Maquette de démonstration Cross-Protect

Jordan AUGE <jordan.auge@francetelecom.com>
Nabil BENAMEUR <nabil.benameur.auge@francetelecom.com>
Abdesselem KORTEBI <abdesselem.kortebi@francetelecom.com>
Sara OUESLATI <sara.oueslati@francetelecom.com>
James ROBERTS <james.roberts@francetelecom.com>

4 avril 2005

1 Présentation de la plate-forme

La plate-forme est constitués de 4 machines. `p-ruche` est configurée en bridge en coupure entre des générateurs et des puits de trafic 1.

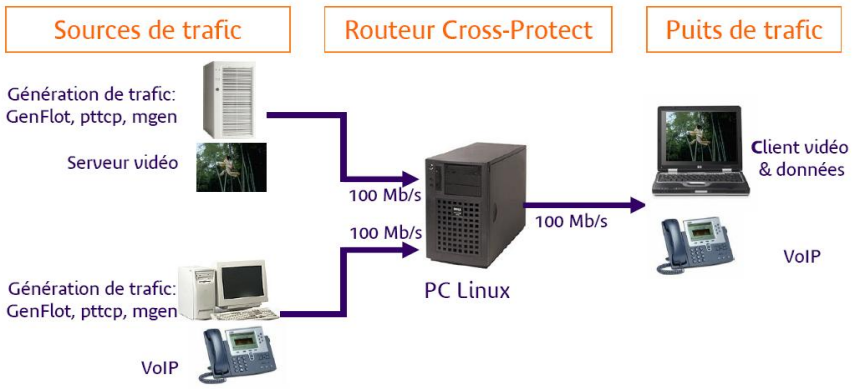

FIG. 1 – Plate-forme d'expérimentation *Cross-Protect*.

Le trafic généré est de trois types :
- du trafic de données,
- du trafic streaming vidéo,
- du trafic streaming audio.

2 Génération de trafic

2.1 Trafic de données

Le trafic de données est généré à différentes charges à l'aide de `genflot`. Les fichiers à mettre en paramètre sont à disposition. En ce qui concerne les paramètres de trafic, il est possible d'en générer de nouveaux à l'aide du script `make_tcp`, qui prend en paramètres la charge, la capacité du lien et la taille moyenne des flots.

Les scénarii présentés lors de la démonstration correspondent à des charges de 90% et 120%. La surcharge sera générée par l'ajout de trafic à la charge existante. Si la charge est générée par une seule machine, les buffers d'émission de la machine source peuvent saturer, et des pertes se produire. Il conviendra ainsi d'assurer la génération par deux machines.

Les fichiers de paramètres peuvent être générés ainsi :

```
$ make_tcp 100 45 5000
$ make_tcp 100 15 5000
```

La taille du buffer d'émission est laissée inchangée sur les machines générant le trafic. L'option permettant de limiter le nombre de retentatives d'établissement d'une connexion TCP est laissée inchangée (`sysctl -w net.ipv4.tcp_syn_retries=0`).

L'une au moins des deux machines doit posséder un noyau 2.6. Si la seconde posssède un 2.4, il est possible de mettre en évidence l'impact du *fair queueing* en établissant par exemple 30 connexions TCP permanentes entre chaque source et le puit à l'aide de `pttcp`. Sans *fair queueing*, le noyau le plus récent possède une version de TCP plus aggressive et réalise de meilleurs débits. Si l'on introduit du *fair queueing*, le détail des débits réalisés par chaque connexion montre l'équité entre chaque flot.

2.2 Trafic vidéo

Le trafic vidéo est streamé au travers du mécanisme à l'aide de `vlc` 2, 3 :

`File/Open File...`

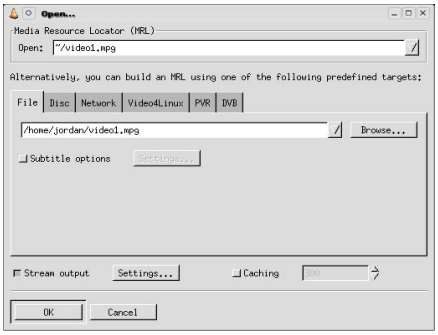

FIG. 2 – Ouverture d'un fichier sous VLC.

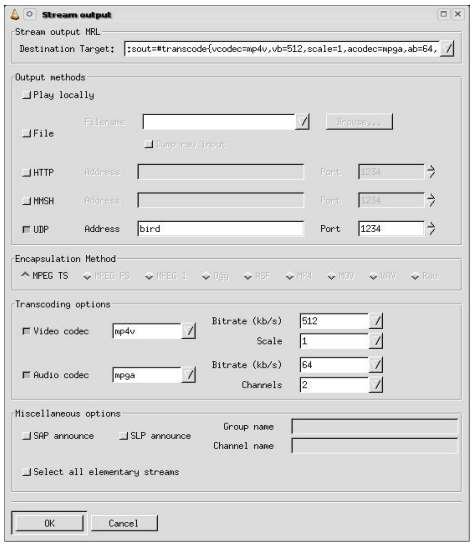

Fig. 3 – Paramètres de streaming VLC (source).

Sur la machine recevant le trafic vidéo, il convient de diminuer la taille du cache 4 :

`Settings/Preferences...`

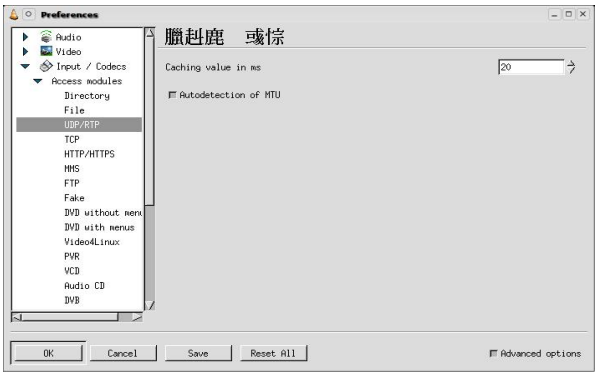

Fig. 4 – Paramètres de streaming VLC (puit).

Il semble important que la machine source soit sous un noyau 2.6, sinon les paquets risquent d'être transmis par bursts. Il serait intéressant d'étudier l'impact que cela a sur le mécanisme *Cross-Protect*.

2.3 Trafic audio

Le trafic audio est généré à l'aide de téléphones IP Cisco 7940. Afin de pouvoir brancher les téléphones sur le routeur Linux, une alimentation externe est nécessaire. Le téléphone se paramètre automatiquement par bootp et dhcp, il récupère notamment l'identificateur du VLAN sur lequel les communications seront transmises, d'où la nécessité d'un bridge afin de gérer le broadcast. Une fois configuré, le téléphone agit comme un switch, il est donc possible de connecter les machines source et puit dessus.

3 Configuration du routeur Linux

3.1 Installation du patch noyau *Cross-Protect*

Le module Cross-Protect est contenu dans les fichiers `sch_xp.c` et `log_table.h`. Ce derniers contient une table de valeur pour le calcul du logarithme utilisé dans l'algorithme *TopN*.

On suppose qu'un lien `/usr/src/linux` pointe vers l'installation courante des sources du noyau Linux.

- Copier `sch_xp.c` et `log_table.h` dans `/usr/src/linux/net/sched`.
- Ajouter les lignes suivantes dans le fichier `/usr/src/linux/net/sched/Makefile` :
  ```
  obj-$(CONFIG_NET_SCH_XP) += sch_xp.o
  ```
- Ajouter les lignes suivantes dans le fichier `/usr/src/linux/net/sched/Kconfig` :
  ```
  config NET_SCH_XP
  tristate "Cross-Protect packet scheduler"
  depends on NET_SCHED
  ---help---
  Cross-Protect packet scheduler
  ```
  Il convient aussi de patcher le driver de la carte réseau de sortie de la plateforme, afin de mettre la valeur du tx_ring la plus faible possible. Les drivers sont situés dans `/usr/src/linux/drivers/net`. Voici les changements qui ont été testés :

fichier	ligne à modifier
3c59x.c	`#define TX_RING_SIZE 4`
pcnet32.c	`#define PCNET32_LOG_TX_BUFFERS 2`

  Il ne reste plus qu'à sélectionner *Cross-Protect packet scheduler* dans la configuration du noyau (`make menuconfig` dans le répertoire `/usr/src/linux`), et de recompiler et installer les modules (`make modules && make modules_install`).

3.2 Patch des outils de configuration user : `tc`

L'utilitaire de configuration `tc` fait partie de l'ensemble `iproute`. Considérons que les sources de la dernière version sont installées dans `/usr/src/iproute2-2.6.9`. Il suffit de rajouter le fichier `q_xp.c` dans le répertoire `/usr/src/iproute2-2.6.9/tc` et de rajouter la ligne suivante dans `Makefile` :

```
TCMODULES += q_cp.o
```

Il suffit alors de compiler le tout et de l'installer.

3.3 Paramètres du module *Cross-Protect*

L'insertion du module dans le noyau se fait par la commande :

```
modprobe sch_xp
```

L'activation du mécanisme sur une interface réseau se fait par l'intermédiaire de l'utilitaire `tc`, avec une syntaxe similaire aux autres disciplines d'ordonnancement (ici l'interface sur laquelle on veut réguler le trafic est `eth1` puisque son driver `pcnet32.c` est celui qui se prête le mieux à la modification de la taille du tx_ring) :

```
tc qdisc add dev eth1 root xp
```

Il est possible d'obtenir un résumé de la syntaxe ainsi :

```
$ tc qdisc add cp help
Usage: ... cp [pfl_limit FLOWS] [pfl_purge_freq MS] [pfl_flow_timeout MS]
              [afl_limit FLOWS] [pifo_limit PACKETS]
              [lc BPS] [wlog LOG_LISSAGE] [debug LEVEL]
              [threshold_fr PERCENT] [delay_fr MS]
              [threshold_pl PERCENT] [delay_pl MS]
              [cac 0|1] [fq 0|1 [pfq 0|1]]
```

Il est ainsi possible de spécifier les paramètres suivants en ligne de commande :

pfl_limit : nombre maximum de flots dans la PFL,

pfl_purge_freq : intervalle entre deux purges de la PFL,

pfl_flow_timeout : durée de vie d'un flot ne recevant plus de paquets dans la PFL,

afl_limit : nombre maximum de flots dans l'AFL,

pifo_limit : nombre de paquets en attente de traitement dans la PIFO,

lc : capacité du lien,

wlog : logarithme en base 2 du coefficient de lissage :

wlog	w	$1-w$
0	1	0
1	0.5	0.5
2	0.25	0.75
3	0.125	0.875
4	0.062	0.938
5	0.031	0.969
...	...	...

debug : niveau de debug : 0 = none, 1 = basic info, 2 = detailed, 3 = full,

threshold_fr : seuil d'admission relatif au *fair rate*,

threshold_pl : seuil d'admission relatif au *priority load*,

delay_fr : intervalle de mise à jour du *fair rate*,

delay_pl : intervalle de mise à jour du *priority load*,

cac : activation du contrôle d'admission,

fq : utilisation du *fair queueing*,

pfq : utilisation du *priority fair queueing*.

Si le contrôle d'admission est activé, mais que le *fair queueing* ne l'est pas, alors l'estimateur utilisé est le *TopN*. Le débit de chaque flot est lissé exponentiellement toutes les *delay_fr* ms. Les valeurs renvoyées pour les indicateurs respectifs *fair rate* et *priority load* sont le débit estimé par le *TopN* ainsi que le taux d'utilisation. Le seuil d'admission utilisé est toujours *threshold_fr*. Le contrôle d'admission n'est activé si besoin qu'au delà d'un certain seuil d'utilisation défini par *threshold_pl*.

Si le contrôle d'admission est activé, ainsi que les *fair queueing*, alors les estimateurs classiques de *fair rate* et de *priority load* sont utilisés.

L'activation du *priority fair queueing* nécessite également celle du *fair queueing*.

Il est possible de changer ces paramètres à la volée à l'aide de :

```
tc qdisc change ... \\
```

3.4 Statistiques

Il est possible d'obtenir la configuration de chaque interface par la commande :

```
tc qdisc (show dev eth1)
```

ainsi que des statistiques détaillées :

```
tc -s qdisc (show dev eth1)
```

Les scripts perl `cpstat` et `cpstatlog` permettent respectivement d'obtenir un affichage permanent à l'ecran de ces valeurs, et d'écrire l'ensemble des résultats dans des fichiers de données, afin de les utiliser dans `gnuplot`. Il est possible de spécifier à `cpstatlog` un paramètres indiquant dans quel répertoire stocker les fichiers de données, faute de quoi il créera un répertoire timestampé avec la date de lancement. Les données sont échantillonées toutes les secondes.

3.5 Configuration des VLANs et du bridging

VLANs Le VLANs 87 est ici réservé pour la téléphonie. Il est recu par le téléphone lors de l'autoconfiguration. Le téléphone taggue avec l'étiquette correspondante de VLAN l'ensemble des paquets qu'il reçoit. Sur le routeur, la création et l'activation se font de la manière suivante :

```
vconfig add eth0 87
vconfig add eth1 87
ifconfig eth0.87 up
ifconfig eth1.87 up
```

bridging On réalise un bridging entre les 3 interfaces du routeur pour le trafic de données :

```
ifconfig eth0 0.0.0.0
ifconfig eth1 0.0.0.0
ifconfig eth2 0.0.0.0
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
brctl addif br0 eth2
ifconfig br0 10.193.163.135 netmask 255.255.255.0 up
```

Les IP des interfaces sont réinitialisées, le bridge `br0` est crée et les 3 interfaces réseau y sont ajoutées.

On procède de même pour les extrémités du VLAN de téléphonie :

```
brctl addbr br1
brctl addif br1 eth0.87
brctl addif br1 eth1.87
ifconfig br1 up
```

Enfin on définit le bridge comme l'interface réseau de la machine :

```
route add -net 10.193.163.0 netmask 255.255.255.0 dev br0
route add default gw 10.193.163.1 dev br0
```

4 Scénarii

4.1 Présentation des scénarii

4.1.1 Scénario 0 : Mise en évidence simple de l'intérêt du *fair queueing*

Comme présenté plus haut, l'utilisation d'un noyau 2.4 sur la machine ne diffusant pas la vidéo permet d'utiliser une version de TCP moins aggressive que celle implémentée sur le noyaux 2.4. Si l'on établit par exemple 30 connexions TCP permanentes à l'aide de `pttcp` entre chaque source et le puit, la bande passante sera répartie inéquitablement à la fois entre chaque machine, mais également entre chaque connexion. En utilisant du *fair queueing*, il apparaîtra clairement dans les statistiques de `pttcp` que ces inégalités disparaissent.

4.1.2 Mise en évidence simple des avantages du *priority fair queueing*

Il suffit de laisser tourner un `ping` entre une machine source et une machine puit. L'insertion du *fair queueing* permettant un RTT variant légèrement autour de quelques millisecondes, alors que le *priority fair queueing* permettra des temps légègement plus faibles et plus stable. L'intêret peut apparaître lorsque plusieurs routeurs consécutifs sont traversée, et notamment si l'on considère des applications sensibles à cette valeur (des jeux en réseau par exemple).

4.1.3 Scénario 1 : Insuffisances d'une FIFO en charge 90%

Les délais qui peuvent être crées sur la maquette ne permettent pas une perturbation suffisante de la vidéo afin d'obtenir un résultat visuel. Les dégradations mises en évidences vont donc être celles dues aux pertes de paquets lors de la saturation du buffer par les connexions TCP.

Pour cela, il convient d'utiliser une taille de buffer de 300 paquets sur l'interface de sortie du routeur (et de même pour l'ordonnanceur) :

```
ifconfig eth1 txqueuelen 300
```

Un script de génération de trafic lance une instance de `gen_flot` sur chaque machine source de trafic, afin de créer une charge de 90%. On constate des dégradations sur la vidéo assez rapidement.

4.1.4 Scénario 2 : Régulation du trafic à l'aide de PFQ

L'activation de PFQ sans contrôle d'admission permet au flux vidéo d'obtenir un débit équitable, un délai fortement réduit, et surtout d'éviter les pertes de paquets. En effet l'algorithme rejette en priorité les flots ayant le backlog le plus important, qui seront dans notre cas les flots de données.

4.1.5 Scénario 3 : Besoin d'un contrôle d'admission en surcharge (120%) pour le bon fonctionnement de PFQ

Si l'on ajoute du trafic afin de se placer en situation de surcharge, PFQ permet de réguler le trafic dans un premier temps, mais l'accumulation des flots dégrade peu à peu la vidéo qui finit par être dégradée.

4.1.6 Scénario 4 : *Cross-Protect*

L'activation de *Cross-Protect* permet de réjeter l'excédent de trafic et permet à la vidéo d'être reçue sans dégradation.

4.1.7 Contrôle d'admission basé sur le *TopN* vs. *Cross-Protect*

Il serait possible de comparer *Cross-Protect* avec les précédentes versions du contrôle d'admission basé sur l'algorithme d'estimation *TopN*. On peut s'attendre à observer le même comportement qu'en charge 90%, c'est-à-dire qu'il ne conviendra pas pour une petite taille de buffer. Cependant les explications à fournir ne sont pas propices à une démonstration, au vu des démonstrations précédentes effectuées sur le prototype *Impact*.

4.2 Scripts de scénario

4.2.1 Génération de trafic

trafic-45pc-bird-5000 `#!/bin/sh`

`./gen -s 500 -H host_bird_5000 -P tcp_45pc_100Mbps_5000ko`

Le même avec le port 8000.

trafic-15pc-bird-5000 `#!/bin/sh`

`./gen -s 500 -H host_bird_5000 -P tcp_15pc_100Mbps_5000ko`

Le même avec le port 8000.

4.2.2 Puits de trafic

```
$ pttcp -r (-B 8000)
```

4.2.3 Routeur

init : Ce script d'initialisation effectue les tâches suivantes :

- Suppression des routes actuellement utilisées,
- Configuration des VLANs et des bridges (voir précédemment),
- Ajout du module `sch_xp` dans le noyau.
- Limitation du buffer à 300 paquets.

scenario1 `#!/bin/sh`

```
interfaces="eth0 eth1 eth2"
extif="eth1"

TC="/usr/src/iproute2-2.6.9/tc/tc"

echo
echo
echo "Scenario 1 : FIFO sans CA"
echo

for i in $interfaces; do
        disc=‘$TC qdisc | grep $i | cut -f 2 -d " "‘
        if [[ $disc != "pfifo_fast" ]]; then
                echo "$TC qdisc del dev $i root $disc"
```

```
                $TC qdisc del dev $i root $disc
        fi
done
```

scenario2

```
#!/bin/sh

interfaces="eth0 eth1 eth2"
extif="eth1"

TC="/usr/src/iproute2-2.6.9/tc/tc"

echo
echo
echo "Scenario 2 : PFQ sans CA"
echo

for i in $interfaces; do
        disc=‘$TC qdisc | grep $i | cut -f 2 -d " "‘
        if [[ $disc != "pfifo_fast" ]]; then
                echo "$TC qdisc del dev $i root $disc"
                $TC qdisc del dev $i root $disc
        fi
done

echo "$TC qdisc add dev eth1 root cp pifo_limit 300 cac 0"
$TC qdisc add dev eth1 root cp pifo_limit 300 cac 0
```

scenario3

```
#!/bin/sh

interfaces="eth0 eth1 eth2"
extif="eth1"

TC="/usr/src/iproute2-2.6.9/tc/tc"

echo
echo
echo "Scenario 3 : PFQ sans CA"
echo

for i in $interfaces; do
        disc=‘$TC qdisc | grep $i | cut -f 2 -d " "‘
        if [[ $disc != "pfifo_fast" ]]; then
                echo "$TC qdisc del dev $i root $disc"
                $TC qdisc del dev $i root $disc
        fi
done

echo "$TC qdisc add dev eth1 root cp pifo_limit 300 cac 0 threshold_fr 2"
$TC qdisc add dev eth1 root cp pifo_limit 300 cac 0 threshold_fr 2
```

scenario4

```
#!/bin/sh

interfaces="eth0 eth1 eth2"
extif="eth1"
```

```
TC="/usr/src/iproute2-2.6.9/tc/tc"

echo
echo
echo "Scenario 4 : Cross-Protect"
echo

for i in $interfaces; do
        disc=‘$TC qdisc | grep $i | cut -f 2 -d " "‘
        if [[ $disc != "pfifo_fast" ]]; then
                echo "$TC qdisc del dev $i root $disc"
                $TC qdisc del dev $i root $disc
        fi
done

echo "$TC qdisc add dev eth1 root cp pifo_limit 300 threshold_fr 2"
$TC qdisc add dev eth1 root cp pifo_limit 300 threshold_fr 2
```

Une interface graphique écrite en ruby/Qt `tc.rb` permet un contrôle des scénarii ainsi qu'une visualisation des statistiques importantes dans une fenêtre graphique.

4.3 Mises en gardes

Le débit fixé dans VLC est un débit moyen. Il est important de vérifier que la vidéo streamée résultante n'aura pas un débit crête supérieur au seuil du *fair rate*. Il est toujours possible auquel cas de changer la valeur du seuil d'admission à 2%.