netkit lab

MPLS VPNs with overlapping address spaces

Version	1.0
Author(s)	S.Filippi, L.Ricci, F.Antonini
E-mail	silvia.filippi@kaskonetworks.it
Web	http://www.kaksonetworks.it/
Description	A lab showing how a Layer 3 MPLS VPN works in case of customer addresses space overlapping

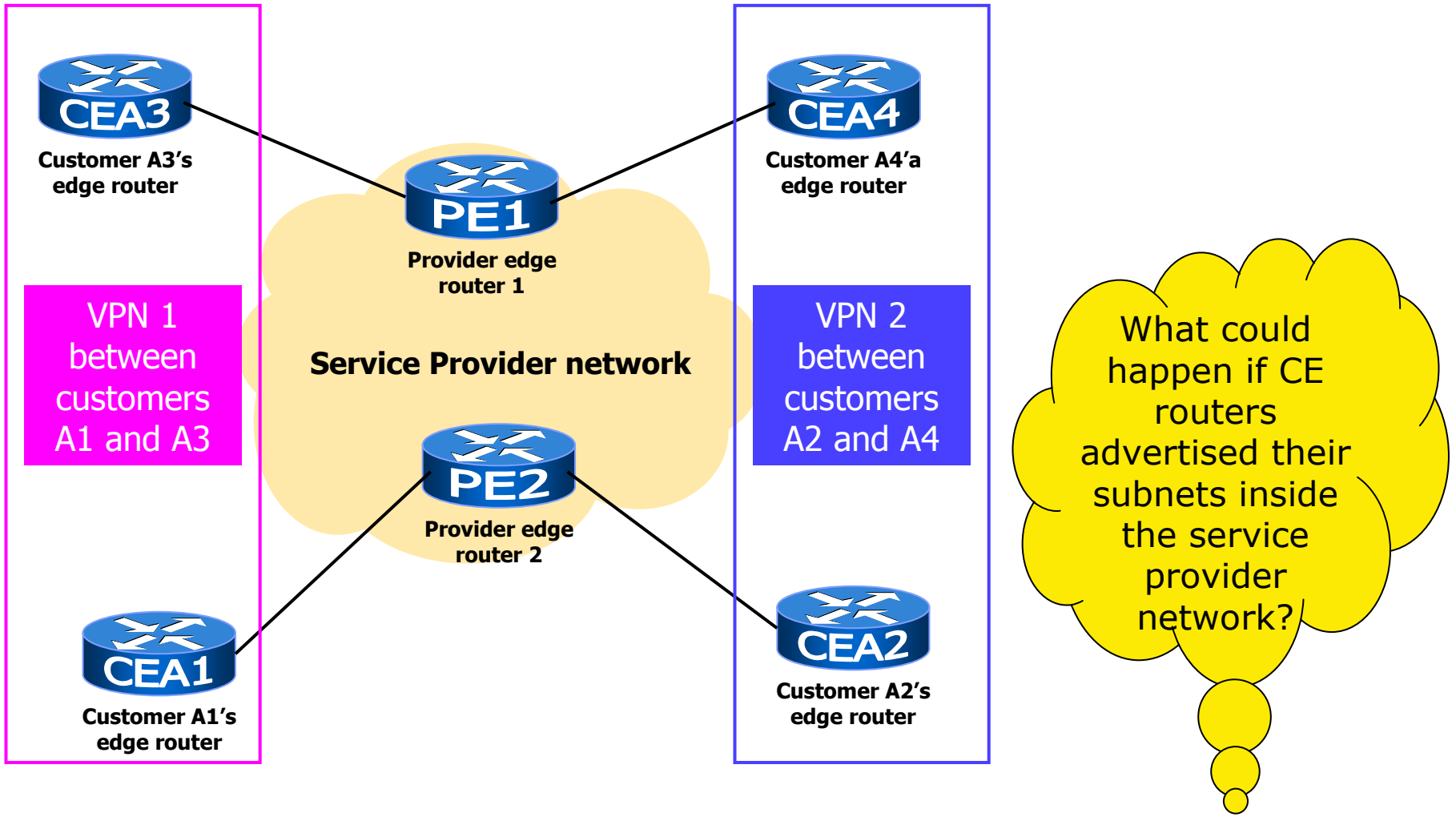
copyright notice

- 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 VPNs

- VPN (Virtual Private Network) is a private network between sites that use a public network (Internet)
- MPLS VPNs are used to separate the traffic, acting as a logical wire berween users (customers), and providing also control of data flows and quality of service (they are so called "Trusted-VPNs"). Security features will be assured by the underlying network. The underlying carrier network is not visible to the cutomer, nor is the user aware of the presence of other customers traversing the same backbone, so all the job of forwarding packets inside the MPLS VPN is carried on by the provider network, no effort is required to the customer
- MPLS VPNs can avoid interferences between customers traffic due to overlapping of customers' subnet addresses, as shown later on

VPNs network example

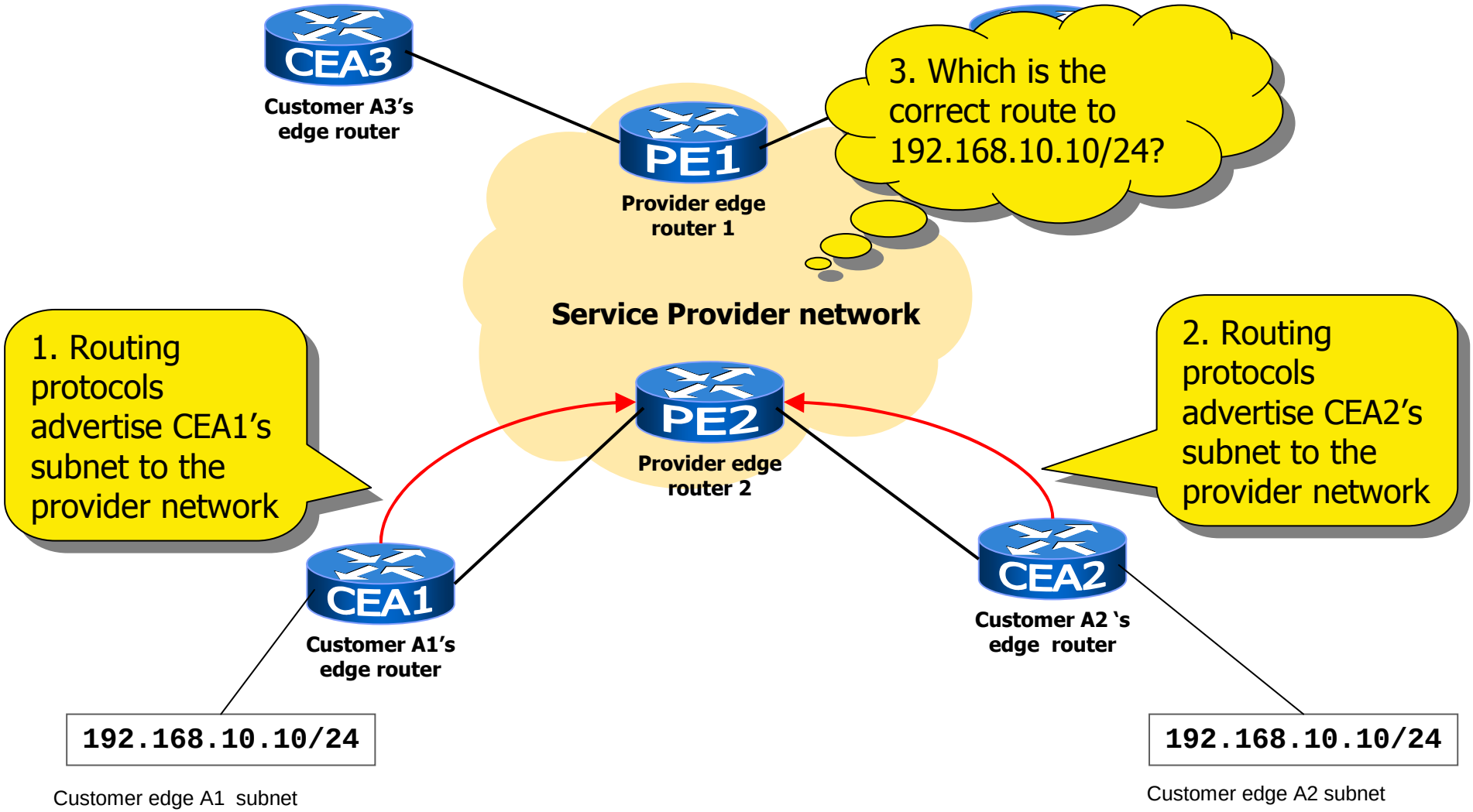


No interference between customers belonging to different VPNs is allowed

Overlapping customer address range

- Due to network protocols action, the SPE would learn address ranges from the various customers and then advertise those routes into the SP's network
- Many customers use the same IP private address ranges
- The SP routers will be confused by overlapping prefixes and network protocols, choosing the best route using their own rules, can route traffic between customers belonging to different VPNs

Overlapping customer address example



Solution: MPLS VPNs

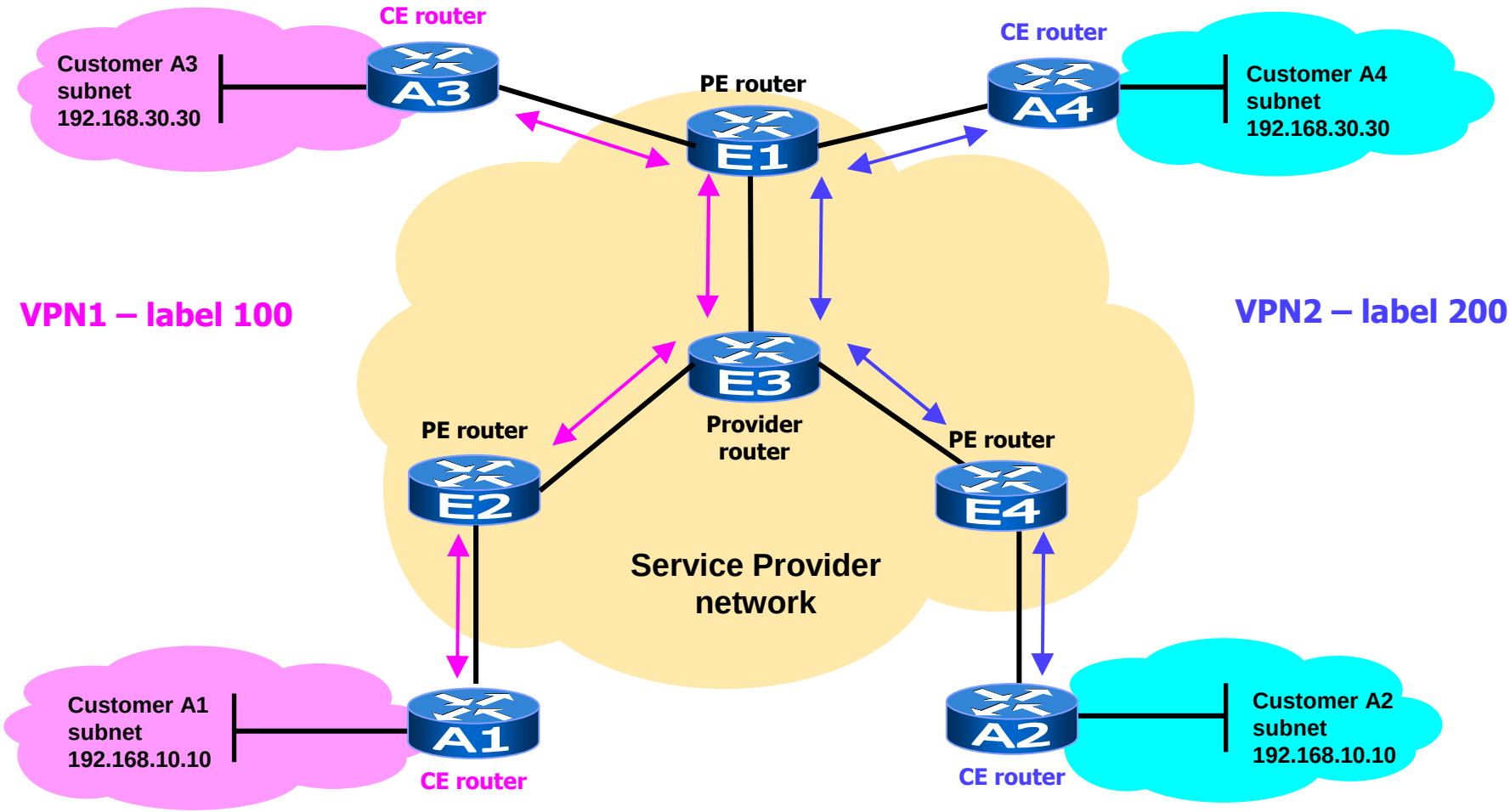
- MPLS VPNs implement the use of VRF (Virtual Routing and Forwarding) tables in order to keep separate customers routes
- These "per-customer" routing tables are stored inside PE routers, and customer routes are never injected inside the provider's network
- The ingress PE router will place 2 labels inside packets:
  - An outer MPLS label (Stack bit = 0), used to switch the packet inside the MPLS VPN network
  - An inner MPLS label (Stack bit = 1), that identifies the VRF table to be used and, as a consequence, the VPN

The Netkit lab

- This lab has been inspired to the examples proposed by by Irina Dumitrascu and Adrian Popa and available at address [1]
- This lab uses static routes. In real life network protocols are used instead to route packets inside the MPLS network and propagate VRF tables between PE routers.

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

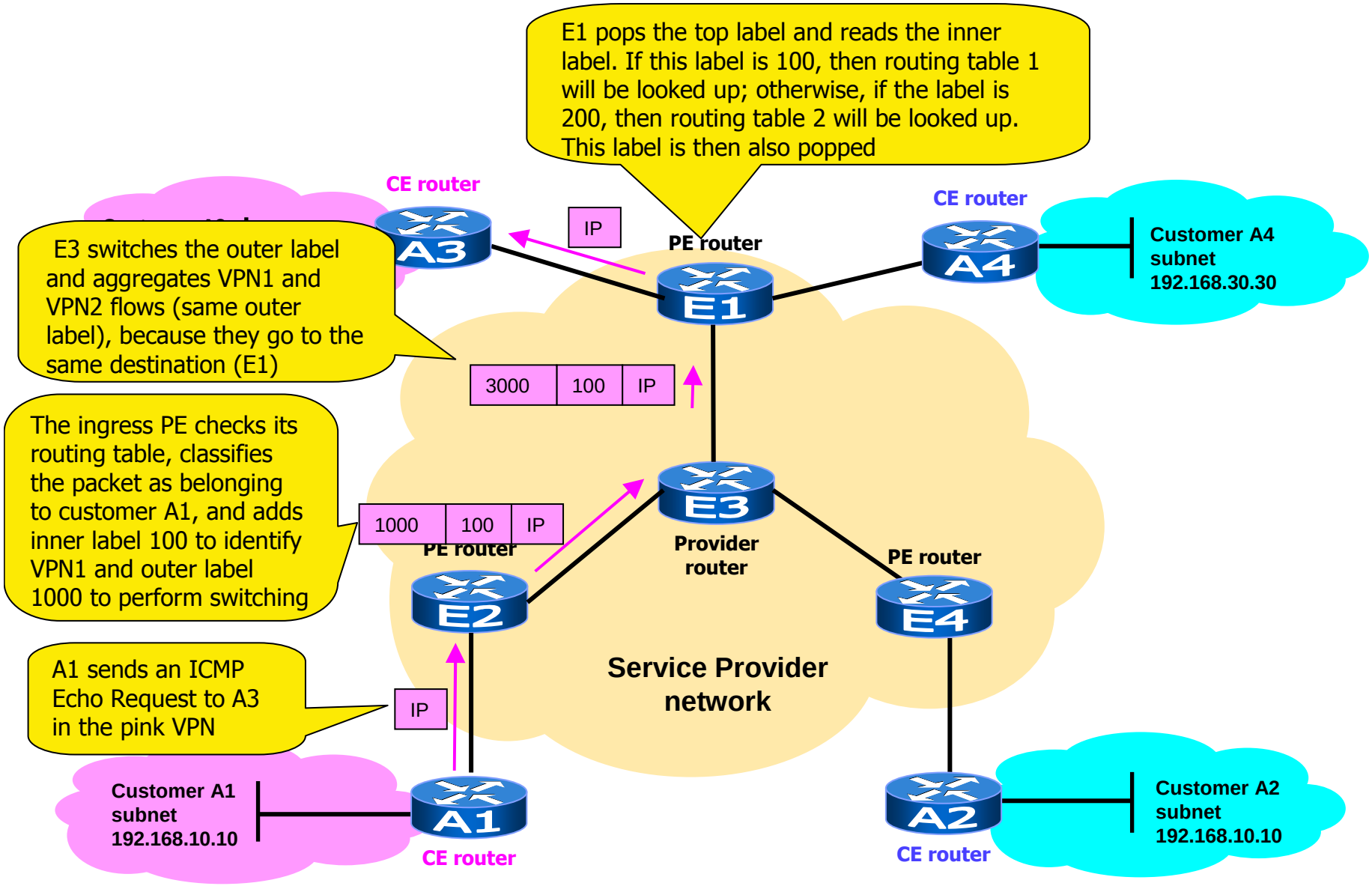
Network topology



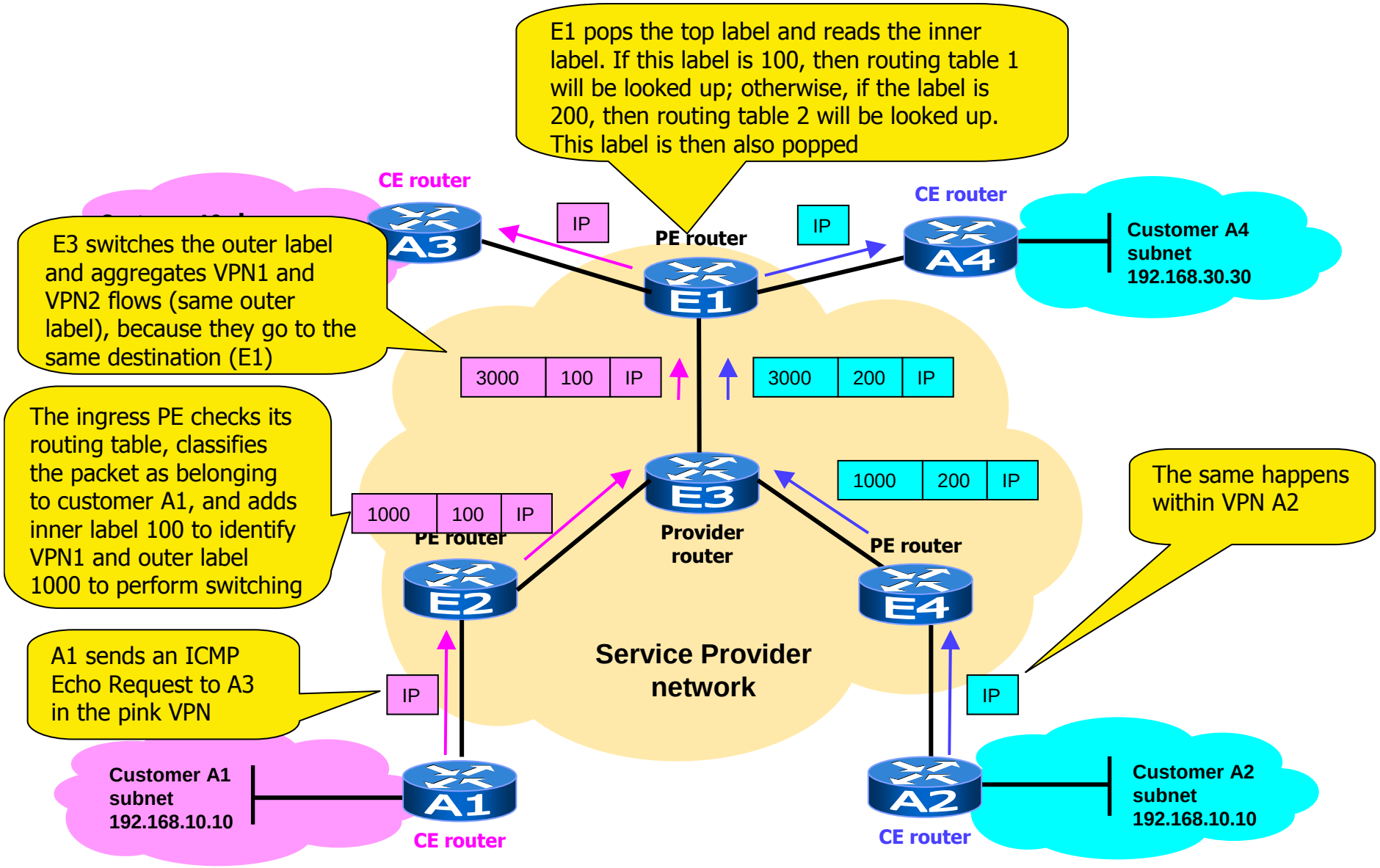
Network configuration

- The Netkit lab will build 2 VPNs (VPN1 and VPN2) in order to keep traffic between CA1 and CA3 (VPN1) separate from that between CA2 and CA4 (VPN2)
- CA1 has the same address space as CA2 (in the lab it is simulated by dummy interface 192.168.10.10)
- CA3 has the same address space as CA4 (in the lab it is simulated by dummy interface 192.168.30.30)
- This address overlapping issue must be resolved by the provider
  - Two MPLS labels in each packet
  - Usage of VRF tables inside provider edge routers E1, E2, and E4
  - VRFs are emulated in Linux by using multiple kernel routing tables
- VPN1 will be identfied by inner label 100
- VPN2 will be identfied by inner label 200

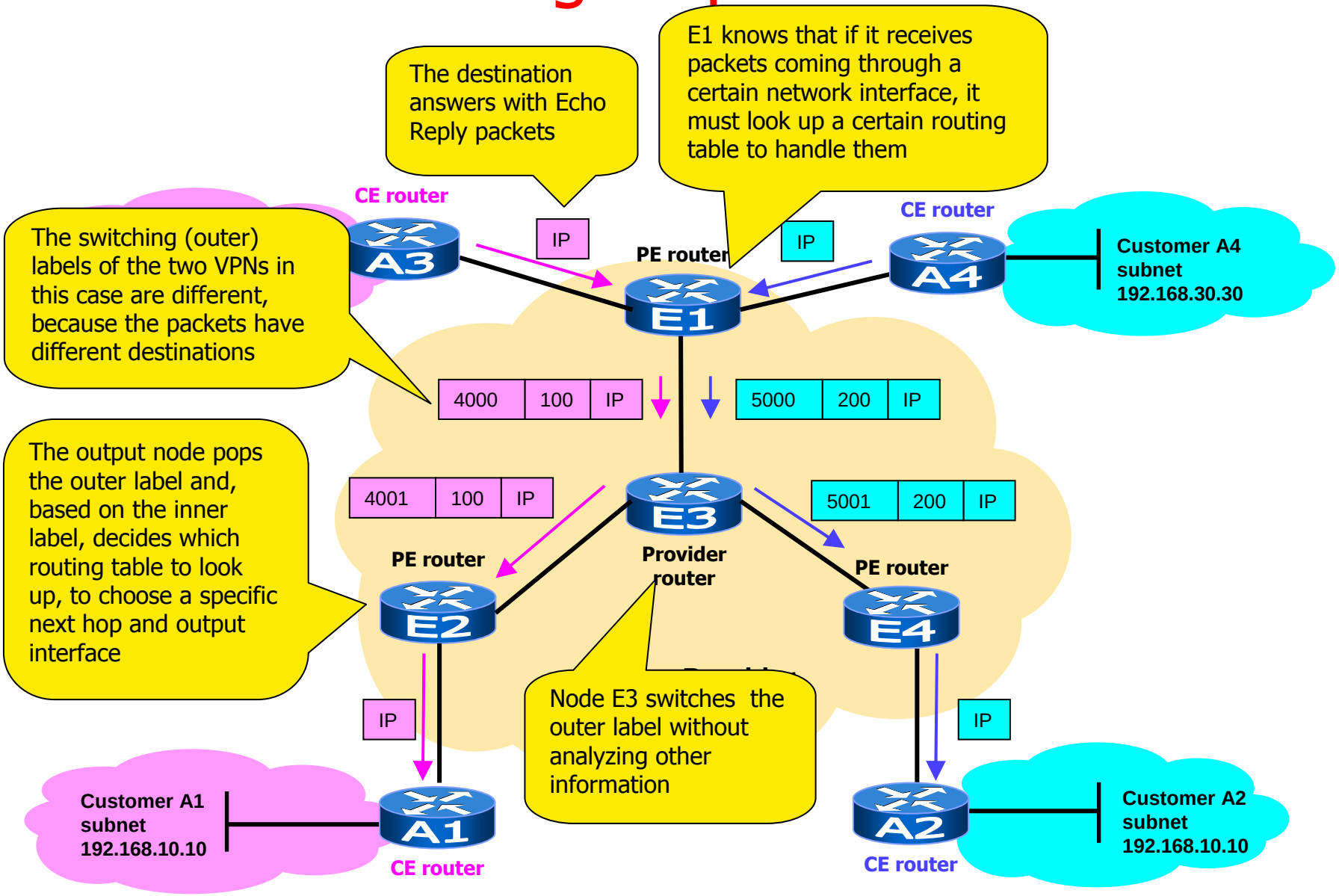
MPLS VPN routing sequence from A1 to A3



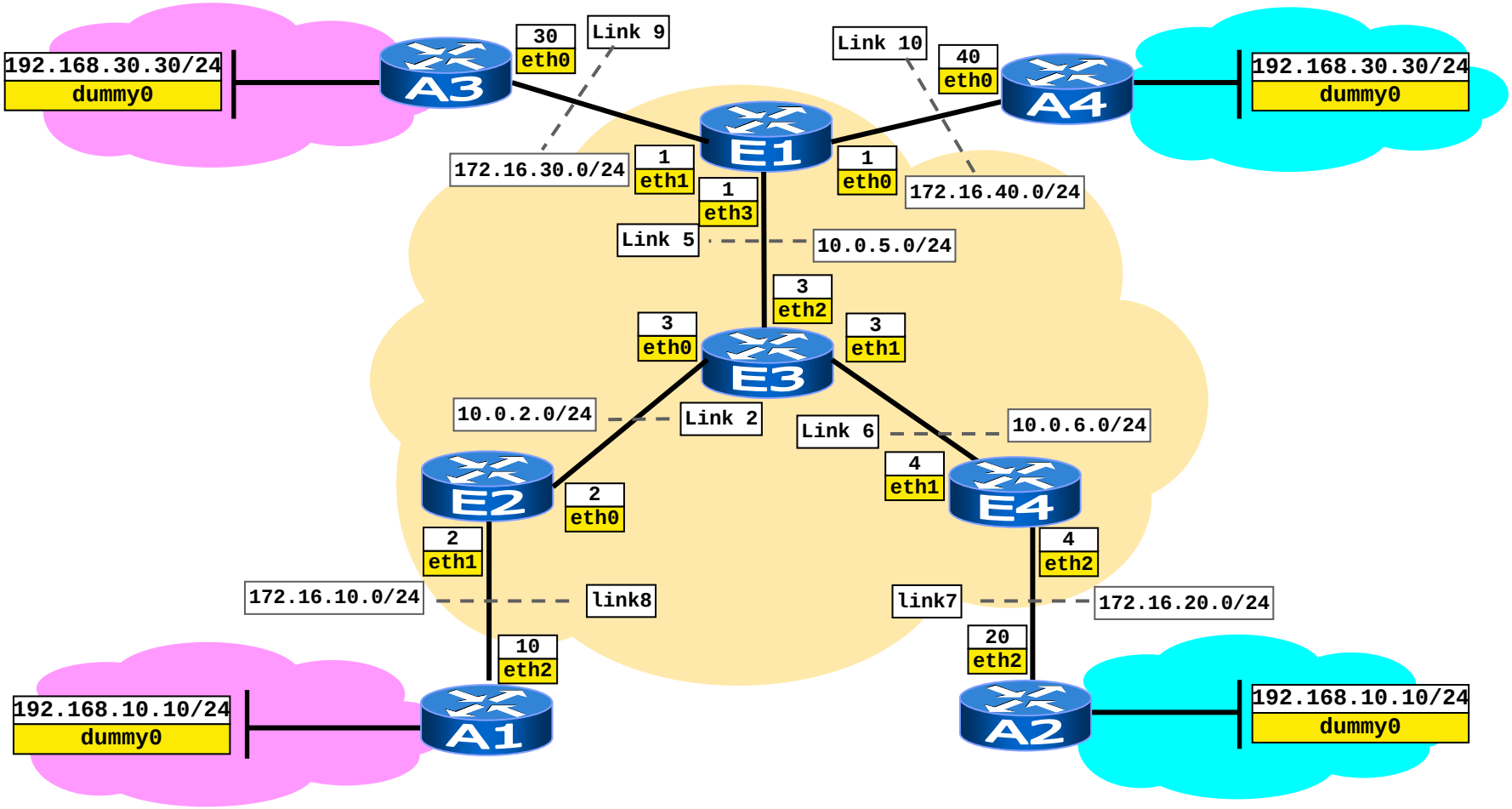
MPLS VPN routing sequence from A2 to A4



MPLS VPN routing sequence from A3 to A1



Network configuration



MPLS configuration: E2

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

ifconfig eth0 10.0.2.2 netmask 255.255.255.0 up
ifconfig eth1 172.16.10.2 netmask 255.255.255.0 up

echo 'A1->A3 VPN1'
#add labels 1000 and 100
key1=`mpls nhlfe add key 0 instructions push gen 1000 nexthop eth0 ipv4 10.0.2.3|grep key |cut -c
        17-26`
key2=`mpls nhlfe add key 0 instructions push gen 100 forward $key1|grep key|cut -c 17-26`
ip route add 172.16.30.0/24 via 10.0.2.3 table 1 mpls $key2
ip route add 192.168.30.0/24 via 10.0.2.3 table 1 mpls $key2

echo 'A3->A1 VPN1'
#for the return path remove the labels and populate the
mpls labelspace set dev eth0 labelspace 0
mpls ilm add label gen 4001 labelspace 0
mpls ilm add label gen 100 labelspace 0
```

Label 1000 is for MPLS label switching
Label 100 defines the VPN

VPN1 is mapped to routing table 1

MPLS configuration: E2

```
ip rule add from 172.16.30.0/24 table 1
ip rule add from 192.168.30.0/24 table 1
ip rule add from 172.16.10.0/24 table 1
ip rule add from 192.168.10.0/24 table 1

ip route add 172.16.10.0/24 dev eth1 table 1
ip route add 10.0.2.0/24 dev eth0 table 1
ip route add 192.168.10.0/24 dev eth1 via 172.16.10.10 table 1
```

MPLS configuration: E1

```
modprobe mpls4
modprobe mplsbr
modprobe mpls_tunnel

ifconfig eth0 172.16.40.1 netmask 255.255.255.0 up
ifconfig eth1 172.16.30.1 netmask 255.255.255.0 up
ifconfig eth3 10.0.5.1 netmask 255.255.255.0 up

#pop label 3000 from E3 incoming packets
mpls labelspace set dev eth3 labelspace 0
mpls ilm add label gen 3000 labelspace 0

#consult various routing tables if packets come from a specific destination
ip rule add from 172.16.10.0/24 table 1
ip rule add from 172.16.30.0/24 table 1
ip rule add from 172.16.20.0/24 table 2
ip rule add from 172.16.40.0/24 table 2
ip rule add from 192.168.30.0/24 dev eth1 table 1
ip rule add from 192.168.30.0/24 dev eth0 table 2
ip rule add from 192.168.10.0/24 table 2
ip rule add from 192.168.10.0/24 table 1
```

These statements emulate a VRF and will avoid overlapping

MPLS configuration: E1

```
echo "A1->A3 in VPN1"
#pop label 100 and send the packet directly to A3 (no routing is done)
mpls ilm add label gen 100 labelspace 0
key1=`mpls nhlfe add key 0 instructions nexthop eth1 ipv4 172.16.30.30| grep key| cut -c 17-26`
mpls xc add ilm_label gen 100 ilm_labelspace 0 nhlfe_key $key1

echo "A3->A1 in VPN1"
#add label 4000 and 100 for packets going to A1
var1=`mpls nhlfe add key 0 instructions push gen 4000 nexthop eth3  ipv4 10.0.5.3|grep key | cut -c
        17-26`
var2=`mpls nhlfe add key 0 instructions push gen 100 forward $var1 |grep key |cut -c 17-26`
#map a FEC to a NHLFE
ip route add 172.16.10.0/24 via 10.0.5.3 table 1 mpls $var2
ip route add 192.168.10.0/24 via 10.0.5.3 table 1 mpls $var2

echo "A2->A4 in VPN2"
#pop label 200 and send the packet directly to A4 (no routing is done)
mpls ilm add label gen 200 labelspace 0
key2=`mpls nhlfe add key 0 instructions nexthop eth0 ipv4 172.16.40.40| grep key| cut -c 17-26`
mpls xc add ilm_label gen 200 ilm_labelspace 0 nhlfe_key $key2
```

MPLS configuration: E1

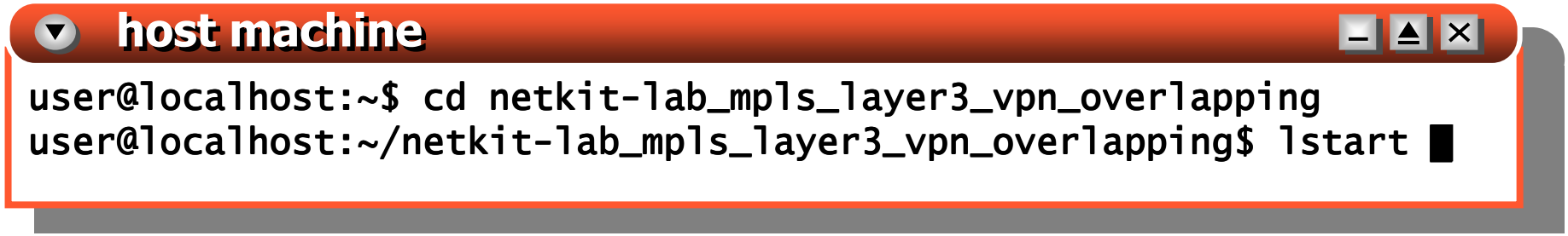
```
echo "A4->A2 in VPN2"
#add labels 5000 and 200 for packets going to A2
var1=`mpls nhlfe add key 0 instructions push gen 5000 nexthop eth3  ipv4 10.0.5.3|grep key | cut -c
        17-26`
var2=`mpls nhlfe add key 0 instructions push gen 200 forward $var1 |grep key |cut -c 17-26`
ip route add 172.16.20.0/24 via 10.0.5.3 table 2 mpls $var2
ip route add 192.168.10.0/24 via 10.0.5.3 table 2 mpls $var2

#populate routing tables
ip route add 172.16.30.0/24 dev eth1 table 1
ip route add 192.168.30.0/24 dev eth1 via 172.16.30.30 table 1
ip route add 10.0.5.0/24 dev eth3 table 1
ip route add 172.16.40.0/24 dev eth0 table 2
ip route add 192.168.30.0/24 dev eth0 via 172.16.40.40 table 2
ip route add 10.0.5.0/24 dev eth3 table 2
```

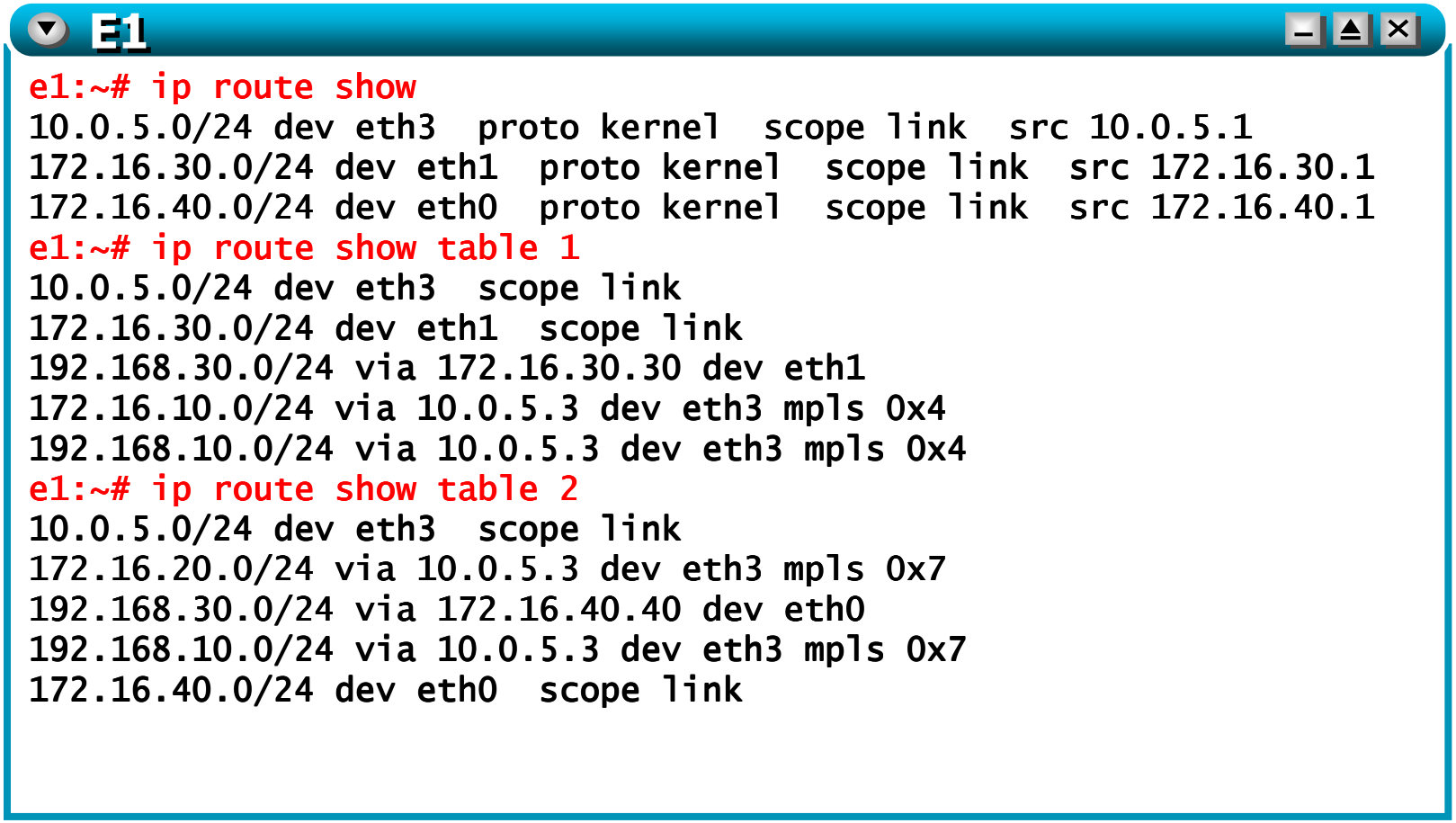
MPLS configuration: E4 – E3

- E4's configuration is very similar to E2's
- E3 just performs label switching, and joins traffic flows from E3 to E1 by labelling them with the same outer MPLS label

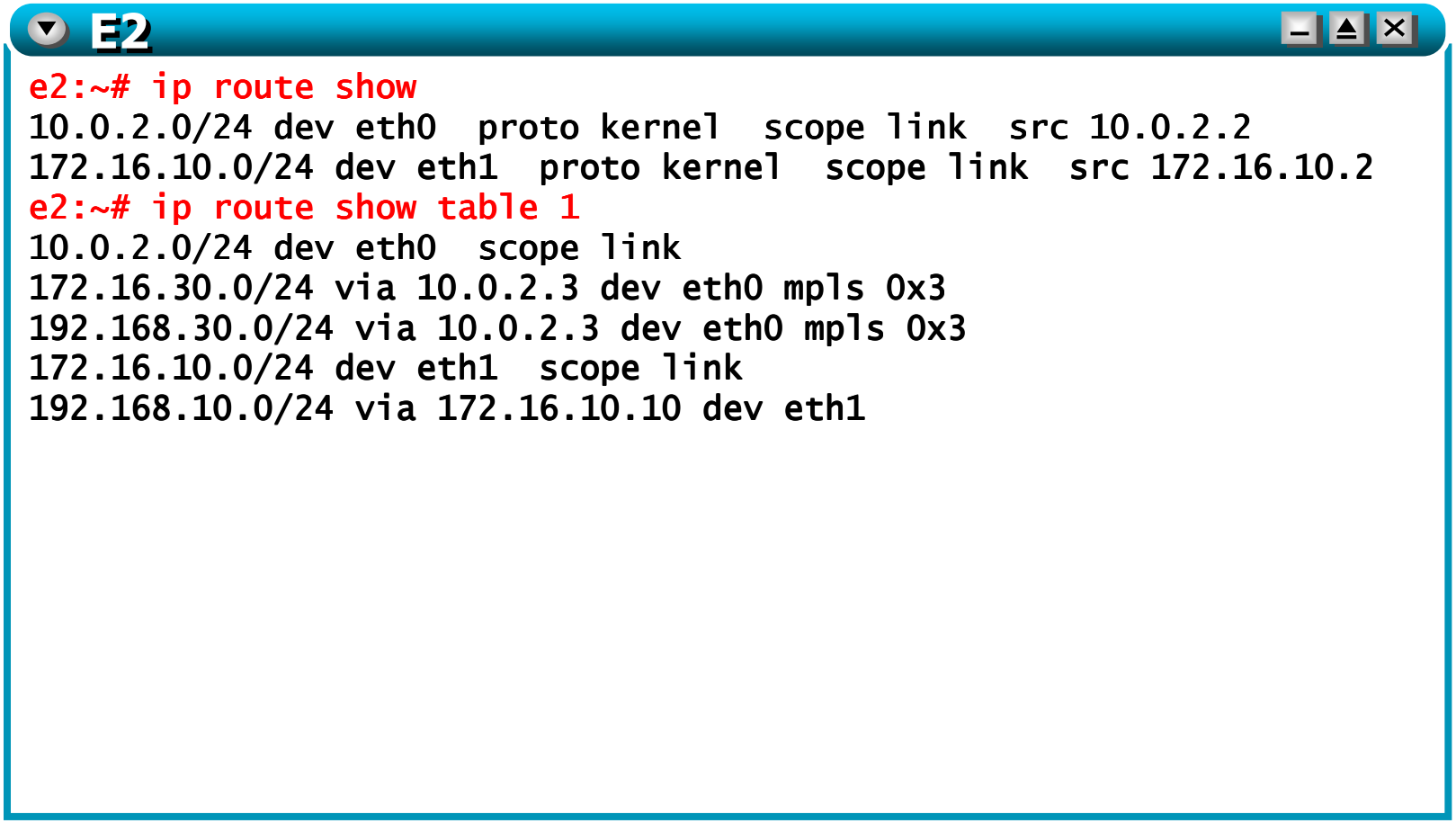
Starting the lab



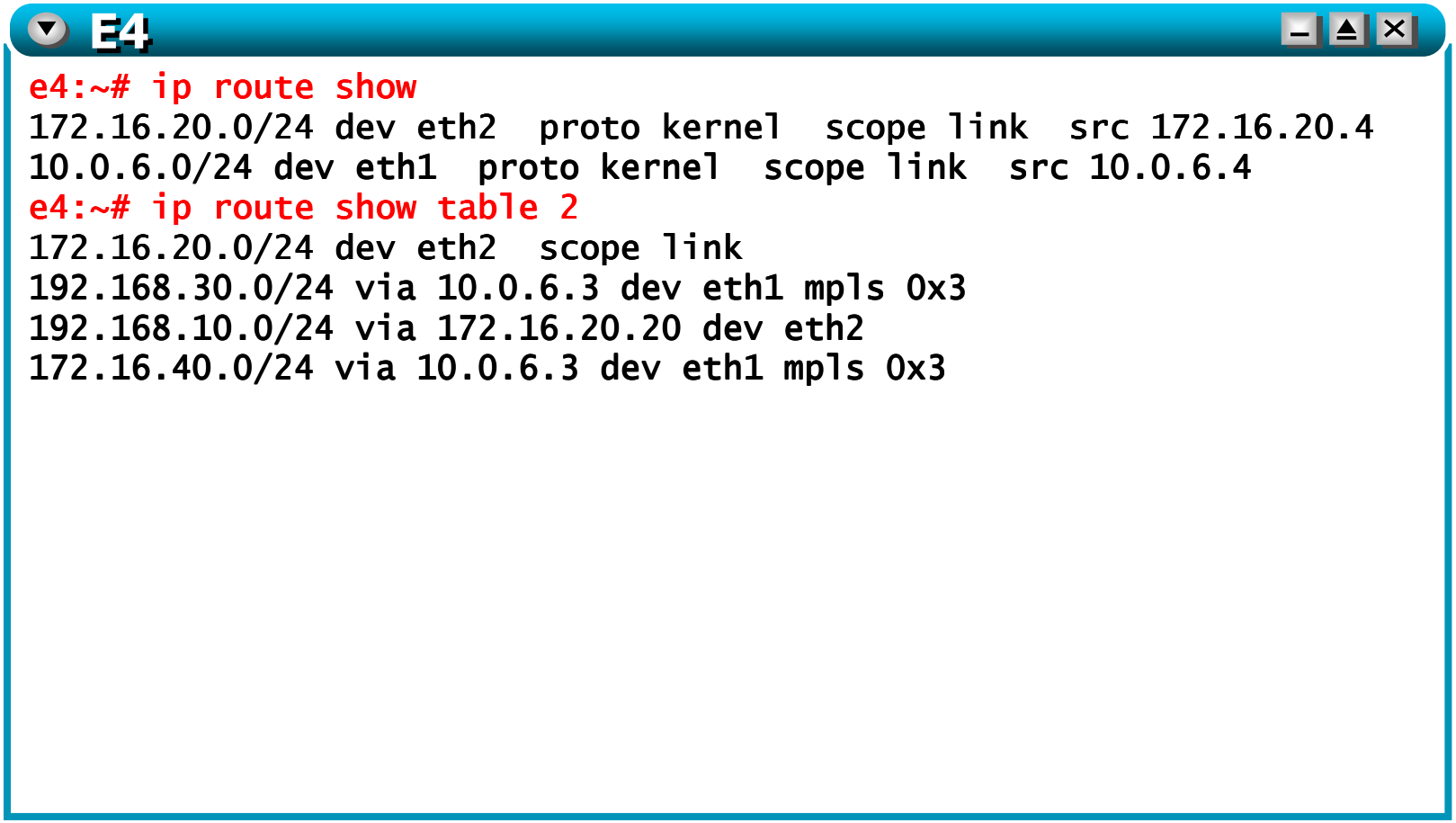
E1 Routing tables



E2 Routing tables

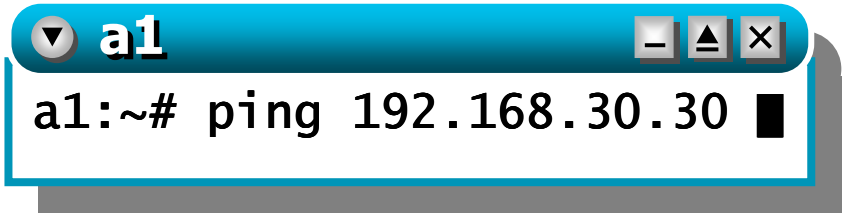


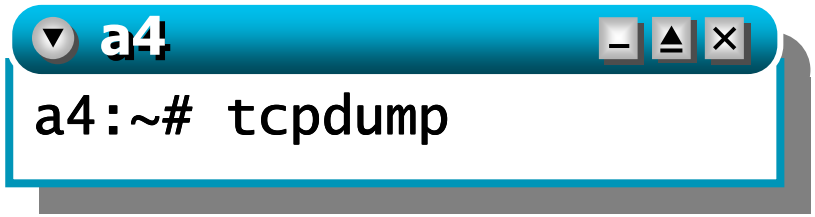
E4 Routing tables



Traffic analysis

Now we'll use VPN1 to ping A3 from A1, and we will observe what A3 and A4 receive





Traffic analysis

a4

```
a1:~# ping 192.168.30.30
PING 192.168.30.30 (192.168.30.30) 56(84) bytes of data.
64 bytes from 192.168.30.30: icmp_seq=1 ttl=61 time=34.3 ms
64 bytes from 192.168.30.30: icmp_seq=2 ttl=61 time=0.733 ms
64 bytes from 192.168.30.30: icmp_seq=3 ttl=61 time=0.605 ms
64 bytes from 192.168.30.30: icmp_seq=4 ttl=61 time=0.586 ms
^C
--- 192.168.30.30 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 0.586/9.078/34.390/14.614 ms
a1:~#
```

A1 sends and receives packets

a4

```
a4:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
a4:~#
```

A4 doesn't receive anything (although it has an interface 192.168.30.30) as it's supposed to be

Traffic analysis

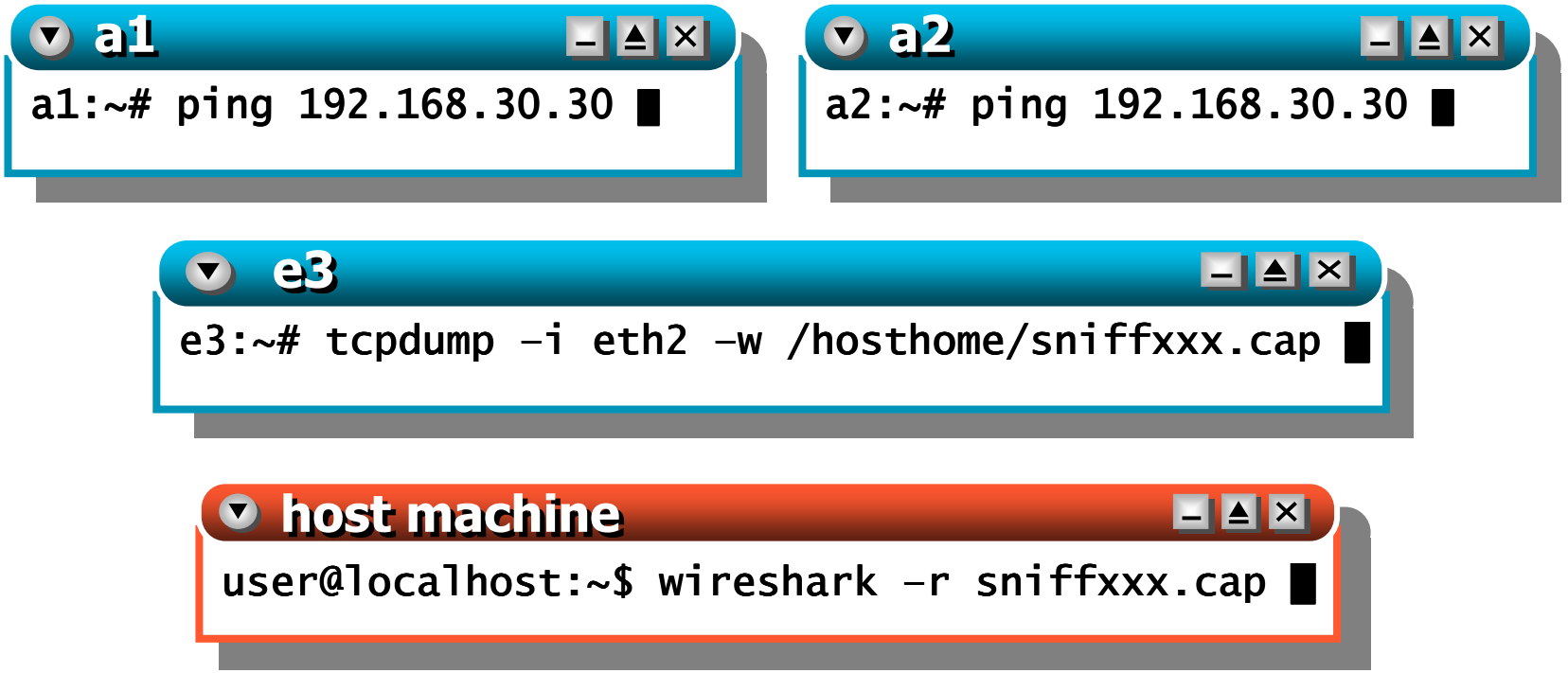
a3

```
a3:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
19:19:09.219770 IP 192.168.10.10 > 192.168.30.30: ICMP echo request, id 3074, seq 1, length 64
19:19:09.234325 arp who-has 172.16.30.1 tell 192.168.30.30
19:19:09.234446 arp reply 172.16.30.1 is-at 3e:56:9f:ce:79:c9 (oui Unknown)
19:19:09.234451 IP 192.168.30.30 > 192.168.10.10: ICMP echo reply, id 3074, seq 1, length 64
19:19:10.219546 IP 192.168.10.10 > 192.168.30.30: ICMP echo request, id 3074, seq 2, length 64
19:19:10.219566 IP 192.168.30.30 > 192.168.10.10: ICMP echo reply, id 3074, seq 2, length 64
19:19:11.219409 IP 192.168.10.10 > 192.168.30.30: ICMP echo request, id 3074, seq 3, length 64
19:19:11.219438 IP 192.168.30.30 > 192.168.10.10: ICMP echo reply, id 3074, seq 3, length 64
19:19:12.219399 IP 192.168.10.10 > 192.168.30.30: ICMP echo request, id 3074, seq 4, length 64
19:19:12.219426 IP 192.168.30.30 > 192.168.10.10: ICMP echo reply, id 3074, seq 4, length 64
19:19:14.213660 arp who-has 172.16.30.30 tell 172.16.30.1
19:19:14.213673 arp reply 172.16.30.30 is-at f2:dd:d9:20:19:6b (oui Unknown)
^C
12 packets captured
12 packets received by filter
0 packets dropped by kernel
a3:~#
```

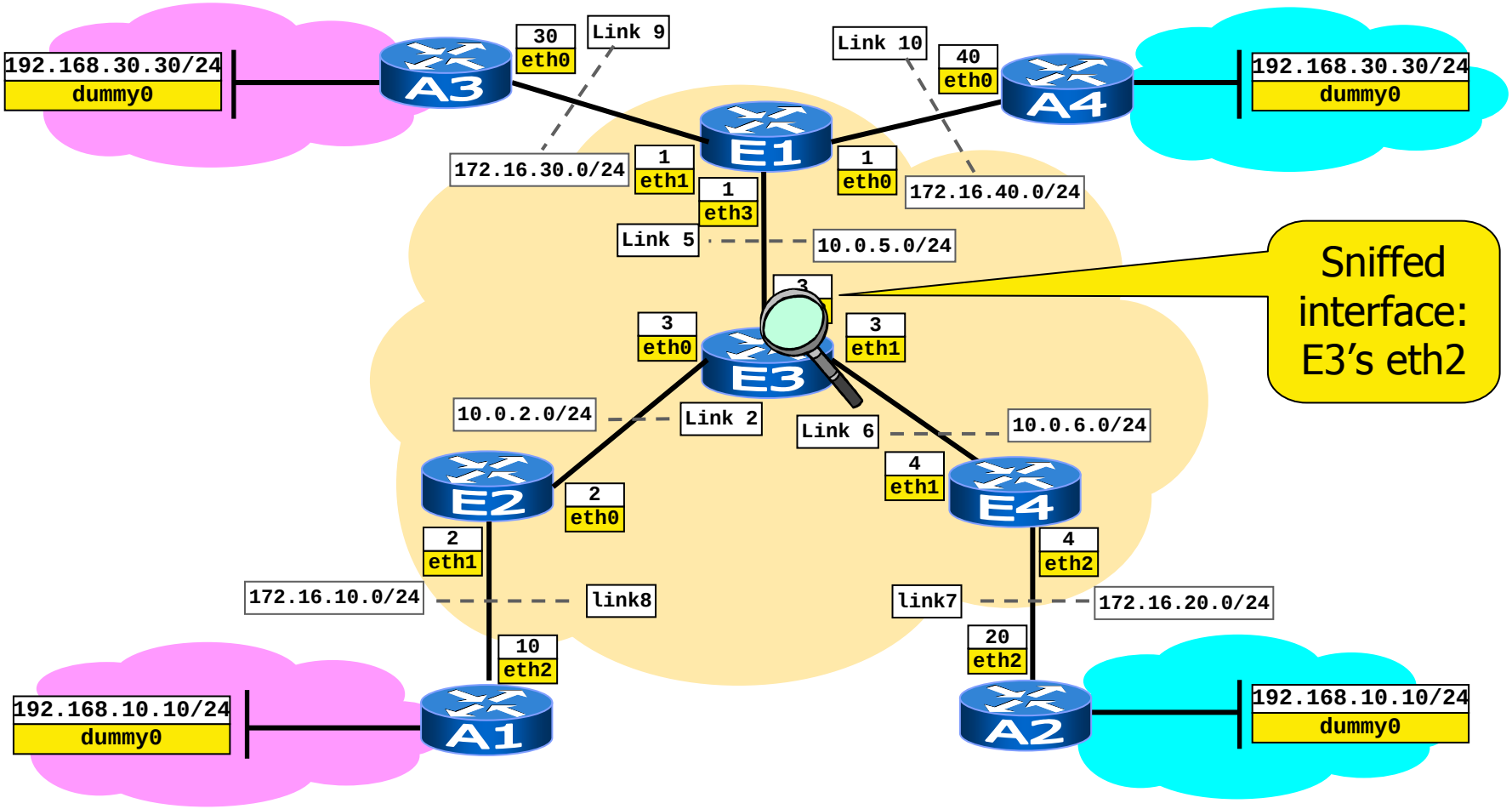
A3 receives packets correctly

Traffic analysis

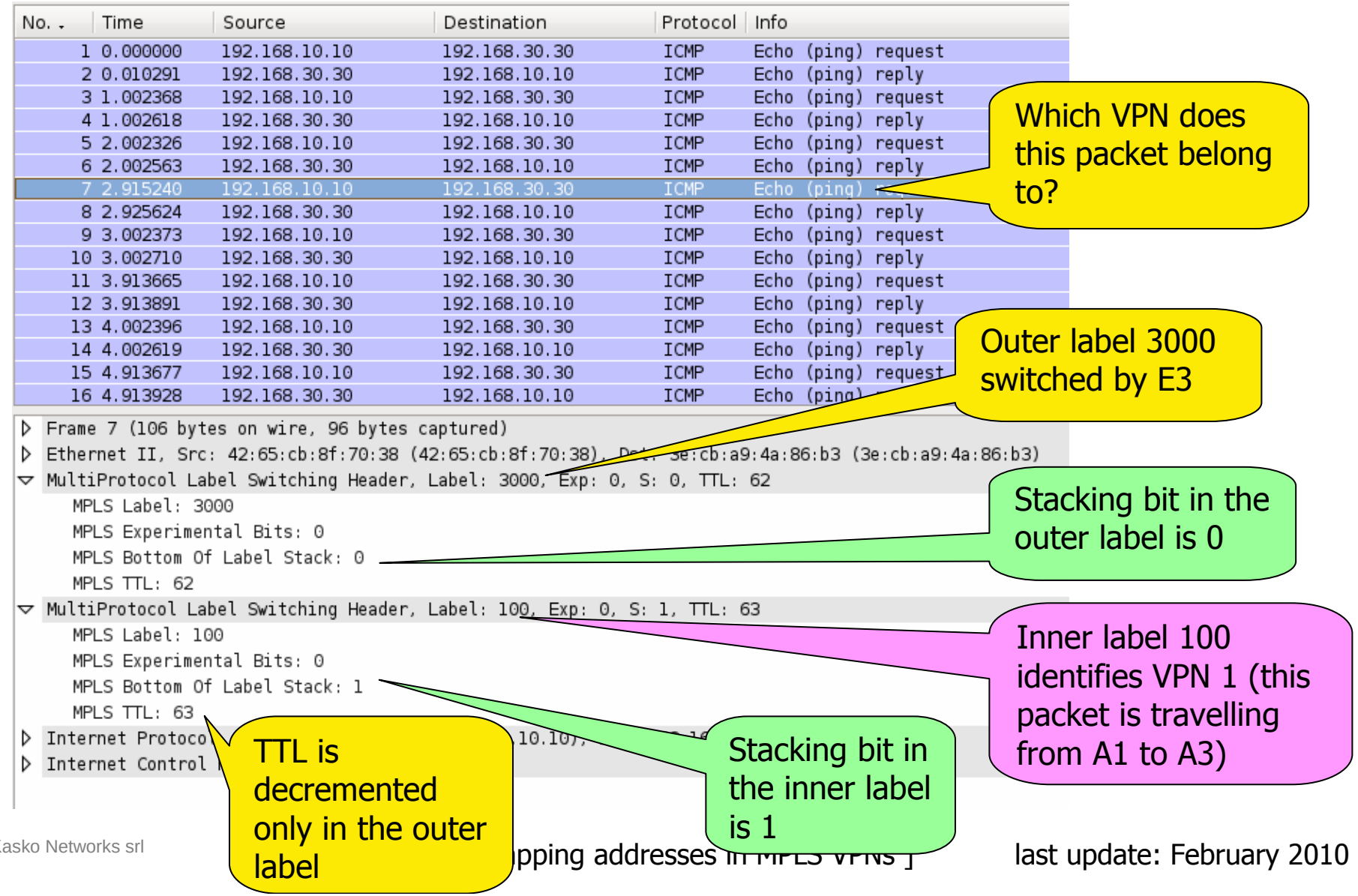
Now we will analyze traffic at E3's eth2 interface using wireshark on the host machine



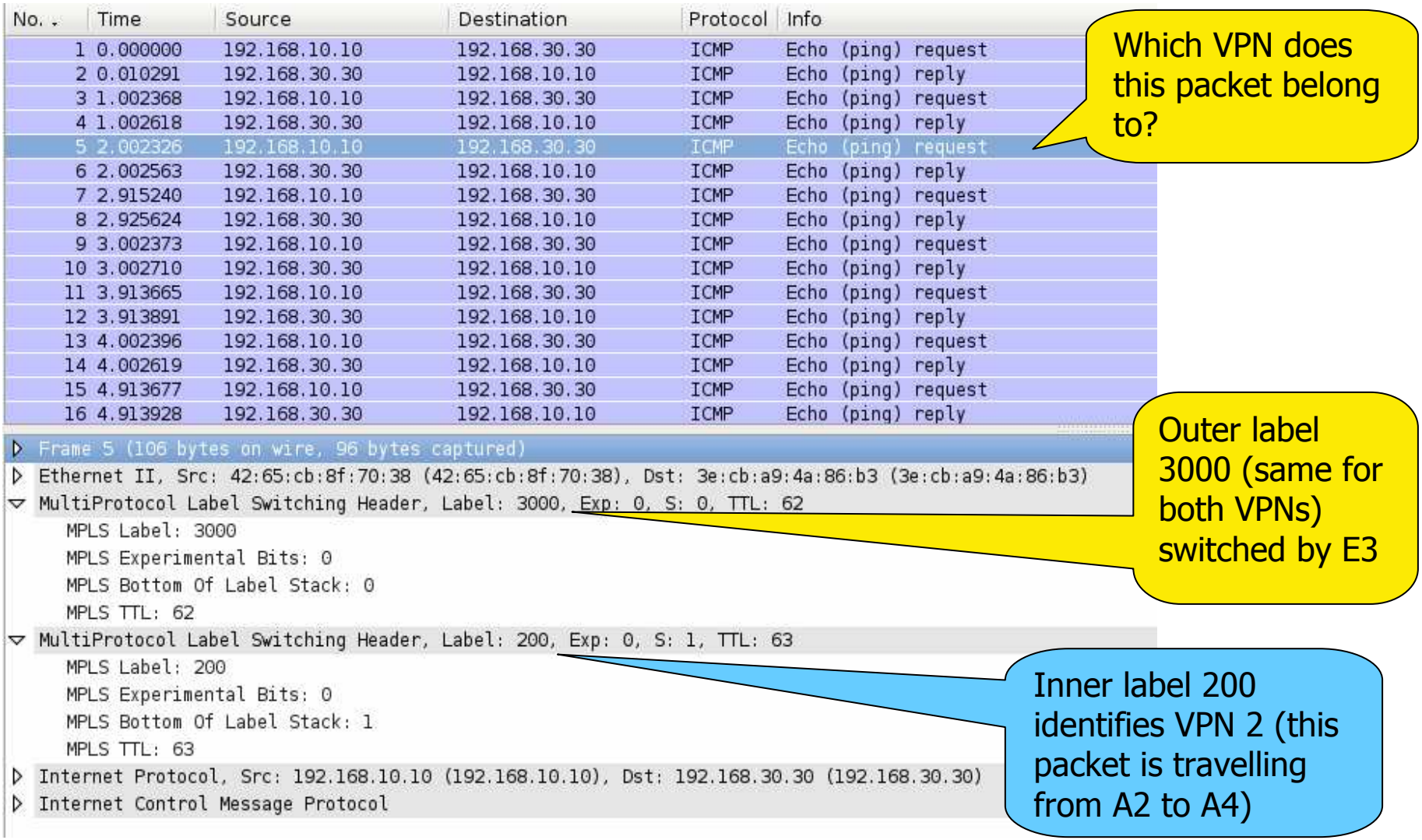
Traffic analysis



Traffic analysis on E3's eth2 interface



Traffic analysis on E3's eth2 interface



Traffic analysis on E3's eth2 interface

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
2	0.010291	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
3	1.002368	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
4	1.002618	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
5	2.002326	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
6	2.002563	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
7	2.915240	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
8	2.925624	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
9	3.002373	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
10	3.002710	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
11	3.913665	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
12	3.913891	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
13	4.002396	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
14	4.002619	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
15	4.913677	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
16	4.913928	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply

```
▷ Frame 8 (106 bytes on wire, 96 bytes captured)
▷ Ethernet II, Src: 3e:cb:a9:4a:86:b3 (3e:cb:a9:4a:86:b3), Dst: 42:65:cb:8f:70:38 (42:65:cb:8f:70:38)
▽ MultiProtocol Label Switching Header, Label: 4000, Exp: 0, S: 0, TTL: 63
    MPLS Label: 4000
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 0
    MPLS TTL: 63
▽ MultiProtocol Label Switching Header, Label: 100, Exp: 0, S: 1, TTL: 63
    MPLS Label: 100
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 63
▷ Internet Protocol, Src: 192.168.30.30 (192.168.30.30), Dst: 192.168.10.10 (192.168.10.10)
▷ Internet Control Message Protocol
```

Echo reply from A3 to A1, recognized by the outer label 4000…

…and inner label 100

Traffic analysis on E3's eth2 interface

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
2	0.010291	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
3	1.002368	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
4	1.002618	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
5	2.002326	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
6	2.002563	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
7	2.915240	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
8	2.925624	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
9	3.002373	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
10	3.002710	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
11	3.913665	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
12	3.913891	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
13	4.002396	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
14	4.002619	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply
15	4.913677	192.168.10.10	192.168.30.30	ICMP	Echo (ping) request
16	4.913928	192.168.30.30	192.168.10.10	ICMP	Echo (ping) reply

```
▷ Frame 4 (106 bytes on wire, 96 bytes captured)
▷ Ethernet II, Src: 3e:cb:a9:4a:86:b3 (3e:cb:a9:4a:86:b3), Dst: 42:65:cb:8f:70:38 (42:65:cb:8f:70:38)
▽ MultiProtocol Label Switching Header, Label: 5000, Exp: 0, S: 0, TTL: 63
    MPLS Label: 5000
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 0
    MPLS TTL: 63
▽ MultiProtocol Label Switching Header, Label: 200, Exp: 0, S: 1, TTL: 63
    MPLS Label: 200
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 63
▷ Internet Protocol, Src: 192.168.30.30 (192.168.30.30), Dst: 192.168.10.10 (192.168.10.10)
▷ Internet Control Message Protocol
```

Echo reply from A4 to A2, recognized by the outer label 5000…

…and inner label 200