CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

Grenoble

Rapport de Stage

Métrologie Temps Réel sur un Réseau Gigabit Ethernet

Stage du 17 juin au 31 août 2002

Maîtres de stage
M. Daniel Gueniche
M. Laurent Neiger

Stagiaires
Olivier CHAUSSINAND
Elise GALLENSTEIN
Gaston MASSALA

Remerciements

Nous tenons à remercier tout particulièrement M. Daniel Gueniche et M. Laurent Neiger pour nous avoir accueillis au sein du CRIC et pour tous les conseils judicieux qu'ils nous ont apportés.

Nous tenons également à remercier M. Dominique Fournier du LEPES, pour son aide et sa sympathie.

Table des matières

Introduction

Le CNRS, comme la majorité des entreprises ou des administrations aujourd'hui, s'appuie sur un grand réseau informatique. Or il est difficile pour les administrateurs d'avoir des informations en temps réel sur l'activité de leur réseau.

C'est donc dans ce contexte que Monsieur Daniel Gueniche, responsable du Centre réseau et Informatique Commun du CNRS (CRIC) nous a proposé notre sujet de stage d'application en entreprise, à savoir, l'analyse et la représentation du réseau du CNRS.

Après un bref historique et la description du réseau du CNRS, nous reviendrons sur cette problématique puis nous présenterons le déroulement du projet avec ces trois phases : collecte des données, stockage des données et interface de présentation. Nous finirons par les apports aussi bien sur le plan professionnel que personnel d'un tel stage.

Présentation de l'entreprise

I. Le CNRS

Créé en 1939 par un décret du président de la république Albert Lebrun, alors que la France est déjà entrée dans la deuxième guerre mondiale, le CNRS a pour vocation de regrouper tous les organismes d'états de la recherche fondamentale ou appliquée, et de coordonner les recherches à l'échelle nationale. Aujourd'hui, le CNRS est composé de plusieurs délégations réparties sur tout le territoire français.

Le CNRS Grenoble regroupe plusieurs laboratoires :

- Le laboratoire de cristallographie
- Le Laboratoire d'Etude des Propriétés Electroniques des Solides (L.E.P.E.S.)
- Le laboratoire Louis Neel
- Le Laboratoire d'Electrostatique et des Métaux Diélectriques (L.E.M.D.)
- Le Centre de Recherche sur les Très Basses Températures (C.R.T.B.T.)
- L'institut Max Planck
- Le Service National des Champs Intenses (S.N.C.I.)
- Le laboratoire sur les Matériaux Supraconductivité Semi-conducteurs (MATFORMAG)
- Le Magistère de physique
- Le Laboratoire de physique et modélisation des milieux condensés

Le CRIC fait partie du site du CNRS à Grenoble, appartenant à une grande entité constituée des organismes scientifiques répartis dans une même zone géographique que l'on appelle le Polygone Scientifique.
Il regroupe, en plus du CNRS :

- Le Commissariat à l'Energie Atomique (C.E.A.)
- L'Institut de Biologie Structurale (I.B.S)
- L'Institut des Sciences Nucléaires (I.S.N.)
- L'European Synchrotron Research Facility (E.S.R.F.)
- L'European Molecular Biology Laboratory (E.M.B.L.)
- L'Institut Laue Langevin (I.L.L.)

II. Le CRIC

Le Centre Réseau et Informatique Commun, dirigé par M. Gueniche assisté de M. Laurent Neiger, a pour vocation la gestion globale du réseau du site. Les laboratoires gardent cependant une autonomie. Le CRIC s'occupe donc d'harmoniser les sous réseaux et d'assurer les connexions vers le réseau mondial, c'est ainsi par exemple que tout le trafic WAN de l'ensemble du Polygone Scientifique passe par le CRIC.

A partir de 1997, les besoins en haut débit ont commencé à se faire sentir.
Pour remédier à cela, les différents partenaires de la plaque métropolitaine (CEA, INRIA, ESRF, …) décident de se réunir en 1998 pour chercher une solution durable.
Après consultation de l'offre de France Télécom et celle de la METRO (communauté de l'agglomération grenobloise), cette dernière s'est révélée beaucoup plus intéressante à moyen et long terme : exploitation de tous les cheminements existants tels les égouts, la voirie, le tram pour y passer des fibres optiques nues afin d'établir un réseau haut débit métropolitain.
Le CRIC, en tant qu'élément fédérateur du réseau du site CNRS, représente les laboratoires dans les discussions entre les partenaires. Il connecte l'ensemble des laboratoires CNRS du site à RENATER (Réseau national pour l'enseignement et la recherche) via ce nouveau réseau.

Le CRIC gère la cohérence et veille au bon fonctionnement de l'ensemble, mais il fournit aussi des services communs :

- Serveur de DNS primaire du CNRS.
- Serveur d'accès distant, grâce à une batterie de modems.
- Manager SNMP pour les équipements d'interconnexions généraux.
- Serveur de messagerie : sendmail et POP. Environ 1200 comptes; plusieurs accès par seconde à POP.
- Serveur HTTP Internet et intranet.
- Serveur FTP.
- Développer différentes applications liées aux besoins des utilisateurs, ainsi qu'à la fiabilité du réseau.

Le schéma global de routage du réseau est présenté sur la figure 1.

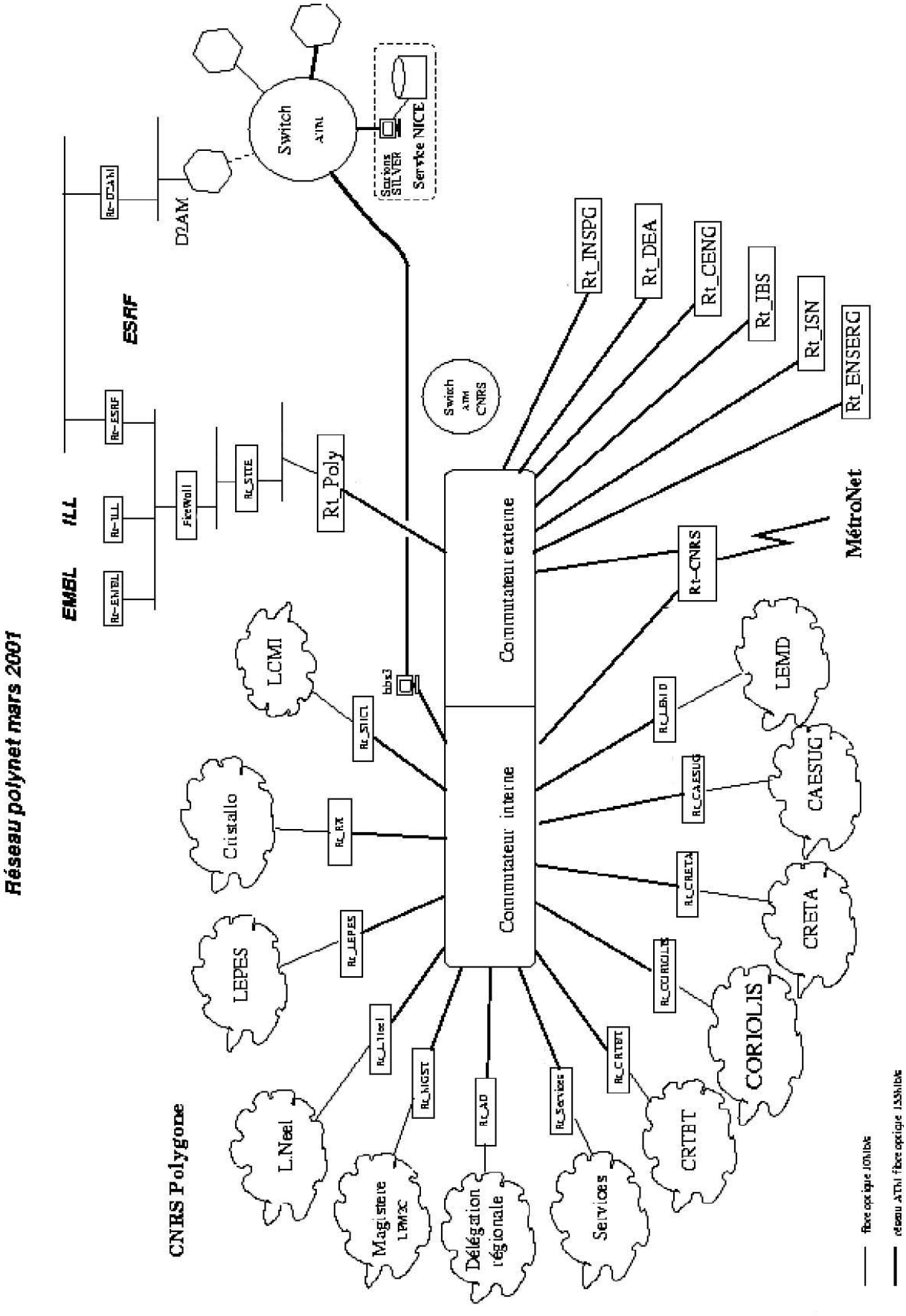

Figure 1: Réseau polynet en mars 2001

Présentation du projet

Le CRIC développe une multitude d'applications liées à l'évolution croissante du réseau informatique, qu'il administre. Ces différentes applications font l'objet des projets proposés aux stagiaires.

Le projet qui nous a été confié concerne la métrologie en temps réel sur le réseau Ethernet Gigabits, actuellement mis en place au polygone scientifique.
L'objectif de ce projet consiste en la mise en œuvre d'une application permettant le suivi du réseau par la visualisation des informations sur l'occupation du réseau en temps réel.
Cette application comporte trois parties :

- ➔ Collecte des informations transitant par le routeur du CNRS.
- ➔ Stockage de celles-ci dans une base de données.
- ➔ Analyse statistique et affichage en temps réel à travers une interface graphique.

Moyens et méthodes mises en oeuvre

La récupération des données transitant par le routeur du CNRS est réalisée grâce à un sniffer. Les données ainsi collectées sont ensuite « parsées » et insérées dans une base de données. Par intervalle de temps déterminé, les données sont chargées dans la base et sont traitées en vue d'obtenir les informations relatives à cette période de temps. Ces informations sont ensuite affichées dans les pages Web, où une alerte est déclenchée lors de l'arrivée d'une donnée anormale.

Déroulement du projet

I. Collecte des données

Cette phase consiste à récupérer toutes les informations transitant par le routeur central du CNRS.
Dans notre étude, les informations pertinentes à connaître concernant chaque flux sont les suivantes :

- Heure de début
- Heure de fin
- Adresse IP source
- Adresse IP destination
- Port destination
- Protocole
- Taille des données

Ce qui permettra d'identifier les 2 laboratoires à qui correspond le flux (la liaison vers l'extérieur du site est assimilé à un labo), de calculer la quantité d'information (en terme de nombre de flux et en nombre de bits) circulant en entrée et sortie de chacun des labos. Ce calcul ne concerne que les flots dits « utiles » correspondants aux protocoles TCP, UDP et ICMP unicast.
Pour réaliser cela, trois techniques différentes ont été essayées.
Durant la phase de développement, pour ne pas perturber le fonctionnement normal du réseau, nous avons testé les diverses solutions sur un sous réseau (figure 2).

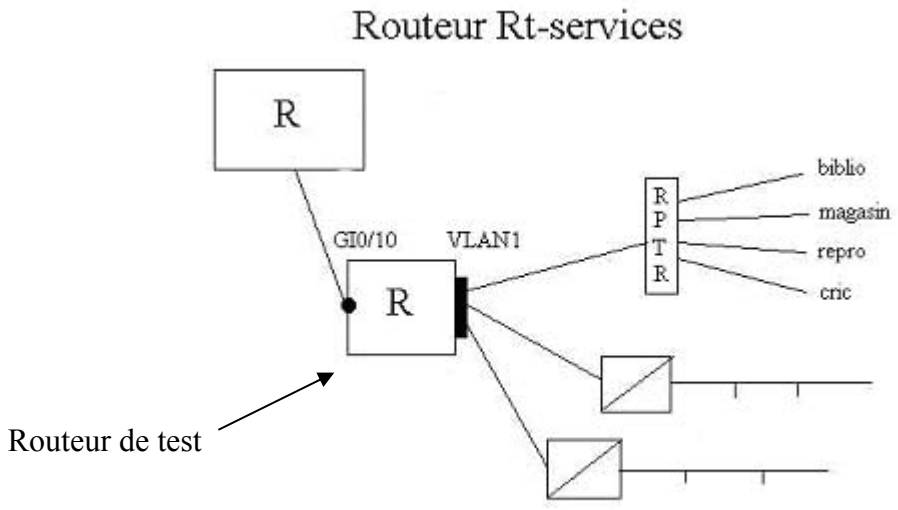

Figure 2 : Sous réseau utilisé pour les tests

Dans un premier temps nous avons utilisé la technique des tickets Netflow embarquée dans les routeurs CISCO, ensuite nous avons mis en place une technique s'appuyant sur le logiciel tcpdump, et finalement nous nous avons écrit un programme basé sur la librairie pcap.

Le calcul de l'occupation réelle du réseau, qui correspond aux types de protocoles déjà cités auxquels s'ajoutent les broadcast et les protocoles moins fréquents, fera l'objet d'une quatrième partie.

A. La technologie Netflow

1. Définition

Netflow est une technique embarquée dans les routeurs et commutateurs routeurs CISCO et a été initialement conçu pour accélérer le passage des paquets dans un routeur.
Netflow est basé sur la notion de flux. Un flux est unidirectionnel et il est défini par :

- une adresse IP source
- une adresse IP destination
- un protocole
- un port source
- un port destination
- une interface d'entrée sur le routeur
- une interface de sortie sur le routeur
- un champ TOS (Type Of Service)

Netflow propose des informations sur les flux en transit dans le routeur ou commutateur routeur. Pour cela il utilise les structures de données manipulées par les mécanismes d'accélération pour construire le cache Netflow.
En effet, dans le routage dit « classique », lorsqu'un paquet entre dans le routeur, celui-ci doit le remonter jusqu'à sa CPU et consulter sa table de routage qui lui indique vers quelle interface rediriger le paquet et il fait cette manipulation pour tous les paquets (figure 3).

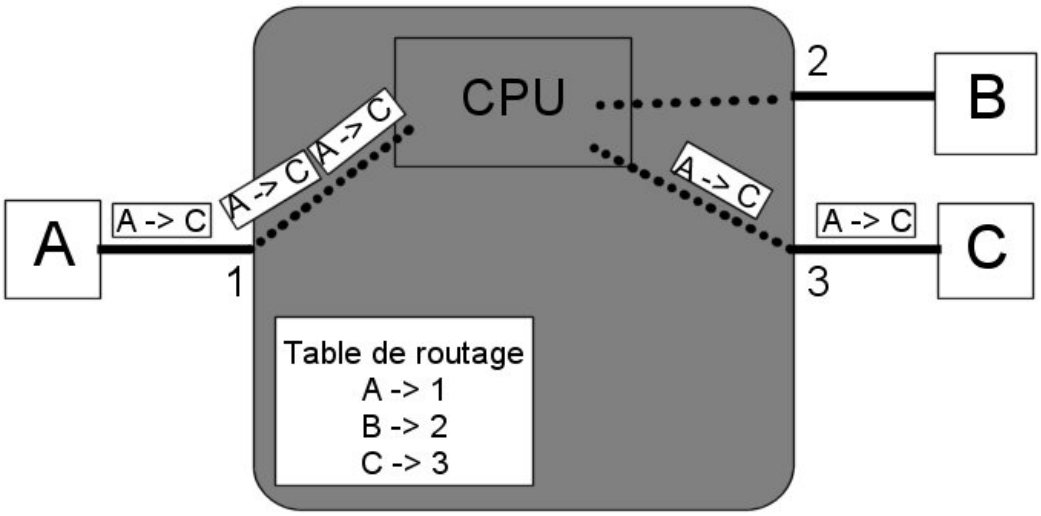

Figure 3 : Le routage classique

Par contre, dans le routage dit « accéléré », lorsqu'un paquet entre dans le routeur, celui-ci consulte son cache Netflow pour savoir si le paquet n'appartient pas à un flux existant.
Si ce n'est pas le cas, le paquet suit la même procédure que dans le routage « classique » et le routeur crée dans son cache Netflow une entrée pour le flux qu'il vient de détecter. Cette entrée indique quel chemin doivent emprunter tous les paquets appartenant à ce flux. Parallèlement, il programme de façon hardware l'acheminement hardware de ce flux entre les deux interfaces (figure 4).

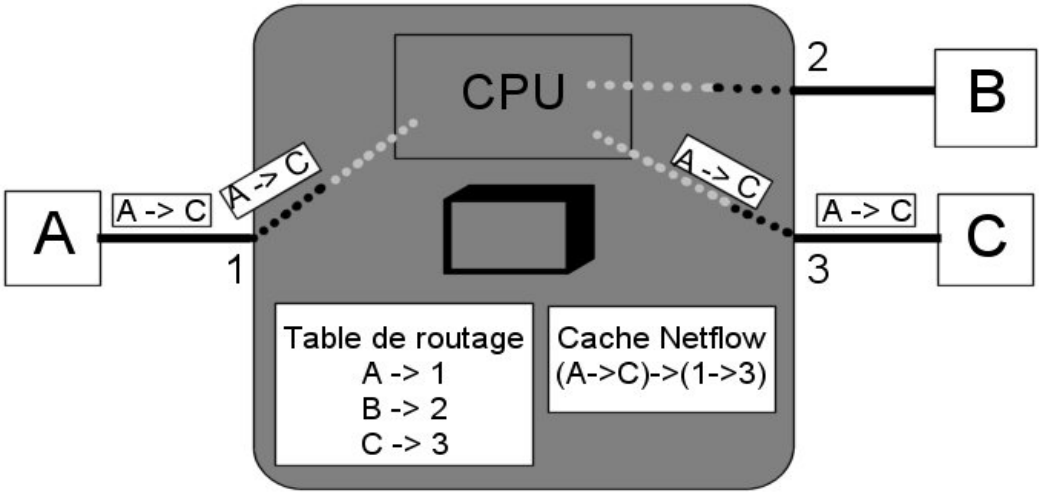

Figure 4 : Le routage accéléré, premier paquet d'un flux

Dans le cas où le paquet appartient à un flux existant, celui-ci ne remonte pas jusqu'à la CPU mais prend le chemin hardware qui a été programmé précédemment et est acheminé directement vers la bonne interface (figure 5).

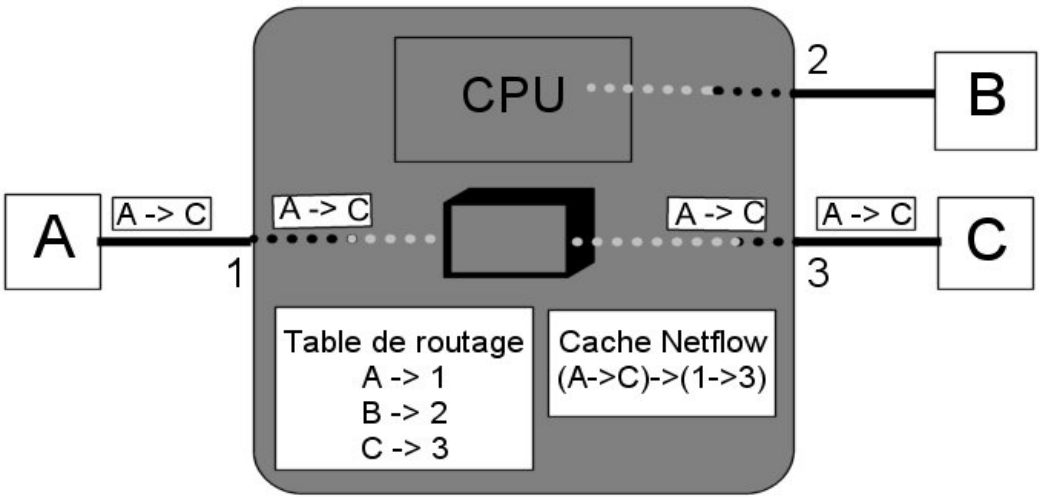

Figure 5 : Le routage accéléré, paquets suivants

Pour récolter les informations sur les flux en cours, il suffit donc d'envoyer le contenu du cache Netflow à une machine de collecte sous la forme de tickets Netflow. Cependant, l'envoi de tickets ne se fait que lorsqu'un flux est terminé, ce qui peut poser problème par exemple dans le cas de protocoles non orientés connexion comme UDP qui ne signale pas quand une communication est terminée. Il existe donc différentes manières de gérer le cache :

- ➔ Terminaison d'un flux après une période d'inactivité (à définir), utile pour les flux UDP
- ➔ Terminaison d'un flux après une période d'activité (à définir), utile pour les longues connexion TCP, telnet par exemple.

Ces deux options sont très utile dans le cas où on souhaite faire de la métrologie en « temps réel ».

2. Mise en œuvre

Le routeur utilisé pour ces tests est le catalyst 3550 qui est commutateur-routeur.
Il a été configuré de la manière suivante :

`ip flow-cache timeout active 5` -> vide le cache toutes les 5 minutes
`ip flow-export version 5` -> exporte les tickets netflow au format 5
`ip flow-export source gi0/10` -> l'interface qui envoie les tickets est x.x
`ip flow-export destination x.x.1.115 8080` -> envoie des tickets à l'adresse 1.115 sur le port 8080

`ip route-cache flow` -> pour toutes les interfaces qu'on veut surveiller (ici Vlan1 et gi0/10)

De plus, nous avons ajouté la ligne :
`no ip route-cache cef`
pour supprimer l'accélération sur le routeur et donc théoriquement voir tous les flux qui le traverse.

on nous a aussi conseillé la commande suivante (disponible sur le Catalyst 6000) :
`mls rp nde-address addr_ip mls flow ip full`
mais cette commande n'est pas reconnue par le Catalyst 3550.

Tests
Nous avons essayé de générer des flux tcp, udp et icmp à partir du réseau 17 sur le réseau 1 et inversement. Les tickets netflow générés par le routeur ne concernent que les 2 interfaces du routeur (y compris les broadcasts). Les flux qui traversent le routeur ne sont pas pris en compte.

3. Conclusion

Ce type de routeur n'est pas officiellement prévu pour exporter du Netflow.
Selon la documentation d'un collecteur de tickets netflow, il est recommandé d'exporter les tickets à la version 7 pour un switch-routeur, ce qui n'est pas possible sur le 3550.
Nous avions cependant essayé de l'utiliser car cette technique était particulièrement adaptée à nos besoins.
Mais, après avoir vainement cherché un équivalent des commandes mls du catalyst 6000 sur le 3550, nous avons finalement décidé de nous réorienter sur une analyse des paquets capturés par tcpdump.

B. Le logiciel Tcpdump

1. Définition

Tcpdump est un logiciel qui permet de capturer les paquets qui transitent par une interface et d'en analyser le contenu. Il met en forme les données reçues par la carte réseau d'une station à différents niveaux (MAC, réseau, transport).
Pour réaliser cela, il doit faire passer l'interface réseau en mode « promiscuous » pour pouvoir la forcer à s'occuper de tous les paquets qui arrivent, y compris ceux qui ne lui sont pas destinés.

Pour chaque paquet, des informations sont affichées à l'écran : les machines source et destination, les ports source et destination, le type de paquet, les données brutes du paquet, etc… En analysant ces informations, on peut reconstituer les données transmises sur le réseau. Comme tous les paquets ne sont pas forcément intéressants, il est possible de définir des filtres pour ne prendre en compte que ceux qui paraissent utiles au phénomène que l'on étudie. On peut filtrer les adresses source ou destination Ethernet; les adresses source ou destination IP de machines ou de réseaux, les numéros de ports, les types de protocole (dans notre cas, seul les paquets de type TCP, UDP et ICMP sont traités).
Tcpdump est un outil qui permet de récupérer toutes les données nécessaires à notre analyse. C'est pourquoi nous avons conçu un parseur, qui lit les sorties de Tcpdump et affiche ces sorties de manière identiques pour tous les protocoles en ne gardant que les informations intéressantes, couplé à un analyseur qui reconstitue les flux et qui gère l'écriture dans la base de données.

2. Mise en œuvre

a. Configuration

Nous avons utilisé la fonctionnalité SPAN (Switch Port ANalyser), fonctionnalité des routeurs CISCO, pour dupliquer et rediriger tous les paquets traversant le routeur. Les commandes à utiliser sont :

```
monitor session <numéro de session> source interface <identifiant de l'interface> both | rx | tx.
```

Où <**numéro de session>** est égal à 1 ou 2,
<identifiant de l'interface> spécifie l'interface à monitorer,
rx pour indiquer que la duplication se fait sur les paquets arrivant sur l'interface
tx pour indiquer que la duplication se fait sur les paquets transmis par l'interface
both pour indiquer que la duplication se fait dans les deux sens

```
monitor session <numéro de session> destination interface <identifiant de l'interface>
```

Où <**numéro de session>** est égal à 1 ou 2,
<identifiant de l'interface> spécifie l'interface sur laquelle rediriger les paquets

Notre configuration est la suivante :

```
monitor session 1 source interface gi0/10 both
monitor session 1 destination interface gi0/09
```

Après cela, le routeur réexpédie tous les paquets sur le port 9 du routeur et il suffit de brancher un PC à l'autre bout et de lancer tcpdump pour retrouver tous les paquets qui circulent dans le routeur.

b. Architecture logicielle

La commande utilisée pour lancer l'analyse est la suivante :

```
tcpdump -nnv udp or tcp or icmp | perl analyseur.pl | ./algo_tcp aa
```

Nous allons maintenant détailler les trois commandes une à une.

i Les options de tcpdump

`tcpdump`

`-n` : empêche la résolution de nom
`-nn` : empêche la résolution des protocole et port
`-q` : affiche les données sous format simple
`-v` : affiche quelques champs de plus pour les paquets
`host nommachine` : affiche les trames en provenance et en direction de nommachine
`udp` : affiche seulement les paquets udp
`icmp` : " " " " icmp
`tcp` : " " " " tcp
`tcp or udp or icmp` : affiche les paquets udp et tcp et icmp
`'tcp[13]&3!=0'` : récupère uniquement les trames SYN et FIN des trames TCP
`- w nomfich` : crée un fichier binaire nomfich contenant les paquets
`- r nomfich` : permet de relire le fichier binaire nomfich créé avec -w

ii Le parseur

Le parseur, écrit en PERL récupère les paquets fournis par tcpdump pour en retirer les informations nécessaires pour l'étude des flots. Il consiste en une analyse syntaxique des sorties de tcpdump. (voir Annexe )

Les informations en sortie sont :
`Num_protocole Date(hh:mm:ss) Adresse_Source Port_Source, Adresse_Destination, Port_Destination, Flag, Taille.`

Pour les paquets icmp on insère dans le champ port destination le type de paquet icmp correspondant.

- echo reply -> 0
- unreachable -> 3
- redirect -> 5
- echo request -> 8
- ...

Pour les paquets udp on récupère simplement les informations souhaitées.
Pour les paquets tcp on doit en plus récupérer les flags.

Exemples :

ICMP:
Entree > 16:14:02.213780 xxx.xxx.1.145 > xxx.xxx.1.15: icmp: host xxx.xxx.148.97 unreachable - admin prohibited filter (ttl 255, id 45345, len 56)
Sortie < 1 16:14:02 xxx.xxx.1.145 0 xxx.xxx.1.15 3 . 56

UDP:
Entree > 16:01:07.823780 xxx.xxx.1.116.53 > xxx.xxx.1.15.1200: 30256 15/2/0 big.oscar.aol.com. A xxx.xxx.210.180, big.oscar.aol.com. (320) (DF) (ttl 64, id 0, len 348)
Sortie < 17 16:01:07 xxx.xxx.1.116 53 xxx.xxx.1.15 1200 . 348

TCP:
Entree > 16:31:10.983780 xxx.xxx.1.15.2929 > xx.xx.180.148.80: S [tcp sum ok]1576163861:1576163861(0) win 5840 (DF) (ttl 64, id 65205, len 60)
Sortie < 6 16:31:10 xxx.xxx.1.15 2929 xx.xx.180.148 80 S 60

iii. L'analyseur

L'analyseur récupère les lignes fournies par le parseur et les traite une à une au fur et à mesure qu'elles arrivent et selon le type de protocole :

- ➔ pour les paquets ICMP, on remplit directement la base (1 paquet = 1 flux)
- ➔ pour les paquets UDP, tous les paquets provenant d'une même adresse source / port source et ayant pour cible la même adresse destination / port destination sont considérés comme venant d'un même flux (mode non connecté).
- ➔ pour les TCP, il est nécessaire de connaître les flags pour détecter le début et la fin d'une connexion (mode connecté).

Il est donc nécessaire d'avoir une structure pour stocker les informations sur les flux en cours dans le cas d'un paquet TCP ou UDP. Cela permet de retrouver à quel flux appartient le paquet en cours de traitement

```
typedef struct {
        char heureDebut[9];
        char heureFin[9];
        char adrSrc[16];
        int portSrc;
        char adrDest[16];
        int portDest;
        int nbOct;
} fluxTCP;

typedef struct {
        char heureDebut[9];
        char heureFin[9];
        char adrSrc[16];
        int portSrc;
        char adrDest[16];
        int portDest;
        int nbOct;
} fluxUDP;
```

Une connexion sera mise en base dans 3 cas :

- ➔ on reçoit un paquet avec le flag F (fin) (cas TCP)
- ➔ on reçoit un paquet avec le flag R (rst) (cas TCP)
- ➔ l'intervalle de temps défini par l'utilisateur entre chaque mise en base s'est écoulé (cas TCP et UDP)

Tests

Sur pc-cric5 (pour limiter les paquets à vérifier), on lance en parallèle notre ligne de commande et tcpdump et on constate qu'aucun paquet n'est perdu.

Tests d'occupation du processeur :

```
Test de dump
     tcpdump normal(à l'écran) : 0.1-2 % débit 10kbits/s
        tcpdump dans fichier(option -r) : pas ou peu de CPU
        tcpdump dans fichier(>> test1) : pas ou peu de CPU
Charge avec ftp d'un gros fichier
        tcpdump normal : charge total 18-22% tcpdump 2-3% debit 7Mo/s
        tcpdump dans fichier(option -r) : 0.5%
        tcpdump dans fichier(>> test1) : 1-2%
```

3. Conclusion

Les problèmes rencontrés à ce stade sont :

- ➔ Il y a un effet de bufferisation entre la sortie tcpdump en temps réel et la sortie de du parseur PERL c'est à dire que le parseur n'affiche rien avant d'avoir reçu une certaine quantité de paquets. Aucune solution n'a été trouvée sur Internet, ni ailleurs.
- ➔ Les options de tcpdump ne sont pas suffisantes selon les protocoles, une option de sortie est meilleure pour tcp et une autre est plus adaptée a icmp et udp.

A ce stade, l'analyse n'était pas complètement satisfaisante. En effet, les programmes fonctionnent correctement et les flux sont bien enregistrés dans la base de données mais le programme ne semble pas totalement optimisé. Il est possible d'adapter l'analyseur pour qu'il capture les paquets lui-même en se servant de la librairie pcap qui est à la base de fonctionnement de Tcpdump.

C. La librairie PCAP

Nous avons finalement décidé de ne pas utiliser Tcpdump mais de s'appuyer sur la libpcap, qui offre des primitives de capture de paquets, en l'intégrant dans notre programme.

1. Définition

La libpcap est donc une librairie qui permet de "capturer" des paquets transitant sur un réseau et de les analyser. Le terme **pcap** signifie "Packet Capturing".
Elle inclut des fonctions de filtrage basées sur le Berkeley Packet Filter (BPF), mécanisme utilisé dans tcpdump, entre autres.
Nous allons détailler les différentes fonctions de la libpcap que nous avons utilisé.

La première étape consiste à choisir l'interface à écouter. On peut le faire manuellement ou laisser le choix à l'utilisateur ou alors utiliser une fonction de la libpcap qui détecte l'interface automatiquement :

```
char *pcap_lookupdev(char *errbuf);
```

où **errbuf** contient le message d'erreur si erreur il y a

Cette fonction retourne un pointeur sur la première interface réseau détectée.
C'est pourquoi, dans le cas où la machine possède deux interfaces, il vaut mieux spécifier manuellement l'interface.

L'étape suivante consiste à récupérer le numéro et le masque de réseau associés à l'interface. C'est le rôle de la commande suivante :

```
int pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32
*maskp, char *errbuf);
```

où **device** est l'interface à écouter
netp est numéro de réseau
maskp est le masque de réseau
errbuf est le message d'erreur s'il y a en une

Il faut maintenant un descripteur pour la capture des paquets qu'on obtient par la fonction

```
pcap_t *pcap_open_live(char *device, int snaplen, int
promisc, int to_ms, char *errbuf);
```

où **device** est l'interface à écouter
snaplen est le nombre maximal d'octets à capturer
promisc permet de dire si l'interface est en promiscuous mode
to_ms est le timeout des lectures
errbuf est le message d'erreur s'il y a en une

Il reste à appliquer des filtres de capture (dont la syntaxe est identique à celle des filtres utilisés par Tcpdump) mais il faut d'abord les traduire en hexadécimal par la fonction :

```
int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int
optimize, bpf_u_int32 netmask)
```

où **p** est le descripteur obtenu avec pcap_open_live()
fp est la structure résultat
str est le filtre écrit sous forme de texte
optimize contrôle l'optimisation du filtre
netmask est le masque obtenu avec pcap_lookupnet()

On peut donc le mettre en place :

```
int pcap_setfilter(pcap_t *p, struct bpf_program *fp)
```

où **p** est le descripteur obtenu avec pcap_open_live()
fp est le filtre obtenu dans l'étape précédente

Finalement il n'y a plus qu'a lire dans le descripteur pour avoir les paquets capturés à l'aide d'une boucle. Cette boucle est réalisée avec la fonction :

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char
*user)
```

où **p** est le descripteur obtenu avec pcap_open_live()
cnt est le nombre de paquets a traiter, -1 pour une boucle infinie
callback est la fonction appelée à chaque paquet capturé
user contient les arguments passés à la fonction callback

C'est dans la fonction **callback** que qu'on réalise le traitement des paquets.

2. Mise en œuvre

Nous avons donc modifié le programme original afin d'intégrer les primitives pcap.
Le programme principal original de notre analyseur a été transformé en sous-routine « callback » appelée par pcap_loop().

L'architecture du programme ainsi que le code source sont disponibles en Annexe

Tests
On le laisse maintenant tourner en permanence sur le collecteur.

3. Conclusion

Les problèmes du chapitre précédent ont disparu avec cette nouvelle version.
En effet il n'y a plus de problème de bufferisation puisqu'il n'y a plus d'intermédiaire entre la capture du paquet et le parseur et les options de tcpdump ne sont plus un problème puisque l'on a directement toutes les informations que l'on souhaite.
Cette solution a été donc conservée et nous l'utilisons pour les tests a plus grande échelle.

D. Occupation réelle du réseau

Les résultats obtenus précédemment ont un rôle important dans la surveillance en temps réel d'un réseau. Cependant il reste une donnée importante qui n'avait pas encore été intégrée dans l'application et qui est d'une égale importance pour permettre d'avoir une idée précise de l'utilisation effective du réseau : c'est le taux de bande passante réellement utilisée à l'instant t.
En effet nos mesures précédentes ne prennent pas en compte tous les octets en circulation mais uniquement les données dites « utiles » c'est à dire celles utilisée par les services de la couche application en particulier. On ne tient absolument pas compte des broadcast ou des paquets de protocole peu fréquents et d'éventuelles données « parasites » qui peuvent perturber le fonctionnement normal d'un réseau.
Pour rendre notre application plus pertinente et plus réaliste, nous avons donc choisit d'utiliser les fonctionnalités du protocole SNMP pour interroger le routeur et connaître le nombre d'octets réel, en entrée et en sortie, transitant par une interface du routeur.

1. Définition

SNMP (Simple Network Management Protocol) permet d'effectuer de la gestion de réseau. Il permet de contrôler un réseau à distance en interrogeant les stations ou les routeurs qui en font partie. Il permet d'obtenir des informations sur leur état, de modifier leur configuration, de faire des tests de sécurité et d'observer différentes informations liées à l'émission de données. C'est le protocole le plus utilisé pour gérer des équipements de réseau

Le SNMP fonctionne avec des requêtes, des réponses et des alertes. Une station envoie des requêtes à un agent et celui-ci doit exécuter la requête et envoyer sa réponse. Il peut aussi y avoir des alertes asynchrones venant des agents lorsqu'ils veulent avertir le station d'un problème

Il existe quatre sortes de requêtes :
- GetRequest : obtenir une variable.
- GetNextRequest : obtenir variable suivante (si existante, sinon retour d'erreur).
- GetBulk : " permet la recherche d'un ensemble de variables regroupées. "
- SetRequest : modifier la valeur d'une variable.

Et deux types de réponses :
- GetResponse : permet à l'agent de retourner la réponse au NMS.
- NoSuchObject : informe le NMS que la variable n'est pas disponible.

Les tables MIB sont des bases de données maintenue par l'agent qui contiennent les informations sur les transmissions de données et sur les composantes de la station ou du routeur, etc. (ex : *uptime*, configuration du routage, état du disque et du port série, nombre de paquets reçus et envoyées, combien de paquets erronés reçus, etc.). Elles contiennent l'ensemble des variables TCP/IP de la station. Ce sont les informations contenues dans ces tables qui sont demandées par la station de gestion afin d'effectuer son travail.
Dans notre application, nous n'utilisons que la table MIB I et uniquement le groupe Interfaces de celle-ci, qui contient des données dynamiques ou statiques sur chaque interface du routeur.

2. Mise en œuvre

Il a fallu configurer le routeur pour qu'il accepte les requêtes SNMP :

A chaque mise à jour de la base, le programme envoie une requête snmp au routeur :

```
snmptable <adresse du routeur> public interfaces.ifTable
```

On récupère le résultat de cette requête grâce à un tube, qui redirige l'affichage écran de la réponse sur un petit parseur qui détermine le nombre d'octets en entrée et en sortie pour chacune des interfaces connectées aux routeurs des laboratoires.
Puis ces résultats sont stockés dans la base de données avec les autres résultats.

3. Conclusion

Cette fonctionnalité est effectivement très intéressante, nous avons pu constater, lors des tests, qu'un réseau en apparence très calme peut être en fait surchargé de données non pertinentes mais qui occupent de la bande passante.

II. Stockage des données

A. Création et accès à la base de donnée

1. Présentation de MySQL

MySQL est un véritable serveur de base de données SQL multi-utilisateur et multi-threaded. SQL est le plus populaire langage de base de données dans le monde. MySQL est une configuration client/serveur ce qui consiste en un serveur daemon mysqld, différents programmes clients et des librairies.
SQL est un langage standardisé qui rend facile le stockage, la mise à jour et l'accès à l'information.
Les principaux objectifs de MySQL sont la rapidité, la robustesse et la facilité d'utilisation.
Bien qu'offrant quelques possibilités en moins que les bases de données Oracle, nous avons choisi d'utiliser MySQL car c'est un logiciel distribué gratuitement.

2. Configuration de MySQL

Notre base de données est implémentée par le serveur de bases de données « MySQL version 3.23.41 ».
Nous avons choisi de créer un nouvel utilisateur dans le serveur MySQL pour éviter d'utiliser le super utilisateur et ainsi pouvoir restreindre ses droits d'accès à la seule base de données utilisée par l'application.
La démarche suivie consiste tout d'abord à ajouter une ligne à la table « user », qui contient la liste des utilisateurs. Cette table permet également de préciser à partir de quelle(s) machine(s) l'utilisateur a accès au serveur MySQL.
Nous avons choisit de créer un utilisateur « netlow » qui peut se connecter au serveur depuis la machine où est installé le serveur.
On doit se connecter en tant que super utilisateur pour taper la requête suivante :

```
    insert into user(host, user, password) values( 'localhost', 'netflow',
password('***')) ;
```

En ajoutant la fonction password(), le mot de passe est stocké de manière cryptée dans la table user.
Ensuite, il faut créer la base de données qu'on souhaite utiliser et l'associer à l'utilisateur créé plus haut. Il faut également spécifier les droits accordés à l'utilisateur sur cette base de données. Dans notre cas, l'utilisateur a tout les droits sur sa base de données.
On tape donc la commande suivante :

```
    insert into db(host, db, user, Select_priv, Insert_priv, Update_priv,
Delete_priv, Create_priv, Drop_priv) values('localhost', 'netflow',
'netflow', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y') ;
```

Maintenant, la connexion à la base de données est réalisée à l'aide de le commande suivante :

```
    mysql -u <login> -p <mot de passe> <base de données>
```

où les 3 paramètres sont définis ci-dessus.

B. Définition des tables

1. Table Flots :

La table Flots contient les informations relatives aux paquets capturés. La clé primaire est numflot.

2. Table Ipreseau :

La table Ipreseau contient les adresses ip associées à chaque laboratoire. Pour les adresses externes une entrée est mise avec pour valeur IP 0.0.0.0. La clé primaire est composée des 4 champs de l'adresse ip.

3. Table Valeurs_labo :

La table Valeurs_labo contient les informations relatives aux valeurs statistiques des différents laboratoires connectés au réseau analysé. La clé primaire est composée du nom du laboratoire et de la date.

4. Table Laboratoires :

La table Laboratoires contient des informations générales sur chaque laboratoire ainsi que l'interface du routeur sur laquelle le routeur du laboratoire est branché, le seuil maximal en terme de nombre de flux et le débit sur ce brin. La clé primaire est le nom du laboratoire.

5. Table Alerte :

La table Alerte contient les informations relatives aux valeurs statistiques des différents laboratoires à chaque fois que ces valeurs déclenche une alerte. La clé primaire est composée du nom du laboratoire et de la date.

6. Table Protocole :

La table Protocole contient la description des protocoles et des applications correspondantes. La clé primaire est le numéro de port.

7. Table Snmp :

La table Snmp contient les informations relatives aux nombres d'octets totaux en entrée et en sorties pour laboratoire.
Nous avons également créé des tables temporaires qui sont utilisées dans les requêtes avec jointure car mysql ne permet pas l'utilisation de vues.

8. Table Flotsrecents :

La table temporaire Flotsrecents contient les informations relatives aux flots du dernier delta de temps entre 2 dumps, remplie au moment des calculs et effacée à la fin de ceux-ci.

9. Table Tablein :

C'est une table temporaire contenant les informations sur les flots en entrée des laboratoires ayant eu au moins une entrée de flots.

10. Table Tableout :

C'est une table temporaire contenant les informations sur les flots en sortie des laboratoires ayant eu au moins une sortie de flots.

11. Table Tableinbis :

C'est une table temporaire contenant les informations sur les flots en entrée des laboratoires pour tous les laboratoires.

12. Table Tableoutbis :

C'est une table temporaire contenant les informations sur les flots en sortie des laboratoires pour tous les laboratoires.

Les requêtes de création des tables ainsi qu'une présentation de leurs structures sont présentes en Annexe. 

C. Requêtes utilisées pour calculer les statistiques

Le remplissage de la table Valeurs_labo, tous les delta-temps entre 2 dumps, s'effectue à l'aide des requêtes suivantes :

On récupère tous les flux reçus depuis le dernier dump et on les stocke dans une table temporaire.

```
INSERT INTO flotsrecents
SELECT numflot, ipsrc1, ipsrc2, ipsrc3, ipsrc4, ipdst1, ipdst2, ipdst3,
ipdst4, taille
FROM flots
WHERE endtime BETWEEN FROM_UNIXTIME(UNIX_TIMESTAMP(date_old) +1) AND
date_new;
```

On associe un laboratoire à chaque adresse IP source, ce qui permet de compter le nombre d'octets et le nombre de flux correspondants au laboratoire. Au cas où l'adresse IP ne correspond pas à un labo, on utilise le labo CICG.

```
INSERT INTO tablein
SELECT IFNULL(nomlabo, \"CICG\"), count(*) flotin, sum(taille) octetin
FROM flotsrecents F LEFT JOIN ipreseau R
ON R.ipsrc1 = F.ipdest1
and R.ipsrc2 = F.ipdest2
and R.ipsrc3 = F.ipdest3
GROUP BY R.nomlabo;
```

On recroise le résultat précédent avec la table Laboratoires pour avoir une entrée pour tous les laboratoires y compris ceux qui n'ont pas eu de flots en entrée.

```
INSERT INTO tableinbis
SELECT distinct(I.nomlabo),flotin, octetin
FROM ipreseau I LEFT JOIN tablein T
ON I.nomlabo=T.nomlabo;
```

On effectue la même opération pour les flots en sortie.

```
INSERT INTO tableout
SELECT IFNULL(nomlabo, "CICG"), count(*) flotout, sum(taille) octetout
FROM flotrecents LEFT JOIN ipreseau R
ON R.ipsrc1 = F.ipsrc1
and R.ipsrc2 = F.ipsrc2
and R.ipsrc3 = F.ipsrc3
GROUP BY R.labo;
INSERT INTO tableoutbis
SELECT distinct(I.nomlabo),flotout, octetout
FROM ipreseau I LEFT JOIN tableout T
ON I.nomlabo=T.nomlabo;
```

On affecte le nombre d'octets en entrée et en sortie sur une interface du routeur au laboratoire correspondant au numéro de port.

```
INSERT INTO snmp
SELECT nomlabo, 8*deltain, 8*deltaout
FROM laboratoires WHERE port = num_port
```

On réunit toutes les valeurs dans la même table (valeurs de sortie, valeurs d'entrée et interrogation snmp).

```
INSERT INTO valeur_labo
SELECT T1.nomlabo, date_new, ROUND(IFNULL(nboctin,0)/delta_tps),
ROUND(IFNULL(nboctout,0)/delta_tps), ROUND(IFNULL(nbflotin, 0)/delta_tps),
ROUND(IFNULL(nbflotout, 0)/delta_tps), ROUND(snmpin/delta_tps),
ROUND(snmpout/delta_tps)
FROM tableinbis T1, tableoutbis T2, snmp S
WHERE T1.nomlabo = T2.nomlabo AND T2.nomlabo = S.nomlabo
```

On supprime toutes les tables temporaires.

```
DELETE FROM flotsrecents;
DELETE FROM tablein;
DELETE FROM tableout;
DELETE FROM tableinbis;
DELETE FROM tableoutbis;
DELETE FROM snmp;
```

Les tables « laboratoires » et « ipreseau » doivent être préalablement remplies pour que le programme puisse fonctionner; une interface Web a été créée pour faciliter la mise à jour de ces tables.

III. Interface graphique

A. Installation et configuration d'Apache

L'interface de l'application doit permettre aux utilisateur d'observer de manière rapide et dynamique les informations collectées et insérées par le sniffer dans la base de données. La mise en œuvre du projet est donc passée par l'installation d'un serveur Web implémentant le PHP.
Les utilisateurs ayant accès à la base de données sont les responsable du CRIC. La consultation de l'interface est donc réalisé en local et pas depuis l'extérieur.
Comme PHP doit impérativement être associé à un serveur Web pour fonctionner, il a fallut choisir quel type de serveur Web installer en fonction du système d'exploitation de la machine destinée à accueillir ce serveur.
Cette dernière étant une machine Linux, notre choix s'est porte vers la référence en la matière: un serveur Apache, gage de fiabilité et dont le code est libre.

1. Configuration du serveur Apache

Pour pouvoir créer notre interface nous avons du configurer un serveur Apache pour qu'il puisse faire fonctionner PHP avec MySQL qui est notre système de base de données. Nous avons aussi ajouté une extension prise en charge par PHP : la bibliothèque GD pour la création et la manipulation d'images.
PHP peut être compilé comme interpréteur indépendant (comme avec tout autre langage de script CGI) ou comme module Apache. Nous avons choisi de compiler PHP en tant que module dynamique Apache en raison de l'amélioration des performance obtenues par ce dernier par rapport à la version CGI.
Afin d'avoir une plus grande liberté dans la configuration de notre serveur nous avons récupéré Apache 1.3.26, PHP 4.2.2 et la bibliothèque GD 2.0.1 sous forme de sources à compiler plutôt que sous forme de package de type RPM, plus facile d'emploi mais figé et peu évolutif.

Nous avons tout d'abord compilé et installé la bibliothèque GD et nous sommes ensuite passés à l'installation du serveur Apache et de PHP.
Après avoir décompressé le fichier source dans un dossier de travail, nous avons lancé le script de configuration du serveur Apache avec les options suivantes :

```
./configure --with-layout=Apache --prefix=/usr/web/server/apache
            --enable-module=mime-magic --enable-shared=mime-magic
            --enable-module=headers --enable-shared=headers
            --enable-module=info --enable-shared=info
            --enable-module=rewrite --enable-shared=rewrite
            --enable-module=speling --enable-shared=speling
```

Puis nous avons réalisé la compilation et l'installation du serveur Apache.

```
Make
Make install
```

Une fois le serveur Apache installé, nous avons construit de même PHP avec la prise en charge d'APXS (*APache eXtenSion* : nouveau support de programme permettant de construire des modules sans recompile Apache) :

```
./configure --with-apxs=/usr/web/server/apache/bin/apxs
            --with-config-file-path=/usr/web/server/php4
            --with-mysql
            --with-gd=/usr/local/apache/htdocs/PHP/gd-2.0.1/
            --with-xml
            --with-jpeg-dir=/usr/lib/
            --with-png-dir=/usr/lib/
            --with-zlib-dir=/usr/include/
            --with-tiff-dir=/usr/local/lib
            --with-ttf-dir=/usr/local
            --with-freetype-dir=/us
            --enable-track-vars
            --enable-magic-quotes
            --enable-debugger
            --enable-gd-imgstrttf
            --enable-gd-native-ttf
```

Suivi de la compilation et de l'installation :

```
Make
Make install
```

Puis nous avons effectué une copie du fichier de configuration de PHP

```
cp php.ini-dist /usr/web/server/php/
```

Il est ensuite nécessaire de modifier le fichier de configuration d'Apache (`/usr/web/server/apache/conf/httpd.conf`), afin que le serveur sache ce qu'il doit faire lorsqu'il rencontre une requête pour un script PHP.
On ajoute un type PHP, ainsi que des directives nécessaires à la version dynamique de PHP :

```
AddType application/x-httpd-php .php .php3
AddType application/x-httpd-php3-source .phps
```

On ajoute une directive qui permet au serveur d'utiliser index.php3 ou index.php comme page par défaut de la même manière que index.html :

```
DirectoryIndex index.html index.shtml index.cgi index.php3 index.php
```

On ajoute une directive afin d'afficher l'icône idoine lorsque l'on examine les fichiers d'un répertoire :

```
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl .php3 .php
.phps
```

Il est ensuite possible de lancer le serveur Apache :

```
/usr/web/server/apache/bin/apachectl start
```

2. Présentation de PHP

Développé en 1994 par Rasmus Lerdorf, PHP (abréviation de PHP Hypertext Preprocessor – Préprocesseur Hypertexte PHP) est un langage de script côté serveur. Cela signifie qu'il fonctionne à l'intérieur d'un document HTML pour lui permettre de générer du contenu dynamique à la demande. Ce langage « Open Source » a amené le développement Web à un nouveau niveau de sophistication. Il est ainsi possible de convertir un site en une application Web, au lieu d'un ensemble de pages statiques plus ou moins actualisées.

PHP a été conçu pour fonctionner sur le Web, ce en quoi il excelle : la connexion et l'interrogation d'une base de données est une tâche simple pouvant être accomplie en deux ou trois lignes de codes. Le moteur de script de PHP est parfaitement optimisé pour les temps de réponses nécessaires à des applications Web, et peut être intégré au serveur Web lui-même pour améliorer encore plus les sorties.

En plus de l'amélioration de la vitesse des script, PHP offre une simplicité d'utilisation, une grande robustesse, une connectivite avec un nombre toujours croissant de serveur Web, la brièveté des cycles de développement et la facilité de création de composants modulaires réutilisables, grâce à la syntaxe et à la construction.

La palette d'applications possible de créer avec PHP est immense : formulaire, système de gestion documentaire fondé sur une base de donnes, système d'aide à la détection de bogues, application de commerce électronique et, enfin, bibliothèques complètes pour un développement rapide et flexible.

La différence principale entre des pages PHP et des pages HTML tient en la façon dont le serveur Web les gère.
Une requête pour une page PHP ajoute une étape supplémentaire à la consultation d'une page HTML. Au lieu de renvoyer une page HTML statique à l'utilisateur, le serveur accomplit certaines actions en fonction du code PHP : PHP doit prendre quelques décisions et créer une page appropriée à la situation. De ce fait les étapes sont les suivantes :

Lecture de la requête du navigateur ;
Identification de la page sur le serveur ;
Exécution des instruction venant de PHP pour modifier la page ;
Envoi de la page au navigateur via Internet (ou intranet).

La différence fondamentale est que le code HTML pur est interprété par le navigateur, et non pas exécuté sur le serveur. En écrivant du code qui sera exécuté sur le serveur Web, il est possible d'accomplir bien plus de choses qu'il n'est normalement possible.

Le traitement et la génération de pages Web côté serveur offrent plusieurs avantages par rapport aux technologies côté client, notamment :

Diminution du trafic réseau en limitant les échanges client/serveur à l'envoi de la requête et de la réponse ;
Réduction du temps de téléchargement : le client ne reçoit qu'une simple page HTML ;
Elimination des problèmes de compatibilité du navigateur ;
Possibilité d'offrir au client des données non présentes chez celui-ci ;
Amélioration des mesures sécuritaires, puisque vous pouvez coder des choses qui ne seront jamais vu par le navigateur.

La librairie GD est une librairie graphique. Elle permet au code de créer rapidement et dynamiquement des images complète au format PNG ou JPEG(mais plus au format GIF depuis la version 1.6, pour des problèmes de droits) comprenant lignes, arcs, textes, couleurs multiple, « copier-coller » à partir d'autres images et diverses structures de remplissage. Elle est très utilisé sur le Web, où le format PNG et JPEG sont deux format accepte comme images en lignes par tous les navigateurs.
Ecrite en C (ANSI), elle peut être utilisée en PHP en compilant les sources du langage avec l'option de configuration –with-gd.

Afin de réaliser facilement des graphiques de statistique nous avons aussi utilise un plug-in PHP appelé Phplot. Phplot est une librairie graphique de scripts PHP pour créer dynamiquement des graphiques scientifiques, commerciaux et statistiques. Compatible PHP3 et PHP4, elle supporte les formats GIF, JPEG, PNG, WBMP selon la version de la librairie GD utilisée et peut inclure les polices TTF si elles sont disponibles. Les graphes générés peuvent de nombreuses sortes : barres 2D/3D, courbes, camemberts, histogrammes...

B. Architecture de l'interface

L'interface de l'application est constituée de nombreuses pages permettant d'avoir différentes visions des résultats obtenus. Ces pages sont installées dans un dossier GEO sur le serveur Apache. L'organisation de l'interface peut être représentée par le schéma suivant :

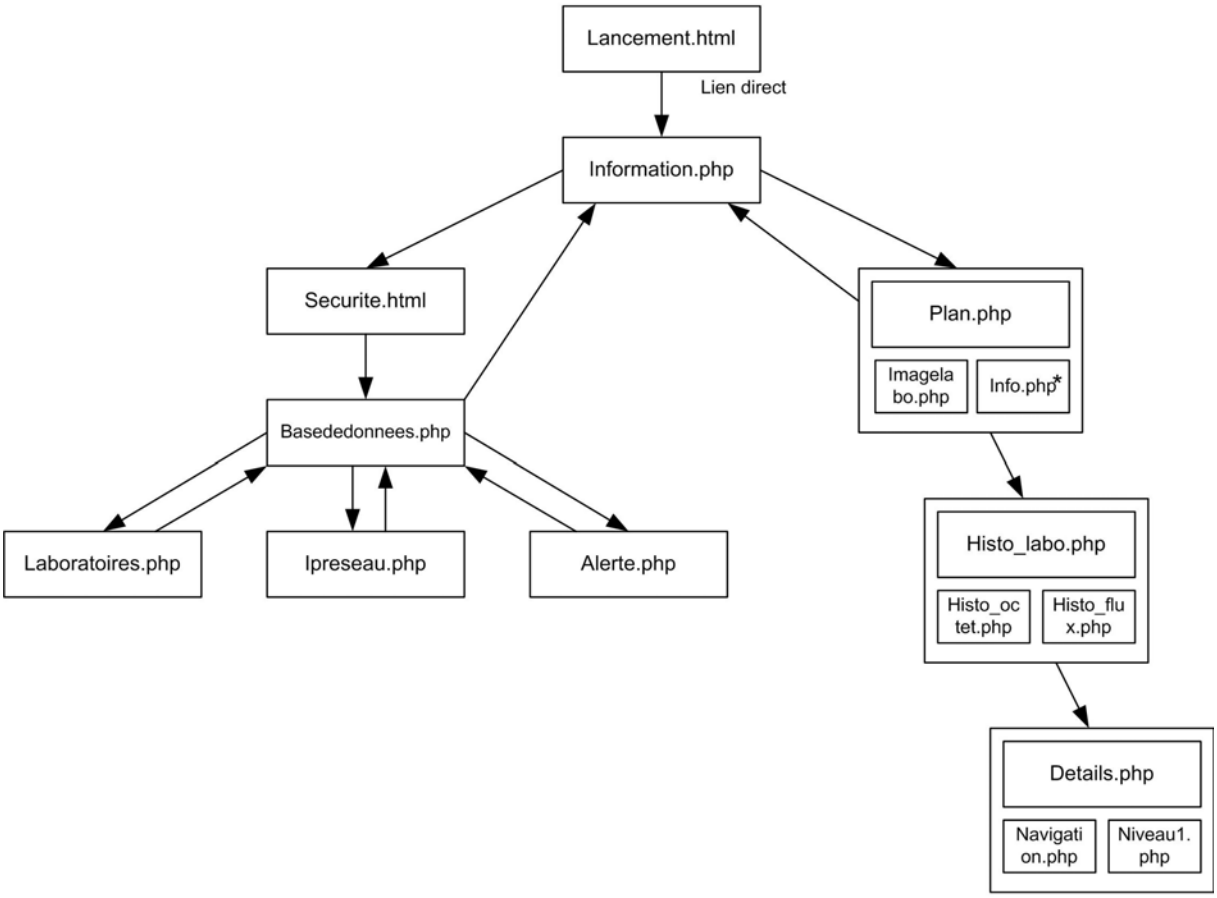

Figure 6 : Schéma de l'architecture de l'interface

La première page **Lancement.html** permet d'avoir accès à toutes les interfaces. Elle n'est pas vraiment une page HTML. Elle permet seulement d'ouvrir la page **Information.php** dans sa propre fenêtre.

La page **Information.php** donne à l'utilisateur les différentes informations nécessaires pour l'utilisation de l'application. Elle permet d'accéder à la présentation des données dans trois interfaces :

- La première permet d'avoir un aperçu rapide de la charge sur le réseau
- La deuxième permet d'avoir un historique de la charge du réseau sur une période de 24h.

➔ La troisième permet une visualisation plus précise des données de la base afin de voir d'où vient le problème.

Un deuxième lien permet d'accéder directement à la base de donnée pour modifier les données.

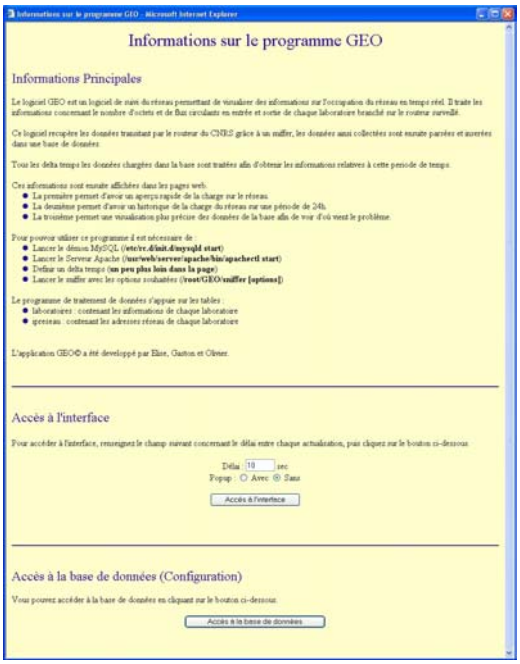

Figure 7 : Aperçu de la page Information.php (copie d'écran en Annexe)

La première présentation **Plan.php**, s'ouvrant dans une nouvelle page, représente les différents laboratoires du site du CNRS de façon géographique. Cette représentation permet d'afficher dans des images dynamique l'occupation du réseau.

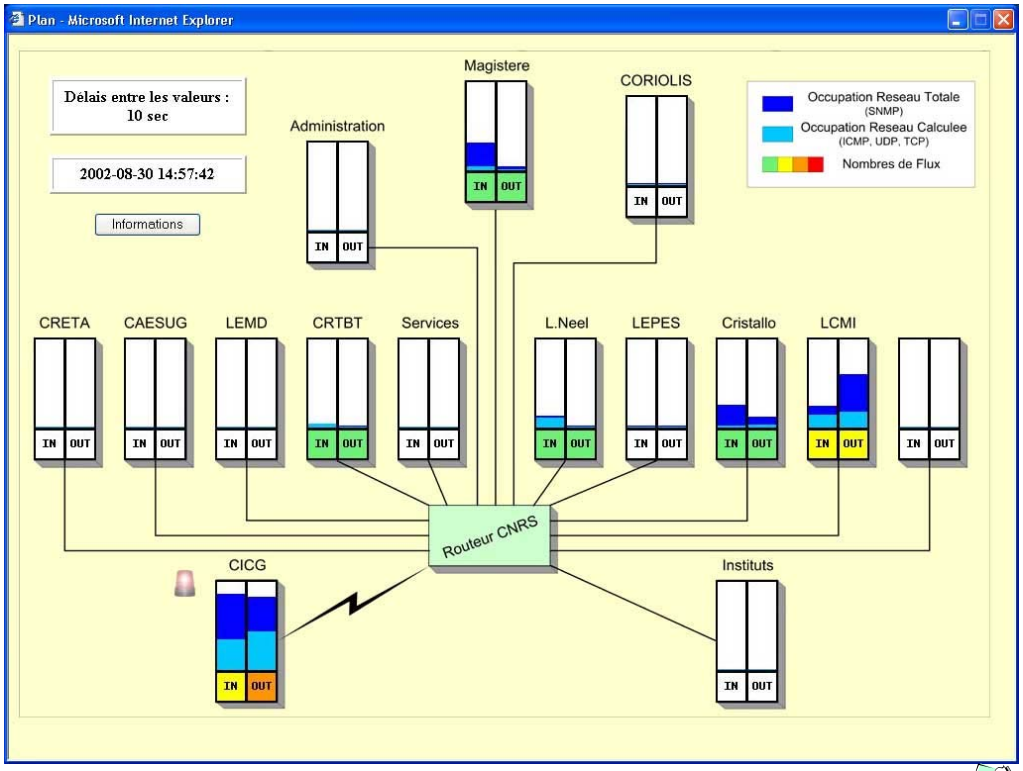

Figure 8 : Aperçu de la page Plan.php (copie d'écran et code en Annexe)

On affiche dans chaque image **Imagelabo.php** le nombre de bits calculé et obtenu avec SNMP ainsi que le nombre de flux en entrée et sortie de chaque laboratoire. Ces informations sont rapportées au débit et nombre de flots maximum indiqués dans la table laboratoires et différents pour chaque laboratoire.

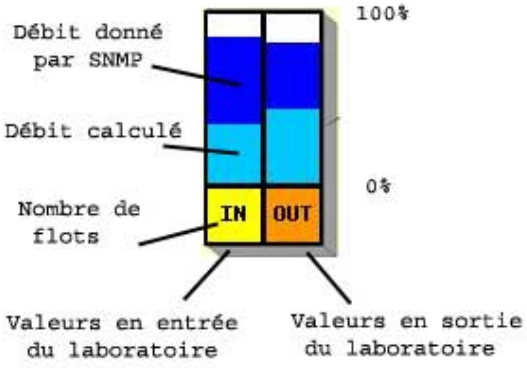

Figure 9 : Présentation des résultats sur chaque laboratoire : Imagelabo.php (code en Annexe)

Si l'occupation du réseau dépasse les 75 % de la bande passante une alerte signalée par un gyrophare s'affiche et une alerte sonore est émise. Cette alerte est créée par l'affichage d'un fichier Flash. Lorsqu'il y a une alerte, les valeurs ayant entraînées l'alerte sont insérées dans la table Alerte.
Si l'option des « popups » a été choisie dans la page Information, lorsque le curseur passe sur un laboratoire une fenêtre (**Info.php**) indiquant les valeurs pour le laboratoire s'ouvre dans un coin de l'écran.
Pour avoir les valeurs en temps réel l'affichage est mis à jour en fonction du temps indiqué dans la page Information au moment de l'accès au plan.

Un clic sur un des laboratoires permet d'accéder à une nouvelle page **Histo_labo.php**. Cette page représente pour chaque laboratoire un historique sur 24h circulantes du nombre de bits et de flots en entrée et sortie de ce laboratoire. Cette page fait appel à 2 images **Histo_octet.php** et **Histo_flux.php** créées dynamiquement grâce au module Phplot.
Cette page étant semblable pour tous les laboratoires une seule page générique a été créée, le laboratoire concerné étant passé en paramètre.

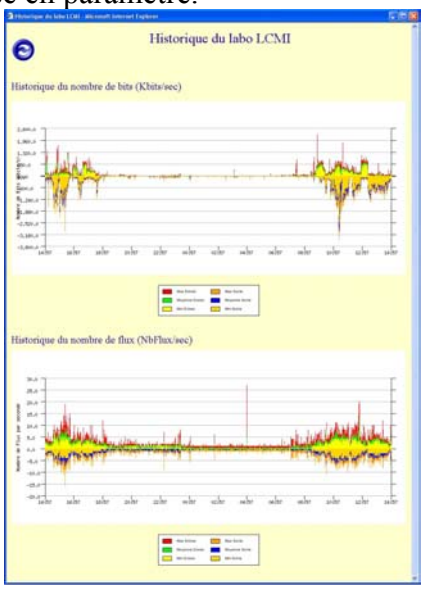

Figure 10 : Aperçu de la page Histo_labo.php (copie d'écran et code en Annexe)

L'historique étant divisé en 144 intervalles, un clic sur un des intervalles permet d'avoir des statistiques sur les 10 minutes correspondantes. La page **Details.php** affiche en effet les flots ayant fait transiter le plus de données en entrée et sortie du laboratoire ainsi les machines ayant réalisées le plus de flux sur cette période. Celle-ci est composé de deux parties, une partie d'affichage **Niveau1.php** et une autre de navigation **Navigation.php** permettant de faire varier l'intervalle de temps des statistiques.

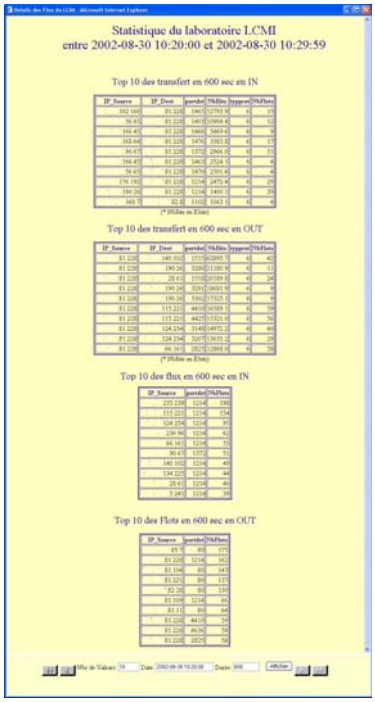

Figure 11 : Aperçu de la page Details.php (copie d'écran en Annexe)

L'autre partie de l'interface correspond à la configuration de la base de données. On y accède par le deuxième lien sur la page d'**Information**. Pour éviter toutes modification intempestive on passe par une page **Securite.html** qui demande le login et le mot de passe. Si le login et le mot de passe sont valides l'utilisateur est dirigé vers la page **Basededonnees** sinon il est renvoyé à la page **Informations**.

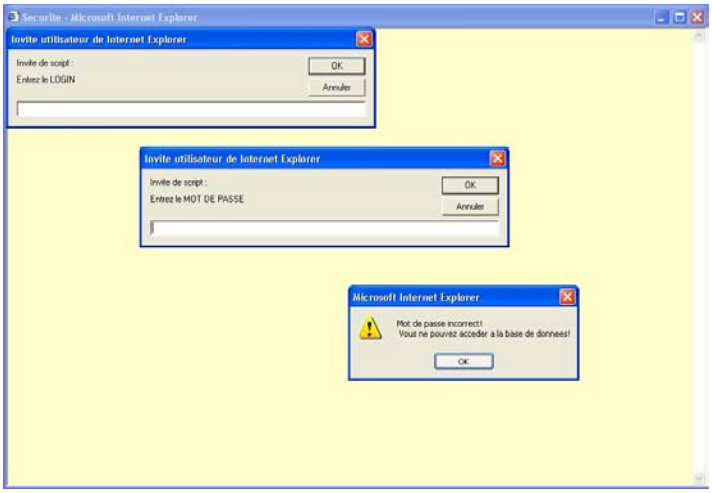

Figure 12 : Aperçu de la page Securite.html (copie d'écran en Annexe)

La page **Basededonnees.php** affiche les informations concernant la base de données utilisée par l'application. L'utilisateur a aussi accès à trois tables :

- ➔ Table Laboratoires
- ➔ Table Ipreseau
- ➔ Table Alerte

Figure 13 : Aperçu de la page Basededonnees.php (copie d'écran en Annexe)

La page **Laboratoires.php** affiche les valeurs de la table laboratoires. Il est possible d'insérer de nouvelles valeurs, d'en supprimer ou de les modifier grâce a un formulaire.

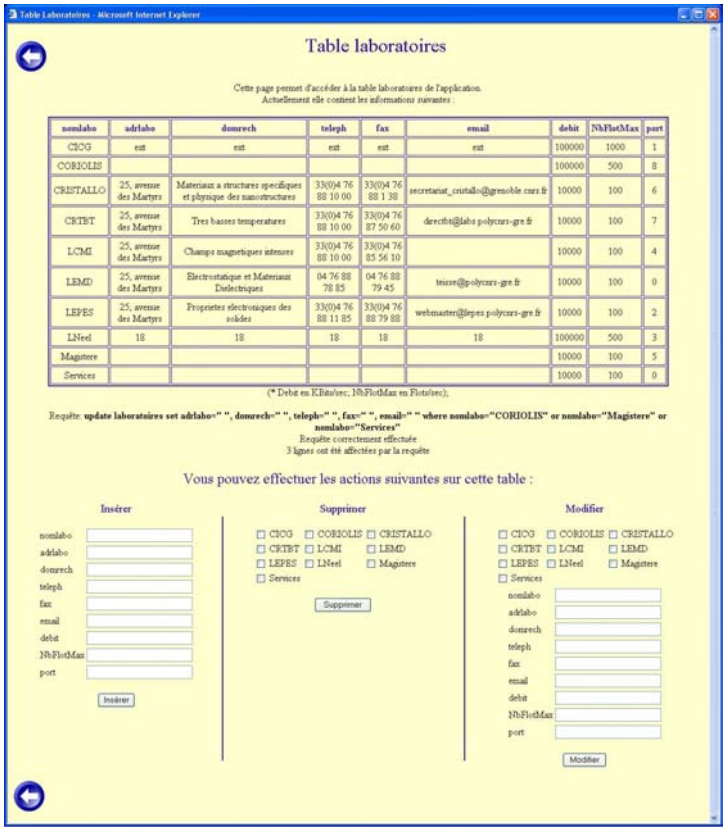

Figure 14 : Aperçu de la page Laboratoires.php (copie d'écran en Annexe)

De même la page **Ipreseau.php** affiche le contenu de la table Ipreseau. Etant donné le nombre important de valeur dans la table Ipreseau, les valeurs ne sont pas affichées par défaut. Il est possible de n'afficher que certaine valeurs, d'en insérer de nouvelles, d'en supprimer ou de les modifier grâce à un formulaire.

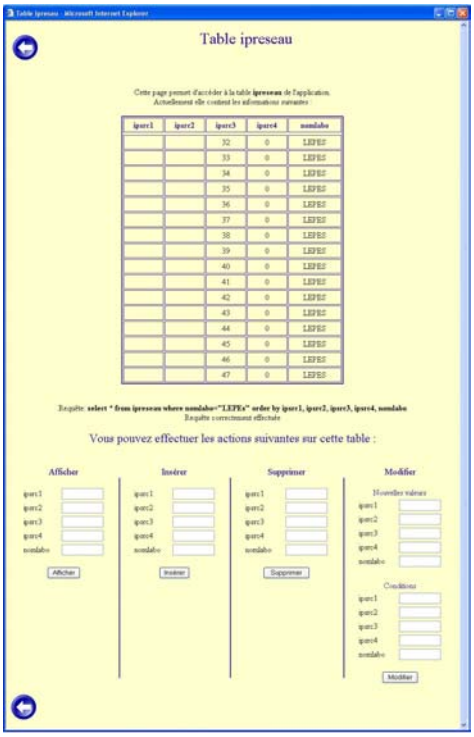

Figure 15 : Aperçu de la page Ipreseau.php (copie d'écran en Annexe)

Une dernière page, la page **Alerte.php** permet d'afficher les différentes alertes enregistrées dans la table alerte. Grâce au formulaire il est possible d'afficher certaines valeurs, d'en supprimer ou de vider la table.

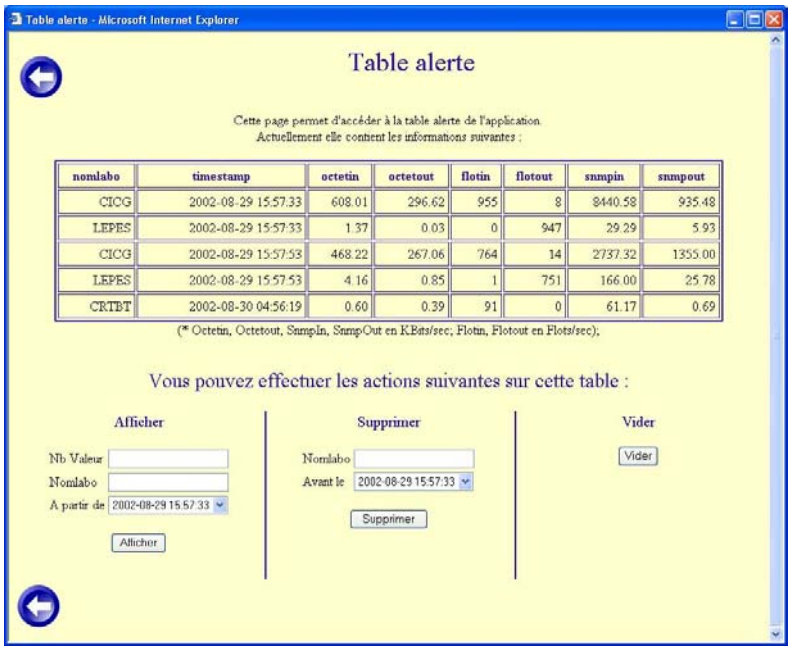

Figure 16 : Aperçu de la page Alerte.php (copie d'écran en Annexe)

C. Conclusion

L'interface réalisée grâce à PHP permet à l'utilisateur d'avoir les différents niveaux de visualisation nécessaires à une analyse du réseau en temps réel. Elle permet aussi la configuration des données pour avoir une analyse adaptée à chaque laboratoire.
L'avantage d'une interface Web est que cela permet l'intégration de nouvelles pages pouvant contenir d'autres analyses statistiques développées ultérieurement…

Conclusion

A l'issue de ce stage, les principaux objectifs définis lors de la présentation du projet sont atteints. En effet, l'application est actuellement en exploitation sur une grande partie du réseau du CNRS. La mise en place sur l'ensemble du réseau sera effectivement réalisée lors de l'arrivée du nouveau switch-routeur qui réalisera la liaison inter-laboratoires et la liaison vers l'extérieur du site.
Des extensions à ce logiciel restent néanmoins possibles telles que l'intégration du traitement d'autres protocoles de transport de données, et la création d'autres interfaces d'affichage statistique.

Bilan du projet

Ce stage au CNRS de Grenoble, sous la direction de Mr Daniel Gueniche nous a beaucoup apporté aussi bien d'un point de vue professionnel que personnel.
Tout d'abord, nous avons pu développer et parfaire nos connaissances dans l'utilisation de Linux et des Unix (les postes utilisés n'ayant pas d'autres systèmes d'exploitation). Cela nous a permis aussi de nous rendre compte des points forts de Linux pour le développement d'applications réseau.

Durant ce projet nous avons aussi pu apprendre à gérer un projet de l'analyse à la livraison du programme en suivant les exigences du client. De plus, la réalisation du programme à plusieurs nous a appris à gérer les difficultés de la gestion d'une équipe, avec des problèmes tels que la répartition des tâches.

Le stage a été une bonne opportunité d'utiliser nos connaissance réseau pour l'analyse des paquets, la configuration des routeurs CISCO ainsi que pour étudier l'organisation du réseau du CNRS. Nous avons aussi pu mettre en pratique nos connaissances en programmation C et SQL et découvrir de nouveaux langages PERL et PHP.

D'un point de vue personnel, nous sommes heureux d'avoir fait des recherches et mis en place un logiciel permettant d'avoir des informations en temps réel sur un réseau privé. Nous avons en plus acquis une certaine autonomie de travail ainsi qu'une plus grande assurance dans les choix techniques et dans la manipulation courante des différents UNIX.

En ce qui concerne la vie en entreprise, il nous a été facile de nous intégrer dans un service composé d'une équipe accueillante, toujours prête à répondre à nos questions.