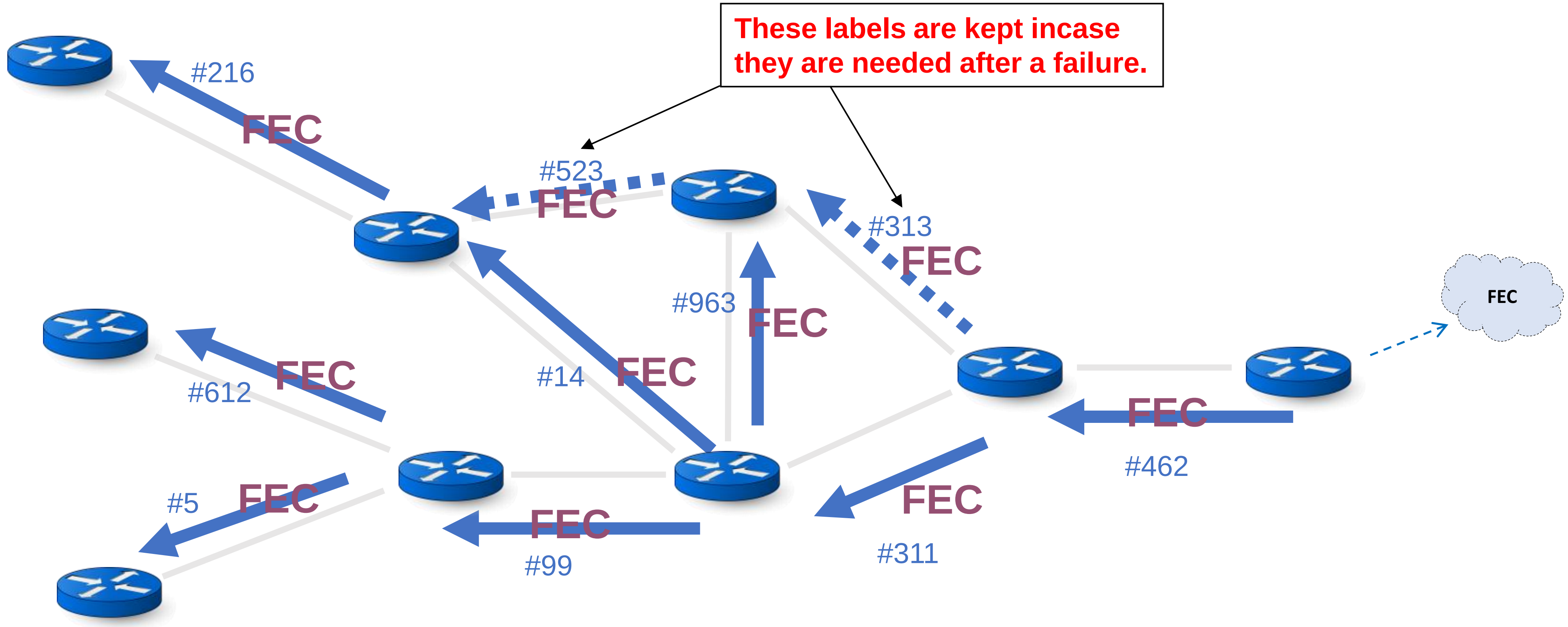
LSR4's Label

~~LSR3's Label~~

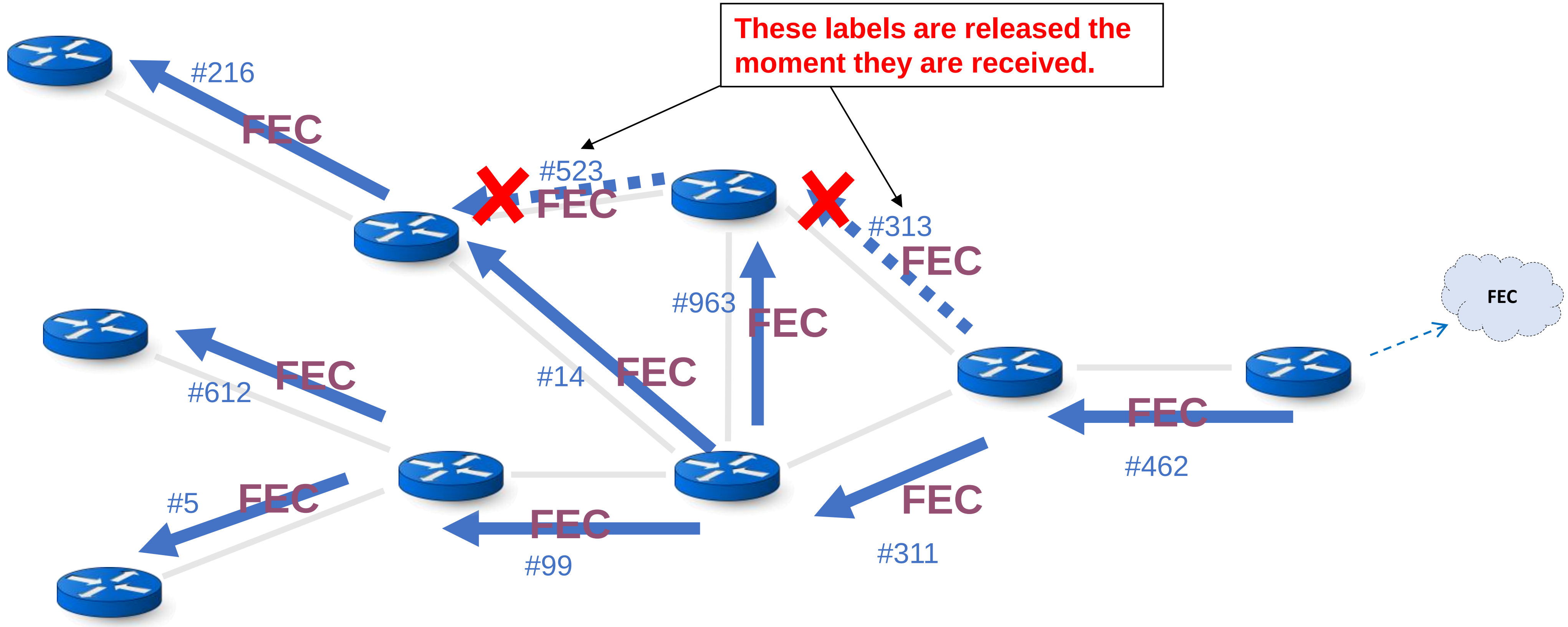
~~LSR2's Label~~

  - 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



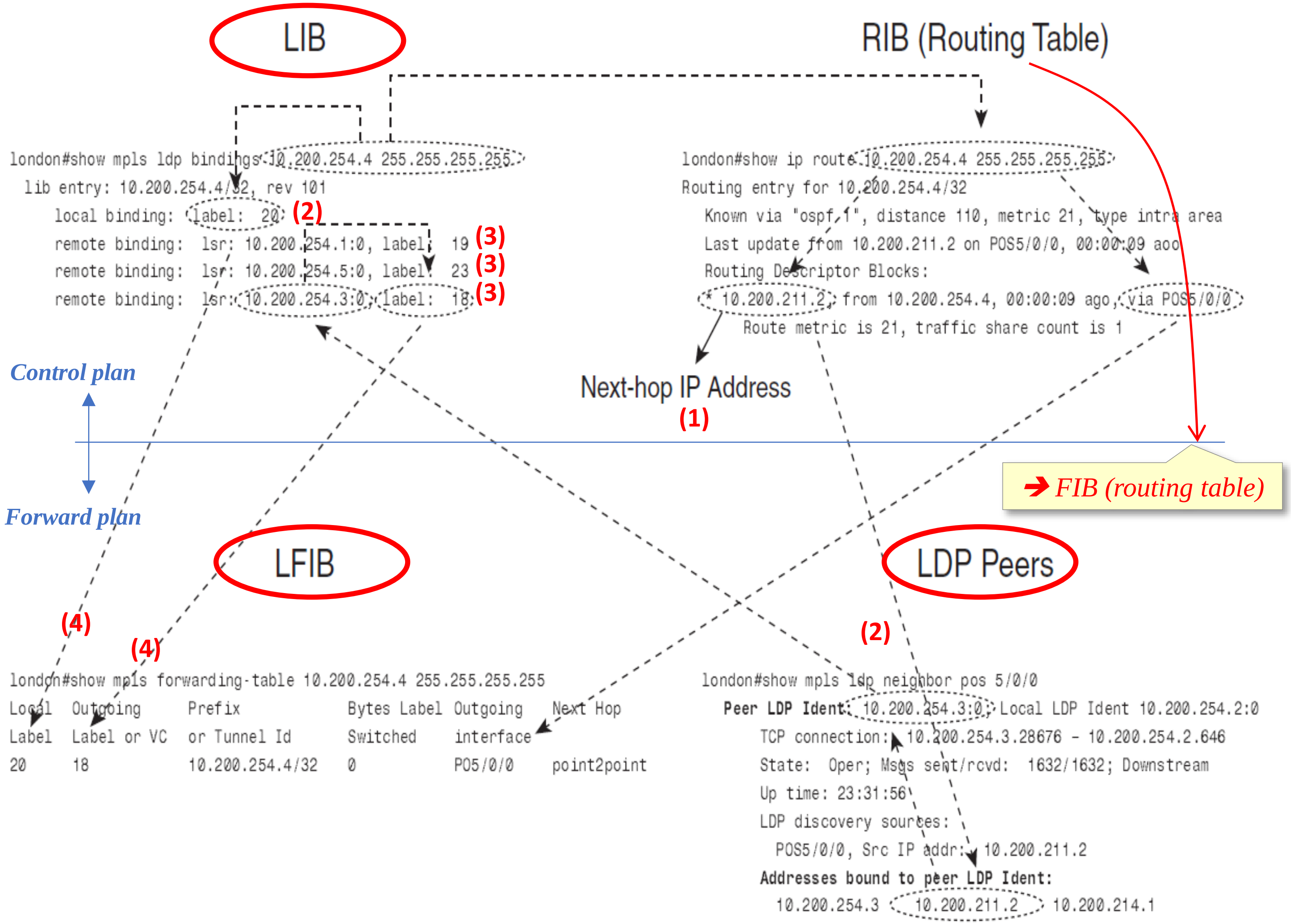
Conservative Retention mode



LDP update flow: RIB ➔ LIB ➔ LFIB

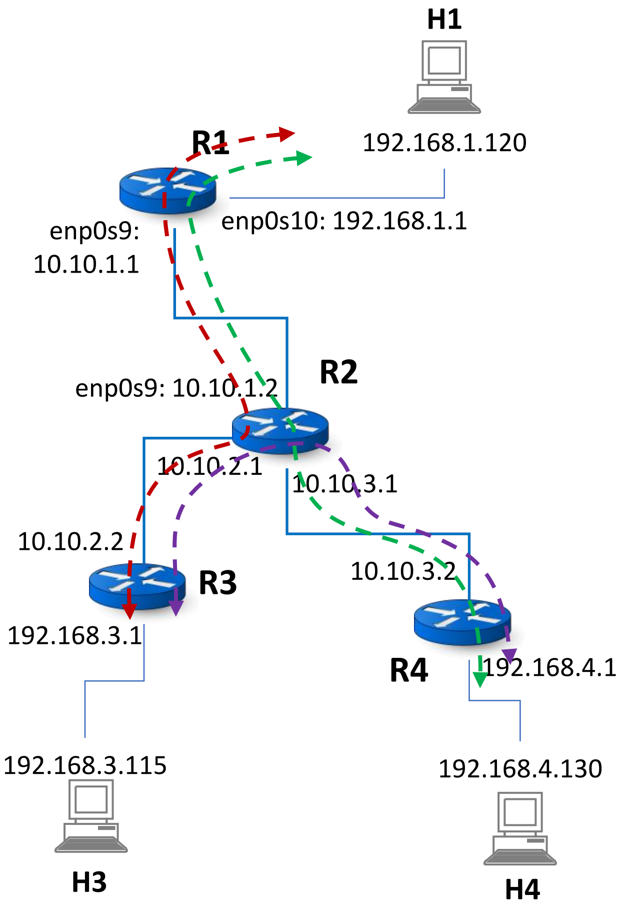
source: MPLS Fundamentals (Cisco press 2007)

- 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 (Libral mode)
  4. Select "best label" to inplement 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



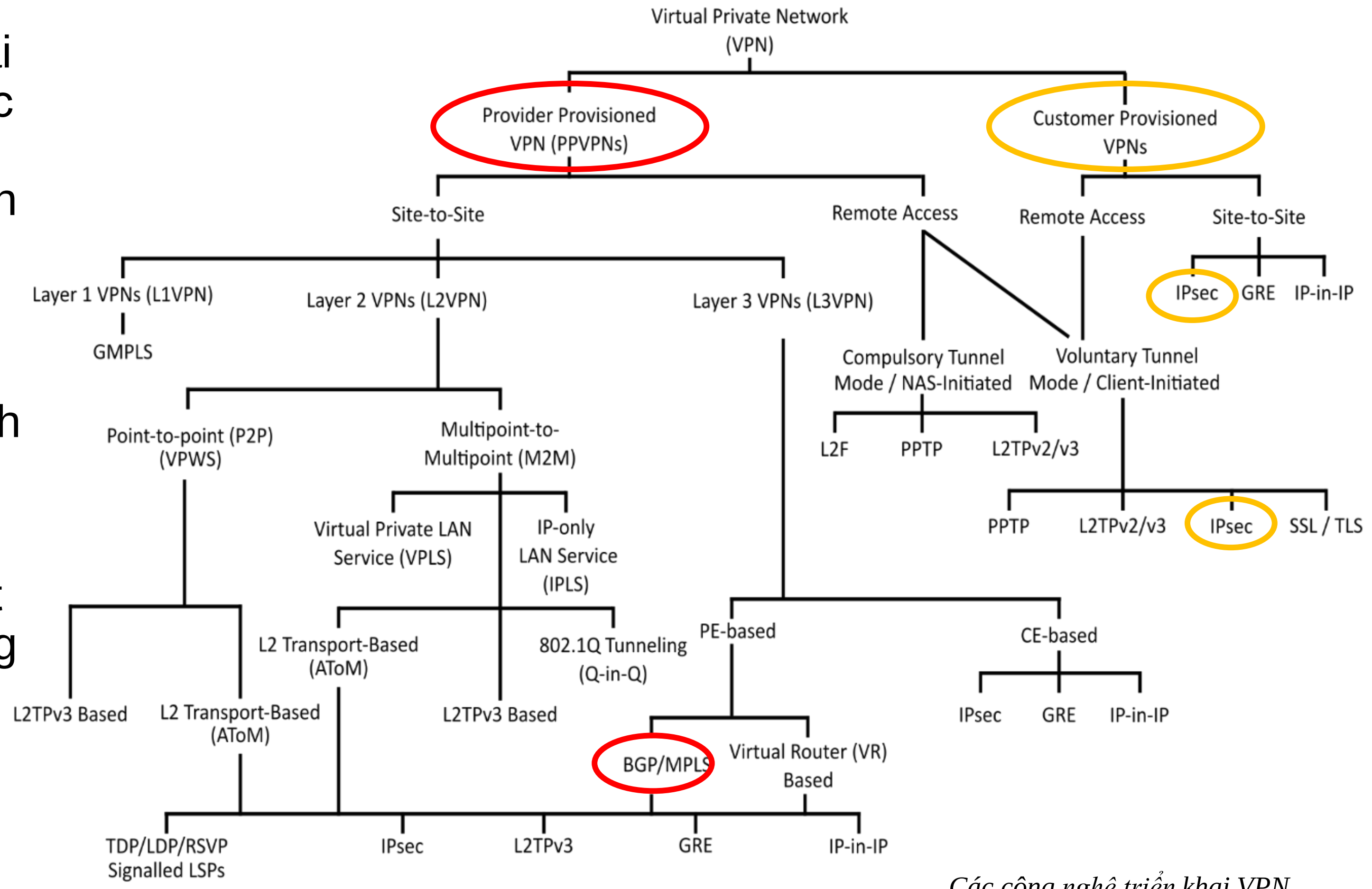
MPLS Applications

VPN

QoS

Các công nghệ triển khai VPN *(nhắc lại)*

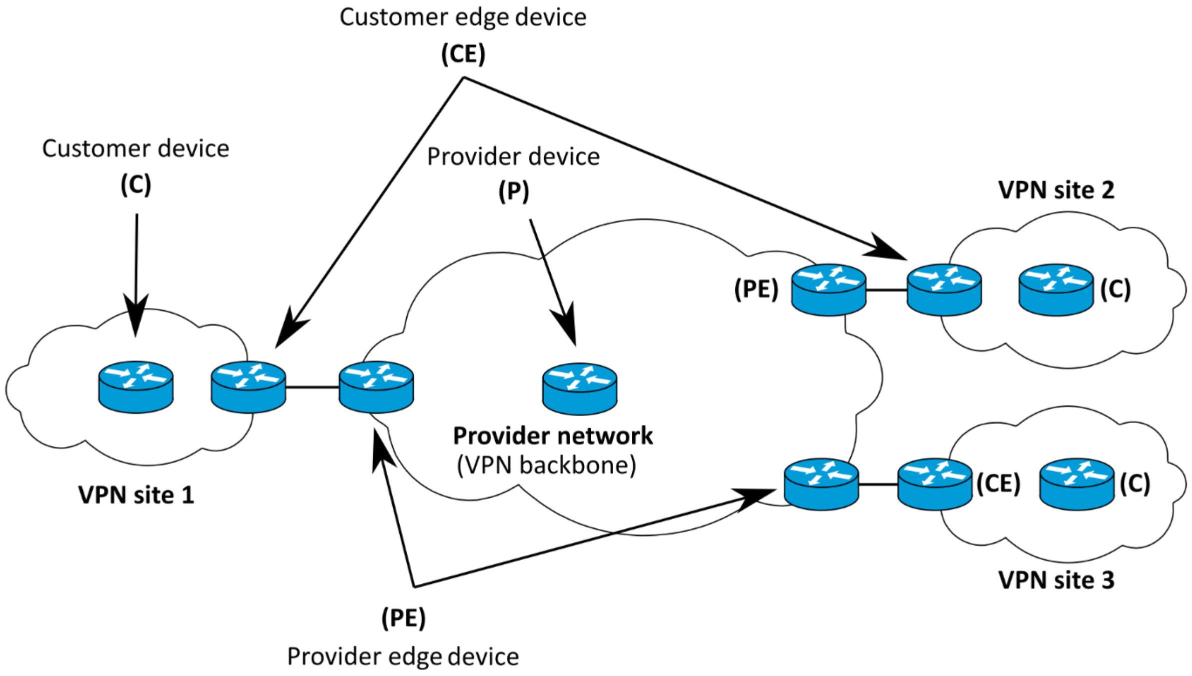
- Công nghệ đường hầm IP để triển khai VPN chỉ là một ví dụ. VPN có thể được triển khai bằng rất nhiều công nghệ khác, và tại các tầng dưới IP hoặc trên IP
- Triển khai VPN phía người dùng (customer provisioned): admin của private network tự thiết lập và vận hành VPN
- Triển khai VPN phía nhà cung cấp (provider provisioned): VPN được thiết lập và vận hành với sự hỗ trợ của công ty cung cấp đường truyền (ví dụ các ISP)



Các công nghệ triển khai VPN
nguồn: Wikipedia

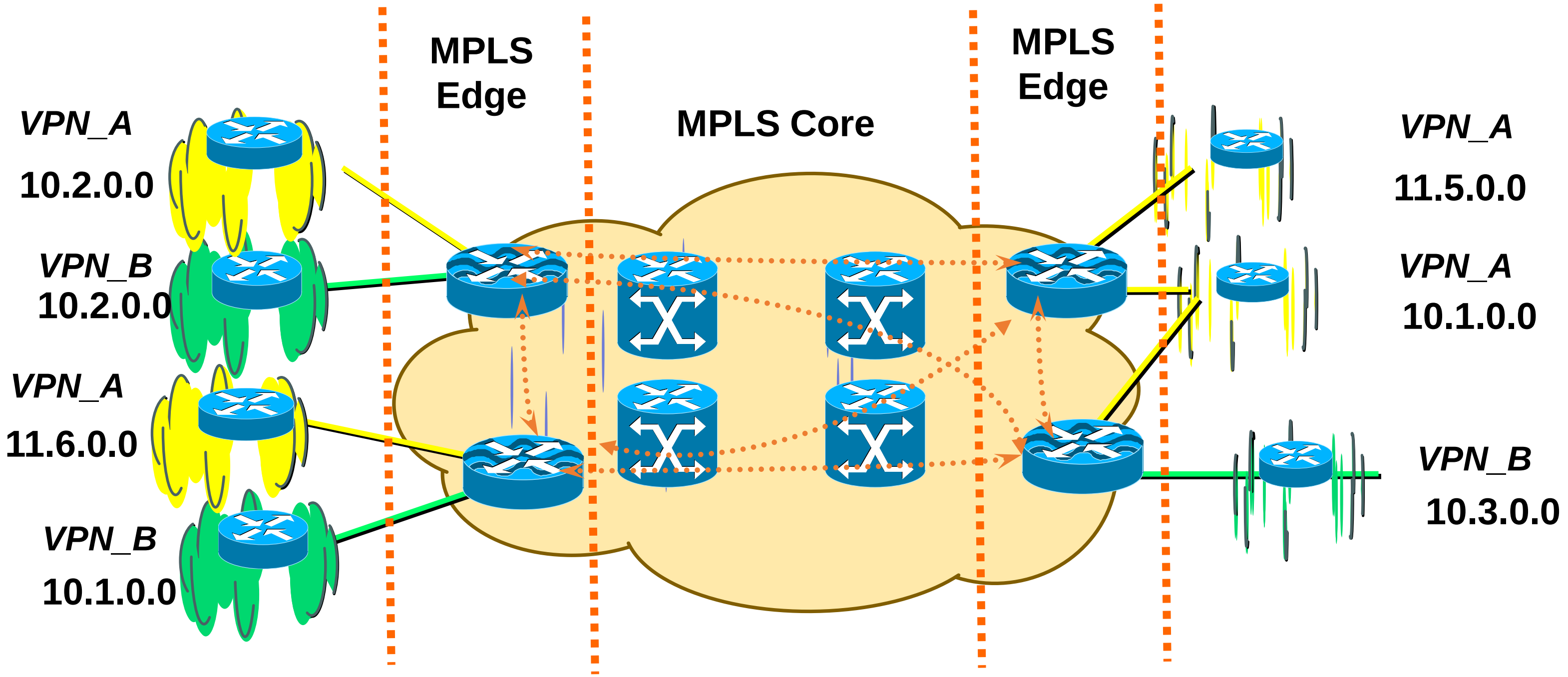
Triển khai VPN phía nhà cung cấp *(nhắc lại)*

- Internet phổ cập toàn cầu và nền kinh tế đòi hỏi công ty phải tổ chức phân tán/cộng tác khiến VPN trở nên cần thiết như một hạ tầng kết nối chuẩn ➔ mô hình triển khai VPN được các nhà cung cấp dịch vụ mạng chuẩn hóa để đáp ứng nhanh nhu cầu từ công ty
- Nhà cung cấp VPN xây dựng hệ thống VPN backbone của mình dựa trên nền tảng Internet và bán dịch vụ VPN cho các công ty
- Private network của các công ty kết nối vào VPN backbone tại điểm truy nhập dịch vụ VPN được triển khai bằng cặp thiết bị Provide edge device (PE) và Customer edge device (CE)
- Tại Việt Nam hiện cũng đã có nhiều nhà cùng cấp dịch vụ VPN độc lập với các ISP *(Google: "top VPN provider in Viet Nam")*



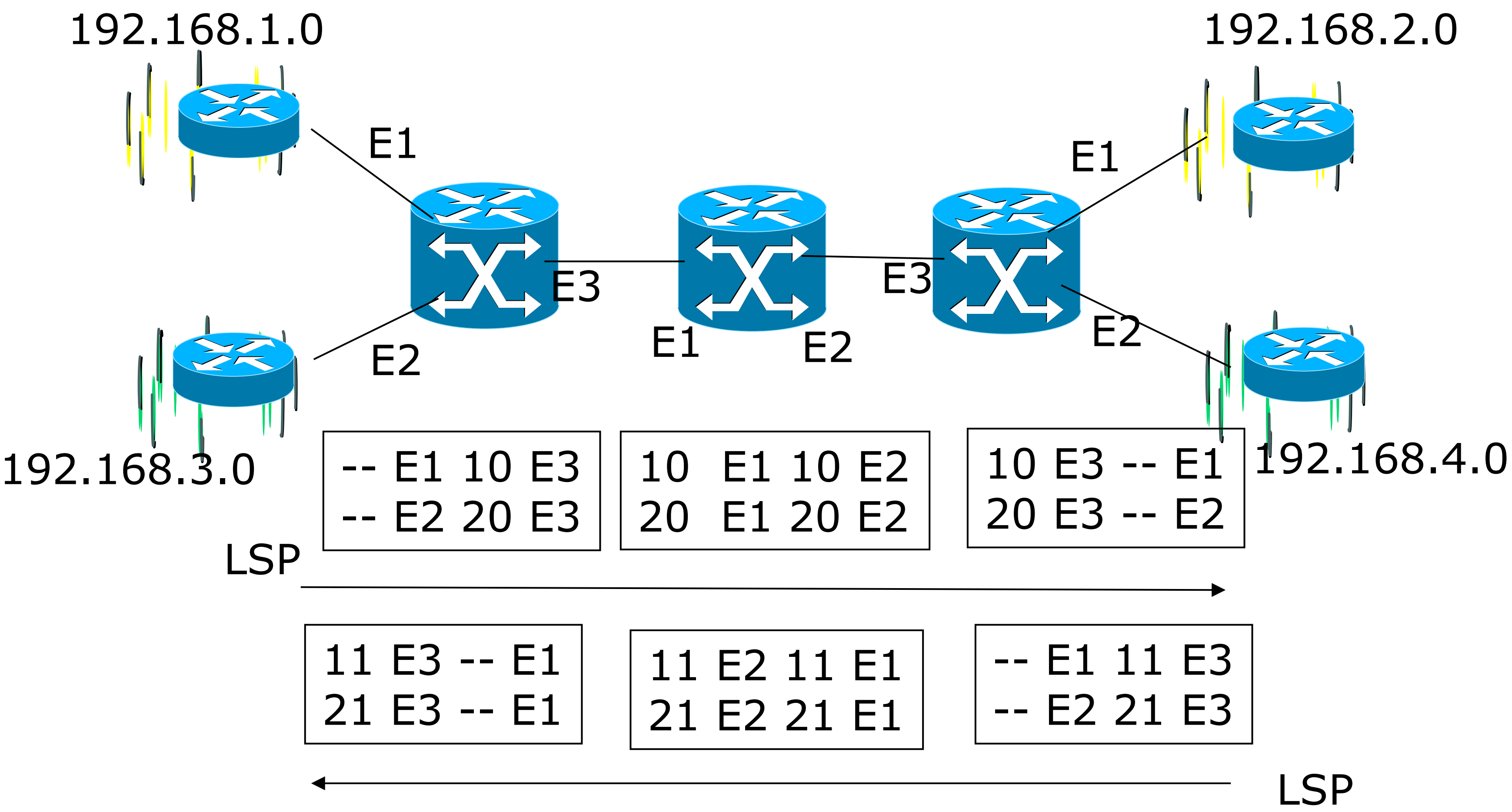
Các thành phần kết nối VPN theo mô hình Provider Provisioned nguồn: Wikipedia

MPLS VPN Connection Model



VPN_A: **10.2.0.0**/24, 11.6.0.0/24, 11.5.0.0/24
VPN_B: **10.2.0.0**/24, 10.1.0.0/24, 10.3.0.0/24

MPLS VPN - Example

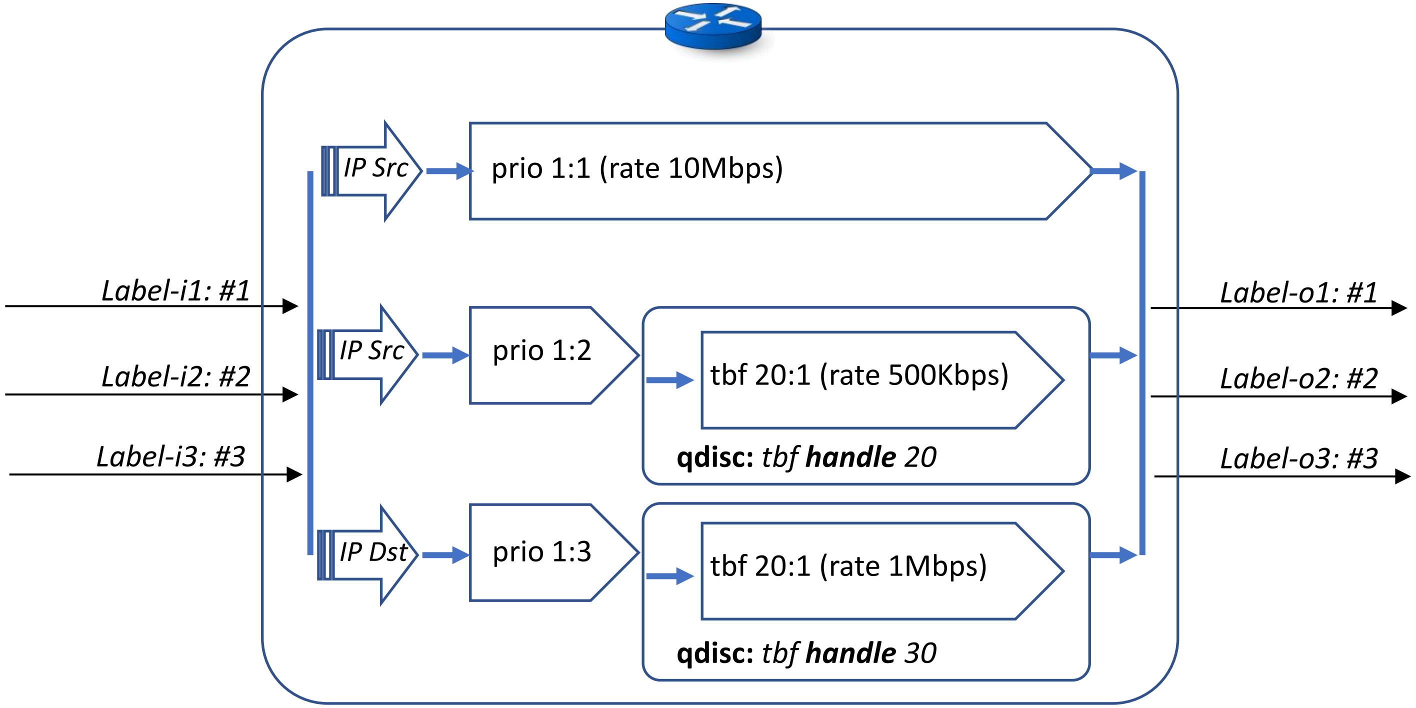


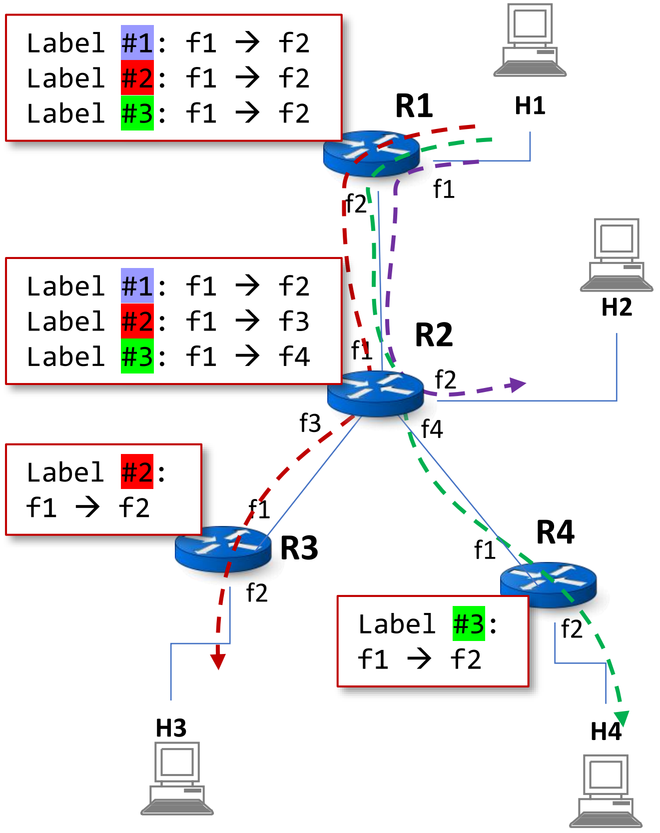
QoS by MPLS

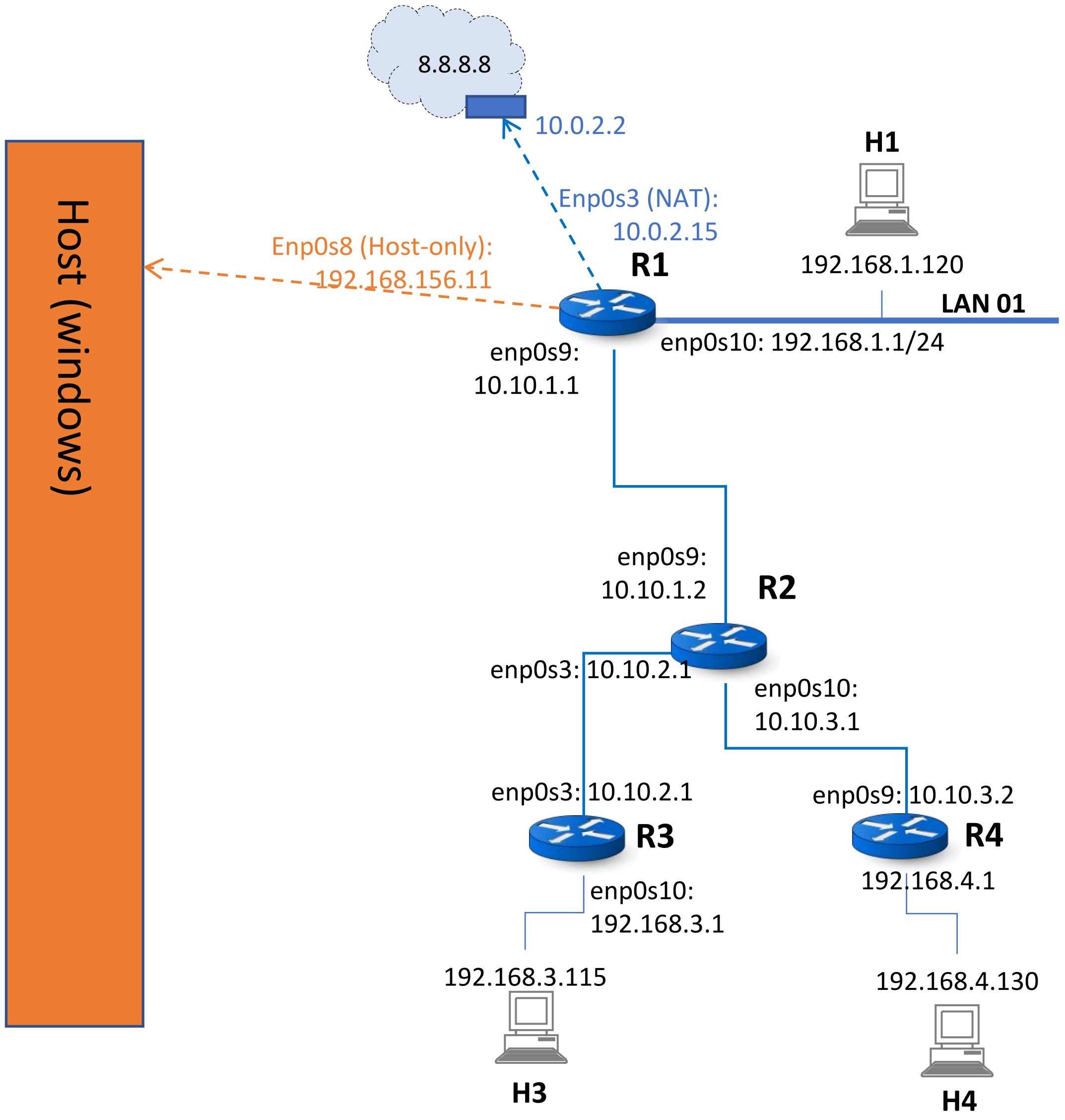
- MPLS doesn’t define a new QoS architecture
- Most of the work on MPLS QoS has focused on supporting current IP QoS architectures
- MPLS QoS uses Differentiated Services (DiffServ) architecture defined for IP QoS
- MPLS DiffServ is defined in RFC3270 (2002)

MPLS Support QoS by DiffServ

- Establish an end-to-end path from source to the destination
- Build a connection-oriented service on the IP network
- Can apply PHB for hop-by-hop forwarding mechanism







8.8.8.8
10.0.2.2
Enp0s3 (NAT):
10.0.2.15
Enp0s8 (Host-only):
192.168.156.11
Host (windows)
R1
H1
192.168.1.120
LAN 01
enp0s10: 192.168.1.1/24
enp0s9:
10.10.1.1
enp0s9:
10.10.1.2
R2
enp0s3: 10.10.2.1
enp0s10:
10.10.3.1
enp0s3: 10.10.2.1
R3
enp0s9: 10.10.3.2
R4
enp0s10:
192.168.3.1
192.168.4.1
192.168.3.115
192.168.4.130
H3
H4