La commutation Ethernet
Les réseaux virtuels

BOURRELIER François *GI04*
FERRARA Arnaud *GI04*
GUILBERT Laurent *GI05*
HUANG Mairaeva *GI04*
LOURD Rodolphe *GI05*
SKORA Ludovic *GI05*
VINZIA Sébastien *GI05*
VIRGA Jérémy *GI04*

Introduction

Ce TP a pour but d'observer les possibilités d'un commutateur Ethernet (switch) et notamment de voir l'une de ses fonctions avancées qui est la mise en place de réseaux virtuels ou VLAN.

Un réseau virtuel est composé d'hôtes physiquement interconnectés et dont les possibilités de communication sont contrôlées. Ces contrôles sont réalisés par des filtres appliqués sur les trames qui traversent le commutateur. Ce filtrage permet de définir des réseaux virtuels qui intègrent généralement des contraintes de sécurité. De plus, le commutateur, tout comme le ferait un pont, améliore la bande passante en limitant les domaines de broadcast.

Remarque : le matériel à utiliser (*commutateur ethernet Hewlett-Packard ProCurve 2424M*) n'étant pas disponible, on utilise une machine disponible à la place, en tant que switch.

1. Mise en place du réseau, utilisation du switch

1.1. Mise en place de l'adressage des machines

On configure les machines avec les adresses suivantes :

- *pc-gi-46 : 192.168.0.222 (eth0)*
- *pc-gi-47 : 192.168.0.100 (eth0/eth1 > switch)*
- *pc-gi-48 : 192.168.0.6 (eth0)*

1.2. Management du switch

La fonction de commutateur est remplie par la seconde machine configurée en tant que telle grâce à l'annexe « Bridging Howto » fournie.

1.3. Mode d'apprentissage des adresses MAC

Pour visualiser les adresses MAC des interfaces réseau du pont et externes au switch, qui sont utilisées pour le routage, nous utilisons la commande :

 # brctl showmacs pont

Cette table d'adressage est très utile et nécessaire ; elle permet en effet au pont ici créé de faire clairement la distinction entre les machines situées de part et d'autres de celui-ci. Elle est également régulièrement mise à jour dès qu'une machine du réseau se connectent ou déconnectent.

Une fois le pont activé, on remarque dans Ethereal l'apparition de trames STP ; ces trames correspondent au protocole 802.1Q. STP sert à détecter les changements de tables (*résilience du lien*), quand d'autres machines se connectent ou déconnectent par exemple ; aussi des telles trames sont constamment envoyées, même en cas d'inactivité du réseau. De même, lorsque nous pingons les machines entre elles, nous observons des trames ARP, pour la résolution de protocole.

2. VLAN non taggé

Par défaut, tous les ports du commutateur appartiennent au même groupe de travail. On le vérifie en faisant un ping sur n'importe quelle adresse IP d'un autre groupe.

On peut créer de tels groupes ; il suffit de cliquer sur le bouton ADD/REMOVE VLAN dans l'onglet Configuration > VLAN. Pour l'ID 802.1Q, on peut mettre n'importe quel nombre, tout en veillant à ce qu'il soit différent de ceux existant et à le noter car il sera utilisé par la suite.

Ensuite on crée un groupe de travail pour un des ports uniquement et on analyse le trafic sur la machine associée à l'aide de **tcpdump**. En pingant depuis l'autre machine on vérifie que les trames n'atteignent pas l'autre port. Après analyse des fichiers log, on observe que les broadcast ARP, générés par un ping d'une machine, d'un VLAN n'atteignent pas les machines du second VLAN.

L'intérêt de la mise en place de plusieurs VLAN permet de diviser efficacement le réseau en plusieurs sous-réseaux sans que ceux-ci interfèrent entre eux. Le trafic est ainsi mieux réparti, l'efficacité du réseau augmentée.

Pour créer une telle configuration, il faut prendre garde à ne pas donner les mêmes classes d'adresse IP. En effet, lors de l'accès à la table d'adresses du switch, on rencontre une difficulté à distinguer les deux sous-réseaux s'ils ont les mêmes adresses IP : on arrive alors à ne communiquer qu'avec l'un des deux sous-réseaux.

3. VLAN tagging IEEE 802.1Q

Le standard IEEE 802.1Q définit un format de trame spécifique pour permettre la diffusion de l'information d'appartenance à un VLAN. Il définit 3 types de trames :

- *trame UNTAGGED : elle ne contient pas de champ Tag après l'adresse source MAC*
- *trame TAGGED : contient un champ Tag qui suit immédiatement l'adresse source MAC*
- *trame PRIORITY-TAGGED : transporte un champ Tag qui contient des informations de priorité, mais qui contient l'identification du VLAN NULL soit 0*

Le format du Tag est divisé ainsi :

- *TPID (Tag Protocol Identifier) : il transporte le 802.1Q TagType égal à 81-00 qui identifie que la trame est taggée*
- *UP (User Priority) : il identifie les 8 niveaux de priorité définis dans le standard IEEE 802.1P*
- *CFI (Canonical Format Indicator) : pour le format Ethernet, si ce bit vaut 1 cela signifie qu'un champ E-RIF (Enhanced Routing Information Field) est présent après le VID*
- *VID (VLAN Identifier) : champ de 12 bits, il identifie le numéro de VLAN parmi 4096 (exception faite de 000, 111 et 001)*

Avec un module intégré sur la version de Linux Debian utilisée, on peut tagger des trames Ethernet en leur rajoutant le tag 802.1Q. Ceci permet d'utiliser l'option Tagged dans le choix de VLAN par port. Sur une machine, on tagge un port par la commande :

 # vconfig add eth0 20

Cette commande crée une nouvelle interface logique avec l'ID de VLAN correspondant. On active l'interface correspondante avec la commande :

 # ifconfig eth0.20 192.168.1.12 netmask 255.255.255.0 up

On peut vérifier le nom et les caractéristiques de l'interface au préalable (tant qu'elle n'est pas up) en tapant **ifconfig –a**. Pour deux ports, on sélectionne un VLAN en Tagged, puis on pinge les machines entre elles. En activant le tag uniquement sur le switch, les machines reçoivent les paquets du switch mais ne les comprennent pas ; il faut en effet activer le tagging sur les machines utilisées afin d'établir correctement la communication.

Avec **tcpdump**, les trames sont analysées sur les deux machines :

 # tcpdump –vvv –n ip src 192.168.1.11

Si l'on met en place le tagging sur l'autre machine avec le même numéro de VLAN (tag), les paquets sont correctement interprêtés et la communication est possible. Le principal intérêt de la norme IEEE 802.1Q et du VLAN tagging est de pouvoir séparer les réseaux grâce au tagging sur les frames et/ou à l'aide de plusieurs VLAN. Un des inconvénients est que cette séparation n'est pas physique ou matérielle, donc il sera toujours possible de capter ces trames puisqu'il suffit d'avoir le bon numéro (facile à trouver par un scan rapide de 0 à 4095).

Conclusion

Ce TP nous a permis de voir un aspect intéressant d'un réseau local : le VLAN. Grâce à cet outil, il possible de regrouper des machines selon un niveau logique et non physique, qui est plus difficile à mettre en œuvre. Le VLAN permet de définir un nouveau réseau au-dessus du réseau physique et à ce titre offre les avantages suivants :

- *plus de souplesse pour l'administration et les modifications du réseau car toute l'architecture peut être modifiée par simple paramétrage des commutateurs*
- *gain en sécurité car les informations sont encapsulées dans un niveau supplémentaire et éventuellement analysées*
- *réduction de la diffusion du trafic sur le réseau*

Associé au Tagging, ce type d'architecture réseau est assez pratique mais reste néanmoins vulnérable aux attaques puisque le tag semble pas trop difficile à trouver pour une personne spécialisée dans l'administration réseau.

ANNEXE : Bridging Howto

Getting the software

Bridging is supported in the current 2.4 (and 2.6) kernels from all the major distributors. The required administration utilities are in the bridge-utils package.

You can also build your own up to date version by getting the latest kernel from kernel.org and build the utilities based on the code on the downloads page.

Setting Up The Bridge

Network cards

Before you start make sure both network cards are set up and working properly. Don't set the IP address, and don't let the startup scripts run DHCP on the ethernet interfaces either. The IP address needs to be set after the bridge has been configured.

The command "ifconfig" should show both network cards, and they should have be DOWN.

Module loading

In most cases, the bridge code is built as a module. If the module is configured and installed correctly, it will get automatically loaded on the first **brctl** command.

If your bridge-utilities have been correctly built and your kernel and bridge-module are OK, then issuing a **brctl** should show a small command synopsis.

```
# brctl
Bridge firewalling registered
commands:
        addbr           <bridge>                add bridge
        addif           <bridge> <device>       add interface to bridge
        delbr           <bridge>                delete bridge
        delif           <bridge> <device>       delete interface from
bridge
        show                                    show a list of bridges
        showmacs        <bridge>                show a list of mac addrs
        showstp         <bridge>                show bridge stp info

        setageing       <bridge> <time>         set ageing time
        setbridgeprio   <bridge> <prio>         set bridge priority
        setfd           <bridge> <time>         set bridge forward delay
        setgcint        <bridge> <time>         set garbage collection
interval
        sethello        <bridge> <time>         set hello time
        setmaxage       <bridge> <time>         set max message age
        setpathcost     <bridge> <port> <cost>  set path cost
        setportprio     <bridge> <port> <prio>  set port priority
        stp             <bridge> <state>        turn stp on/off
```

Creating a bridge device

The command

brctl addbr *bridgename*

creates a logical bridge instance with the name *bridgename*. You will need at least one logical instance to do any bridging at all. You can interpret the logical bridge being a container for the interfaces taking part in the bridging. Each bridging instance is represented by a new network interface.

Deleting a bridge device

The corresponding "shutdown" command is:

brctl delbr *bridgename*

Adding devices to a bridge

The command

brctl addif *bridgename device*

adds the network device *device* to take part in the bridging of **bridgename**. All the devices contained in a bridge act as one big network. It is not possible to add a device to multiple bridges or bridge a bridge device, because it just wouldn't make any sense! The bridge will take a short amount of time when a device is added to learn the Ethernet addresses on the segment before starting to forward.

Deleting devices from a bridge

The corresponding command to take an interface out of the bridge is:

brctl delif *bridgename device*

Showing devices in a bridge

The **brctl show** command gives you a summary about the overall bridge status, and the instances running as shown below:

```
# brctl addbr br549
# brctl addif br549 eth0
# brctl addif br549 eth1
# brctl show
bridge name     bridge id               STP enabled     interfaces
br549           8000.00004c9f0bd2       no              eth0
                                                        eth1
```

Once a bridge is running the **brctl showmacs** will show information about network addresses of traffic being forwarded (and the bridge itself).

```
# brctl showmacs br549
port no mac addr                is local?       ageing timer
  1     00:00:4c:9f:0b:ae       no                17.84
  1     00:00:4c:9f:0b:d2       yes                0.00
  2     00:00:4c:9f:0b:d3       yes                0.00
  1     00:02:55:1a:35:09       no                53.84
  1     00:02:55:1a:82:87       no                11.53
  1     00:02:b3:09:eb:de       no                43.12
  1     00:02:b3:11:d4:8d       no                 2.66
  1     00:02:b3:11:e1:1e       no                13.19
  1     00:02:b3:11:e9:ad       no                26.67
  1     00:02:b3:11:f0:ab       no                 0.66
  1     00:02:b3:11:f1:99       no                35.34
```

```
  1     00:02:b3:3d:d1:08       no                12.80
  1     00:02:b3:3d:d1:4b       no                21.76
  1     00:02:e3:00:08:ff       no                43.12
  1     00:02:e3:00:0c:19       no                46.68
  1     00:03:47:4c:4b:4c       no                40.86
  1     00:03:93:73:cd:dc       no                38.91
  1     00:04:80:26:b9:60       no                 0.85
  1     00:04:80:29:5a:00       no                10.22
  1     00:08:83:dc:38:e7       no                55.61
  1     00:09:6b:58:c6:1d       no                16.53
  1     00:30:c1:ad:70:c5       no                13.68
  1     00:b0:d0:b0:63:ca       no                 9.29
  1     00:c0:a8:7b:b6:14       no                64.16
  1     00:d0:b7:a9:3f:b8       no                 4.91
  1     00:d0:b7:a9:44:60       no                 0.02
```

The aging time is the number of seconds a MAC address will be kept in the forwarding database after having received a packet from this MAC address. The entries in the forwarding database are periodically timed out to ensure they won't stay around forever. Normally there should be no need to modify this parameter, but it can be changed with:

```
brctl setageing time
```

Time is in seconds.

Spanning Tree Protocol

If you are running multiple or redundant bridges, then you need to enable the Spanning Tree Protocol (STP) to handle multiple hops and avoid cyclic routes. More information about on the STP page.

Enabling STP is done via:

```
brctl stp bridgename on
```

You can see the STP paramaters with:

```
# brctl showstp br549
br549
 bridge id              8000.00004c9f0bd2
 designated root        0000.000480295a00
 root port                 1                    path cost                104
 max age                  20.00                 bridge max age       200.00
 hello time                2.00                 bridge hello time     20.00
 forward delay           150.00                 bridge forward delay  15.00
 ageing time             300.00                 gc interval            0.00
 hello timer               0.00                 tcn timer              0.00
 topology change timer     0.00                 gc timer               0.33
 flags


eth0 (1)
 port id                8001                   state            forwarding
 designated root        0000.000480295a00      path cost               100
 designated bridge      001e.00048026b901      message age timer     17.84
 designated port        80c1                   forward delay timer    0.00
```

```
 designated cost           4                    hold timer              0.00
 flags

eth1 (2)
 port id                8002                    state                disabled
 designated root        8000.00004c9f0bd2       path cost                 100
 designated bridge      8000.00004c9f0bd2       message age timer       0.00
 designated port        8002                    forward delay timer     0.00
 designated cost           0                    hold timer              0.00
 flags
```

STP tuning

Each bridge has a relative priority and cost. Each interface is associated with a port (number) in the STP code. Each has a priority and a cost, that is used to decide which is the shortest path to forward a packet. The lowest cost path is always used unless the other path is down. If you have multiple bridges and interfaces then you may need to adjust the priorities to achieve optimium performance.

brctl setbridgeprio *bridgename priority*

The bridge with the lowest priority will be elected as the root bridge. The root bridge is the "central" bridge in the spanning tree.

brctl setfd *bridge time*

Set forwarding delay time is the time spent in each of the Listening and Learning states before the Forwarding state is entered.

brctl sethello *bridge time*

Sets the hello time. Every (this number) seconds, a hello packet is sent out by the Root Bridge and the Designated Bridges. Hello packets are used to communicate information about the topology throughout the entire Bridged Local Area Network.

brctl maxage *bridge time*

Sets the maximum message age. If the last seen (received) hello packet is more than this number of seconds old, the bridge in question will start the takeover procedure in attempt to become the Root Bridge itself.

brctl setpathcost *bridge port cost*

Sets the cost of sending a packet on this interface. Faster interfaces should have lower path costs. These values are used in the computation of the minimal spanning tree. Paths with lower costs are likelier to be used in the spanning tree than high-cost paths (As an example, think of a gigabit line with a 100Mbit or 10Mbit line as a backup line. You don't want the 10/100Mbit line to become the primary line there.)

The Linux implementation currently sets the path cost of all eth* interfaces to 100, the nominal cost for a 10Mbit connection. There is unfortunately no easy way to discern 10Mbit from 100Mbit from 1Gbit Ethernet cards, so the bridge cannot use the real interface speed.

brctl setportprio *bridgename port priority*

Sets the priority of ports with eqaul cost. You can use this to control which port gets used when there are redundant paths.

These parameters are only of interest, if you have more than one bridge in your LAN and stp enabled.

Sample setup

The basic setup of a bridge is done like:

1. Zero IP the interfaces. The bridge needs the network devices to be operational, but without TCP/IP running on them.
   ```
   # ifconfig eth0 0.0.0.0
   # ifconfig eth1 0.0.0.0
   ```
2. Create the bridge interface.
   ```
   # brctl addbr mybridge
   ```
3. Add interfaces to the bridge.
   ```
   # brctl addif mybridge eth0
   # brctl addif mybridge eth1
   ```
4. Put up the bridge.
   ```
   # ifconfig mybridge up
   ```

   This will set the host up as a pure bridge, it will not have an IP address for itself, so it can not be remotely accessed (or hacked) via TCP/IP.

   Optionally you can configure the virtual interface mybridge to take part in your network. It behaves like one interface (like a normal network card). Exactly that way you configure it, replacing the previous command with something like: `# ifconfig mybridge 192.168.100.5 netmask 255.255.255.0 up`