Configuration des matériels du réseau

Dans la plupart des tests, j'ai utilisé sept à huit machines. Voici une liste des noms de ces machines avec leurs cartes réseaux correspondantes. L'ordre des cartes dans le tableau correspond bien à celui qui se trouve physiquement sur la boite de la machine de **haut en bas** - je mentionne des informations sur les cartes que j'ai rencontré des problèmes avec ou bien que je l'ai installées. Pour les autres cartes qui ont déjà en état de fonctionnement quand j'ai configuré mon réseau, je l'ai laissées comme telles. L'alias entre le module d'une carte et son nom se trouve dans le fichier « /etc/modules.conf ».

Tantale
Eth1 – carte PCI 3Com (10M) - module à charger c'est 3c-59x
Eth0 – carte PCI 3Com (10M) - module à charger c'est 3c-59x
Eth2 – carte intégrée avec la carte mère (100M)

Troll
Eth2 – (10M)
Eth0 – (10M)
Eth1 – carte intégrée avec la carte mère (100M)

Leone
Eth1 – (10M)
Eth2 – carte ISA 3Com (10M) - module à charger c'est 3c-509
Eth0 – carte intégrée avec la carte mère (100M)

Rigel
Eth1 – carte PCI 3Com (10M) - module à charger c'est 3c-59x
Eth0 – carte PCI 3Com (10M) - module à charger c'est 3c-59x
Eth2 – carte intégrée avec la carte mère (100M)

Cook
Eth3 – carte ISA Smc-ultra (10M) - module à charger c'est smc-ultra
Eth2 – carte PCI 3Com (10M) - module à charger c'est 3c-59x
Eth0 – (10M)
Eth1 – carte intégrée avec la carte mère (100M)

Popy
Eth1 – carte ISA Smc-ultra (10M) - module à charger c'est smc-ultra
Eth0 – carte intégrée avec la carte mère (100M)

Montrachet
Eth1 – carte ISA Smc-ultra (10M) - module à charger c'est smc-ultra - **il y a un problème avec cette carte** – elle envoie des paquets mais elle ne les reçoit
Eth0 – carte intégrée avec la carte mère (100M)

Centropyge
Eth1 – carte ISA Smc-ultra (10M) - module à charger c'est smc-ultra – **il y a un probleme avec cette carte**
Eth0 – carte intégrée avec la carte mère (100M)

L'installation d'un module se fait avec la commande : « modprobe nom_du_module » qui permet de charger en mémoire un module ainsi que tous les modules dont il dépend. Le nom d'un module d'une carte se trouve sur la plus large puce de cette carte. Après, on doit taper « depmod –a » qui remet à jour le fichier modules.dep, en fonction du ou des nouveaux modules. Pour afficher la liste des modules chargés, les dépendances entre les modules chargés, on peut utiliser la commande « lsmod ». Une fois on a installé un module, on doit ajouter ça au fichier « /etc/modules.conf » pour éviter qu'à chaque fois on démarre notre machine de le réinstaller pour cette carte. Comme exemple, la syntaxe sera la suivante pour une carte 3Com qui est reconnu par le noyau sous l'interface zéro :
« alias eth0 3c59x ».

Enfin de connecter nos machines par des liaisons point à point, on a utilisé soit des câbles croisés, soit des VLans.
Avec un câble croisé, on peut connecter directement une interface d'une machine avec une autre d'une autre machine. Une autre façon d'établir une liaison point à point au moyen des câbles croisés – notamment les courts - c'est par la connexion d'un câble croisé partant d'une interface avec un autre droit partant d'une autre interface au moyen d'un adaptateur femelle femelle. On peut établir ça aussi par la connexion de deux câbles droits par un adaptateur femelle femelle.

L’autre manière d’établir des liaisons point à point c’était au moyen des Lans virtuels. En fait, on a configuré des lans virtuels entre les portes adjacentes de switch qu’on a disposé - switch fws-u2-1. Les vlans entre les portes sont configurées de la façon suivante :

*portes 23 et 24
*portes 21 et 22
*portes 19 et 20
*portes 17 et 18
*portes 15 et 16
*portes 12, 13 et 14
*portes 9, 10 et 11.

Les autres portes restantes sont connectées au réseau interne de l’Irisa.

Configurations logicielles du réseau

Tout d’abord, le mot de passe pour ‘root’ sur toutes les machines c’est : **rootroot**

Le hostname

Pour changer le nom d’une machine, on peut procéder avec la commande « hostname nom_machine ». Après, pour sauvegarder le changement du nom, il faut ajouter cette commande dans le fichier « /etc/sysconfig/network » (sur Redhat), et « /etc/hostname » (sur Debian).

L'adressage IP

Pour connaître des informations à propos des interfaces d’une machine, une commande à taper c’est «ifconfig » qui va lister des informations détaillées sur les interfaces reconnues par le noyau. Cette commande va afficher quelque chose comme ça :

```
root@leone:~# ifconfig
eth0      Lien encap:Ethernet  HWaddr 00:C0:4F:17:80:12
          inet adr:131.254.200.20  Bcast:131.254.255.255
Masque:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:134783 errors:0 dropped:0 overruns:0 frame:0
          TX packets:123012 errors:0 dropped:0 overruns:0 carrier:9
          collisions:0 lg file transmission:100
          RX bytes:25909866 (24.7 MiB)  TX bytes:32930086 (31.4 MiB)
          Interruption:14 Adresse de base:0xcc00
```

```
eth1      Lien encap:Ethernet  HWaddr 00:A0:24:75:99:4A
          inet adr:10.0.23.2  Bcast:10.0.23.255  Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:224187 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10920 errors:0 dropped:0 overruns:0 carrier:0
          collisions:582 lg file transmission:100
          RX bytes:75964863 (72.4 MiB)  TX bytes:4009473 (3.8 MiB)
          Interruption:11 Adresse de base:0xd8e0

eth2      Lien encap:Ethernet  HWaddr 00:60:97:39:4D:A9
          inet adr:10.0.3.2  Bcast:10.0.3.255  Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3454435 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2525804 errors:0 dropped:0 overruns:0 carrier:2
          collisions:66 lg file transmission:100
          RX bytes:208137776 (198.4 MiB)  TX bytes:119123234 (113.6
MiB)
          Interruption:5 Adresse de base:0x240

lo        Lien encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:84992 errors:0 dropped:0 overruns:0 frame:0
          TX packets:84992 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:14335516 (13.6 MiB)  TX bytes:14335516 (13.6 MiB)
```

Les informations fournies par la commande :

eth0,eth1, lo...	nom de l'interface ; le mot « lo » tient pour l'interface locale.
HWaddr	adresse MAC de carte
inet adr	adresse IP liée de l'interface
Bcast	adresse de broadcast
Masque	masque de réseau
UP	état de l'interface s'elle est active – l'autre cas c'est DOWN, c.à.d. interface non active.
MTU	taille maximum des trames physiques- Maximum Transmitted Unit.
RX packets	nombre de paquets reçues, perdus ou non reçus à cause d'erreur ou de débordements (arrivent de manière trop rapide pour pouvoir être traités par le noyau).
TX packets	nombre de paquets transmis, perdus ou non reçus à destination à cause d'erreurs ou de débordements.
collisions	La collision se produit lorsque 2 machines émettent en même temps sur le réseau. Il n'y a pas de souci tant que le rapport « nombre de paquets total / nombre de collisions » reste inférieur à 30%.

Pour afficher toutes les interfaces actives et non actives, il suffit de taper « ifconfig –a » – a tient pour 'all'. Avec la commande ifconfig, on peut aussi configurer les adresses IP d'une interface. Par exemple, si on souhaite configurer l'interface eth0 avec l'adresse

10.0.2.1 sur le réseau de classe C 10.0.2.0, et possédante comme adresse de broadcast le 10.0.2.255, on peut procéder comme le suivant :

```
ifconfig eth0 10.0.2.1 netmask 255.255.255.0 broadcast 10.0.2.255
```

Par défaut, l'exécution de cette commande va mettre l'interface dans l'état actif. Si on désire éteindre cette interface, on doit taper : « ifconfig eth0 down ». Pour revenir à l'état actif de l'interface, on doit alors taper « ifconfig eth0 up ». On remarque dans ce cas qu'après l'extinction d'une interface, puis après l'activation, les règles de routage qui ont été auparavant associées à cette interface sont disparues. Le seul moyen de recharger ces règles au lieu de les retaper, c'est de les sauvegarder dans un fichier de routage – on va parler de ces fichiers après – et ensuite, lancer un script de redémarrage du réseau.

Pour la configuration des tables de routage, on peut procéder soit avec la commande « ip » soit avec la commande « route ».

La syntaxe de la commande « ip » est la suivante :

```
ip r [ a| d ] [-host|-net [netmask]] via gateway
```

où « r » tient pour route, « a » pour ajouter (add) , « d » pour supprimer (delete), « s » pour afficher (show), « host » si la destination de la route est une machine, « net » si la destination de la route est un réseau, « gateway » peut désigner l'adresse IP ou le nom de machine de la passerelle. On peut aussi spécifier l'interface où on veut appliquer cette règle par l'ajout à la ligne précédente « dev ethx ». Par exemple, si on désire ajouter une route sur une interface eth0 pour le réseau Classe C 10.0.2.0 via la passerelle définit par x.y.z.t, alors, on peut procéder avec :

```
ip r a 10.0.2.0/24 via x.y.z.t dev eth0
```

Si on désire ajouter une passerelle par défaut pour toute destination qui ne dispose pas d'une entrée dans la table de routage, on peut alors spécifier le mot « défaut » dans l'adresse de destination :

```
ip r a default via x.y.z.t
```

On peut aussi configurer une interface avec « ip » au lieu de « ifconfig ». La syntaxe est la suivante :

```
ip a [ a | d ] ip_interface dev ethx
```

la première lettre « a » tient pour adresse, la deuxième pour ajouter, la « d » pour supprimer. On peut aussi afficher les tables de routage avec « ip » en tapant :

```
ip r s
```

L'utilisation de la commande « route » pour gérer les tables de routage est pareille à celle de « ip ». La syntaxe de la commande est :

```
route [add|del] [-host|-net [netmask]] gw gateway dev [ethx]
```

Si on revient à l'exemple précédent, l'ajout de la route avec « route » sera :

```
route add -net 10.0.2.0/24 gw x.y.z.t dev eth0
```

L'ajout d'une passerelle pour les destinations inconnues sera :

```
route add default gw x.y.z.t dev eth0
```

Ces configurations qu'on vient de décrire sont à retaper chaque fois qu'on démarre nos PCs ou qu'on active une interface éteinte. Pour débarrasser de ce problème, il faut les sauvegarder dans des fichiers de configurations. Le nom, la location et la syntaxe de ces fichiers dépendent de la distribution de Linux.

Pour les machines qui tournent RedHat comme système d'exploitation, les fichiers de configuration des interfaces se situent dans « /etc/sysconfig/network-scripts ». Pour l'interface nommée ethx, le nom du fichier sera « ifcfg-ethx ». La syntaxe du fichier est la suivante :

```
DEVICE=ethx
IPADDR=x.y.z.t
NETMASK=255.255.255.0
NETWORK= x.y.z.0
BROADCAST= x.y.z.255
ONBOOT=yes
```

Pour la configuration des tables de routage, il faut les éditer dans le fichier « /etc/sysconfig/static-routes ». La syntaxe de ce fichier et d'autre fichiers de configuration du réseau sur RedHat se trouve dans la répertoire « ../linux/Documentation/networking/ ». Le suivant est un exemple de la syntaxe de ce fichier sur une de mes machines :

```
eth2 net 10.0.20.0 netmask 255.255.255.0 gw 10.0.22.2
eth0 net 224.1.1.1 netmask 255.255.255.255 gw 10.0.24.2
```

Une remarque à considérer ici est que ces fichiers (celles des configurations des interfaces et l'autre pour les routes) doivent être exécutables pour qu'ils puissent être initialisés par un script du réseau. Si ce n'est pas le cas, on doit alors procéder à changer le flag d'exécution du fichier en tapant « chmod a+x nom_fichier ».
Une fois on a édité ces fichiers, on peut alors recharger nos configurations en tournant un script de démarrage du réseau comme « /etc/init.d/network ».
Un mot encore pour les machines RedHat à propos du fichier de configuration de « ip_forwarding ». Par défaut, le plupart des noyaux linux ne permettent pas l'expédition des paquets IP entre ses interfaces. Cependant, on peut changer ça en mettant à 1 le flag correspondant en tapant :

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Pour sauvegarder notre configuration pour ce flag, il faut mettre à 1 la variable « net.ipv4.ip_forward » dans le fichier « /etc/sysctl.conf ».

Pour les machines qui tournent RedHat comme système d'exploitation, les fichiers de configuration des interfaces se situent dans « /etc/sysconfig/network-scripts ». Pour l'interface nommée ethx, le nom du fichier sera « ifcfg-ethx ». La syntaxe du fichier est la suivante :

```
DEVICE=ethx
IPADDR=x.y.z.t
NETMASK=255.255.255.0
NETWORK= x.y.z.0
BROADCAST= x.y.z.255
ONBOOT=yes
```

Pour la configuration des tables de routage, il faut les éditer dans le fichier « /etc/sysconfig/static-routes ». La syntaxe de ce fichier et d'autre fichiers de configuration du réseau sur RedHat se trouve dans la répertoire « ../linux/Documentation/networking/ ». Le suivant est un exemple de la syntaxe de ce fichier sur une de mes machines :

```
eth2 net 10.0.20.0 netmask 255.255.255.0 gw 10.0.22.2
eth0 net 224.1.1.1 netmask 255.255.255.255 gw 10.0.24.2
```

Une remarque à considérer ici est que ces fichiers (celles des configurations des interfaces et l'autre pour les routes) doivent être exécutables pour qu'ils puissent être initialisés par un script du réseau. Si ce n'est pas le cas, on doit alors procéder à changer le flag d'exécution du fichier en tapant « chmod a+x nom_fichier ».
Une fois on a édité ces fichiers, on peut alors recharger nos configurations en tournant un script de démarrage du réseau comme « /etc/init.d/network ».
Un mot encore pour les machines RedHat à propos du fichier de configuration de « ip_forwarding ». Par défaut, le plupart des noyaux linux ne permettent pas l'expédition des paquets IP entre ses interfaces. Cependant, on peut changer ça en mettant à 1 le flag correspondant en tapant :

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Pour sauvegarder notre configuration pour ce flag, il faut mettre à 1 la variable « net.ipv4.ip_forward » dans le fichier « /etc/sysctl.conf ».

Pour les machines qui tournent Debian comme système d'exploitation, le fichier de configuration se situe dans « /etc/network ». Ce fichier, unique pour les configurations des interfaces et des routes, s'appelle « interfaces ». Un exemple de la syntaxe de ce fichier est le suivant :

```
iface eth1 inet static
        address 10.0.23.2
        netmask 255.255.255.0
        network 10.0.23.0
        broadcast 10.0.23.255
        up ip r a 10.0.22.0/24 via 10.0.23.1
        up ip r a 10.0.20.0/24 via 10.0.23.1

iface eth2 inet static
        address 10.0.3.2
        netmask 255.255.255.0
        network 10.0.3.0
        broadcast 10.0.3.255
        up ip r a 224.1.1.1/32 via 10.0.3.1
```

Comme dans le cas des fichiers de configuration sur RedHat, ce fichier doit être aussi exécutable. Le script de redémarrage des configurations du réseau sur Debian est « /etc/init.d/networking ».

Aussi, c'est pareil pour le flag d'expédition des paquets IP qui doit être mis à 1. Pourtant, la sauvegarde de la configuration de ce flag sur Debain se fait dans le fichier « /etc/network/options ».

Outils de diagnostique et d'analyse du réseau

ping : l'outil le plus célèbre pour le diagnostic du réseau. Il permet de détecter un bon nombre de problèmes concernant la configuration IP : adressage des cartes réseaux, configuration des routes.

traceroute : L'autre outil important pour le diagnostic des problèmes du routage, c'est la commande traceroute, qui comme son nom l'indique, permet d'afficher la route suivie par les paquets de données vers la destination. La route est constituée de tous les routeurs traversés pour arriver à destination.

netstat : Cet outil permet entre autres choses d'obtenir d'informations détaillées sur les connections ouvertes. L'options les plus couramment utilisés avec cette commande c'est « antup » où la lettre « a » tient pour l'affichage de tous les connections actives et non actives, « n » pour l'affichage des adresses numériques au lieu des noms des machines, « t » pour l'affichage des portes utilisant tcp, « u » pour l'affichage des portes utilisant udp, « p » pour l'affichage des noms des programmes correspondants aux connections ouvertes. Un exemple de cette commande est le suivant :

```
root@leone:~# netstat -antup
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale          Adresse distante
Etat        PID/Program name
tcp        0      0 0.0.0.0:6000            0.0.0.0:*
LISTEN     23676/X
tcp        0      0 0.0.0.0:2646            0.0.0.0:*
LISTEN     9695/multreeldp
```

```
tcp        0      0 0.0.0.0:22              0.0.0.0:*
LISTEN     22533/sshd
udp        0      0 0.0.0.0:2646            0.0.0.0:*
9695/multreeldp
```

tcpdump : Il s'agit d'un outil très efficace pour la diagnostic et l'analyse du trafic sur le réseau. Cet outil permet d'afficher la description et le contenu des paquets qui transitent sur les interfaces. L'option pour l'affichage du contenu d'un paquet se fait avec « vvv ». Mais la caractéristique importante de cet outil c'est qu'il permet d'élaborer des filtres en fonction d'informations recherchées. En voici quelques uns des filtres possibles:

and, or : permet de combiner les filtres

src, dst : permet de choisir les paquets provenant de / à destination d'une interface réseau.

host,net, port : filtrer l'affichage des paquets selon un nom de machine, un réseau et/ou un port

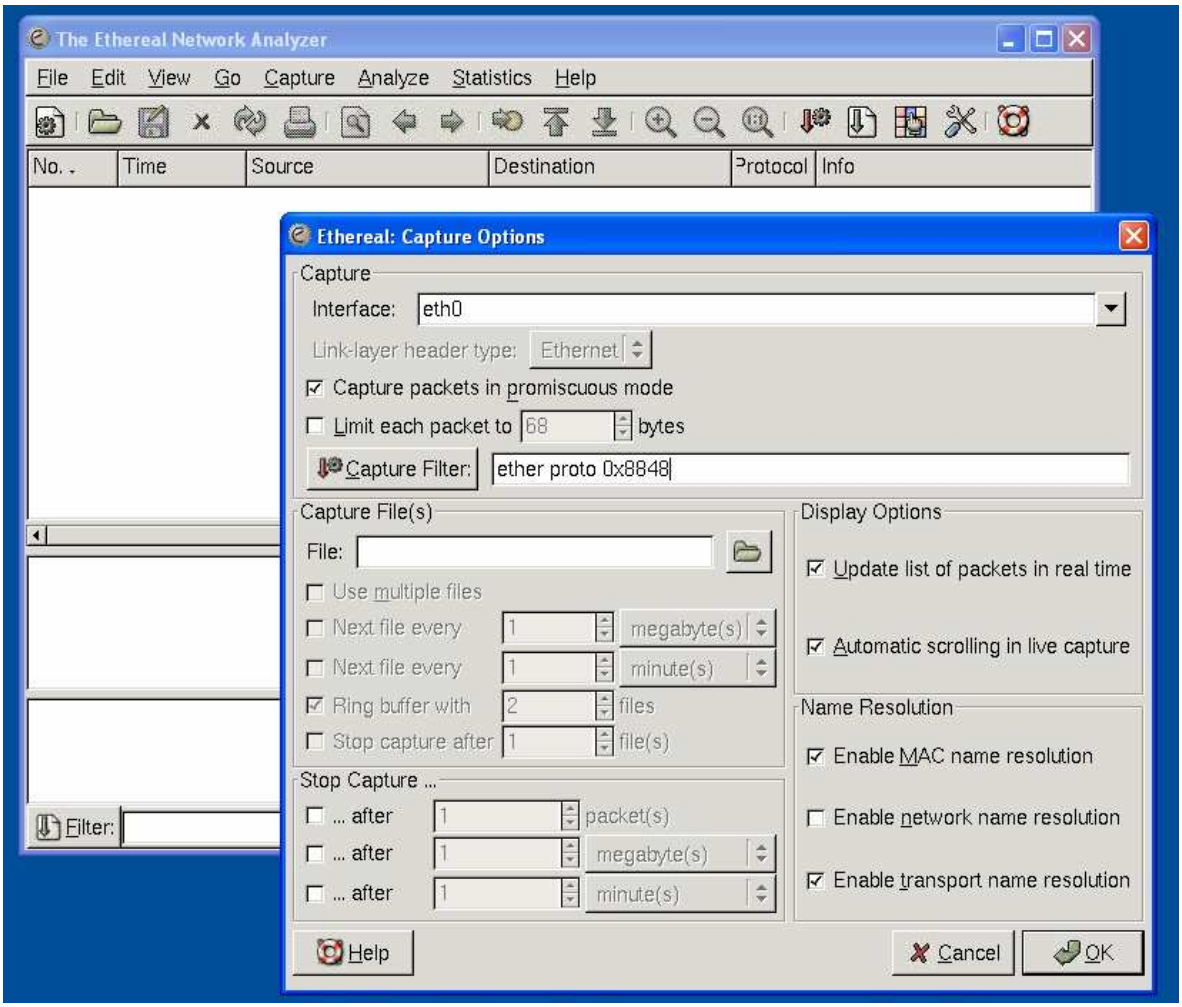
ether proto `protocole`: filtrer l'affichage des paquets ethernet qui sont de type `protocole`. Notons que `protocole` peut être un nom de protocole (à l'exception des noms des protocoles qui sont des mots clés pour cette commande comme ip ou arp et dont leurs noms doivent alors être précéder par une barre oblique \) ou bien un numéro de protocole. La deuxième option de ce filtre est intéressant lorsque on va filtrer les paquets multicast MPLS puisque ce nouveau protocole multreeldp n'apparaît pas dans la liste des protocoles de tcpdump, et on peut alors procéder avec le numéro du protocole 0x8848. Comme exemple, pour afficher les paquets multicast MPLS qui passent sur l'interface eth2 de Cook, on peut faire le suivant :

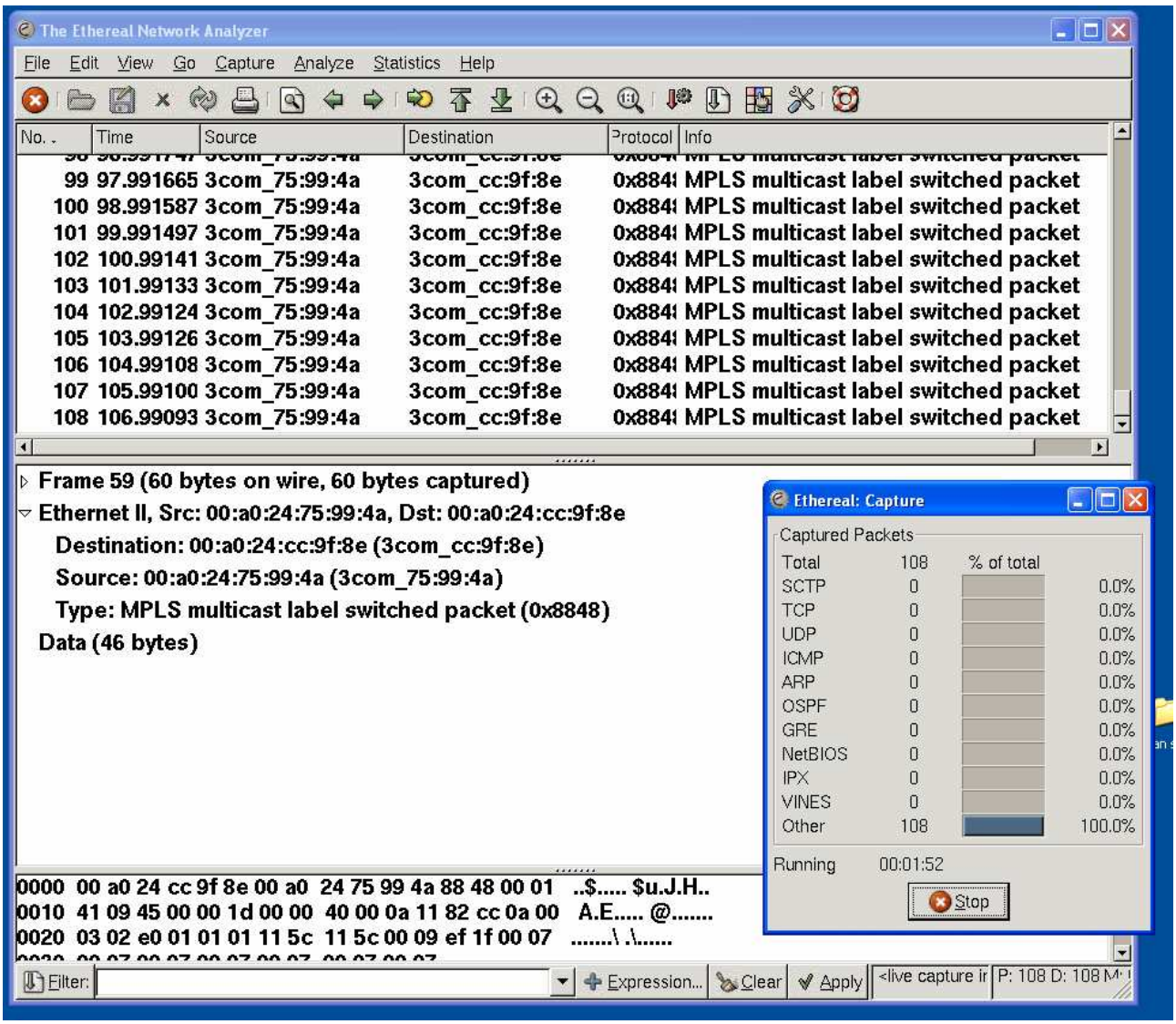
```
root@cook:/home# tcpdump -i eth2 ether proto 0x8848
tcpdump: listening on eth2
17:27:13.825155 MPLS (label 0x14[S] TTL 8)
17:27:14.824862 MPLS (label 0x14[S] TTL 8)
17:27:15.824913 MPLS (label 0x14[S] TTL 8)
17:27:16.824944 MPLS (label 0x14[S] TTL 8)
17:27:17.824993 MPLS (label 0x14[S] TTL 8)
```

L'option « i » permet de spécifier une interface sur laquelle on désire filtrer les paquets.

Ethereal : Similaire à tcpdump, cet outil permet aussi d'afficher et d'analyser d'une façon graphique le trafic circulant sur le réseau. Avant de commencer, on doit ajuster notre filtre selon nos besoins. Pour cela, on appuie sur le bouton « Capture », on spécifie l'interface sur laquelle on désire analyser le trafic, et ensuite, on définit notre filtre dans le champ « Capture Filter ». Pour définir l'expression d'un filtre, on procède pareillement au cas de tcpdump. On peut aussi manipuler l'affichage des paquets en activant les options dans « Display Options ».

Cet outil est équipé sur les quatre machines qui tournent Debian.


The Ethereal Network Analyzer
File Edit View Go Capture Analyze Statistics Help
No. Time Source Destination Protocol Info
Filter:
Ethereal: Capture Options
Capture
Interface: eth0
Link-layer header type: Ethernet
Capture packets in promiscuous mode
Limit each packet to 68 bytes
Capture Filter: ether proto 0x8848
Capture File(s)
File:
Use multiple files
Next file every 1 megabyte(s)
Next file every 1 minute(s)
Ring buffer with 2 files
Stop capture after 1 file(s)
Display Options
Update list of packets in real time
Automatic scrolling in live capture
Name Resolution
Enable MAC name resolution
Enable network name resolution
Enable transport name resolution
Stop Capture ...
... after 1 packet(s)
... after 1 megabyte(s)
... after 1 minute(s)
Help
Cancel
OK

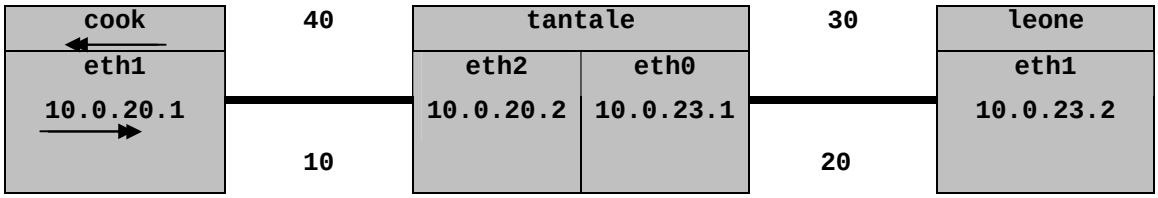


mplsadm : Cet outil permet de gérer des LSPs pour le MPLS. J'ai utilisé cet outil pour tester le patch de MPLS. La version que j'ai utilisée a été fournie avec le code du programme. Les options de cet outil sont les suivantes :

```
tantale:~# mplsadm -h
usage: mplsadm [ADBUdhvT:f:L:I:O:i:o:]
```

```
-A add modifier
-B bind modifier
-D delete modifier
-U unbind modifier
-d toggle debug
-h this message
-v verbose info
-T <tunnel name>:<dest addr>
-f <prefix/len>
-L <interface name>:<label space> set the label space for an interface
(-1 disables)
-I <gen|atm|fr>:<label>:<label space> create|delete an incoming label
-O <gen|atm|fr>:<label>:<out interface>[:<next type>:<next hop>]
-i <opcode:opcode_data>+
-o <opcode:opcode_data>+
```

J'ai établie avec cet outil un LSP bidirectionnel. La topologie pour le test c'était le suivant :



Les numéros sur les flèches indiquent les étiquettes utilisées. La syntaxe est la suivante pour chaque machine :

cook :

```
root@cook:~# mplsadm -L eth1:1
root@cook:~# mplsadm -v -A -I gen:40:1
In label input: gen:40:1
root@cook:~#  mplsadm  -v  -A  -B  -O  gen:10:eth1:ipv4:10.0.20.2  -f
10.0.23.0/24
Out label input: gen:10:eth1:ipv4:10.0.20.2
FEC input: 10.0.23.0/24
ipv4
root@cook:~# cat /proc/net/mpls_labelspace
```

```
lo      1
eth0    2
eth1    1
eth2    4
eth3    5
root@cook:~# cat /proc/net/mpls_in
4000a001 gen 40 1 0/0/0 POP DLV
root@cook:~# cat /proc/net/mpls_out
```

tantale :

```
tantale:~# mplsadm -L eth0:0
tantale:~# mplsadm -L eth2:2
tantale:~# mplsadm -v -A -I gen:10:2 -O gen:20:eth0:ipv4:10.0.23.2 -B
In label input: gen:10:2
Out label input: gen:20:eth0:ipv4:10.0.23.2
ipv4
tantale:~# mplsadm -v -A -I gen:30:0 -O gen:40:eth2:ipv4:10.0.20.1 -B
In label input: gen:30:0
Out label input: gen:40:eth2:ipv4:10.0.20.1
ipv4
tantale:~# cat /proc/net/mpls_labelspace
lo      1
eth0    0
eth1    3
eth2    2
tantale:~# cat /proc/net/mpls_in
40007800 gen 30 0 0/0/0 POP FWD(4000a004)
tantale:~# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) SET(eth0)
4000a004 0/0/0 PUSH(gen 40) SET(eth2)
```

leone :

```
root@leone:~# mplsadm -L eth1:1
root@leone:~# mplsadm -v -A -I gen:20:1
In label input: gen:20:1
root@leone:~# mplsadm -v -A -B -O gen:30:eth1:ipv4:10.0.23.1 -f
10.0.20.0/24
```

```
Out label input: gen:30:eth1:ipv4:10.0.23.1
FEC input: 10.0.20.0/24
ipv4
root@leone:~#
root@leone:~# cat /proc/net/mpls_labelspace
lo      1
eth0    2
eth1    1
eth2    4
root@leone:~# cat /proc/net/mpls_in
40005001 gen 20 1 0/0/0 POP DLV
root@leone:~# cat /proc/net/mpls_out
40007803 0/0/0 PUSH(gen 30) SET(eth1)
```

Pourvu que j'ai quelques entrées qui ne s'affichent pas dans les tables (champ de sortie pour l'interface eth1 de cook, et champ d'entrée pour l'interface eth2 de tantale malgré que ces entrées apparaissent sur les autres interfaces), quand j'ai fait un ping à partir de cook pour l'adresse 10.023.2 et j'ai filtré les paquets MPLS sur tantale/eth2 et sur leone/eth1, j'ai eu le suivant :

```
root@cook:~# ping 10.0.23.2
PING 10.0.23.2 (10.0.23.2): 56 data bytes
64 bytes from 10.0.23.2: icmp_seq=0 ttl=253 time=0.8 ms
64 bytes from 10.0.23.2: icmp_seq=1 ttl=253 time=0.7 ms
64 bytes from 10.0.23.2: icmp_seq=2 ttl=253 time=0.7 ms

--- 10.0.23.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.7/0.7/0.8 ms
root@cook:~#
**********************************************************************
****
tantale:~# tcpdump -i eth2 ether proto 0x8847
tcpdump: listening on eth2
16:09:41.676201 MPLS (label 0x28[S] TTL 253)
16:09:42.668203 MPLS (label 0x28[S] TTL 253)
16:09:43.668100 MPLS (label 0x28[S] TTL 253)
```

```
3 packets received by filter
0 packets dropped by kernel
tantale:~#
************************************************************************
****
root@leone:~# tcpdump -i eth1 ether proto 0x8847
tcpdump: listening on eth1
01:38:52.288226 MPLS (label 0x1e[S] TTL 254)
01:38:53.280325 MPLS (label 0x1e[S] TTL 254)
01:38:54.280307 MPLS (label 0x1e[S] TTL 254)

3 packets received by filter
0 packets dropped by kernel
root@leone:~#
```

Après, j'ai fait l'inverse,c.à.d. j'ai fait un ping à partir de leone pour l'adresse 10.0.20.1 et j'ai filtré les paquets MPLS sur tantale/eth0 et sur cook/eth1, j'ai eu le suivant :

```
root@leone:~# ping 10.0.20.1
PING 10.0.20.1 (10.0.20.1): 56 data bytes
64 bytes from 10.0.20.1: icmp_seq=0 ttl=254 time=0.7 ms
64 bytes from 10.0.20.1: icmp_seq=1 ttl=254 time=0.7 ms
64 bytes from 10.0.20.1: icmp_seq=2 ttl=254 time=0.7 ms

--- 10.0.20.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.7/0.7/0.7 ms
root@leone:~#
root@leone:~#
************************************************************************
****
tantale:~# tcpdump -i eth0 ether proto 0x8847
tcpdump: listening on eth0
16:05:57.586486 MPLS (label 0x28[S] TTL 62)
16:05:58.578671 MPLS (label 0x28[S] TTL 62)
16:05:59.578568 MPLS (label 0x28[S] TTL 62)
```

```
3 packets received by filter
0 packets dropped by kernel
tantale:~#
tantale:~#
************************************************************************
****
root@cook:~# tcpdump -i eth1 ether proto 0x8847
tcpdump: listening on eth1
22:18:59.286641 MPLS (label 0x45000 TTL 84) (label 0x4 TTL 0) (label
0x3e01f exp 0x6[S] TTL 166)
22:19:00.278863 MPLS (label 0x45000 TTL 84) (label 0x4 TTL 0) (label
0x3e01f exp 0x6[S] TTL 166)
22:19:01.278807 MPLS (label 0x45000 TTL 84) (label 0x4 TTL 0) (label
0x3e01f exp 0x6[S] TTL 166)

3 packets received by filter
0 packets dropped by kernel
root@cook:~#
root@cook:~#
```

Les résultats que j'ai obtenus montrent clairement que le patch MPLS marche très bon. Pour vider les tables de MPLS, on peut taper « multreeldp –f ».

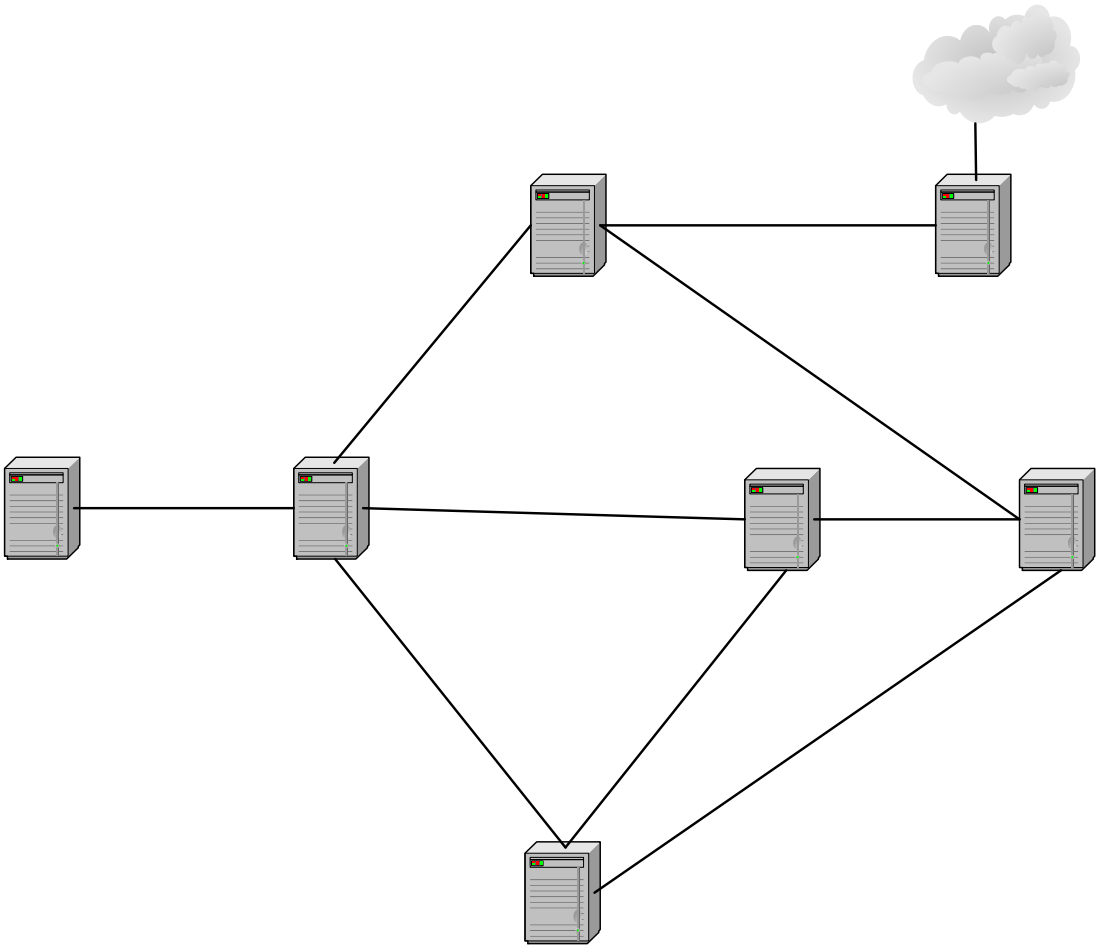
L’exécution du programme :

Tout d’abord, on dispose des quatre machines, tantal, cook, troll et leone, qui tournent Debian comme système. Les autres machines, montrachet, rigel et popy, tournent RedHat 7.3. Lorsqu’on doit compilé notre programme sur une machine et après, procédé à la distribution de la version compilée sur les autres machines, il faut tenir compte de ce point, c.à.d., que les machines tournantes Debian n’acceptent pas une version compilé sur RedHat, et vice versa. Pour cela, j’ai distribué la code source du programme - et des versions ultérieurs - sur tantal, pour les machines Debain, et sur rigel, pour les machines RedHat. On peut utiliser une disquette pour copier après le bianire en tapant « tar cvf /dev/fd0 nom_fichier_bianire » pour copier le binaire sur la disquette, et « tar xvf /dev/fd0 » pour copier le binaire de la disquette vers le répertoire courant.

La version que j’ai décrite dans ce test c’est la version que j’ai adaptée pour la protection d’une feuille. En fait, cette version c’était un test simplifié pour la version ultérieure conçue pour plusieurs chemins de réserve. Le programme va calculer un chemin protégé et établir un chemin de réserve pour ce chemin protégé. La protection sera donc pour une seule feuille. Cette version a très bien marché. L’autre version que j’ai décrite dans le test suivant c’est la version pour le calcul de plusieurs chemins de réserves (pour plus qu’une feuille) et en conséquence, l’annone de plusieurs chemin protégés.

L’idée c’est de trier les feuilles de l’arbre dans un tableau et les autres nœuds de l’arbre dans un autre tableau, et après, essayer de chercher pour chaque feuille un chemin de réserve vers un nœud dans le deuxième tableau. Si on trouve ce chemin, alors on peut utiliser les mêmes mécanismes qui étaient utilisées dans la version originale du programme. La raison pour la séparation des nœuds en deux tableaux réside dans le fait d’éviter au maximum l’utilisation d’une deuxième ou d’une troisième interface pour un LSR feuille, parce que dans le patch MPLS qu’on dispose, un LSR feuille ne peut pas appliquer l’instruction « push » pour envoyer le trafic MPLS que sur une seule interface. On va discuter ce problème après dans les tests en dessous.

Pour toutes les versions que j'ai modifiées, il faut mettre à 1 un argument (starIndex = starCenterDBLookupFec(address, preLen, 1) d'une fonction (in_path_merge fichier multreeldp.c) dans les versions compilées du programme sur les LSR feuilles et sur les LSRs appartenants au réseau et pas à l'arbre. On va parler de ce cas dans la section traitante le code. La topologie pour notre exemple est la suivante :



L'arbre multicast est formée par les routeurs suivants : montrachet, cook, troll, popy, rigel, leone. Il y a une seule machine, tantal, qui ne se trouve pas dans l'arbre. Avant de commencer à tourner le programme, on est obligé à configurer l'adresse multicast sur tous les LERs. En plus, le routeur qui va commencer à annoncer le chemin primaire et après les chemins des réserves (backup), doit être configuré de façon à accéder à toutes les autres machines. Comme n'importe quelle machine, qui se dispose des fichiers du

eth2
10.0.21.1

eth2
10.0.21.2

eth0
10.0.2.1

eth3
10.0.2.2

eth1
10.0.20.1

montrachet

cook

réseau et de l'arbre, peut annoncer les LSPs, alors il est recommandé de configurer statiquement sur tous les routeurs des routes vers les autres réseaux.

Pour cet exemple, j'ai la configuration IP suivante sur les machines.

Sur montrachet :

```
[root@montrachet root]# ip r s
224.1.1.1 via 10.0.2.2 dev eth0
10.0.2.0/24 dev eth0  scope link
127.0.0.0/8 dev lo  scope link
[root@montrachet root]#
```

Sur cook :

```
root@cook:~# ip r s
10.0.20.0/24 dev eth1  proto kernel  scope link  src 10.0.20.1
10.0.21.0/24 dev eth2  proto kernel  scope link  src 10.0.21.2
10.0.22.0/24 via 10.0.20.2 dev eth1
10.0.23.0/24 via 10.0.20.2 dev eth1
10.0.2.0/24 dev eth3  proto kernel  scope link  src 10.0.2.2
10.0.3.0/24 dev eth0  proto kernel  scope link  src 10.0.3.1
root@cook:~#
```

Sur leone :

```
root@leone:~# ip r s
224.1.1.1 via 10.0.3.1 dev eth2
10.0.20.0/24 via 10.0.23.1 dev eth1
10.0.21.0/24 via 10.0.23.1 dev eth1
10.0.22.0/24 via 10.0.23.1 dev eth1
10.0.23.0/24 dev eth1  proto kernel  scope link  src 10.0.23.2
10.0.3.0/24 dev eth2  proto kernel  scope link  src 10.0.3.2
10.0.26.0/24 dev eth0  proto kernel  scope link  src 10.0.26.2
root@leone:~#
```

Sur tantale :

```
tantale:~# ip r s
10.0.20.0/24 dev eth2  proto kernel  scope link  src 10.0.20.2
10.0.21.0/24 via 10.0.20.1 dev eth2
10.0.22.0/24 dev eth1  proto kernel  scope link  src 10.0.22.2
```

```
10.0.23.0/24 dev eth0  proto kernel  scope link  src 10.0.23.1
tantale:~#
```

Sur rigel :

```
[root@rigel root]# ip r s
224.1.1.1 via 10.0.24.2 dev eth0
10.0.20.0/24 via 10.0.22.2 dev eth2
10.0.21.0/24 via 10.0.22.2 dev eth2
10.0.22.0/24 dev eth2  scope link
10.0.23.0/24 via 10.0.22.2 dev eth2
10.0.24.0/24 dev eth0  scope link
10.0.26.0/24 dev eth1  scope link
127.0.0.0/8 dev lo  scope link
[root@rigel root]#
```

Sur troll:

```
root@troll:~# ip r s
10.0.20.0/24 via 10.0.21.2 dev eth2
10.0.21.0/24 dev eth2  proto kernel  scope link  src 10.0.21.1
10.0.22.0/24 via 10.0.21.2 dev eth2
10.0.23.0/24 via 10.0.21.2 dev eth2
10.0.24.0/24 dev eth0  proto kernel  scope link  src 10.0.24.2
10.0.25.0/24 dev eth1  proto kernel  scope link  src 10.0.25.2
root@troll:~#
```

Sur popy :

```
[root@popy root]# ip r s
224.1.1.1 via 10.0.25.2 dev eth0
10.0.24.0/24 via 10.0.25.2 dev eth0
10.0.25.0/24 dev eth0  scope link
131.254.0.0/16 dev eth1  scope link
127.0.0.0/8 dev lo  scope link
default via 131.254.1.1 dev eth1
[root@popy root]#
```

En plus, le flag de “ip_forwarding” est mis à “1” sur toutes les routeurs. Ces configurations ip sont nécessaires seulement lors de l’établissement de l’arbre, et après, lors de l’annonce des chemins de réserve. Vous pouvez référer à la section **5.2** de la thèse qui décrit les mécanismes et les types de messages échangés lors de l’établissement de chemin primaire et des chemins de secours, et les messages de notifications.

Avant de commencer à tourner le programme, on doit définir notre réseau et notre arbre dans des fichiers séparés. Le fichier du réseau pour notre exemple c’est :

```
router1 10.0.2.1 router2 10.0.2.2
router2 10.0.21.2 router3 10.0.21.1
router2 10.0.20.1 router6 10.0.20.2
router2 10.0.3.1 router5 10.0.3.2
router3 10.0.25.2 router4 10.0.25.1
router3 10.0.24.2 router7 10.0.24.1
router6 10.0.22.2 router7 10.0.22.1
router6 10.0.23.1 router5 10.0.23.2
router5 10.0.26.2 router7 10.0.26.1
```

les noms router1, router2, … sont arbitrairement choisies.

Le fichier de définition de l’arbre c’est :

```
10.0.21.2(10.0.2.1L)(10.0.21.1(10.0.25.1L)(10.0.24.1L))(10.0.3.2L)
```

Vous pouvez référer à la section **5.2.1** qui décrit la syntaxe de ce dernier fichier. J’ai ajouté à cette syntaxe la lettre « L » pour l’identification d’une feuille.

Pour tourner le programme, on doit taper « multreeldp –s & » sur toutes les machines de l’arbre et sur les machines se trouvant dans le réseau et pas dans l’arbre, mais qui vont être utilisées à un moment ultérieure pour l’établissement des chemins de réserve. L’exécution de cette commande à pour effet d’initialiser les paramètres et les threads du programme, de vider les tables de MPLS existantes, et de mettre les machines dans une situation d’écoute sur la porte tcp 2646.

Sur montrachet :

```
[root@montrachet root]# cd /home
[root@montrachet home]# ./multreeldp -s &
[1] 10597
[root@montrachet home]# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646
```

```
[root@montrachet home]#
```

Sur cook :

```
root@cook:~# cd /home
root@cook:/home# ./multreeldp -s &
[1] 6279
root@cook:/home# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646

root@cook:/home#
```

Sur leone :

```
root@leone:~# cd /home
root@leone:/home# ./multreeldp -s &
[1] 16327
root@leone:/home# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646

root@leone:/home#
```

Sur tantal:

```
tantale:~# cd /home
tantale:/home# cd multreeldp-test
tantale:/home/multreeldp-test# ./multreeldp -s &
[1] 2340
tantale:/home/multreeldp-test# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646

tantale:/home/multreeldp-test#
```

Sur rigel :

```
[root@rigel root]# cd /home
[root@rigel home]# cd multreeldp-test/
```

```
[root@rigel multreeldp-test]# ./multreeldp -s &
[1] 16706
Starting in server mode.
[root@rigel multreeldp-test]# Server started on port 2646
Link Monitor started on port 2646

[root@rigel multreeldp-test]#
```

Sur troll :

```
root@troll:/home# ./multreeldp -s &
[1] 15221
root@troll:/home# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646

root@troll:/home#
```

Sur popy :

```
[root@popy home]# ./multreeldp -s &
[1] 4362
[root@popy home]# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646

[root@popy home]#
```

Une fois on a terminé le lancement du programme sur les machines, on choisie un routeur qui va décoder ces deux fichiers et générer à partir deux trois messages de type « demande d'établissement d'une arbre ». Le premier arbre c'est l'arbre multicast, tandis que le deuxième et le troisième sont les chemins de réserves qui sont sous forme d'un arbre avec un seul enfant pour chaque nœud, et ayant comme parent un de deux PSLs. Le premier message est envoyé vers la source de l'arbre qui est dans notre test « cook », tandis que les deux autres sont envoyés vers les deux PSLs qui vont être dans ce test « cook » et « leone ». A la réception d'un tel message, le parent de l'arbre va générer des messages pour demander et annoncer les labels. L'arbre multicast sera

bidirectionnel, tandis que les arbres pour les chemins de réserves seront unidirectionnels, à l'exception de la partie commune qui, par conséquence, va être bidirectionnelle.
Dans ce test, j'ai choisie la machine « troll » pour ce rôle. Sur cette machine, on tape :

```
root@troll:/home# ./multreeldp -s &
[1] 15455
root@troll:/home# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646

root@troll:/home# ./multreeldp -b 224.1.1.1 -n network -t tree
FEC e0010101

Multicast tree :
==============
PREFIX 0a001502 - Parent: 00000000
Child 0
        PREFIX 0a000201 (leaf) - Parent: 0a001502
Child 1
        PREFIX 0a001501 - Parent: 0a001502
        Child 0
                PREFIX 0a001901 (leaf) - Parent: 0a001501
        Child 1
                PREFIX 0a001801 (leaf) - Parent: 0a001501
Child 2
        PREFIX 0a000302 (leaf) - Parent: 0a001502

First Backup path tree :
======================
PREFIX 0a001702 (leaf) - Parent: 00000000
[unique child]
        PREFIX 0a001701 - Parent: 00000000
        [unique child]
                PREFIX 0a001401 - Parent: 00000000

Second Backup path tree:
=======================
PREFIX 0a001401 - Parent: 00000000
[unique child]
        PREFIX 0a001402 - Parent: 00000000
        [unique child]
                PREFIX 0a001702 - Parent: 00000000
                [unique child]
                        PREFIX 0a000301 - Parent: 00000000

Protected Path:
==============
0a000302        0a000301
root@troll:/home#
```

Les numéros affichés se sont les adresses IP en hexadécimales. Premièrement, c'est l'arbre multicast qui est affiché. En deuxième lieu, c'est le chemin de secours établi dans la première direction de leone vers tantal vers cook. Troisièmement, c'est le chemin de secours établi dans la deuxième direction de cook vers tantal vers leone vers cook. L'asymétrie du chemin résulte du fait qu'on a conservé la notion du Sprime – voir section 4.4. Finalement, c'est le chemin protégé qui est affiché et qui est la liaison entre cook et leone.

Les tables MPLS d'entrée et de sortie pour les sept routeurs avec les attributs des labels pour les interfaces sont les suivants:

```
[root@montrachet home]# cat /proc/net/mpls_labelspace
lo      1
eth0    2
[root@montrachet home]# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP DLV
[root@montrachet home]# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
[root@montrachet home]#
```

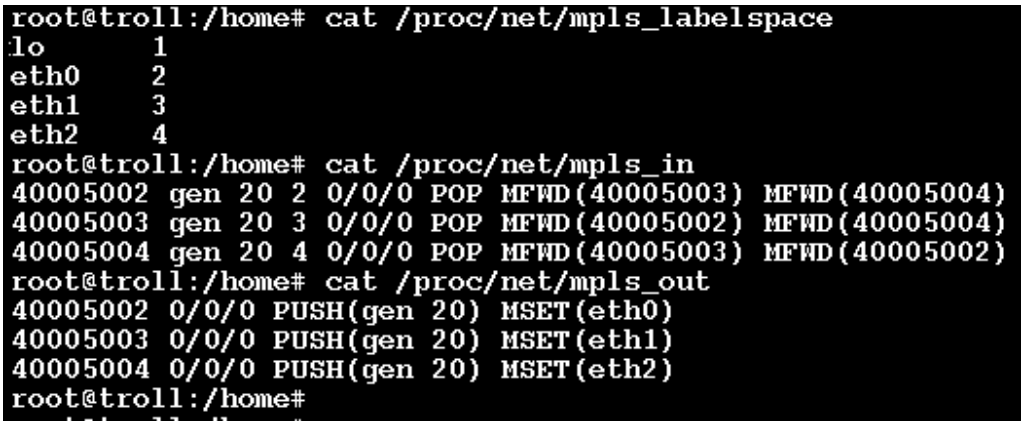
```
root@cook:/home# cat /proc/net/mpls_labelspace
lo      1
eth0    2
eth1    3
eth2    4
eth3    5
root@cook:/home# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005003 gen 20 3 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005004 gen 20 4 0/0/0 POP MFWD(40005002) MFWD(40005005)
40005005 gen 20 5 0/0/0 POP MFWD(40005002) MFWD(40005004)
40005402 gen 21 2 0/0/0 POP MFWD(40005004) MFWD(40005005)
root@cook:/home# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005005 0/0/0 PUSH(gen 20) MSET(eth3)
```

```
root@leone:/home# cat /proc/net/mpls_labelspace
lo      1
eth0    2
eth1    3
eth2    4
root@leone:/home# cat /proc/net/mpls_in
40005003 gen 20 3 0/0/0 POP MFWD(40005404)
40005004 gen 20 4 0/0/0 POP DLV
root@leone:/home# cat /proc/net/mpls_out
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005404 0/0/0 PUSH(gen 21) MSET(eth2)
root@leone:/home#
```

```
tantale:/home/multreeldp-test# cat /proc/net/mpls_labelspace
lo      1
eth0    2
eth1    3
eth2    4
tantale:/home/multreeldp-test# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP MFWD(40005004)
40005004 gen 20 4 0/0/0 POP MFWD(40005002)
tantale:/home/multreeldp-test# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
tantale:/home/multreeldp-test#
```

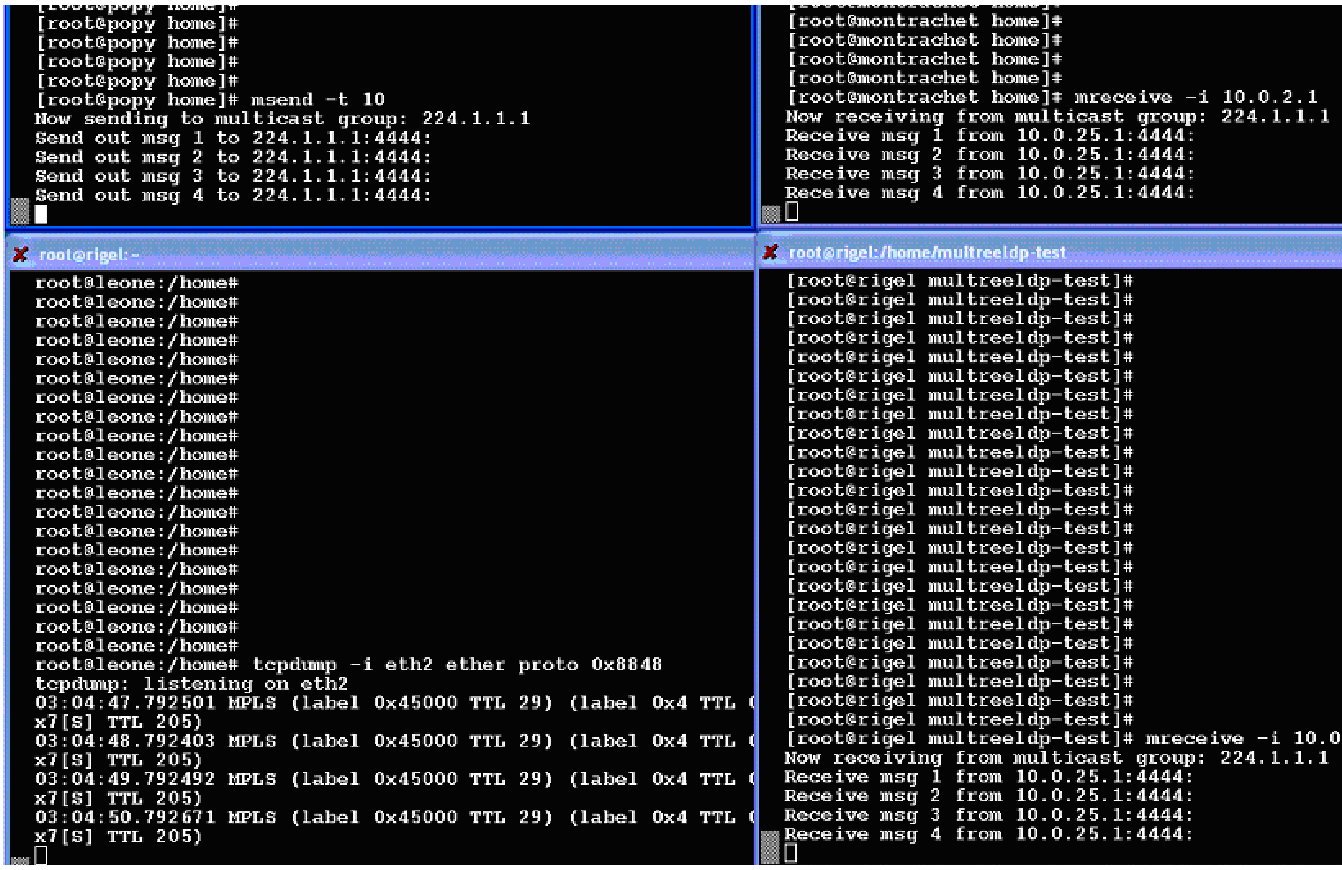
```
[root@rigel multreeldp-test]# cat /proc/net/mpls_labelspace
lo      1
eth0    2
eth1    3
eth2    4
[root@rigel multreeldp-test]# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP DLV
[root@rigel multreeldp-test]# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
[root@rigel multreeldp-test]#
```

```
[root@popy home]# cat /proc/net/mpls_labelspace
lo      1
eth0    2
eth1    3
[root@popy home]# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP DLV
[root@popy home]# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
[root@popy home]#
```



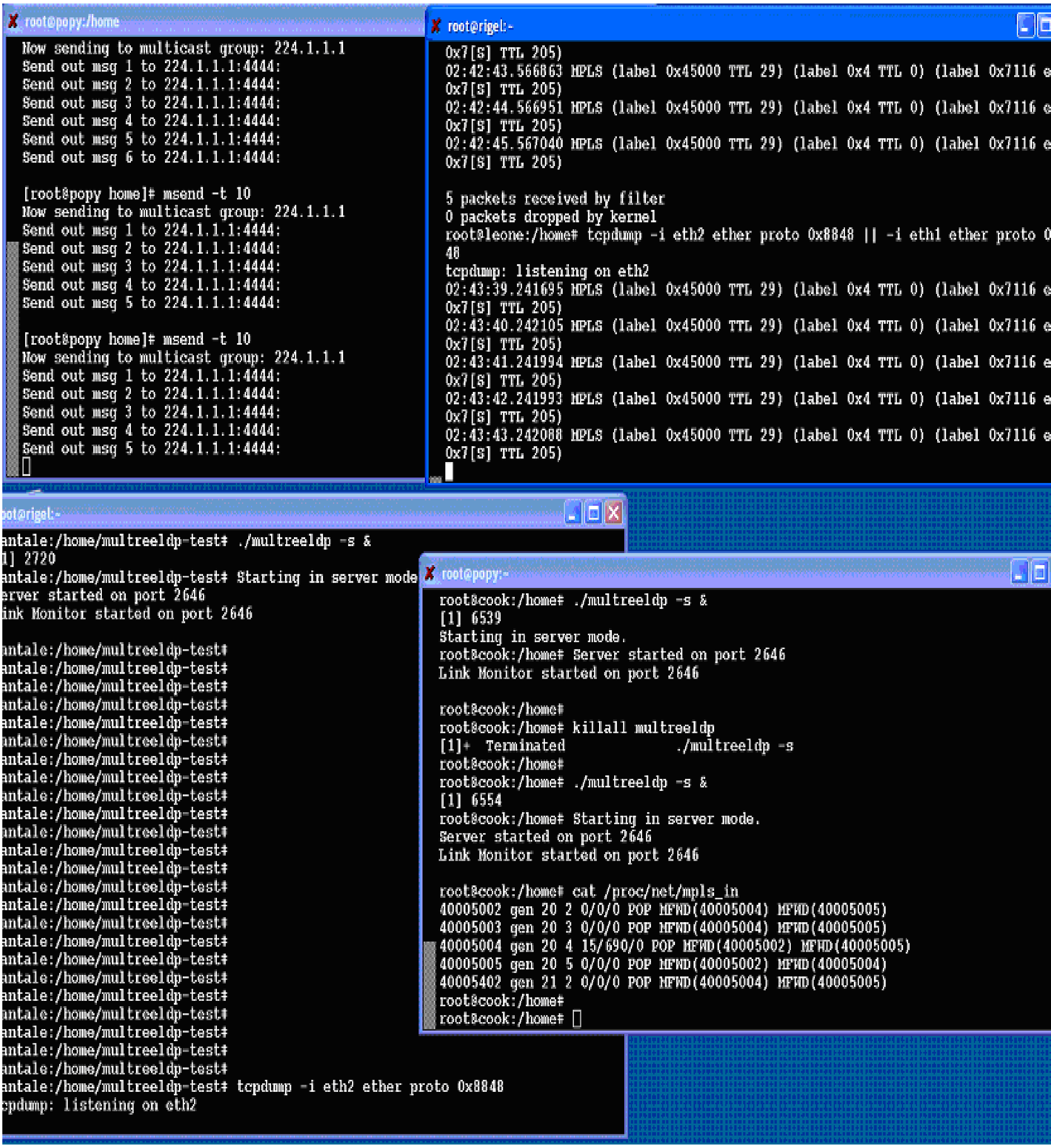
Considérons la dernière figure. Le premier numéro dans mpls_in et mpls_out c'est l'index de l'entrée dans la table. « gen » c'est « generic label ». «20 » c'est le label d'entrée. « 2 » c'est le « labelspace » pour l'interface d'entrée. Pour l'expression « 0/0/0 », le premier champ indique le nombre des paquets reçus, cas table d'entrée, et le nombre des paquets envoyés, cas table de sortie ; le deuxième indique le volume en Bytes des données reçues ou envoyées. La première ligne dans la table de mpls_in signifie que lorsqu'on reçoit un paquet multicast mpls sur l'interface eth0 avec un label de 20, alors faisant un « POP » puis envoyant le sur l'interface eth1 avec un label 20 et sur l'interface eth2 avec un label 20.

Pour simuler une transmission multicast, on va générer du trafic avec l'outil « msend » sur popy par exemple, et on va recevoir ce trafic sur les autres feuilles soit avec l'outil « mreceive » soit avec « tcpdump » en filtrant les paquets multicast MPLS.



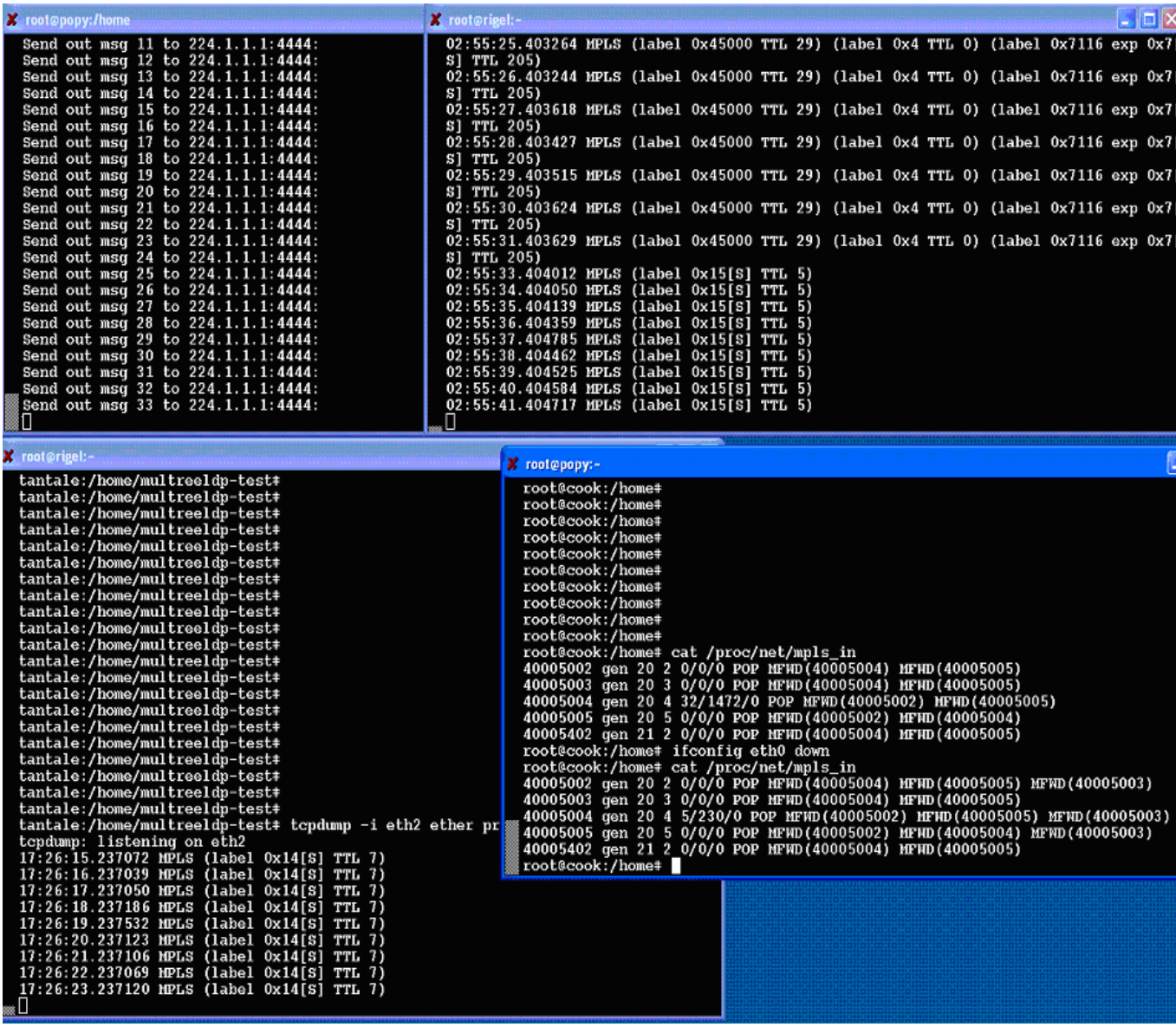
Maintenant, on va éteindre une interface sur le chemin protégé – l'interface eth0 de cook - pour simuler la coupure d'une liaison. On va afficher les tables de MPLS sur un des PSL – cook- suite à cette coupure, puis après on va afficher ces tables suite au recouvrement de cette interface. On va attendre les paquets utilisant le chemin de réserve sur l'interface eth2 de tantal.

<u>Avant la coupure de l'interface :</u>



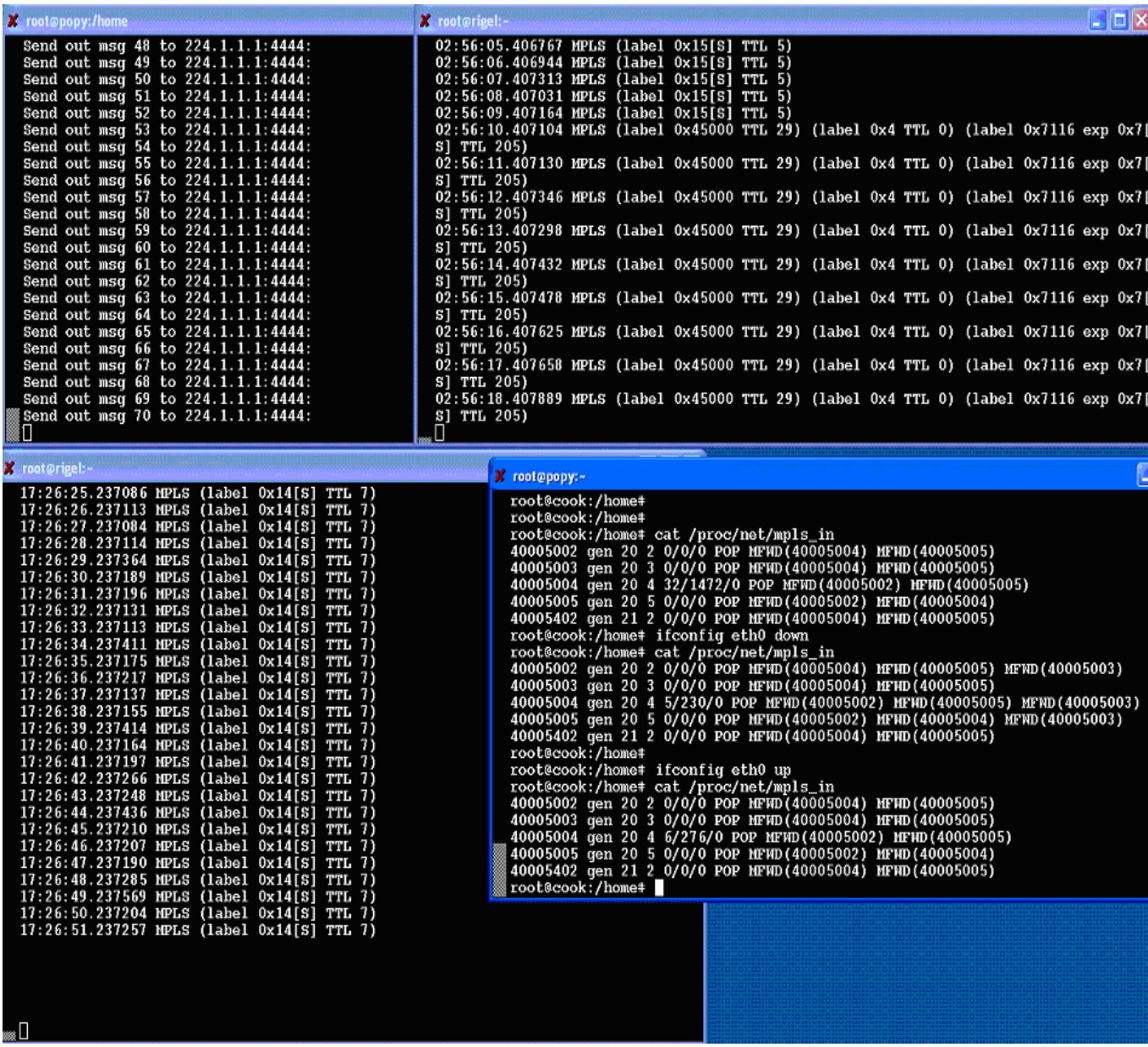
Après la coupure de l’interface :

Dans le deuxième écran, on voit que la réception du paquet sur leone n’a pas influencé par la coupure de la liaison. Dans le quatrième écran, on voit que les tables d’entrée de MPLS sont changées de façon à tenir en compte le chemin de réserve puisque cook est un PSL pour le chemin où l’interface est coupé. Sur le troisième écran, on commence à recevoir du trafic sur l’interface eth2 de tantal.,



Après le recouvrement :

On voit que le trafic est arrêté de circuler sur le chemin de réserve. Cela est montré par l'arrêt de la réception sur tantal. La table MPLS d'entrée sur cook est changée suite à ce recouvrement.



Comme c'est prévu avec ce patch, on a des problèmes avec l'envoie du trafic sur un LSR feuille ayant plus qu'une interface pour envoyer le trafic MPLS. C'est le cas du leone par exemple. En effet, si on essaye de générer du trafic avec « msend » sur cette machine, le trafic est envoyé sur l'interface eth1 qui se trouve sur le chemin de réserve au lieu d'être envoyer sur l'autre l'interface prévu pour l'envoie du trafic sur l'arbre multicast.

Une autre chose, un paquet reçu sur l'interface eth2 de leone et destiné, suite à une coupure, à être envoyé sur le chemin de réserve via l'interface eth1 de cette machine, est empêché de faire ça parce que on ne dispose dans l'entrée sur l'interface eth2 de leone que d'une instruction « POP » suivie d'une instruction « DLV ».

Le test suivant qu'on va décrit c'est le test avec la version conçue pour calculer plusieurs chemins de réserve pour plusieurs chemins protégés. Cette version a bien calculée les chemins de réserves et les chemins protéges, mais on a des problèmes avec cette version dans la table d'entrée de MPLS en cas de switchover suite à une coupure sur un chemin protégé. En plus, elle n'est pas trop testé et génère parfois des messages d'erreur « Segmentation fault ».

On va reprendre le test précèdent. On va générer du trafic sur popy, et on va le recevoir sur Leone. On va écouter aussi sur l'interface eth2 de tantal qui est prévu pour recevoir le trafic en cas de 'switchover' sur cook. On va afficher la table d'entrée sur cook avant et après le 'switchover' suite à une coupure de liaison sur un chemin protégé. On va remarquer qu'il y a un 'switchover' suite à cette coupure. Le problème c'est lorsqu'on va couper une liaison sur l'autre chemin protégé, alors pas de 'switchover' et la table d'entrée de MPLS du PSL cook reste inchangeable.

Pareil à l'exemple précèdent, on va taper la commande suivante sur troll après le lancement du programme sur toutes les machines :

```
root@troll:/home/multi# ./multreeldp -b 224.1.1.1 -n /home/network -t /home/tree
FEC e0010101
Multicast tree :
==============
PREFIX 0a001502 - Parent: 00000000
Child 0
        PREFIX 0a000201 (leaf) - Parent: 0a001502
Child 1
        PREFIX 0a001501 - Parent: 0a001502
        Child 0
                PREFIX 0a001901 (leaf) - Parent: 0a001501
        Child 1
                PREFIX 0a001801 (leaf) - Parent: 0a001501
Child 2
        PREFIX 0a000302 (leaf) - Parent: 0a001502

Backup path number 1 :
====================
PREFIX 0a001601 (leaf) - Parent: 00000000
[unique child]
        PREFIX 0a001602 - Parent: 00000000
        [unique child]
                PREFIX 0a001401 - Parent: 00000000

PREFIX 0a001401 - Parent: 00000000
[unique child]
        PREFIX 0a001402 - Parent: 00000000
        [unique child]
                PREFIX 0a001601 - Parent: 00000000
                [unique child]
                        PREFIX 0a001802 - Parent: 00000000
                        [unique child]
                                PREFIX 0a001502 - Parent: 00000000

Protected Path number 1 :
=======================
0a001801        0a001802        0a001501        0a001502


Backup path number 2 :
====================
PREFIX 0a001702 - Parent: 00000000
[unique child]
        PREFIX 0a001701 - Parent: 00000000
        [unique child]
                PREFIX 0a001401 - Parent: 00000000

PREFIX 0a001401 (leaf) - Parent: 00000000
[unique child]
        PREFIX 0a001402 - Parent: 00000000
        [unique child]
                PREFIX 0a001702 - Parent: 00000000
                [unique child]
                        PREFIX 0a000301 - Parent: 00000000

Protected Path number 2 :
=======================
0a000302        0a000301

root@troll:/home/multi#
```

On voit sur cet écran les chemins de réserve établie dans les deux sens pour les deux chemins protégés. Le premier chemin protégé est formé par les liens entre cook, troll et rigel, et le deuxième chemin est formé par le seul lien entre cook et leone. Les fichiers du réseau et de l'arbre sont aussi les même.

On va afficher maintenant les tables de MPLS pour quelques machines qui nous intéressent. Les attributs de label sont les même que dans le cas précèdent.

Sur rigel :

```
[root@rigel multreeldp-test-multi]# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP DLV
40005004 gen 20 4 0/0/0 POP MFWD(40005402)
[root@rigel multreeldp-test-multi]# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005402 0/0/0 PUSH(gen 21) MSET(eth0)
[root@rigel multreeldp-test-multi]#
```

Sur troll :

```
root@troll:/home/multi# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP MFWD(40005003) MFWD(40005004)
40005003 gen 20 3 0/0/0 POP MFWD(40005002) MFWD(40005004)
40005004 gen 20 4 0/0/0 POP MFWD(40005002) MFWD(40005003)
40005402 gen 21 2 0/0/0 POP MFWD(40005003) MFWD(40005404)
root@troll:/home/multi# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005404 0/0/0 PUSH(gen 21) MSET(eth2)
```

Sur tantal :

```
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP MFWD(40005404)
40005003 gen 20 3 0/0/0 POP MFWD(40005004)
40005004 gen 20 4 0/0/0 POP MFWD(40005003)
40005404 gen 21 4 0/0/0 POP MFWD(40005002)
tantale:/home/multreeldp-test-multi# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005404 0/0/0 PUSH(gen 21) MSET(eth2)
```

Sur leone :

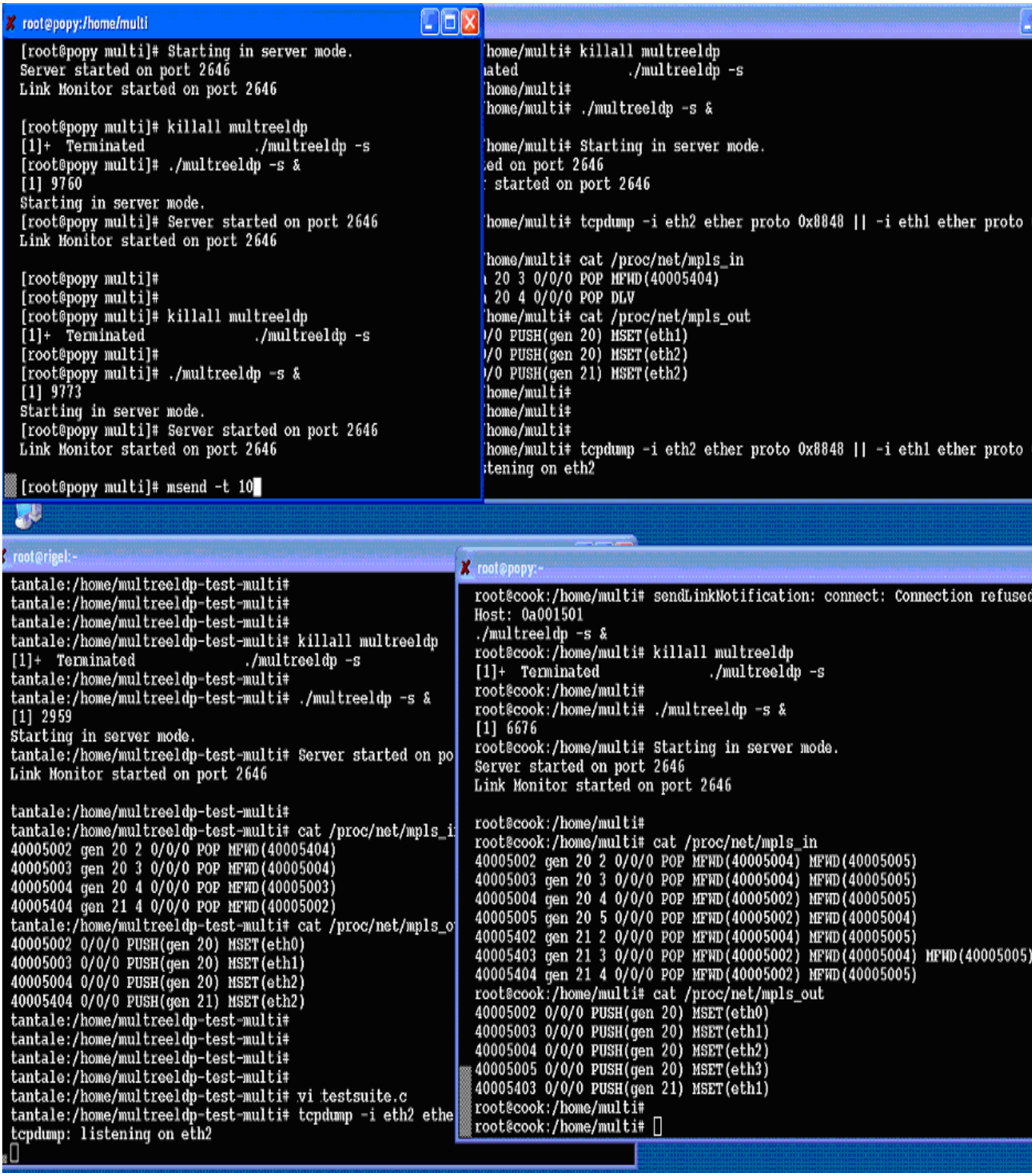
```
root@leone:/home/multi# cat /proc/net/mpls_in
40005003 gen 20 3 0/0/0 POP MFWD(40005404)
40005004 gen 20 4 0/0/0 POP DLV
root@leone:/home/multi# cat /proc/net/mpls_out
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005404 0/0/0 PUSH(gen 21) MSET(eth2)
```

Sur cook:

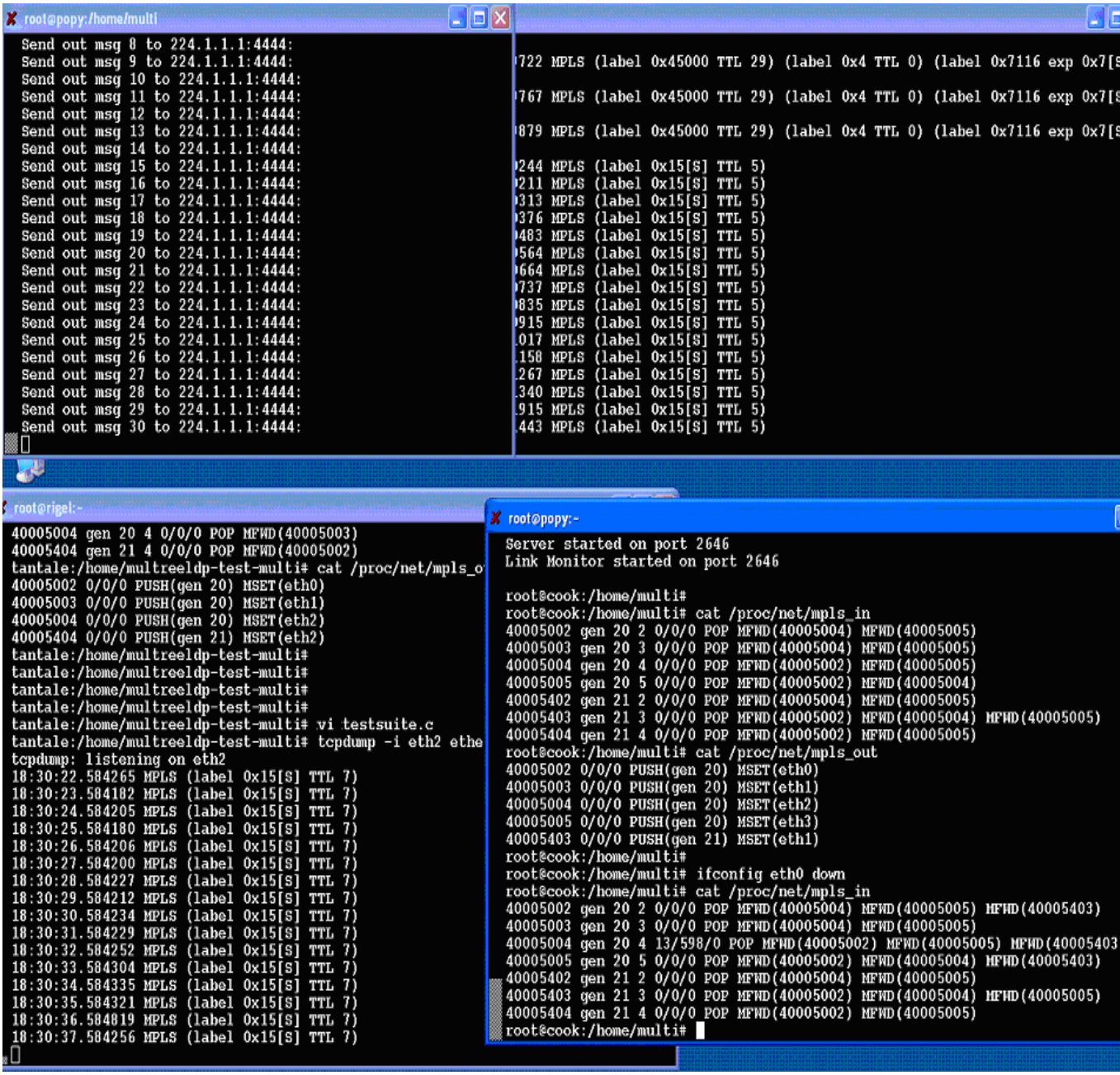
```
root@cook:/home/multi#
root@cook:/home/multi# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005003 gen 20 3 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005004 gen 20 4 0/0/0 POP MFWD(40005002) MFWD(40005005)
40005005 gen 20 5 0/0/0 POP MFWD(40005002) MFWD(40005004)
40005402 gen 21 2 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005403 gen 21 3 0/0/0 POP MFWD(40005002) MFWD(40005004) MFWD(40005005)
40005404 gen 21 4 0/0/0 POP MFWD(40005002) MFWD(40005005)
root@cook:/home/multi# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005005 0/0/0 PUSH(gen 20) MSET(eth3)
40005403 0/0/0 PUSH(gen 21) MSET(eth1)
root@cook:/home/multi#
```

Les différences entre ces tables et les tables du test précèdent résident seulement dans les tables sur cook, tantal et rigel.

On va simuler maintenant une transmission multicast. La figure suivante montre les outils utilisés.

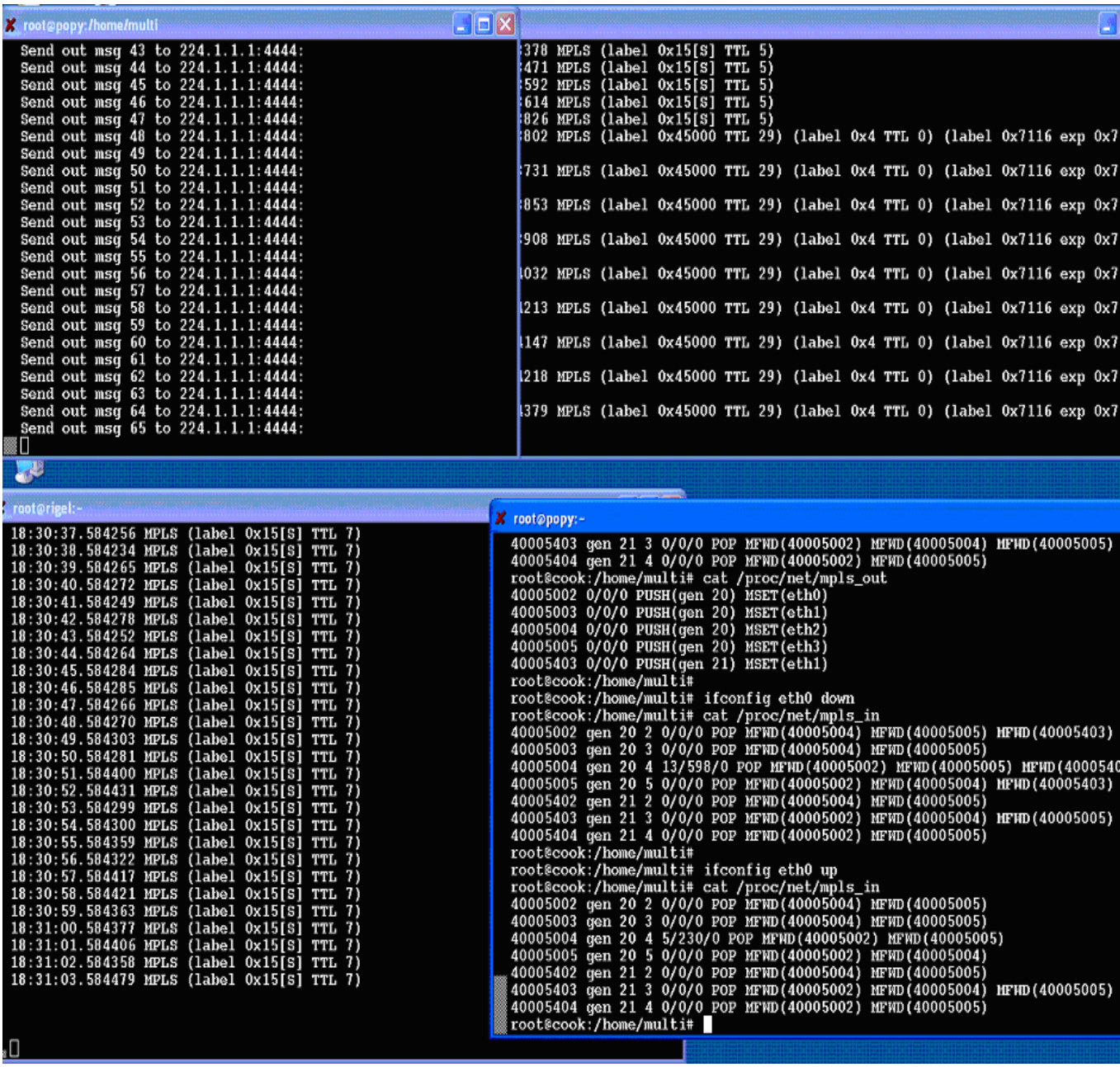

root@popy:/home/multi
[root@popy multi]# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646
[root@popy multi]# killall multreeldp
[1]+ Terminated ./multreeldp -s
[root@popy multi]# ./multreeldp -s &
[1] 9760
Starting in server mode.
[root@popy multi]# Server started on port 2646
Link Monitor started on port 2646
[root@popy multi]#
[root@popy multi]#
[root@popy multi]# killall multreeldp
[1]+ Terminated ./multreeldp -s
[root@popy multi]#
[root@popy multi]# ./multreeldp -s &
[1] 9773
Starting in server mode.
[root@popy multi]# Server started on port 2646
Link Monitor started on port 2646
[root@popy multi]# msend -t 10
home/multi# killall multreeldp
ated ./multreeldp -s
home/multi#
home/multi# ./multreeldp -s &
home/multi# Starting in server mode.
ed on port 2646
started on port 2646
home/multi# tcpdump -i eth2 ether proto 0x8848 || -i eth1 ether proto
home/multi# cat /proc/net/mpls_in
20 3 0/0/0 POP MFWD(40005404)
20 4 0/0/0 POP DLV
home/multi# cat /proc/net/mpls_out
/0 PUSH(gen 20) MSET(eth1)
/0 PUSH(gen 20) MSET(eth2)
/0 PUSH(gen 21) MSET(eth2)
home/multi#
home/multi#
home/multi#
home/multi# tcpdump -i eth2 ether proto 0x8848 || -i eth1 ether proto
tening on eth2
root@rigel:~
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi# killall multreeldp
[1]+ Terminated ./multreeldp -s
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi# ./multreeldp -s &
[1] 2959
Starting in server mode.
tantale:/home/multreeldp-test-multi# Server started on po
Link Monitor started on port 2646
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi# cat /proc/net/mpls_i
40005002 gen 20 2 0/0/0 POP MFWD(40005404)
40005003 gen 20 3 0/0/0 POP MFWD(40005004)
40005004 gen 20 4 0/0/0 POP MFWD(40005003)
40005404 gen 21 4 0/0/0 POP MFWD(40005002)
tantale:/home/multreeldp-test-multi# cat /proc/net/mpls_o
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005404 0/0/0 PUSH(gen 21) MSET(eth2)
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi#
tantale:/home/multreeldp-test-multi# vi testsuite.c
tantale:/home/multreeldp-test-multi# tcpdump -i eth2 ethe
tcpdump: listening on eth2
root@popy:~
root@cook:/home/multi# sendLinkNotification: connect: Connection refused
Host: 0a001501
./multreeldp -s &
root@cook:/home/multi# killall multreeldp
[1]+ Terminated ./multreeldp -s
root@cook:/home/multi#
root@cook:/home/multi# ./multreeldp -s &
[1] 6676
root@cook:/home/multi# Starting in server mode.
Server started on port 2646
Link Monitor started on port 2646
root@cook:/home/multi#
root@cook:/home/multi# cat /proc/net/mpls_in
40005002 gen 20 2 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005003 gen 20 3 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005004 gen 20 4 0/0/0 POP MFWD(40005002) MFWD(40005005)
40005005 gen 20 5 0/0/0 POP MFWD(40005002) MFWD(40005004)
40005402 gen 21 2 0/0/0 POP MFWD(40005004) MFWD(40005005)
40005403 gen 21 3 0/0/0 POP MFWD(40005002) MFWD(40005004) MFWD(40005005)
40005404 gen 21 4 0/0/0 POP MFWD(40005002) MFWD(40005005)
root@cook:/home/multi# cat /proc/net/mpls_out
40005002 0/0/0 PUSH(gen 20) MSET(eth0)
40005003 0/0/0 PUSH(gen 20) MSET(eth1)
40005004 0/0/0 PUSH(gen 20) MSET(eth2)
40005005 0/0/0 PUSH(gen 20) MSET(eth3)
40005403 0/0/0 PUSH(gen 21) MSET(eth1)
root@cook:/home/multi#
root@cook:/home/multi#

La figure suivante montre le ‘switchover’ suite à une coupure du lien reliant cook et leone. Le trafic, redirigé par cook sur le chemin de réserve, passe avant d’être reçu par leone par l’interface eth2 de tanal. Le quatrième écran montre la table MPLS d’entrée sur cook qui est changée de façon à prendre en considération le chemin de réserve établie pour le chemin protégé dont le lien coupé en appartient.



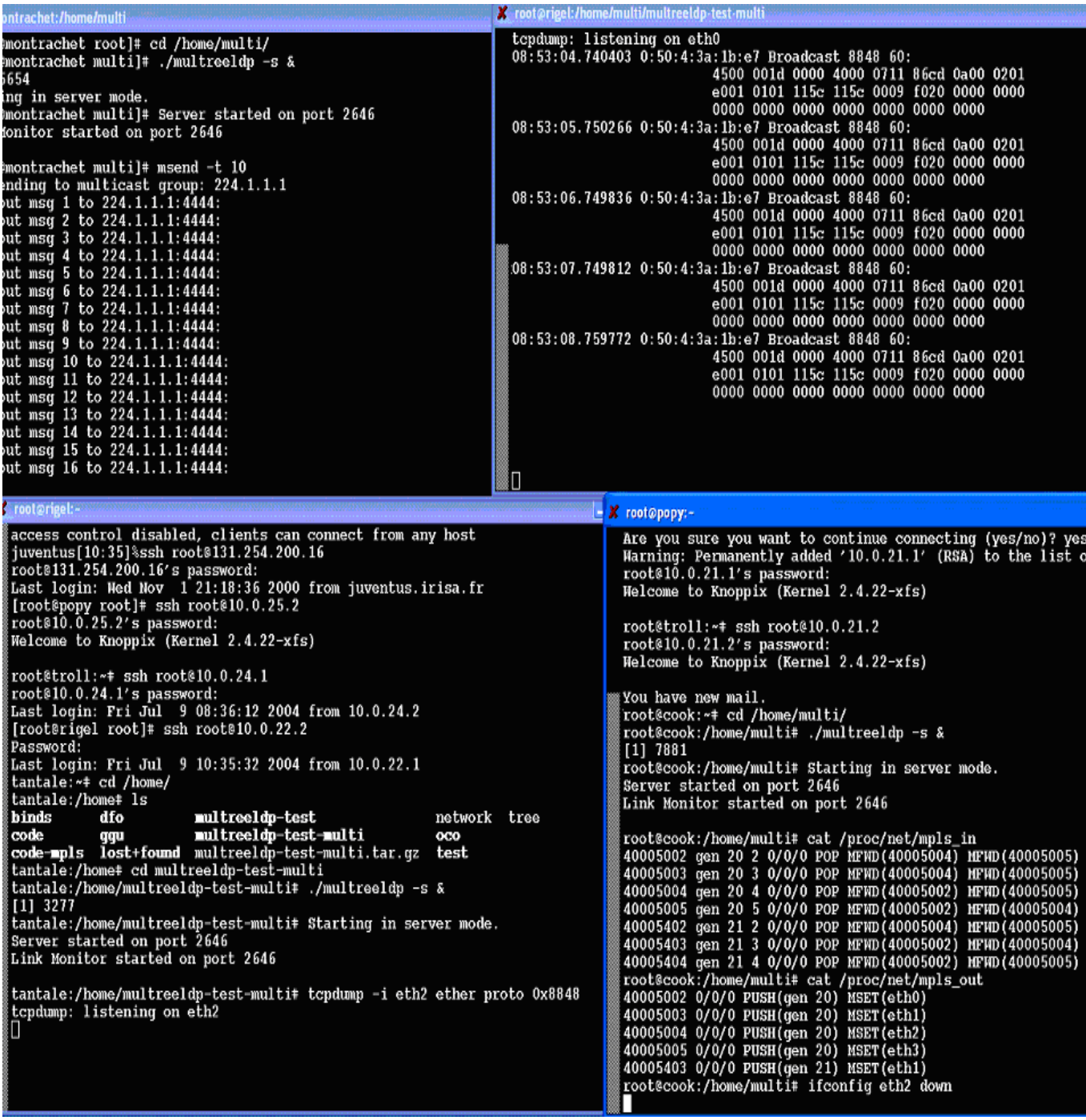
La figure suivante montre le ‘switchback’ suite au recouvrement du lien reliant cook et leone. Le trafic, redirigé par cook sur le chemin initial, est reçu par leone. On voit aussi sur tantal une coupure dans la réception du trafic via l’interface eth2. Le quatrième écran

montre la table MPLS d'entrée sur cook qui est changée de façon à prendre en considération le recouvrement du lien.



Dans notre dernier test, on va générer du trafic sur montrachet, et on va l'attendre sur rigel. On va couper une interface sur l'autre chemin protégé. Comme on a mentionné

auparavant, un arrêt de réception va avoir lieu sur rigel après la coupure de cette liaison. Avant la coupure de la liaison, on voit la réception du trafic sur rigel (écran haut en droite). Après la coupure de l'interface eth2 de cook, on voit un arrêt de réception sur rigel, et en plus, pas de réception sur le chemin de réserve formé par les liens joignants cook – tantal - rigel et dont l'interface eth2 de tantal fait partie. La table MPLS d'entrée du cook reste inchangeable suite à cette coupure comme rien ne se passe.



Debbugage du code :