netkit lab

MPLS for Linux

Version	1.1
Author(s)	S.Filippi, F.Antonini (Kasko Networks S.r.l.) Reviewed by M. Rimondini (Roma Tre Univ.)
E-mail	silvia.filippi@kaskonetworks.it
Web	http://www.kaskonetworks.it/
Description	An example with 2 lers, 1 lsr and 3 hosts to show how mpls works on ipv4

copyright notice

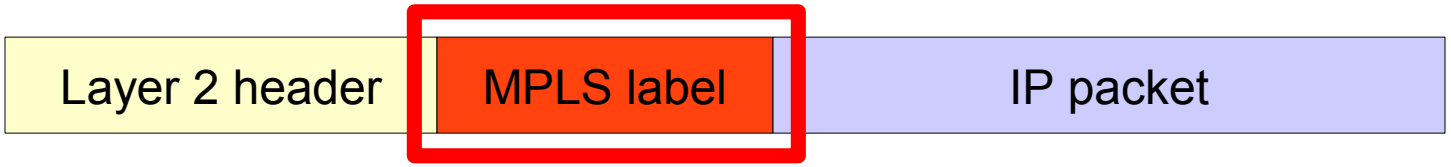
- All the pages/slides in this presentation, including but not limited to, images, photos, animations, videos, sounds, music, and text (hereby referred to as “material”) are protected by copyright.
- This material, with the exception of some multimedia elements licensed by other organizations, is property of the authors and/or organizations appearing in the first slide.
- This material, or its parts, can be reproduced and used for didactical purposes within universities and schools, provided that this happens for non-profit purposes.
- Information contained in this material cannot be used within network design projects or other products of any kind.
- Any other use is prohibited, unless explicitly authorized by the authors on the basis of an explicit agreement.
- The authors assume no responsibility about this material and provide this material “as is”, with no implicit or explicit warranty about the correctness and completeness of its contents, which may be subject to changes.
- This copyright notice must always be redistributed together with the material, or its portions.

MPLS basics (1/3)

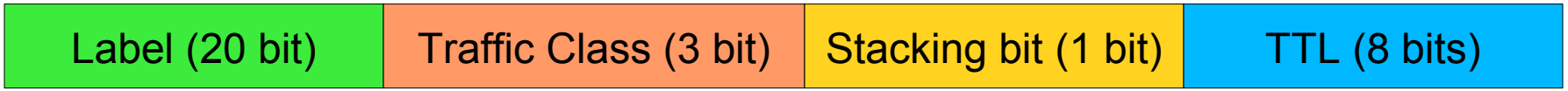
- MPLS stands for Multi Protocol Label Switching
- MPLS forwards packets inside the network using the “label switching” mechanism
- The mechanism is as follows:
  - Traffic that is supposed to be forwarded inside an MPLS network is classified (e.g., all the packets having the same destination address)
  - Each classified packet is assigned a label (label binding) and injected inside the MPLS network
  - Forwarding inside the MPLS network happens by just swapping the label (i.e., no more “routing by network address”)
    - “swapping”=replacing the label with another (possibly different) one; the scope of each label is a single link
  - The label is removed at the egress from the MPLS network

MPLS basics (2/3)

- After label insertion, the routed packet will look like this...



...with the *MPLS label* consisting of these fields:



- Label: the label value
- Traffic Class: used to differentiate priority among different types of traffic
- Stacking bit: a packet may be assigned a stack of labels, instead of a single one; this bit marks the end of the stack
- TTL: the usual Time to Live

MPLS basics (3/3)

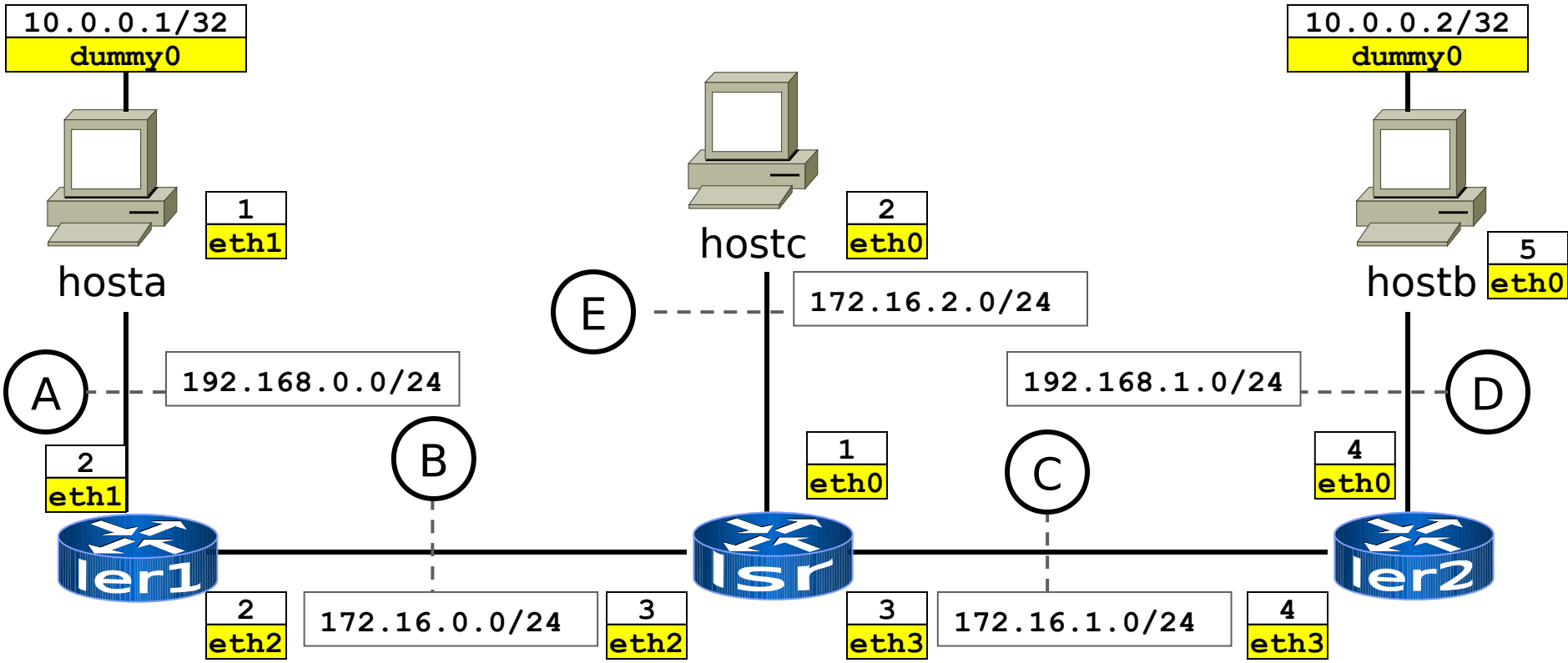
- In order to perform packet forwarding, MPLS routers will check some tables:
  - NHLFE (Next Hop Label Forwarding Entry): contains information about how to forward a packet in MPLS, including:
    - actual IP next hop for the packet
    - operations (push/pop) to be performed on the label stack
  - ILM (Incoming Label Map): maps incoming labelled packets to a NHLFE, i.e., tells how to forward labelled packets
  - XC (Cross Connect): maps an entry in the ILM to a NHLFE; tells the router how to perform label swapping

The MPLS Lab

- This lab is (highly) inspired to the example "*MPLS for Linux: IPv4 over MPLS: two LER one LSR example for mpls-linux-1.95x*" from [1]
- Some more MPLS terminology:
  - LER (Label Edge Router): entry and exit routers for MPLS networks. They push labels in packets entering the MPLS network and pop labels from packets that exit the MPLS network.
  - LSR (Label Switching Router): performs routing inside an MPLS network, based on solely swapping labels.
  - FEC (Forwarding Equivalence Class): a set of IP packets which are forwarded in the same manner

[1] http://sourceforge.net/apps/mediawiki/mpls-linux/index.php?title=Examples

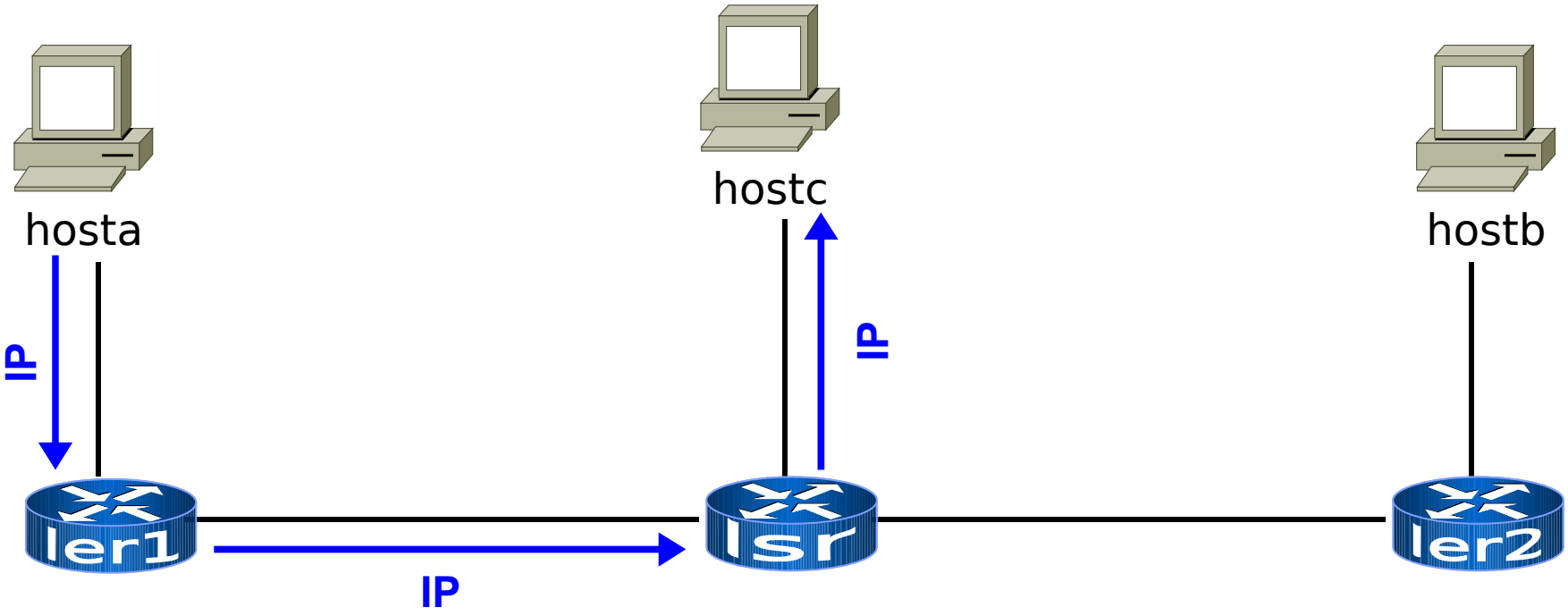
Topology



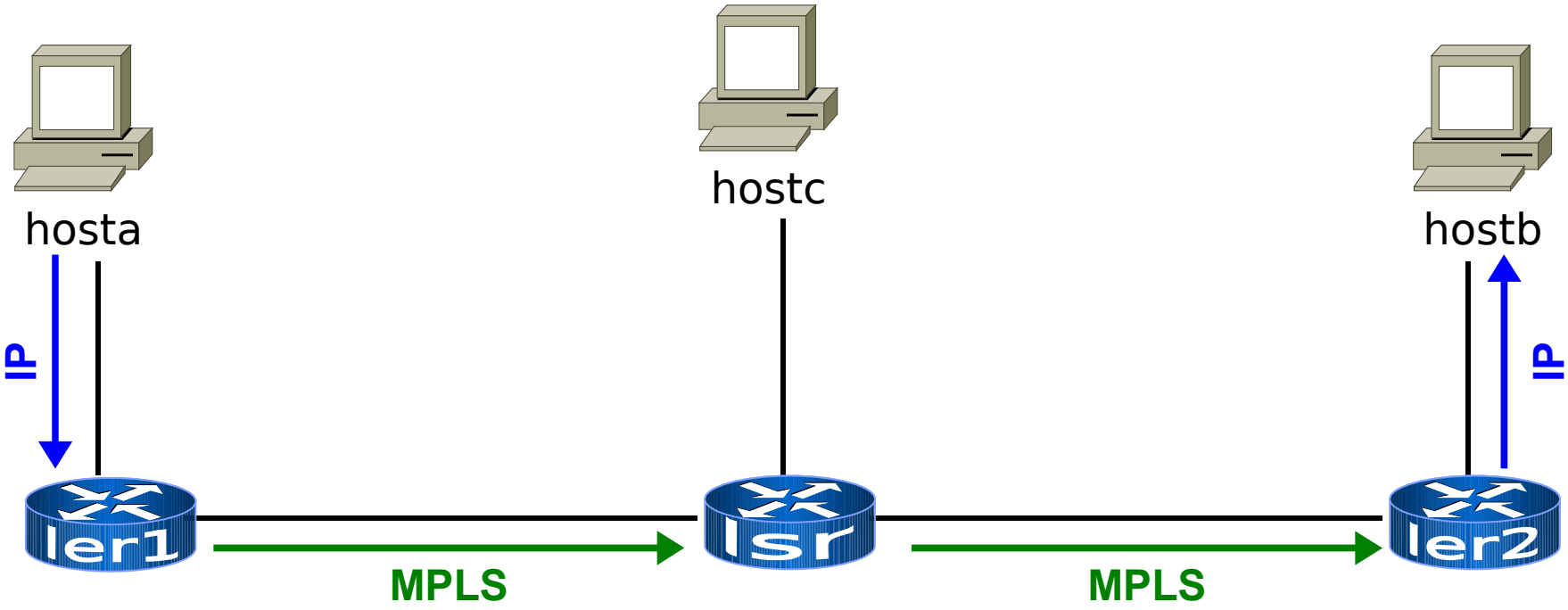
Network topology

- **hosta** is connected by an MPLS network (**ler1** – **lsr** – **ler2**) to **hostb**.
- **ler1**, **lsr**, and **ler2** must be configured in order to allow exchange of packets between **hosta** and **hostb** using MPLS.
- At the same time, **hosta** is connected to **hostc** via IP only. Therefore, IP routes have to be added to **hosta**, **hostc**, **ler1**, and **lsr** in order to support the exchange of conventional IP traffic.

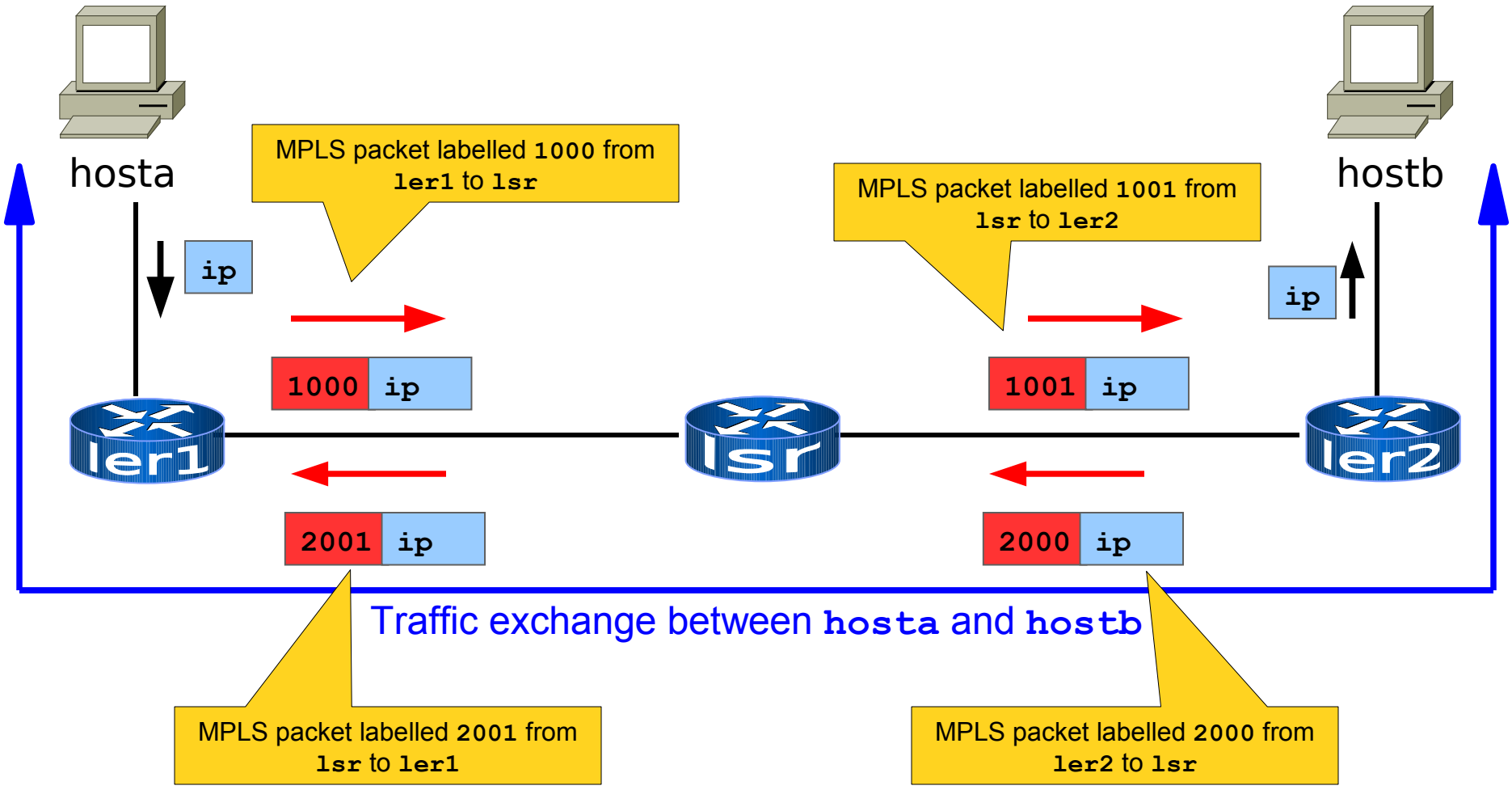
Traffic types



Traffic types



Traffic types



Basic configuration

hosta.startup

```
ifconfig eth1 192.168.0.1 netmask 255.255.255.0 broadcast 192.168.0.255 up
ifconfig dummy0 10.0.0.1/32
ip route add 10.0.0.2/32 via 192.168.0.2 src 10.0.0.1

#traffic to hostc
route add -net 172.16.2.0 netmask 255.255.255.0 gw 192.168.0.2 dev eth1
```

hostb.startup

```
ifconfig eth0 192.168.1.5 netmask 255.255.255.0 broadcast 192.168.1.255 up
ifconfig dummy0 10.0.0.2/32
ip route add 10.0.0.1/32 via 192.168.1.4 src 10.0.0.2
```

hostc.startup

```
ifconfig eth0 172.16.2.2 netmask 255.255.255.0 broadcast 172.16.2.255 up

#traffic to hosta
route add default gw 172.16.2.1
```

MPLS configuration: `ler1`

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route add   .0.0.2/32 via 172.16.0.3 mpls 0x2
```

load MPLS modules

new nhlfe entry

MPLS configuration: ler1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route add      0.2/32 via 172.16.0.3 mpls 0x2
```

load MPLS modules

sequential number identifying the entry (0="new entry": a number will be automatically assigned)

MPLS configuration: ler1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route    l 10.0.0.2/32 via 172.16.0.3 mpls 0x2
```

load MPLS modules

push a label of type “gen” and value 1000...

MPLS configuration: ler1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route add 10.0.0.2/ via 172.16.0.3 mpls 0x2
```

load MPLS modules

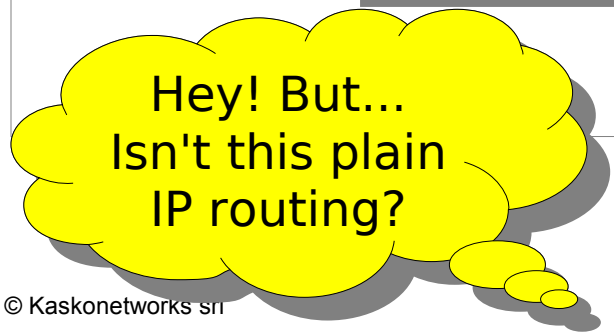
...and forward the packet to a certain router

MPLS configuration: ler1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route add 10.0.0.2/ via 172.16.0.3 mpls 0x2
```

load MPLS modules

...and forward the packet to a certain router



MPLS configuration: ler1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route add 10.0.0.2/ via 172.16.0.3 mpls 0x2
```

load MPLS modules

...and forward the packet to a certain router

Hey! But... Isn't this plain IP routing?

Explanation coming in next slide...

MPLS configuration: `ler1`

- Machines outside the MPLS network...
  - route by IP addresses
  - only know paths to ingress points to the MPLS network
- Routers inside the MPLS network...
  - route by label swapping
  - only know how to reach edge routers in the MPLS network...
    - ...and they achieve this knowledge by borrowing information from the IP routing tables built by an underlying IGP protocol (e.g., RIP, OSPF, etc.)
  - in this case, we simply use static routes
- This allows a complete separation of the routing inside and outside the MPLS network

MPLS configuration: ler1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route add 10.0.0.2/32 via 172.16.0.3 mpls 0x2
```

load MPLS modules

label binding: instruct the router to use the previously created nhlfe to forward the packet

MPLS configuration: ler1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#*********** MPLS TRAFFIC FROM HOSTA TO HOSTB  ****************

mpls nhlfe add key 0 instructions \
        push gen 1000 nexthop eth2 ipv4 172.16.0.3

ip route add 10.0.0.2/32 via 172.16.0.3 mpls 0x2
```

load MPLS modules

this is the key returned by the previous **mpls nhlfe add** command

note: here we are defining a fec

MPLS configuration: `ler1`

```
#************** MPLS TRAFFIC FROM HOSTB TO HOSTA ******************

mpls labelspace set dev eth2 labelspace 0

mpls ilm add label gen 2001 labelspace 0

mpls nhlfe add key 0 instructions nexthop eth1 ipv4 192.168.0.1

mpls xc add ilm_label gen 2001 ilm_labelspace 0 nhlfe_key 0x3
```

enable eth2 to receive mpls traffic

MPLS configuration: ler1

```
#************** MPLS TRAFFIC FROM HOSTB TO HOSTA ******************

mpls labelspace set dev eth2 labelspace 0

mpls ilm add label gen 2001 labelspace 0

mpls nhlfe add key 0 instructions nexthop eth1

mpls xc add ilm_label gen 2001 ilm_labelspace 0
```

put label 2001 in the ilm

required in order to recognize the incoming label and be able to put an entry in the xc list later on

MPLS configuration: ler1

```
#************** MPLS TRAFFIC FROM HOSTB TO HOSTA ******************

mpls labelspace set dev eth2 labelspace 0

mpls ilm add label gen 2001 labelspace 0

mpls nhlfe add key 0 instructions nexthop eth1 ipv4 192.168.0.1

mpls xc add ilm_label gen 2001 ilm_labelspace 0 nhlfe_key 0x3
```

a nhlfe entry that tells where the packet is to be forwarded

MPLS configuration: `ler1`

```
#************** MPLS TRAFFIC FROM HOSTB TO HOSTA ******************

mpls labelspace set dev eth2 labelspace 0

mpls ilm add label gen 2001 labelspace 0

mpls nhlfe add key 0 instructions nexthop eth1 ipv4 192.168.0.1

mpls xc add ilm_label gen 2001 ilm_labelspace 0 nhlfe_key 0x3
```

perform label "swapping":
upon receiving a packet with label 2001 we execute the nhlfe indexed by key 0x3 (returned by the last **`mpls nhlfe add`** command)

note: this instruction "consumes" (=pops) label 2001, which is all we need to do because packets directed to **`hosta`** are exiting the MPLS network

MPLS configuration: `ler2`

- Very similar to **`ler1`**'s

MPLS configuration: `lsr`

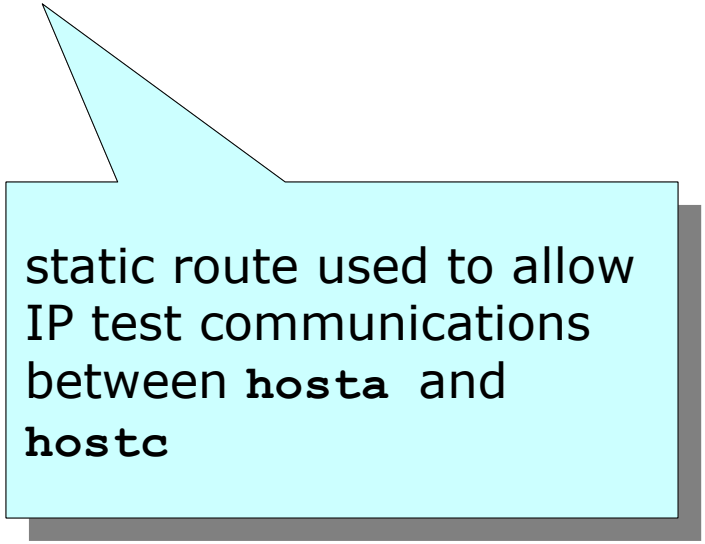
```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

#********** MPLS TRAFFIC FROM HOSTA TO HOSTB

mpls labelspace set dev eth2 labelspace 0
mpls ilm add label gen 1000 labelspace 0
mpls nhlfe add key 0 instructions \
       push gen 1001 nexthop eth3 ipv4 172.16.1.4
mpls xc add ilm_label gen 1000 ilm_labelspace 0 nhlfe_key 0x2


#********** MPLS TRAFFIC FROM HOSTB TO HOSTA

mpls labelspace set dev eth3 labelspace 0
mpls ilm add label gen 2000 labelspace 0
mpls nhlfe add key 0 instructions \
       push gen 2001 nexthop eth2 ipv4 172.16.0.2
mpls xc add ilm_label gen 2000 ilm_labelspace 0 nhlfe_key 0x3
```

swap incoming label **1000** with **1001** and forward on to **ler2**

swap incoming label **2000** with **2001** and forward on to **ler1**

MPLS configuration: `lsr`

```
route add -net 192.168.0.0 netmask 255.255.255.0 gw
172.16.0.2 dev eth2
```



Starting the lab

host machine

```
user@localhost:~$ cd netkit-lab_mpls
user@localhost:~/netkit-lab_mpls$ lstart
```

Commands for MPLS traffic analysis

hosta

```
hosta:~# ping 10.0.0.2
```

lsr

```
lsr:~# tcpdump -i eth3
```

ler2

```
ler2:~# tcpdump -i eth0
```

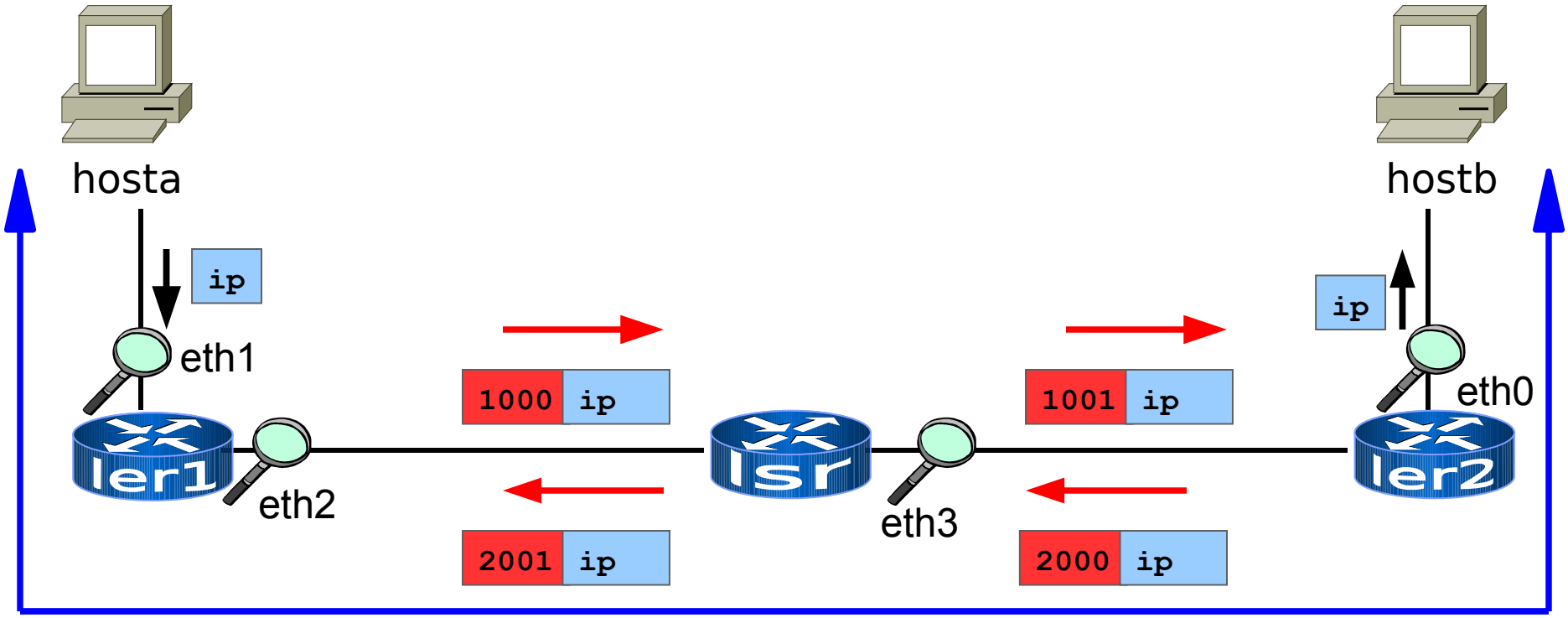
ler1

```
ler1:~# tcpdump -i eth1
```

ler1

```
ler1:~# tcpdump -i eth2
```

MPLS Traffic analysis



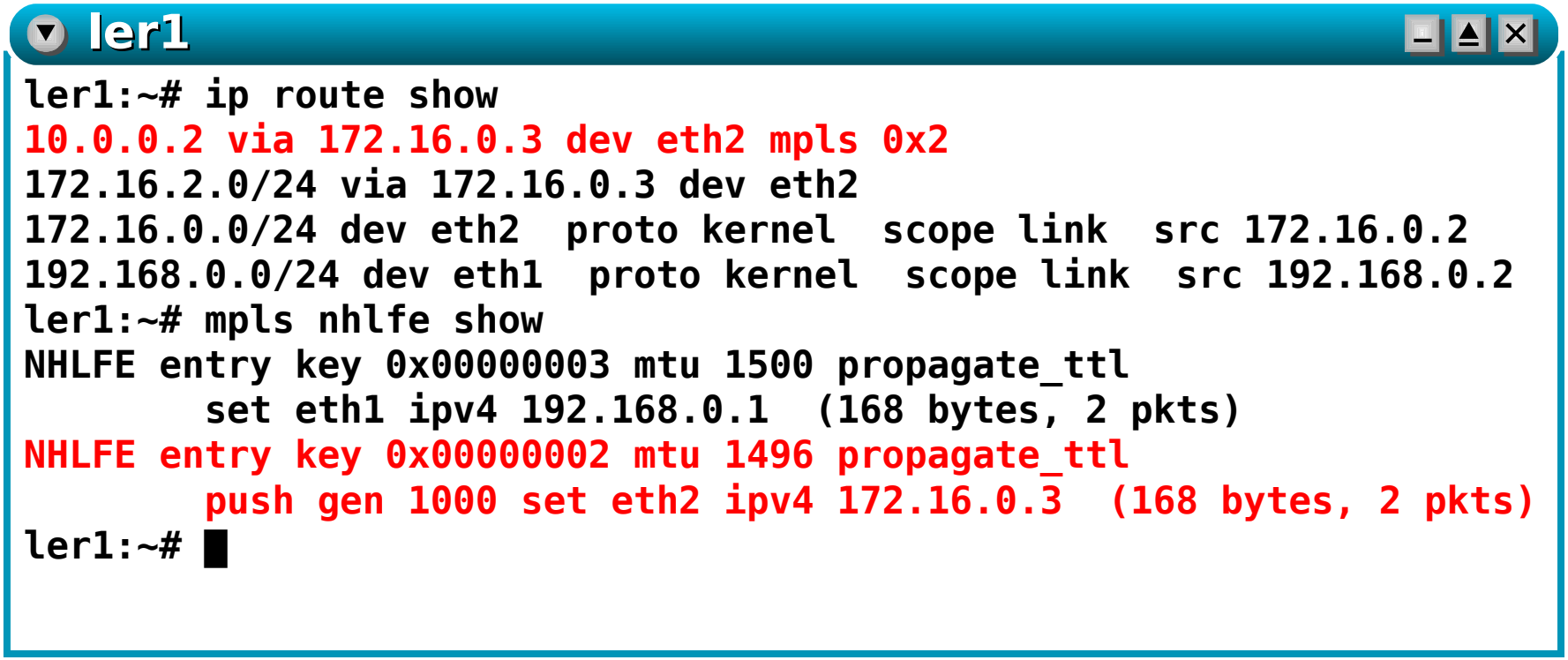
Traffic exchange between **hosta** and **hostb**

MPLS Traffic analysis

- Some checkpoints have been identified along the path from **`hosta`** to **`hostb`**
  - **`ler1`**: both interfaces, to observe traffic before and after the insertion of the MPLS header
  - **`lsr`**: interface **`eth3`**, to observe label switching
  - **`ler2`**: interface **`eth0`**, where the MPLS header is removed
- Similar checkpoints can be considered for the traffic from **`hostb`** to **`hosta`**

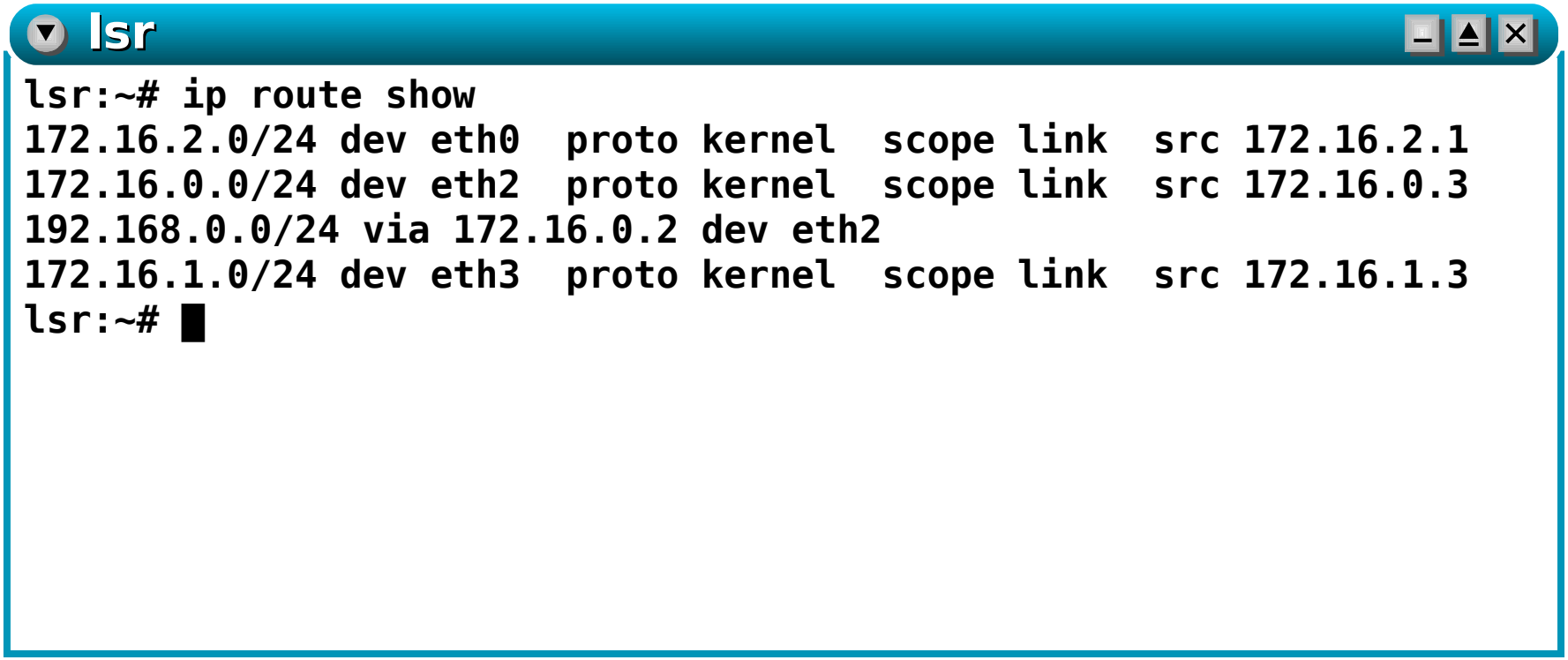
MPLS Traffic analysis

- **ler1** applies NHLFE 0x2 (="push label **1000**") to traffic directed to **10.0.0.2** and forwards it to **lsr**



MPLS Traffic analysis

- **`lsr`**'s IP (correctly) knows nothing about `10.0.0.0`s, but...



MPLS Traffic analysis

- ...**lsr**'s MPLS knows how to forward labeled packets!

lsr

```
lsr:~# mpls nhlfe show
NHLFE entry key 0x00000003 mtu 1496 propagate_ttl
        push gen 2001 set eth2 ipv4 172.16.0.2  (168 bytes, 2 pkts)
NHLFE entry key 0x00000002 mtu 1496 propagate_ttl
        push gen 1001 set eth3 ipv4 172.16.1.4  (168 bytes, 2 pkts)
lsr:~# mpls ilm show
ILM entry label gen 2000 labelspace 0 proto ipv4
        pop forward key 0x00000003  (176 bytes, 2 pkts)
ILM entry label gen 1000 labelspace 0 proto ipv4
        pop forward key 0x00000002  (176 bytes, 2 pkts)
lsr:~# mpls xc show
XC entry ilm_label gen 2000 ilm_labelspace 0 nhlfe_key 0x00000003
XC entry ilm_label gen 1000 ilm_labelspace 0 nhlfe_key 0x00000002
lsr:~#
```

MPLS Traffic analysis

- incoming packets with label **1000** have their label recognized and popped...

lsr

```
lsr:~# mpls nhlfe show
NHLFE entry key 0x00000003 mtu 1496 propagate_ttl
        push gen 2001 set eth2 ipv4 172.16.0.2  (168 bytes, 2 pkts)
NHLFE entry key 0x00000002 mtu 1496 propagate_ttl
        push gen 1001 set eth3 ipv4 172.16.1.4  (168 bytes, 2 pkts)
lsr:~# mpls ilm show
ILM entry label gen 2000 labelspace 0 proto ipv4
        pop forward key 0x00000003  (176 bytes, 2 pkts)
ILM entry label gen 1000 labelspace 0 proto ipv4
        pop forward key 0x00000002  (176 bytes, 2 pkts)
lsr:~# mpls xc show
XC entry ilm_label gen 2000 ilm_labelspace 0 nhlfe_key 0x00000003
XC entry ilm_label gen 1000 ilm_labelspace 0 nhlfe_key 0x00000002
lsr:~#
```

MPLS Traffic analysis

- ...and are forwarded to **ler2** after swapping the label with **1001**

lsr

```
lsr:~# mpls nhlfe show
NHLFE entry key 0x00000003 mtu 1496 propagate_ttl
        push gen 2001 set eth2 ipv4 172.16.0.2  (168 bytes, 2 pkts)
NHLFE entry key 0x00000002 mtu 1496 propagate_ttl
        push gen 1001 set eth3 ipv4 172.16.1.4  (168 bytes, 2 pkts)
lsr:~# mpls ilm show
ILM entry label gen 2000 labelspace 0 proto ipv4
        pop forward key 0x00000003  (176 bytes, 2 pkts)
ILM entry label gen 1000 labelspace 0 proto ipv4
        pop forward key 0x00000002  (176 bytes, 2 pkts)
lsr:~# mpls xc show
XC entry ilm_label gen 2000 ilm_labelspace 0 nhlfe_key 0x00000003
XC entry ilm_label gen 1000 ilm_labelspace 0 nhlfe_key 0x00000002
lsr:~#
```

MPLS Traffic analysis

- also **ler2**'s IP (correctly) knows nothing about **10.0.0.2**, but...

ler2

```
ler2:~# ip route show
10.0.0.1 via 172.16.1.3 dev eth3 mpls 0x3
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.4
172.16.1.0/24 dev eth3  proto kernel  scope link  src 172.16.1.4
ler2:~#
```

MPLS Traffic analysis

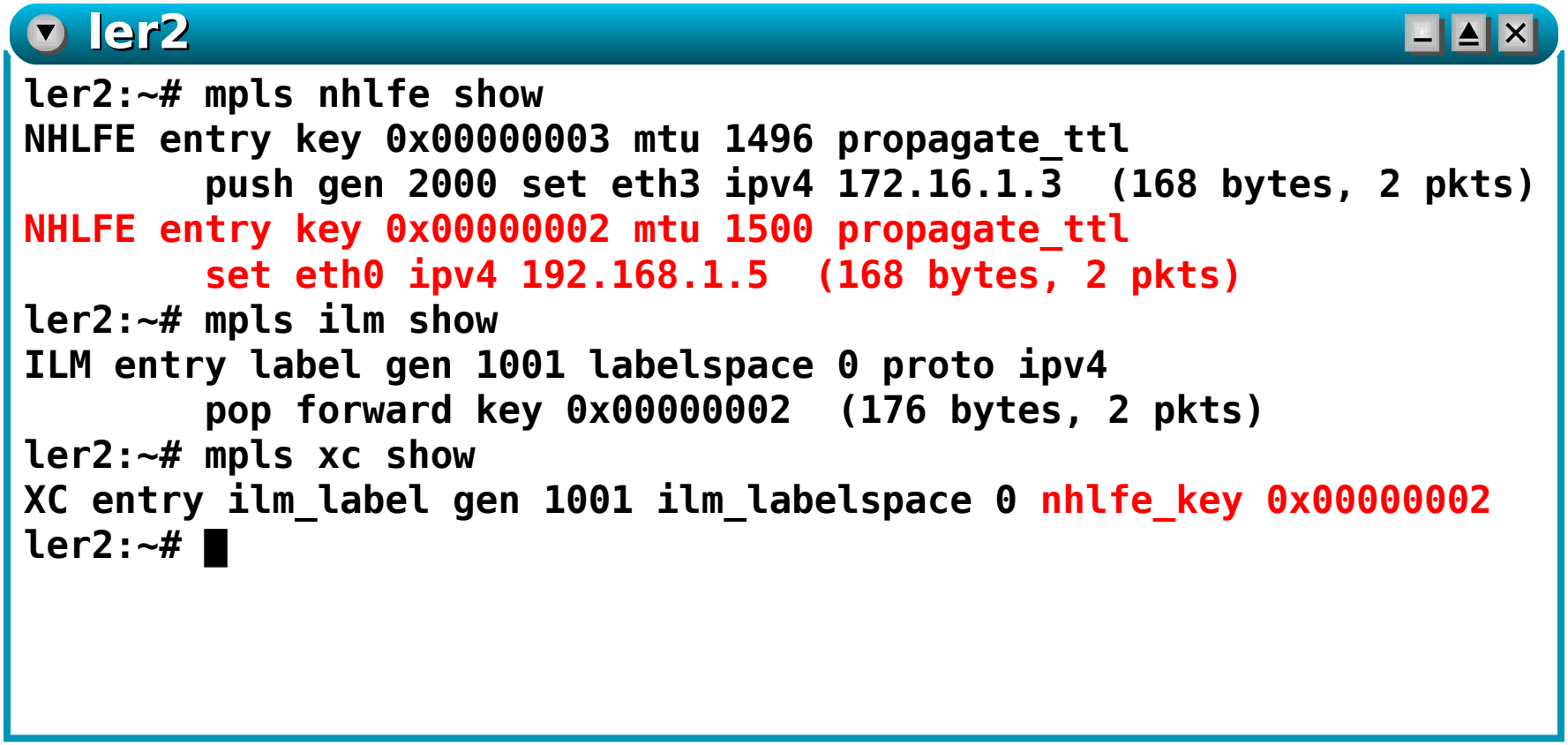
- ...incoming packets with label 1001 have their label recognized and popped...

ler2

```
ler2:~# mpls nhlfe show
NHLFE entry key 0x00000003 mtu 1496 propagate_ttl
        push gen 2000 set eth3 ipv4 172.16.1.3  (168 bytes, 2 pkts)
NHLFE entry key 0x00000002 mtu 1500 propagate_ttl
        set eth0 ipv4 192.168.1.5  (168 bytes, 2 pkts)
ler2:~# mpls ilm show
ILM entry label gen 1001 labelspace 0 proto ipv4
        pop forward key 0x00000002  (176 bytes, 2 pkts)
ler2:~# mpls xc show
XC entry ilm_label gen 1001 ilm_labelspace 0 nhlfe_key 0x00000002
ler2:~#
```

MPLS Traffic analysis

- ...and are forwarded to **hostb**



MPLS Traffic analysis

- We sniff packets using tcpdump and examine the dumps on the host using wireshark

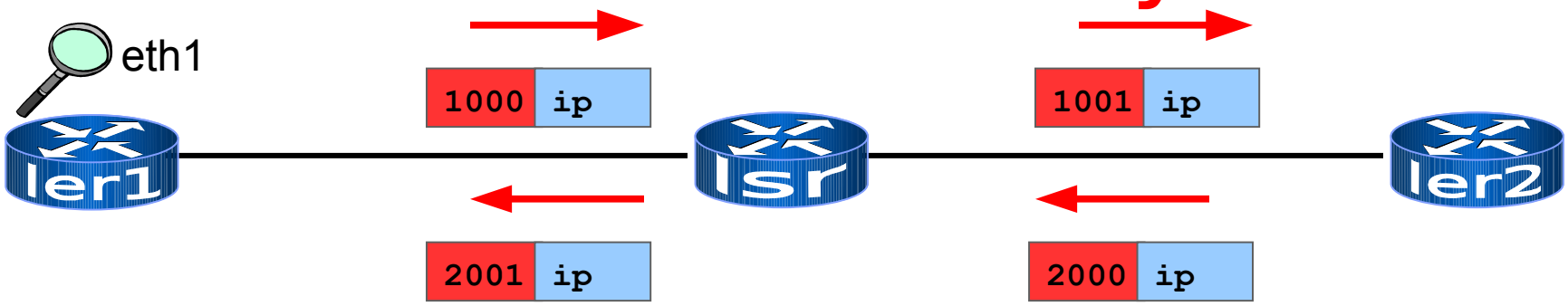
lsr

```
lsr:~# tcpdump -i ethX -w /hostlab/sniffXXX.cap -s 1500
```

host machine

```
user@localhost:~$ cd netkit-lab_mpls
user@localhost:~/netkit-lab_mpls$ wireshark -r sniffXXX.cap
```

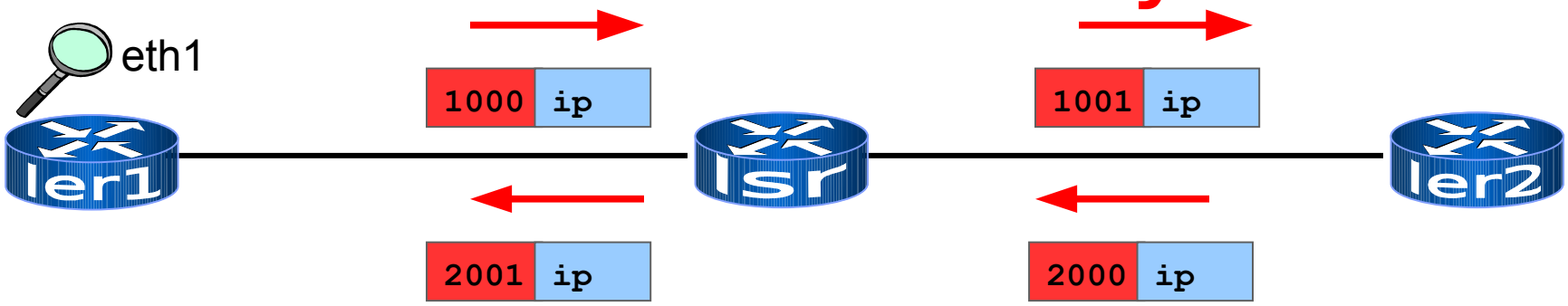
MPLS Traffic analysis



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000540	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.010003	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010808	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	2.020014	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
6	2.020541	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
7	3.030045	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
8	3.030628	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
9	4.040040	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
10	4.040623	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
11	5.050014	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
12	5.050569	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply

```
▷ Frame 1 (98 bytes on wire, 98 bytes captured)
▷ Ethernet II, Src: 4a:b6:3f:db:98:62 (4a:b6:3f:db:98:62), Dst: 5e:93:ba:fa:7c:1c (5
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol
```

MPLS Traffic analysis

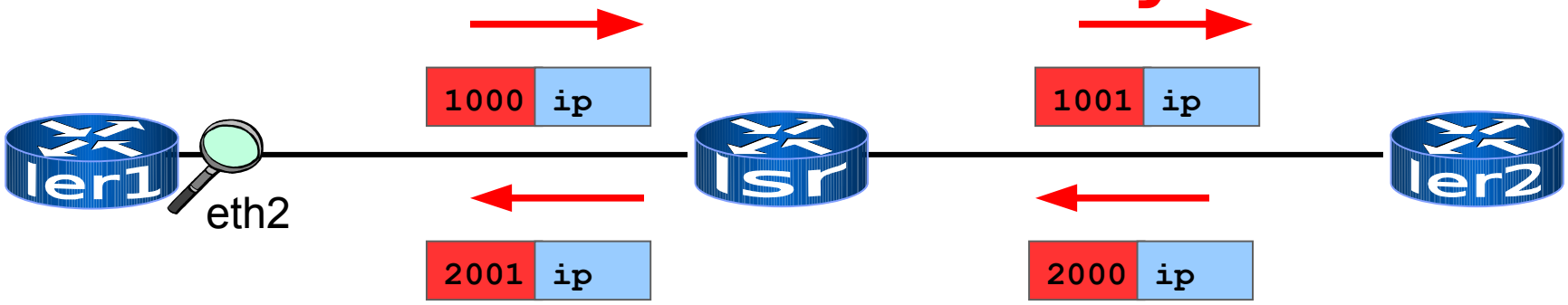


No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000540	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.010003	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010808	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	2.020014	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
6	2.020541	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
7	3.030045	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
8	3.030628	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
9	4.040040	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
10	4.040623	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
11	5.050014	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
12	5.050569	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply

plain ICMP packets

```
▷ Frame 1 (98 bytes on wire, 98 bytes captured)
▷ Ethernet II, Src: 4a:b6:3f:db:98:62 (4a:b6:3f:db:98:62), Dst: 5e:93:ba:fa:7c:1c (5
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol
```

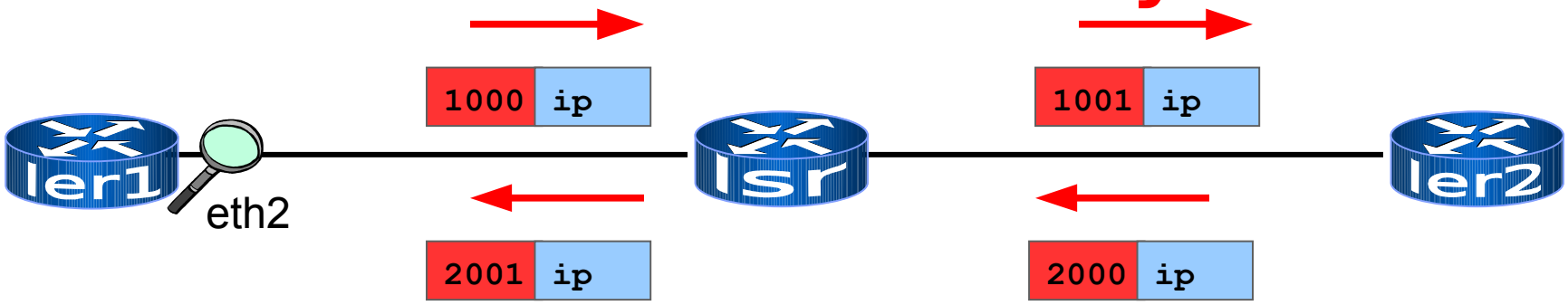
MPLS Traffic analysis



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000440	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.009669	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010171	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	2.009923	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
6	2.010422	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
7	2.879181	86:59:f1:c8:89:f8	6a:a8:77:72:7a:d8	ARP	Who has 172.16.0.2? Te
8	2.879205	6a:a8:77:72:7a:d8	86:59:f1:c8:89:f8	ARP	172.16.0.2 is at 6a:a8:
9	3.019863	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
10	3.020483	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
11	4.029677	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
12	4.030141	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply

```
▷ Frame 1 (102 bytes on wire, 102 bytes captured)
▷ Ethernet II, Src: 6a:a8:77:72:7a:d8 (6a:a8:77:72:7a:d8), Dst: 86:59:f1:c8:89:f8 (8
▽ MultiProtocol Label Switching Header, Label: 1000, Exp: 0, S: 1, TTL: 63
    MPLS Label: 1000
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 63
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol
```

MPLS Traffic analysis

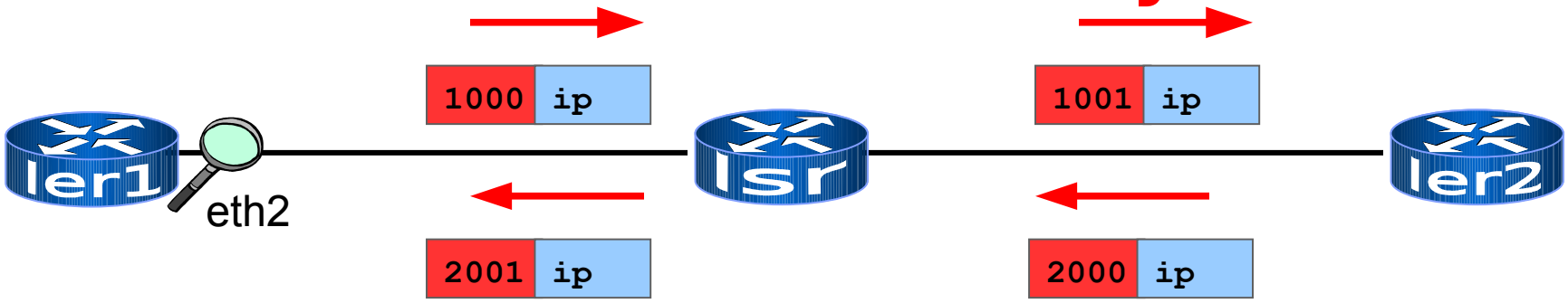


No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000440	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.009669	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010171	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	2.009923	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
				ICMP	Echo (ping) reply
			a:a8:77:72:7a:d8	ARP	Who has 172.16.0.2? Te
			d8 86:59:f1:c8:89:f8	ARP	172.16.0.2 is at 6a:a8:
			10.0.0.2	ICMP	Echo (ping) request
			10.0.0.1	ICMP	Echo (ping) reply
11			10.0.0.2	ICMP	Echo (ping) request
12	4.030141	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply

ICMP packets...

```
Frame 1 (102 bytes on wire, 102 bytes captured)
Ethernet II, Src: 6a:a8:77:72:7a:d8 (6a:a8:77:72:7a:d8), Dst: 86:59:f1:c8:89:f8 (8
MultiProtocol Label Switching Header, Label: 1000, Exp: 0, S: 1, TTL: 63
    MPLS Label: 1000
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 63
Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
Internet Control Message Protocol
```

MPLS Traffic analysis



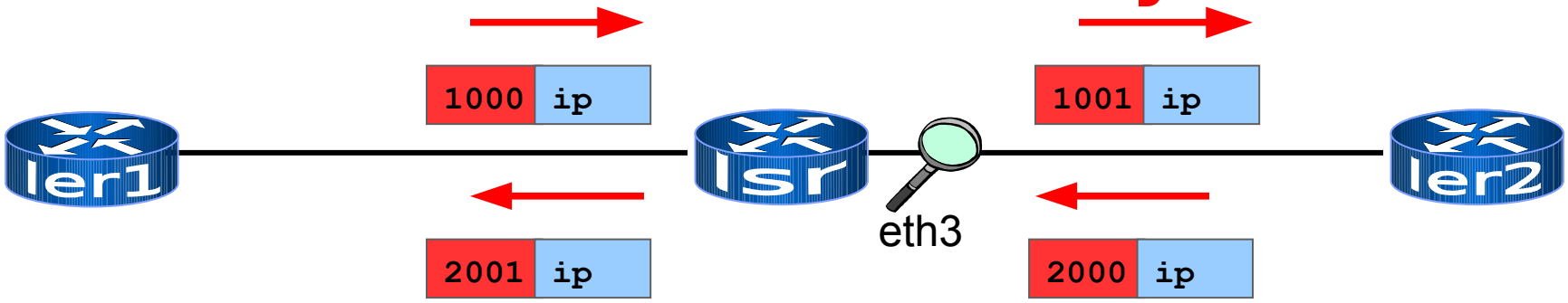
No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000440	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.009669	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010171	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	2.009923	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
				ICMP	Echo (ping) reply
			:a8:77:72:7a:d8	ARP	
			:d8 86:59:f1:c8:89:f8	AR	
			10.0.0.2	IC	
			10.0.0.1	IC	
11			10.0.0.2	IC	
12	4.030141	10.0.0.2	10.0.0.1	ICM	

ICMP packets...

...encapsulated in MPLS

```
Frame 1 (102 bytes on wire, 102 bytes captured)
Ethernet II, Src: 6a:a8:77:72:7a:d8 (6a:a8:77:72:7a:d8), Dst: 86:59:  c8:89:f8 (8
MultiProtocol Label Switching Header, Label: 1000, Exp: 0, S: 1, TTL: 63
    MPLS Label: 1000
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 63
Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
Internet Control Message Protocol
```

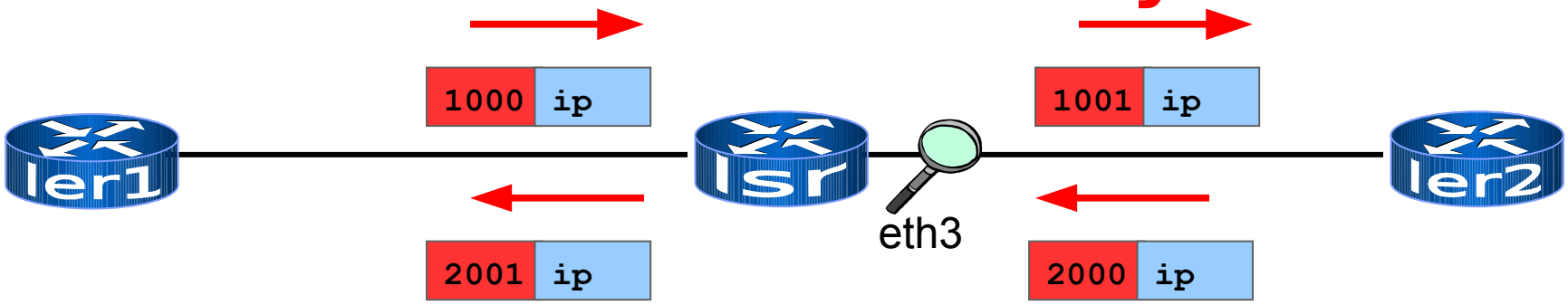
MPLS Traffic analysis



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000313	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.009854	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010204	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	1.160649	aa:83:be:20:c1:bd	d2:8b:a4:86:7e:fd	ARP	Who has 172.16.1.4? Te
6	1.160859	d2:8b:a4:86:7e:fd	aa:83:be:20:c1:bd	ARP	172.16.1.4 is at d2:8b:
7	2.020003	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
8	2.020365	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
9	3.029933	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
10	3.030262	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
11	4.039937	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
12	4.040333	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply

```
▷ Frame 1 (102 bytes on wire, 102 bytes captured)
▷ Ethernet II, Src: aa:83:be:20:c1:bd (aa:83:be:20:c1:bd), Dst: d2:8b:a4:86:7e:fd (d
▽ MultiProtocol Label Switching Header, Label: 1001, Exp: 0, S: 1, TTL: 62
    MPLS Label: 1001
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 62
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol
```

MPLS Traffic analysis



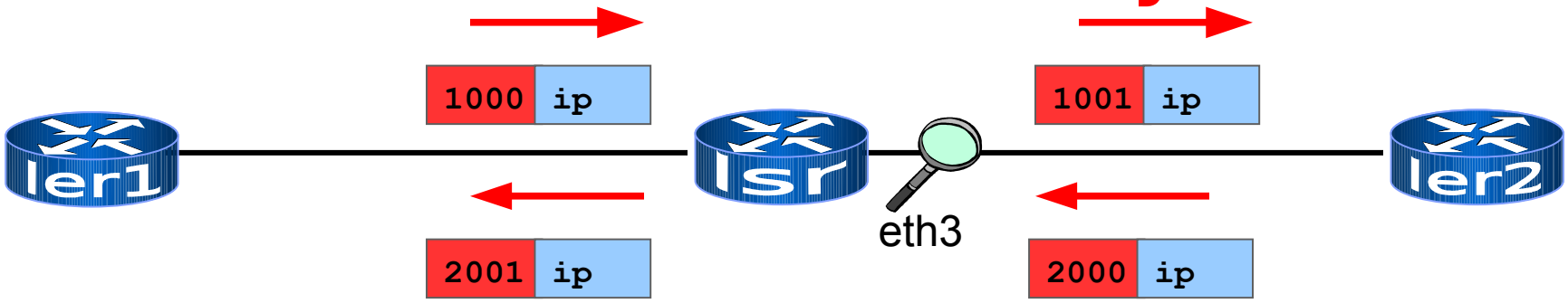
No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000313	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.009854	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010204	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	1.160649	aa:83:be:20:c1:bd	d2:8b:[illegible]:7e:fd	ARP	Who has 172.16.1.4? Te
		[illegible]:be:20:c1:bd		ARP	172.16.1.4 is at d2:8b:
			10.0.0.2	ICMP	
			10.0.0.1	ICMP	
			10.0.0.2	ICMP	
			10.0.0.1	ICMP	
11	4.039937	10.0.0.1	10.0.0.2	ICMP	
12	4.040333	10.0.0.2	10.0.0.1	ICMP	

ICMP packets...

...encapsulated in MPLS...

```
▷ Frame 1 (102 bytes on wire, 102 bytes captured)
▷ Ethernet II, Src: aa:83:be:20:c1:bd (aa:83:be:20:c1:bd), Dst: d2:8b:[illegible]6:7e:fd (d
▽ MultiProtocol Label Switching Header, Label: 1001, Exp: 0, S: 1, TTL: 62
    MPLS Label: 1001
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 62
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol
```

MPLS Traffic analysis



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000313	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.009854	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010204	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	1.160649	aa:83:be:20:c1:bd	d2:8b:[illegible]:7e:fd	ARP	Who has 172.16.1.4? Te
		[illegible]:be:20:c1:bd		ARP	172.16.1.4 is at d2:8b:
			10.0.0.2	ICMP	
			10.0.0.1	ICMP	
			10.0.0.2	ICMP	
			10.0.0.1	ICMP	
11	4.039937	10.0.0.1	10.0.0.2	ICMP	
12	4.040333	10.0.0.2	10.0.0.1	ICMP	

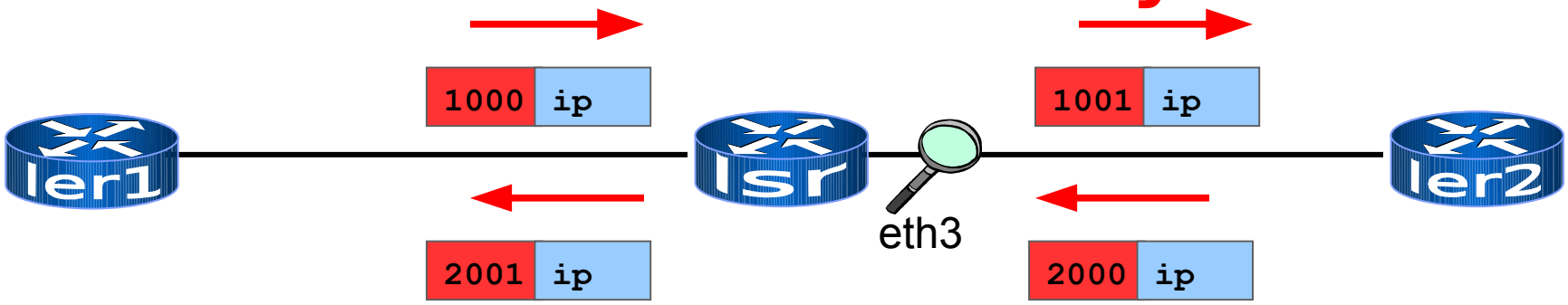
ICMP packets...

...encapsulated in MPLS...

```
▷ Frame 1 (102 bytes on wire, 102 bytes captured)
▷ Ethernet II, Src: aa:83:be:20:c1:bd (aa:83:be:20:c1:bd), Dst: d2:8b:a[illegible]:86:7e:fd (d
▽ MultiProtocol Label Switching Header, Label: 1001, Exp: 0, S: 1, TTL: 62
    MPLS Label: 1001
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 62
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2
▷ Internet Control Message Protocol
```

...with a swapped label!

MPLS Traffic analysis

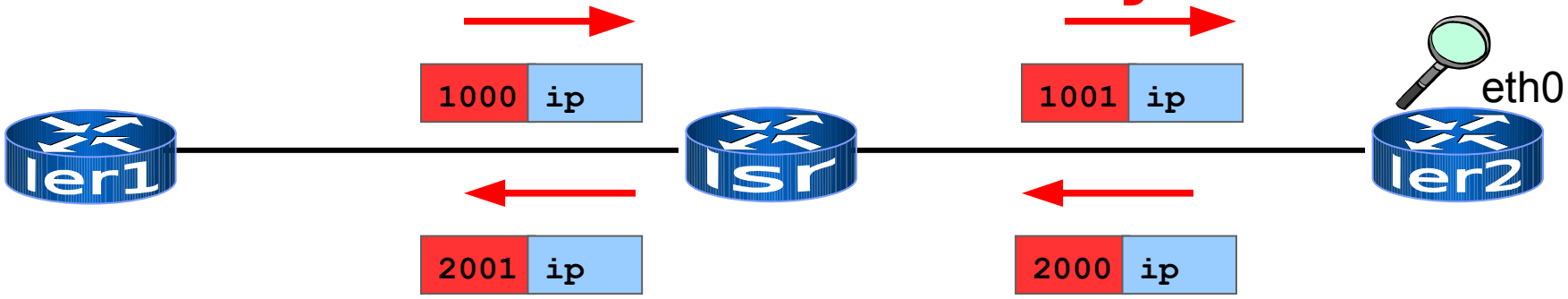


No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000313	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.009854	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010204	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	1.160649	aa:83:be:20:c1:bd	d2:8b:a4:86:7e:fd	ARP	Who has 172.16.1.4? Te
		aa:83:be:20:c1:bd		ARP	172.16.1.4 is at d2:8b:
		.0.0.2		ICMP	Echo (ping) request
		.0.0.1		ICMP	Echo (ping) reply
		.0.0.2		ICMP	Echo (ping) request
		.0.0.1		ICMP	Echo (ping) reply
		.0.0.2		ICMP	Echo (ping) request
		.0.0.1		ICMP	Echo (ping) reply

echo replies travel with a different label

```
▷ Frame 2 (102 bytes on wire, 102 bytes captured)
▷ Ethernet II, Src: d2:8b:a4:86:7e:fd (d2:8b:a4:86:7e:fd), Dst: aa:83:be:20:c1:bd (a
▽ MultiProtocol Label Switching Header, Label: 2000, Exp: 0, S: 1, TTL: 63
    MPLS Label: 2000
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 63
▷ Internet Protocol, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.1 (10.0.0.1)
▷ Internet Control Message Protocol
```

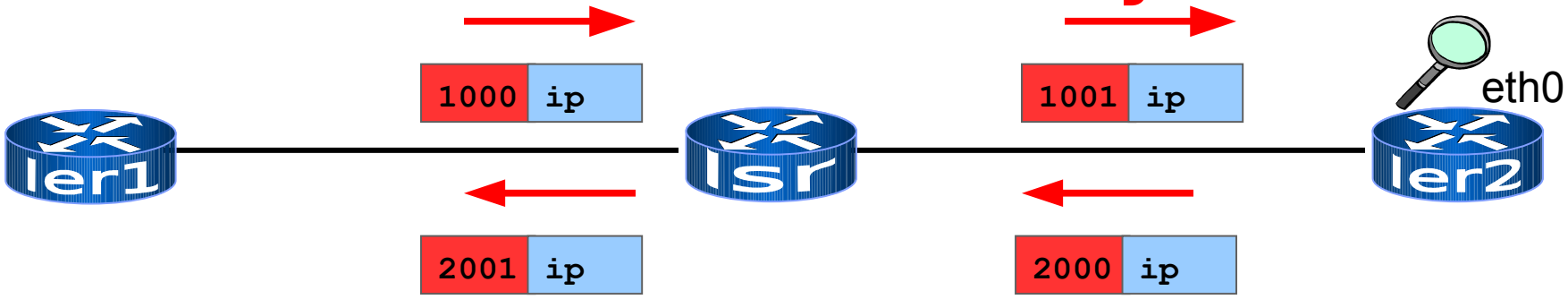
MPLS Traffic analysis



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000121	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.010061	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010313	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	2.020049	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
6	2.020245	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
7	3.030085	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
8	3.030287	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
9	4.040128	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
10	4.040307	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
11	5.049959	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
12	5.050155	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply

```
▷ Frame 1 (98 bytes on wire, 98 bytes captured)
▷ Ethernet II, Src: ae:48:62:f5:63:c6 (ae:48:62:f5:63:c6), Dst: de:e5:92:ad:6f:e4 (d
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol
```

MPLS Traffic analysis

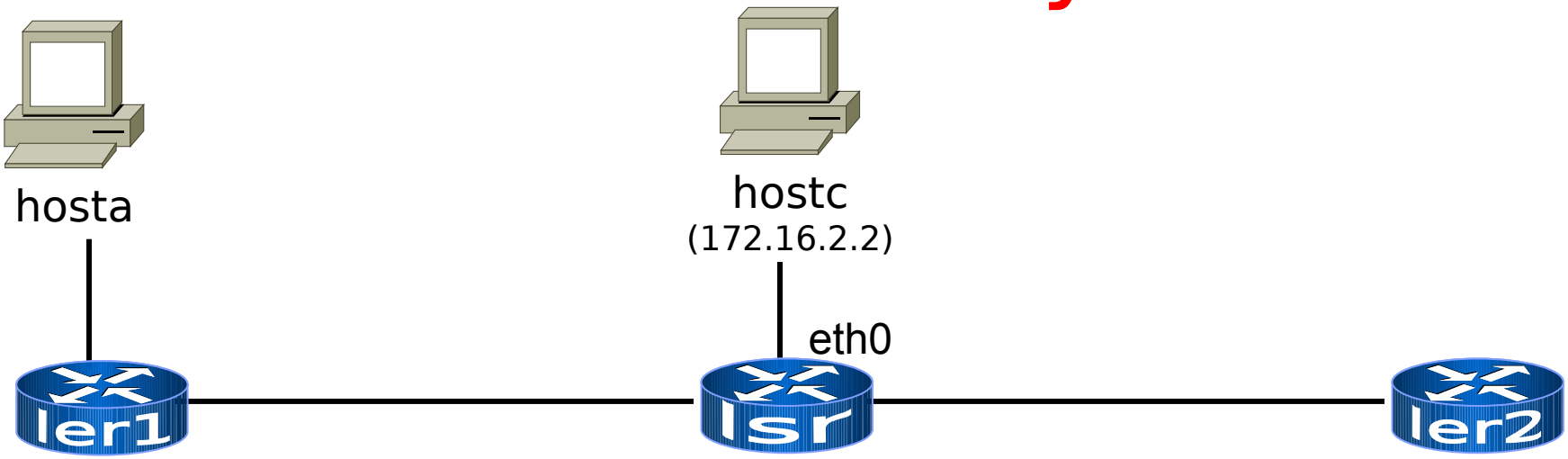


No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
2	0.000121	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
3	1.010061	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
4	1.010313	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
5	2.020049	10.0.0.1	10.0.0.2	ICMP	Echo (ping) request
6	2.020245	10.0.0.2	10.0.0.1	ICMP	Echo (ping) reply
7	3.030085		2	ICMP	Echo (ping) request
8	3.030287		1	ICMP	Echo (ping) reply
9	4.040128		2	ICMP	Echo (ping) request
10	4.040307		1	ICMP	Echo (ping) reply
11	5.049959		2	ICMP	Echo (ping) request
12	5.050155		1	ICMP	Echo (ping) reply

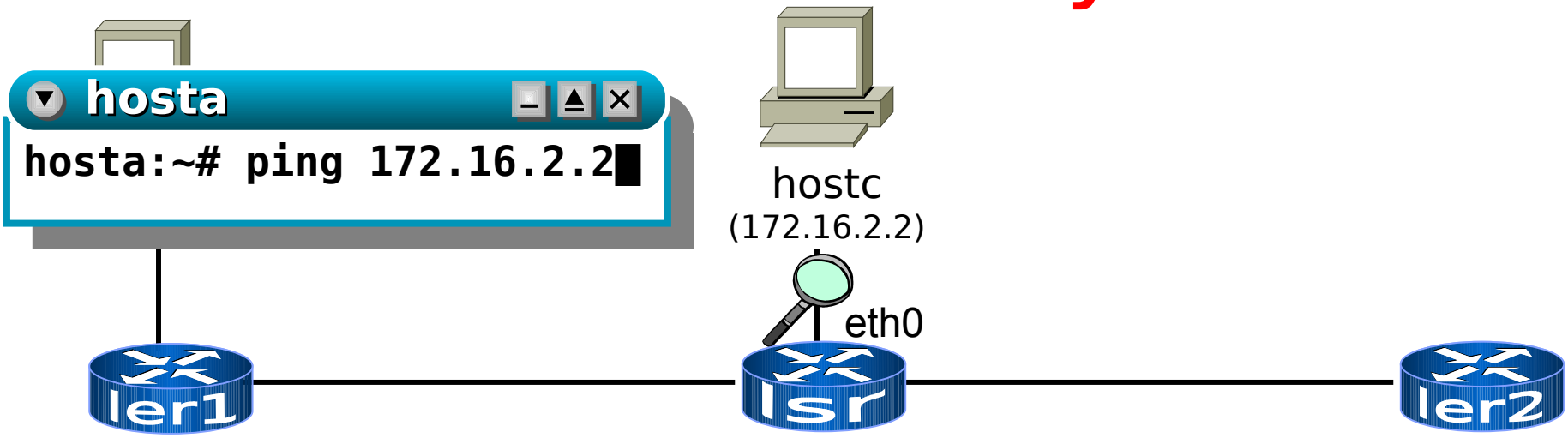
(back to) plain ICMP packets

```
▷ Frame 1 (98 bytes on wire, 98 bytes captured)
▷ Ethernet II, Src: ae:48:62:f5:63:c6 (ae:48:62:f5:63:c6), Dst: de:e5:92:ad:6f:e4 (d
▷ Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol
```

IP Traffic analysis



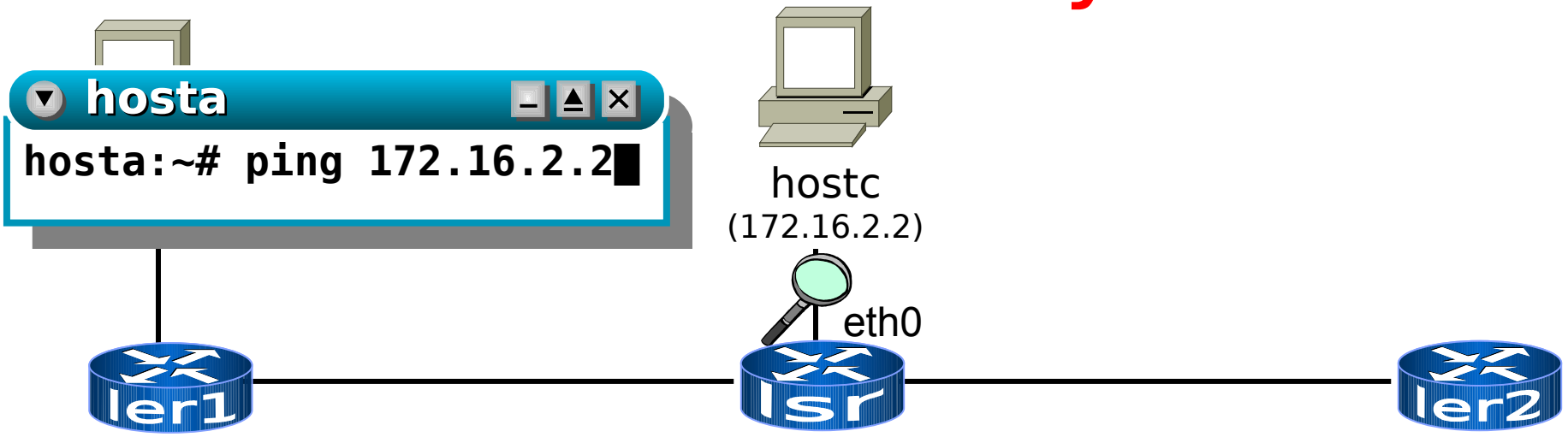
IP Traffic analysis



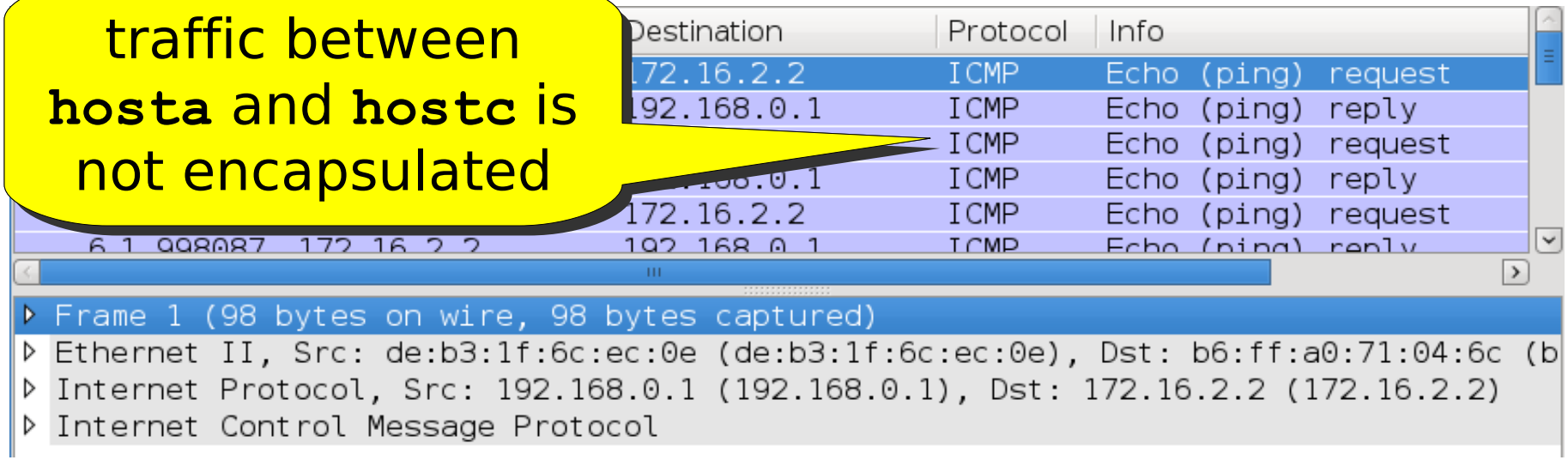
No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.1	172.16.2.2	ICMP	Echo (ping) request
2	0.000074	172.16.2.2	192.168.0.1	ICMP	Echo (ping) reply
3	0.998815	192.168.0.1	172.16.2.2	ICMP	Echo (ping) request
4	0.998933	172.16.2.2	192.168.0.1	ICMP	Echo (ping) reply
5	1.997949	192.168.0.1	172.16.2.2	ICMP	Echo (ping) request
6	1.998087	172.16.2.2	192.168.0.1	ICMP	Echo (ping) reply

```
▷ Frame 1 (98 bytes on wire, 98 bytes captured)
▷ Ethernet II, Src: de:b3:1f:6c:ec:0e (de:b3:1f:6c:ec:0e), Dst: b6:ff:a0:71:04:6c (b
▷ Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 172.16.2.2 (172.16.2.2)
▷ Internet Control Message Protocol
```

IP Traffic analysis



traffic between **hosta** and **hostc** is not encapsulated



Proposed exercises

- Sniff on all the interfaces with wireshark
- On a virtual router, try to give the commands stated in the .startup “by hand”, one by one directly on the virtual machine in order to observe the returned keys
- Try to change NHLFE tables on the running lab, adding or deleting entries (use “**`mpls -help`**” inside virtual machines to see the help)