MPLS VPNs with overlapping address spaces

<table>
<thead>
<tr>
<th>Version</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>S.Filippi, L.Ricci, F.Antonini</td>
</tr>
<tr>
<td>E-mail</td>
<td><a href="mailto:silvia.filippi@kaskonetworks.it">silvia.filippi@kaskonetworks.it</a></td>
</tr>
<tr>
<td>Web</td>
<td><a href="http://www.kaskonetworks.it/">http://www.kaskonetworks.it/</a></td>
</tr>
<tr>
<td>Description</td>
<td>A lab showing how a Layer 3 MPLS VPN works in case of customer addresses space overlapping</td>
</tr>
</tbody>
</table>
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 between 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 customer, 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
What could happen if CE routers advertised their subnets inside the service provider network?

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

1. Routing protocols advertise CEA1’s subnet to the provider network

Customer A1’s edge router

Provider edge router 1

Service Provider network

Provider edge router 2

Customer A2’s edge router

2. Routing protocols advertise CEA2’s subnet to the provider network

Customer edge A1 subnet

192.168.10.10/24

Customer A3’s edge router

CEA3

Customer edge A2 subnet

192.168.10.10/24

CEA1

Customer edge A1 subnet

192.168.10.10/24

CEA2

Customer edge A2 subnet

192.168.10.10/24

PE1

PE2

3. Which is the correct route to 192.168.10.10/24?
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 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.

Network topology

VPN1 – label 100

Customer A1 subnet 192.168.10.10

Customer A3 subnet 192.168.30.30

VPN2 – label 200

Customer A2 subnet 192.168.10.10

Customer A4 subnet 192.168.30.30

Service Provider network
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 identified by inner label 100
- VPN2 will be identified by inner label 200
MPLS VPN routing sequence from A1 to A3

E1 pops the top label and reads the inner label. If this label is 100, then routing table 1 will be looked up; otherwise, if the label is 200, then routing table 2 will be looked up. This label is then also popped.

E3 switches the outer label and aggregates VPN1 and VPN2 flows (same outer label), because they go to the same destination (E1).

The ingress PE checks its routing table, classifies the packet as belonging to customer A1, and adds inner label 100 to identify VPN1 and outer label 1000 to perform switching.

A1 sends an ICMP Echo Request to A3 in the pink VPN.

Customer A1 subnet 192.168.10.10

Customer A2 subnet 192.168.10.10

Customer A3 subnet 192.168.30.30

Customer A4 subnet 192.168.30.30
MPLS VPN routing sequence from A2 to A4

The ingress PE checks its routing table, classifies the packet as belonging to customer A1, and adds inner label 100 to identify VPN1 and outer label 1000 to perform switching.

E3 switches the outer label and aggregates VPN1 and VPN2 flows (same outer label), because they go to the same destination (E1).

A1 sends an ICMP Echo Request to A3 in the pink VPN.

The same happens within VPN A2.

E1 pops the top label and reads the inner label. If this label is 100, then routing table 1 will be looked up; otherwise, if the label is 200, then routing table 2 will be looked up. This label is then also popped.
MPLS VPN routing sequence from A3 to A1

The destination answers with Echo Reply packets.

E1 knows that if it receives packets coming through a certain network interface, it must look up a certain routing table to handle them.

The switching (outer) labels of the two VPNs in this case are different, because the packets have different destinations.

The output node pops the outer label and, based on the inner label, decides which routing table to look up, to choose a specific next hop and output interface.

Node E3 switches the outer label without analyzing other information.

Customer A3 subnet 192.168.30.30
Customer A4 subnet 192.168.30.30
Customer A1 subnet 192.168.10.10
Customer A2 subnet 192.168.10.10
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 routing table
mpls ilm add label gen 4001 labelspace 0
mpls ilm add label gen 100 labelspace 0

VPN1 is mapped to routing table 1
Label 1000 is for MPLS label switching
Label 100 defines the VPN
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

```
user@localhost:~$ cd netkit-lab_mpls_layer3_vpn_overlapping
user@localhost:~/netkit-lab_mpls_layer3_vpn_overlapping$ lstart
```
E1 Routing tables

```
e1:~# ip route show
ten 0.5.0/24 dev eth3  proto kernel  scope link  src 10.0.5.1
172.16.30.0/24 dev eth1  proto kernel  scope link  src 172.16.30.1
172.16.40.0/24 dev eth0  proto kernel  scope link  src 172.16.40.1

e1:~# ip route show table 1
10.0.5.0/24 dev eth3  scope link
172.16.30.0/24 dev eth1  scope link
192.168.30.0/24 via 172.16.30.30 dev eth1
172.16.10.0/24 via 10.0.5.3 dev eth3 mpls 0x4
192.168.10.0/24 via 10.0.5.3 dev eth3 mpls 0x4

e1:~# ip route show table 2
10.0.5.0/24 dev eth3  scope link
172.16.20.0/24 via 10.0.5.3 dev eth3 mpls 0x7
192.168.30.0/24 via 172.16.40.40 dev eth0
192.168.10.0/24 via 10.0.5.3 dev eth3 mpls 0x7
172.16.40.0/24 dev eth0  scope link
```

© Kasko Networks srl

netkit – [ Overlapping addresses in MPLS VPNs ]  
last update: February 2010
## E2 Routing tables

<table>
<thead>
<tr>
<th>Route Details</th>
<th>Table Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.2</td>
<td>10.0.2.0/24 scope link</td>
</tr>
<tr>
<td>172.16.10.0/24 dev eth1 proto kernel scope link src 172.16.10.2</td>
<td>172.16.10.0/24 via 172.16.10.10 dev eth1</td>
</tr>
<tr>
<td>192.168.30.0/24 via 10.0.2.3 dev eth0 mpls 0x3</td>
<td></td>
</tr>
<tr>
<td>192.168.30.0/24 via 10.0.2.3 dev eth0 mpls 0x3</td>
<td></td>
</tr>
<tr>
<td>172.16.10.0/24 dev eth1 scope link</td>
<td></td>
</tr>
<tr>
<td>192.168.10.0/24 via 172.16.10.10 dev eth1</td>
<td></td>
</tr>
</tbody>
</table>
E4 Routing tables

```
e4:~# ip route show
172.16.20.0/24 dev eth2 proto kernel scope link src 172.16.20.4
10.0.6.0/24 dev eth1 proto kernel scope link src 10.0.6.4

e4:~# ip route show table 2
172.16.20.0/24 dev eth2 scope link
192.168.30.0/24 via 10.0.6.3 dev eth1 mpls 0x3
192.168.10.0/24 via 172.16.20.20 dev eth2
172.16.40.0/24 via 10.0.6.3 dev eth1 mpls 0x3
```
Traffic analysis

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

```
a1:~# ping 192.168.30.30
```

```
a3:~# tcpdump
```
```
a4:~# tcpdump
```
Traffic analysis

A1 sends and receives packets

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

```
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
```

A3 receives packets correctly
Traffic analysis

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

```
a1:~# ping 192.168.30.30

a2:~# ping 192.168.30.30

e3:~# tcpdump -i eth2 -w /hosthome/sniffxxx.cap

user@localhost:~$ wireshark -r sniffxxx.cap
```
Traffic analysis
Traffic analysis on E3’s eth2 interface

Which VPN does this packet belong to?

- Inner label 100 identifies VPN 1 (this packet is travelling from A1 to A3)
- Outer label 3000 switched by E3
- Stacking bit in the inner label is 1
- Stacking bit in the outer label is 0
- TTL is decremented only in the outer label
Traffic analysis on E3’s eth2 interface

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0000000</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>2</td>
<td>0.010291</td>
<td>192.168.30.30</td>
<td>192.168.10.10</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>3</td>
<td>1.002368</td>
<td>102.168.10.10</td>
<td>102.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>4</td>
<td>1.002618</td>
<td>192.168.30.30</td>
<td>192.168.10.10</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
</tbody>
</table>

Frame 5 (116 bytes on wire, 96 bytes captured)


MultiProtocol Label Switching Header, Label: 3000, Exp: 0, S: 0, TTL: 62
- MPLS Label: 3000
- MPLS Experimental Bits: 0
- MPLS Bottom Of Label Stack: 0
- MPLS TTL: 62

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 Control Message Protocol

Outer label 3000 (same for both VPNs) switched by E3

Inner label 200 identifies VPN 2 (this packet is travelling from A2 to A4)

Which VPN does this packet belong to?
Traffic analysis on E3’s eth2 interface

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

- Frame 8 (105 bytes on wire, 96 bytes captured)
- 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 Control Message Protocol
Traffic analysis on E3’s eth2 interface

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>2</td>
<td>0.0010291</td>
<td>192.168.30.30</td>
<td>192.168.10.10</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>3</td>
<td>1.002368</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>4</td>
<td>4.002610</td>
<td>192.168.30.30</td>
<td>192.168.10.10</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>5</td>
<td>2.002326</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>6</td>
<td>2.002563</td>
<td>192.168.30.30</td>
<td>192.168.10.10</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>7</td>
<td>2.915240</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>8</td>
<td>2.925624</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>9</td>
<td>3.002373</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>10</td>
<td>3.002710</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>11</td>
<td>3.913565</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>12</td>
<td>3.913981</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>13</td>
<td>4.002396</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>14</td>
<td>4.002619</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
<tr>
<td>15</td>
<td>4.913677</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) request</td>
</tr>
<tr>
<td>16</td>
<td>4.913928</td>
<td>192.168.10.10</td>
<td>192.168.30.30</td>
<td>ICMP</td>
<td>Echo (ping) reply</td>
</tr>
</tbody>
</table>

Frame 4 (106 bytes on wire, 96 bytes captured)

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 Control Message Protocol

© Kasko Networks srl