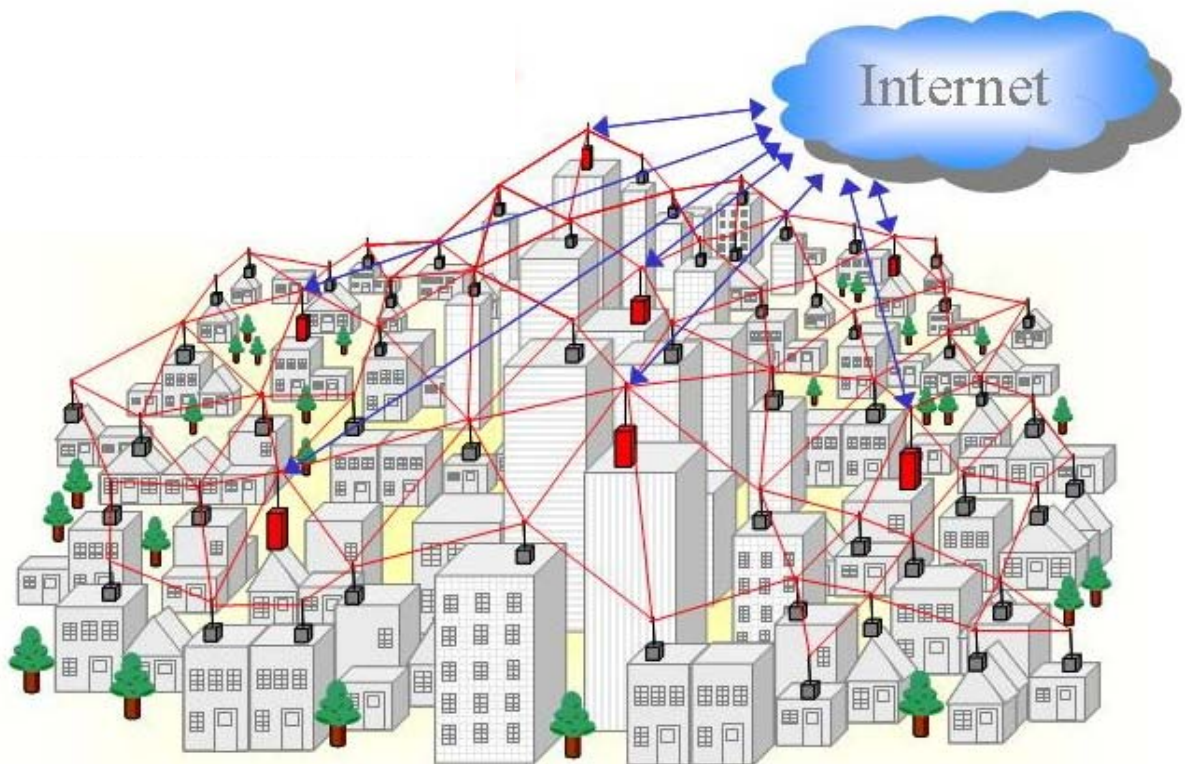


# PROJET DE CONCEPTION

## Rapport

\*\*\*\*\*



*Découverte de topologie et  
surveillance réseau par protocole SNMP*

*Responsable de projet : Patrick Mercier*

## **Introduction**

Dans le cadre de notre cursus d'élèves de deuxième année à l'Ecole Supérieure d'Electricité, un module est consacré à la réalisation d'un projet de conception. Le projet se déroule sur une séquence, soit environ deux mois durant lesquels 8 séances de 4h chacune sont dédiés à sa réalisation.

Nous avons choisi de continuer un projet commencé en début d'année lors du projet de développement logiciel, ce projet permettant de représenter graphiquement et automatiquement la topologie d'un réseau informatique via le protocole SNMP.

Nous commencerons donc par introduire le protocole SNMP lui-même avant de présenter la partie réalisation. Nous avons séparé cette réalisation en deux tâches parallèles : la première consistait en la recherche du matériel présent sur le réseau étudié grâce au protocole SNMP, et la seconde s'occupait de développer l'interface graphique qui allait permettre de visualiser le travail réalisé par le premier groupe.

Mais avant toute chose, nous souhaitons remercier les personnes suivantes pour avoir réalisé la première version de ce projet :

DECOBECQ Vincent

INCATASCIATO Benoit

LEGRAND Thomas

POL Sabine

RICHARD Charles-André

Table des matières

<b>INTRODUCTION .....</b>	<b>2</b>
<b>I. LES ENJEUX DU PROJET .....</b>	<b>6</b>
A. La connaissance du réseau .....	6
B. La surveillance du réseau .....	6
C. Les réponses apportées par notre projet .....	7
D. Les outils utilisés.....	7
1. Le SNMP .....	7
a) Trame SNMP .....	9
b) Premier format pour les PDU .....	10
c) Second format pour les PDU .....	10
2. L'architecture L.A.M.P. ....	10
<b>II. CONCEPTION ET METHODES DE DEVELOPPEMENT .....</b>	<b>12</b>
A. Architecture de l'application.....	12
B. Organisation du développement .....	13
<b>III. MOTEUR DE DECOUVERTE.....</b>	<b>15</b>
A. Découverte de la topologie du réseau.....	15
1. Détection initiale des équipements réseaux .....	15
2. Détection des terminaux .....	16
3. Détection des liaisons entre switches .....	17
4. Détection des switches non gérables .....	18
5. Détection des routeurs .....	19
6. Auto-apprentissage du type des équipements.....	20
7. Cas des clusters .....	20
8. Cas des VLAN.....	21
9. Recherche en profondeur (Mode poursuite) .....	22
10. Modularité du script .....	23
11. Processus de découverte .....	24
B. Suivre des modifications sur le réseau .....	25
C. Stockage des topologies.....	25
D. Implémentation .....	28
1. Débogage.....	28
2. Gestion des erreurs .....	29
3. Dépendance et organisation des différentes librairies .....	29

4.	Librairie SNMP .....	31
a)	Emploi des MIB et des OID .....	31
b)	Arguments des fonctions SNMP.....	31
c)	Fonction GetCorrespondancePortMacBySNMP .....	32
d)	Fonction GetMembersOfClusterBySNMP .....	34
e)	Fonction GetSwitchMacAdrBySNMP.....	35
f)	Fonction GetInterfaceMacAdrBySNMP .....	35
g)	Fonction GetTypeServicesBySNMP .....	35
h)	Fonction GetObjectIDBySNMP .....	36
i)	Fonction GetArpTableBySNMP.....	36
j)	Fonction GetIfSpeedBySNMP .....	37
k)	Fonction GetNextRouterBySNMP .....	37
l)	Fonction IPScanBySNMP .....	37
m)	Fonction GetVlanTableBySNMP .....	38
5.	Librairie Réseau .....	39
a)	Fonction MacFormat .....	39
b)	Fonction ComparaisonIPs.....	39
c)	Fonction GetLocalArpTable .....	40
d)	Fonction GetNumericOID .....	40
6.	Librairie Topologie.....	41
a)	Fonction GetSwitchLinks .....	41
b)	Fonction GetNoManageableDevices .....	43
c)	Fonction ComparaisonSwitchsAvecListeSwitchs .....	43
d)	Fonction ComparaisonRouteursAvecListeSwitchs.....	43
e)	Fonction FusionCorrespondancesVlans.....	43
f)	Fonction CreationTableauLiaisons .....	43
g)	Fonction CreationTableauEquipements.....	44
h)	Fonction RechercheDNS .....	44
i)	Fonction ExportationVersMysql.....	44
j)	Fonction ModificationEtatReseau .....	44
k)	Fonction DecouverteReseau .....	47
<b>E.</b>	<b>Limitations.....</b>	<b>51</b>
1.	Détection des routeurs .....	51
2.	Détection des liaisons inter-switchs .....	51
3.	Détection des nœuds non gérables.....	51
4.	Utilisation de MIB propriétaires.....	52
5.	Surveillance de l'état du réseau .....	52
6.	Bornes Wifi .....	52
7.	Fonctionnalités supplémentaires.....	53

<b>IV.</b>	<b>INTERFACE UTILISATEUR ET REPRESENTATION GRAPHIQUE.</b>	<b>54</b>
<b>A.</b>	<b>Présentation de l'interface utilisateur .....</b>	<b>54</b>
1.	Description de l'interface .....	54
2.	Structure des menus.....	55
a)	Gestion des utilisateurs .....	55
b)	Gestion des réseaux .....	55
c)	Gestion des équipements .....	56
d)	Affichage de la topologie et monitoring réseau .....	56
3.	Structures des fichiers.....	57
a)	Description du fichier racine.....	57
<b>B.</b>	<b>Gestion des formulaires .....</b>	<b>58</b>
<b>C.</b>	<b>Affichage de la topologie du réseau .....</b>	<b>60</b>
1.	Fonction d'affichage de la topographie .....	60
2.	Positionnement des images.....	62
	<b>CONCLUSION.....</b>	<b>63</b>
	<b>ANNEXES.....</b>	<b>64</b>
	Préambule.....	64
	Annexe 1 : Librairie SNMP et fonctions .....	64
	Annexe 2 : Librairie Réseau et fonctions .....	68
	Annexe 3 : Librairie Topologie et fonctions.....	69
	Annexe 4 : OID et MIB utilisées .....	80
	Annexe 5 : Récupération des correspondances MAC/Ports.....	81

## **I. Les enjeux du projet**

Dans la plupart des centres universitaires, d'éducation et de recherche, comme dans toute entreprise ayant atteint une certaine taille, l'utilisation de réseaux d'ordinateurs est aujourd'hui devenue incontournable, tant ces derniers ont su s'imposer comme un outil indispensable, facilitant l'échange d'informations, entre personnes du même site ou vers l'extérieur, et augmentant la productivité.

### **A. La connaissance du réseau**

Dans ce contexte, les réseaux utilisés ont le plus souvent connu une expansion rapide, dictée par le besoin toujours croissant de nombre de postes à utiliser, de services et rendre et de capacité à soutenir un trafic de données toujours plus élevé.

Par conséquent, d'une part un réseau à toujours une architecture qui n'est pas cent pour cent logique, ou qui en tout cas ne répond qu'à une « certaine » logique (celle de son concepteur le plus souvent) et d'autre part son évolution est rapide, pour ne pas dire constante et instance.

Et le paradoxe le plus important réside dans le fait que plus un réseau est important, donc plus il est difficile et long d'en connaître l'organisation, plus celui-ci est soumis à de nombreuses et rapides variations (ajout ou changement de matériel, extension du réseau, offre de nouveaux services).

Pour ajouter à la complexité du problème, il faut encore préciser que la connaissance fine de l'architecture de son réseau est impérative, et même essentielle lorsque celui-ci est de taille conséquente.

Pourtant, il est souvent courant, surtout après une phase où l'on a du faire face à une extension rapide du réseau, de ne plus avoir qu'une idée top peu précise de son organisation et de son implantation physique. Qui ne s'est jamais retrouvé perdu devant une baie de brassage où les correspondances étaient périmées ou approximatives, sans savoir qui était relié à quoi et comment ?

### **B. La surveillance du réseau**

Un autre problème est de pouvoir à chaque instant garantir une qualité de service optimale sur son réseau : c'est-à-dire de garantir le fonctionnement de tous les services offerts par le réseau et de toutes les machines qui le composent.

Mais, soyons réalistes, aucun équipement n'étant parfait, les pannes et les interruptions de services arrivent (les administrateurs réseaux les plus pessimistes vont même jusqu'à dire qu'elles sont monnaie courante...).

Là encore, s'il est facile de diagnostiquer la panne sur un réseau composé de peu d'équipement et offrant peu de services, la tâche devient beaucoup plus difficile lorsqu'il s'agit de surveiller un réseau d'une taille beaucoup plus conséquente où la défaillance d'un équipement peut engendrer des perturbations beaucoup plus complexes à interpréter que la simple « coupure » réseau (la logique du « tout ou rien » ne s'applique plus).

D'ailleurs, et ceci est notamment le cas dans les réseaux où les équipements sont redondants, une panne matérielle peut ne pas entraîner de conséquence visible pour l'utilisateur. Pour autant, il est important de détecter cette panne et d'y remédier au plus vite, ne serait-ce que pour maintenir la fiabilité et la continuité des services du réseau.

Aussi, les administrateurs réseau cherchent, par divers moyen, de surveiller (ou de « monitorer », pour faire un anglicisme) leur réseau ; c'est-à-dire d'effectuer, si possible en temps réel, un diagnostic le plus complet possible de l'état de santé de leurs équipements afin de savoir rapidement où et comment agir pour pallier à d'éventuels dysfonctionnements.

### ***C. Les réponses apportées par notre projet***

C'est afin de répondre à ces deux problèmes cruciaux de découverte de topologie et d'audit de l'état d'un réseau que nous avons développé notre projet. En effet, si plusieurs outils existent, comme par exemple NeDi (Network Discovery) ou Cacti, ils ne permettent pas de faire simultanément de la découverte de topologie et de la surveillance au sein d'une même application, ce que nous nous proposons de faire.

En essayant de nous appuyer sur une conception innovante, nous allons tenter de répondre à ce double-problème au travers d'une application qui se voudra conviviale et facile d'utilisation.

L'objectif est donc d'une part de réaliser un « moteur » de découverte : c'est-à-dire implémenter une série de fonctions permettant de découvrir le réseau (c'est-à-dire d'en établir une topologie : nœuds et liens entre ces nœuds), et d'autre part de présenter et d'exploiter au mieux ces résultats : c'est-à-dire réaliser une interface graphique pratique et conviviale dans laquelle seront regroupées et mises en forme les informations issues du moteur de découverte. Notamment, il s'agira de représenter de manière graphique la topologie du réseau afin de permettre à l'utilisateur d'identifier de manière immédiate.

De plus, la notion de « monitoring » du réseau sous-entend le fait que l'on puisse comparer l'état courant du réseau avec un état précédent afin de pouvoir mettre en exergue les différences entre ces eux « instantanés » de l'état du réseau : ce que nous nommerons par la suite « évènement » et qui sont les modifications dans la structure ou l'état des nœuds et de liaisons dans le réseaux et qui devront interpeller l'administrateur soucieux du bon fonctionnement de son réseau.

### ***D. Les outils utilisés***

Afin de répondre au cahier des charges que nous venons de décrire succinctement, nous avons besoin de divers outils que nous rangerons en deux catégories.

Certains outils nous serviront à découvrir la topologie du réseau, en dialoguant d'une façon ou d'une autre avec les équipements du réseau afin de récupérer un maximum d'information sur leur état, leurs propriétés, leurs relations avec les autres équipements du réseau.

Les autres outils serviront à automatiser l'interrogation des équipements, à stocker les résultats de ces interrogations, à les exploiter puis enfin à les présenter à l'utilisateur.

## **1. Le SNMP**

Pour diverses raisons, il peut être nécessaire de connaître la topologie d'un réseau informatique (optimisation de la répartition des tâches afin d'utiliser au mieux la bande passante, administration, etc ...).

Pour ce faire, plusieurs outils sont à la disposition des informaticiens. Parmi eux, le protocole SNMP (Simple Network Management Protocol) est l'un des plus répandus et utilisés.

Le système de gestion de réseau est basé sur deux éléments principaux: un superviseur et des agents. Le superviseur est la console qui permet à l'administrateur réseau d'exécuter des

requêtes de management. Les agents sont des entités qui se trouvent au niveau de chaque interface connectant l'équipement managé au réseau et permettant de récupérer des informations sur différents objets.

Switchs, hubs, routeurs, et serveurs sont des exemples d'équipements contenant des objets manageables. Ces objets manageables peuvent être des informations matérielles, des paramètres de configuration, des statistiques de performance et autres objets qui sont directement liés au comportement en cours de l'équipement en question. Ces objets sont classés dans une sorte de base de donnée appelée **MIB** ("*Management Information Base*"). SNMP permet le dialogue entre le superviseur et les agents afin de recueillir les objets souhaités dans la MIB.

L'architecture de gestion du réseau proposée par le protocole SNMP est donc basée sur trois principaux éléments :

- Les **équipements managés** (**managed devices**) sont des éléments du réseau (ponts, hubs, routeurs ou serveurs), contenant des "objets de gestion" (*managed objects*) pouvant être des informations sur le matériel, des éléments de configuration ou des informations statistiques ;
- Les **agents**, c'est-à-dire une application de gestion de réseau résidant dans un périphérique et chargé de transmettre les données locales de gestion du périphérique au format SNMP ;
- Les **systèmes de management de réseau** (*network management systems* notés **NMS**), c'est-à-dire une console au travers de laquelle les administrateurs peuvent réaliser des tâches d'administration.

Pour se retrouver dans la foule d'informations proposées par chaque agent, on a défini une structure particulière pour les informations appelée SMI. Chacune des informations de la MIB peut être retrouvée soit à partir de son nom de variable, soit à partir d'un arbre de classification. Cela revient à parcourir des sous-dossiers et dossiers d'un disque dur...

Supposons que vous souhaitiez consulter la variable *System* d'un hôte, vous pouvez soit lui demander la variable *System* directement, soit lui demander la variable ayant pour OID (Object IDentification) 1.3.6.1.2.1.1... correspondant à l'arborescence de la variable (ISO, Identified Organization, dod, Internet, Management, MIB2, System).

Ca parait très lourd à première vue, mais le nombre de variable étant important, on ne peut se souvenir de chaque nom. Par contre, il existe de nombreux logiciel permettant d'explorer la MIB de façon conviviale, en utilisant cette classification



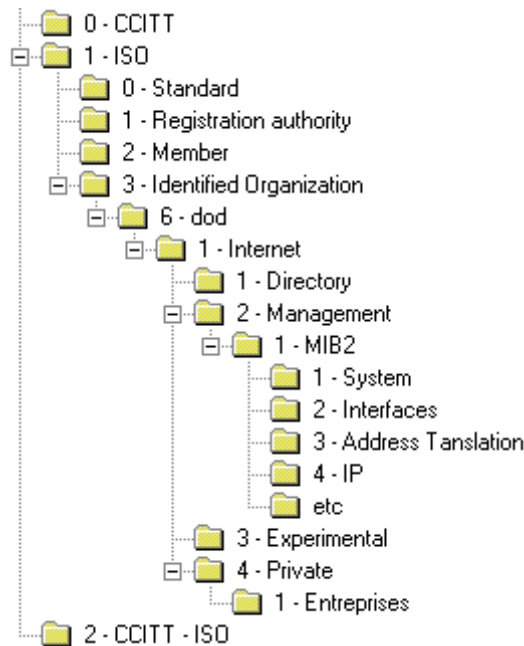


Figure 1 : Structure de la MIB

Au bout d'un moment, les variables choisies pour la MIB (puis la MIB2) se sont avérées insuffisantes pour plusieurs applications. On va donc trouver deux autres types de MIB que sont les Private MIB et les MIB R-MON (Remote network MONitoring). Les Private MIB, représentées en 1.3.6.1.4 dans la classification SMI, permettent aux entreprises de rajouter des variables pour une implémentation particulière des agents SNMP. Cela leur permet d'ajouter de nouvelles variables en fonctions des applications qu'elles veulent développer.

Les MIB R-MON permettent par exemple de placer des agents SNMP sur des supports physiques. Sur un câbles, on peut connecter une sonde R-MON qui va enregistrer tout se passe et que le l'administrateur pourra interroger pour avoir des informations sur les collisions, les débits à un endroit précis.

Deux situations sont possibles pour les échanges de données. Soit l'administrateur réseau demande une information à un agent et obtient une réponse, soit l'agent envoie de lui-même une alarme (trap) à l'administrateur lorsqu'un événement particulier arrive sur le réseau. Il est donc possible que l'agent prévienne l'administrateur de son propre chef si un incident survient.

### a) Trame SNMP

Le format de la trame SNMP est décrit ci-dessous :

Version	Communauté	PDU
---------	------------	-----

Figure 2 : Format de la trame SNMP

**Version** : numéro de version SNMP. Le manager et l'agent doivent utiliser le même numéro.  
**Communauté** : ce champ sert à identifier auprès du manager l'agent avant de lui accorder un accès.

**PDU** : il y a 5 types de PDU : GetRequest, GetNextRequest, GetResponse, SetRequest, et TRAP.

Une description de ces PDU est donnée ensuite.

### b) Premier format pour les PDU

Un premier format est utilisé pour les PDU du genre GET, ou SET :

Type de PDU	ID de requête	Statut d'erreur	Index d'erreur	Obj 1, val 1
-------------	---------------	-----------------	----------------	--------------

Figure 3 : Premier format de TRAP PDU

#### Type de PDU :

- 0 : GetRequest
- 1 : GetNextRequest
- 2 : GetResponse
- 3 : SetRequest

**ID de requête** : champ qui coordonne la requête du manager et la réponse de l'agent.

**Statut d'erreur** : entier qui indique une opération normale (cas 0) ou bien une erreur (cas 1)

**Index d'erreur** : identifie les entrées avec la liste des variables qui ont causé l'erreur.

**Obj/Val** : association du nom de la variable à transmettre avec sa valeur.

### c) Second format pour les PDU

Un second format est utilisé pour la TRAP PDU :

Type de PDU	Entreprise	Adresse Agent	Type Generique	Type Specifique	Timestamp	Obj 1, val 1
-------------	------------	---------------	----------------	-----------------	-----------	--------------

Figure 4 : Second format de TRAP PDU

**Type de PDU** : dans ce cas toujours égal à 4.

**Entreprise** : identifie l'entreprise de management qui a défini la Trap.

**Adresse Agent** : adresse IP de l'agent.

**Type Générique** : décrit quel type de problème est survenu. (7 valeurs sont possibles).

**Type Spécifique** : est utilisé afin d'identifier une TRAP spécifique à une entreprise.

**Timestamp** : contient la valeur de l'objet sysUptime représentant le temps écoulé depuis la dernière initialisation.

**Obj/Val** : association du nom de la variable à transmettre avec sa valeur.

## 2. L'architecture L.A.M.P.

Pour réaliser notre application, nous avons bien entendu le choix entre plusieurs architectures et plusieurs langages pour la réaliser. Lors du précédent projet auquel notre réalisation fait suite, le moteur de découverte avait été réalisé en PHP (Personal Hypertext Preprocessor). Comme ce langage était connu de nous deux, nous avons choisi de le conserver.

En outre, il possédait la particularité de répondre à plusieurs exigences que nous avons fixées à notre application.

Comme la plupart des logiciels réalisant des fonctions similaires et que nous avons cités précédemment, nous souhaitons que notre application soit disponible depuis n'importe quel poste du réseau et sans avoir d'installation à faire sur le poste de consultation.

Aussi, nous avons adopté le modèle dit du « client léger » (aujourd'hui adopté par de nombreux progiciels d'entreprise accessibles via l'intranet développé au sein des sociétés) qui limite au maximum l'utilisation de ressources sur le poste client ce qui ouvre la possibilité de d'utiliser notre service sur des appareils mobiles, comme un PDA par exemple, ce qui peut s'avérer pratique quand il est nécessaire d'intervenir « sur site » tout en contrôlant en temps quasi-réel l'impact sur le réseau des modifications que l'on effectue.

De plus, les nombreuses bibliothèques de fonctions développées par PHP incluent des fonctions de dialogue en SNMP et de nombreux outils graphiques (inclus dans la librairie GD2 de PHP) ce qui va nous permettre de faciliter notre développement qui reposera grandement sur l'utilisation de ces fonctions.

Et puis, nous avons tout à l'heure évoqué la nécessité de recourir à l'utilisation d'un système de bases de données performant afin de gérer le stockage des informations concernant les états successifs du réseau. Cet impératif vient renforcer le choix d'une architecture « trois tiers », où dans notre cas l'exécution des requêtes et le stockage s'effectue sur la même machine mais auraient très bien pu se faire sur des équipements différents. D'un point de vue technique, notre choix s'est arrêté sur MySQL, système de gestion de bases de données libre et mondialement connu pour ses performances et sa robustesse.

Notre application sera donc exécutée sur un serveur, pour lequel nous avons fait le choix d'une solution libre : Linux Fedora pour le système d'exploitation et Apache pour le serveur web. Cette solution, libre, stable et éprouvée, vient se marier parfaitement avec l'utilisation du couple PHP/MySQL.

## II. Conception et méthodes de développement

Maintenant que le choix des technologies et de l'architecture générale de l'application a été arrêté, il reste à définir l'architecture interne de l'application, à savoir la manière dont vont dialoguer entre eux les différents éléments que nous avons déjà esquissés, à savoir : le moteur de recherche, l'interface utilisateur incluant la représentation graphique des données et la base de données servant au stockage des applications.

### A. Architecture de l'application

Afin d'optimiser le développement de notre application, vaste sujet à couvrir dans un temps imparti relativement restreint, nous avons choisi de paralléliser les tâches et de développer de manière relativement indépendante chacune de nos parties.

En fait, cette mise en parallèle correspondait aussi avec notre profil et notre expérience passée : l'un d'entre nous avait déjà développé le moteur de découverte au cours d'un projet précédent, connaissait bien le fonctionnement de ce dernier et du SNMP et n'avait que des optimisations à apporter à son fonctionnement, l'autre avait déjà développé plusieurs sites web dynamiques et pouvait donc mettre à profit cette expérience pour développer dans les temps impartis l'interface la plus complète possible.

Nous avons donc choisi d'organiser notre application de la sorte, en s'appuyant sur les deux couches qui semblaient apparaître naturellement : le moteur et l'interface utilisateur.

Pour interfacier ces deux éléments, nous avons choisi d'utiliser la base de données. Le moteur, une fois la découverte du réseau effectuée, stockera dans une base de données les informations issues de cette découverte. Par la suite, l'interface utilisateur utilisera ces données pour les traiter et les afficher. Elle permettra aussi de lancer la découverte topologique avec les paramètres adéquats.

On voit donc qu'il est uniquement nécessaire de bien spécifier l'architecture des bases de données pour que les deux « briques » principales de notre application s'interfacent parfaitement. De même, il suffit de s'accorder sur le nombre et la nature des paramètres de la fonction de découverte de topologie (fonction de plus haut niveau du moteur de découverte) pour pouvoir permettre à nos parties de communiquer.

On aboutit alors à l'architecture présentée en Figure 5.

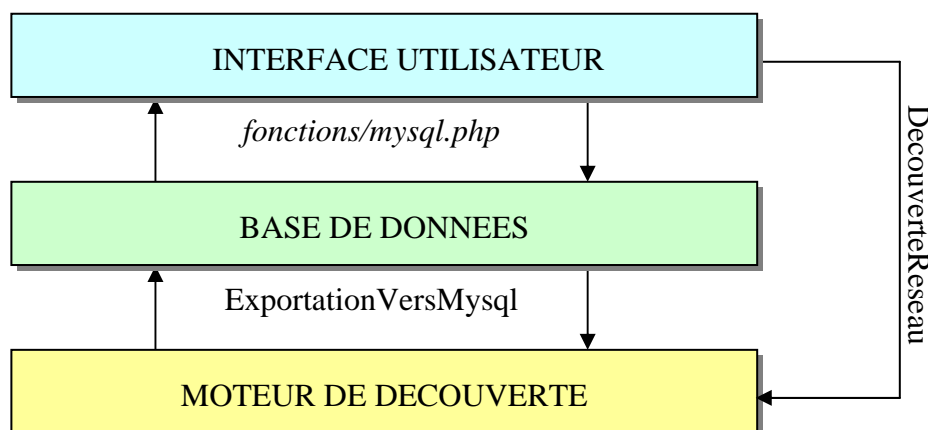


Figure 5 : Architecture générale de l'application

## ***B. Organisation du développement***

Comme nous l'avons vu précédemment, nous avons voulu organiser de la manière la plus optimale possible le développement de notre application, en parallélisant les tâches. Le diagramme présenté en Figure 6 résume succinctement les différentes étapes de réalisation en indiquant leur durée relative.

Le moteur de découverte ayant déjà été codé, une grande partie du temps disponible a été consacré à la relecture de son code, à l'optimisation des fonctions lorsque cela était possible, ainsi qu'à la simplification de certaines autres. D'autre part il a fallu rajouter les quelques fonctions permettant la surveillance des réseaux, et exporter le résultat du moteur non pas vers des fichiers textes mais vers une base de données. Le reste du temps disponible a permis de reprendre le précédent rapport afin de le corriger, compléter, et rajouter des annexes afin qu'il constitue une documentation solide sur ce moteur. Le but est qu'il ne soit pas nécessaire de se pencher sur le code source pour comprendre son fonctionnement ou utiliser une fonction des librairies, à moins ce qu'il soit nécessaire de modifier leur code. Dans ce dernier cas, les dernières informations utiles se trouvent sous forme de commentaire dans les codes sources.

Concernant la partie interface utilisateur, le développement a principalement consisté à implémenter différentes librairies permettant de concevoir et réaliser facilement les différentes parties de cette interfaces : les différents formulaires par exemples (dont certains présentent des fonctionnalités particulières comme le téléchargement d'images sur le serveur, la création et la suppression de bases de données à la volée, ...)

D'autre part, la principale difficulté de ce développement a été de représenter correctement et de manière agréable la topologie du réseau. La réalisation de cette fonction étant complexe, il a fallu deux tentatives successives pour y parvenir. Plus de détails concernant cette partie sont données dans la suite de ce rapport.

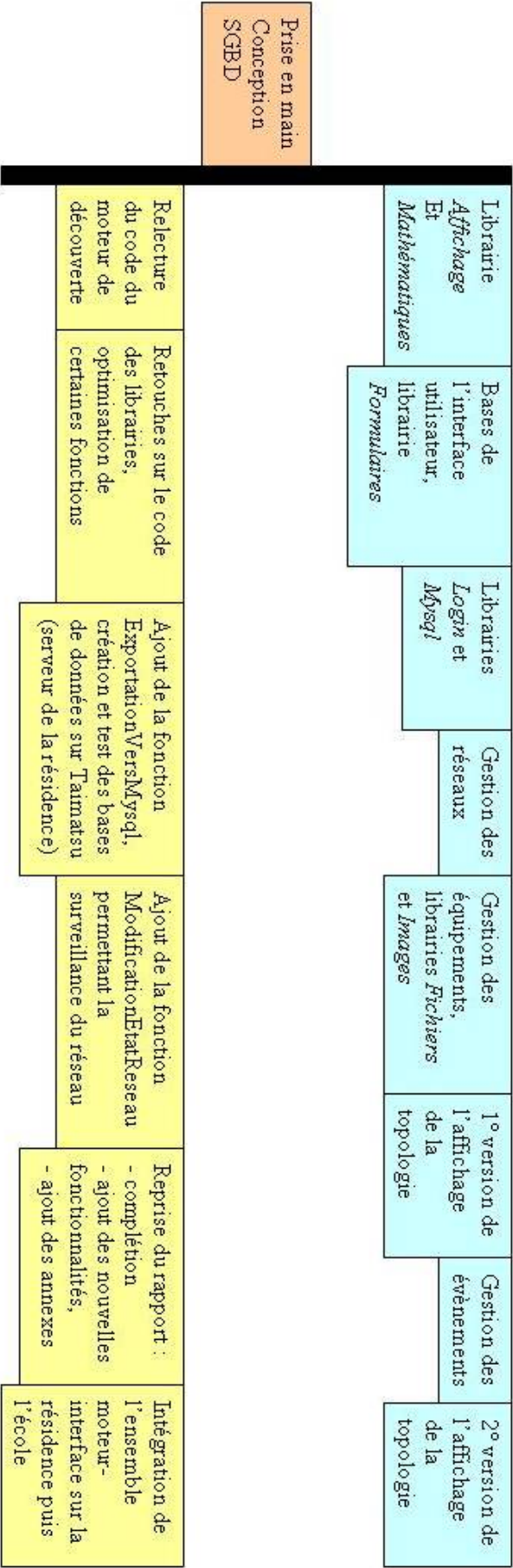


Figure 6 : Déroulement de la réalisation

## III. Moteur de découverte

### A. Découverte de la topologie du réseau

Un réseau informatique est caractérisé par un ensemble de terminaux reliés entre eux par des liaisons, et éventuellement par des nœuds. Les terminaux sont les ordinateurs (ou encore station de travail) des utilisateurs, les imprimantes réseaux, les serveurs (Web ou FTP par exemple), etc. Les nœuds sont quant à eux les commutateurs (switchs en anglais) souvent utilisés dans l'épine dorsale du réseau, les concentrateurs (hubs) utilisés en bout de réseau, les routeurs, etc. Finalement, un réseau se résume donc à un certain nombre d'équipements et de liaisons entre ceux-ci, et découvrir sa topologie revient à trouver ces deux couples d'éléments.

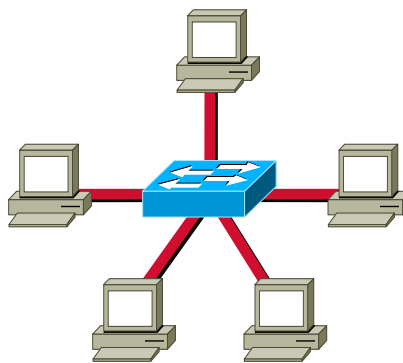


Figure 7 : Réseau (en étoile)

Nous allons détailler dans les paragraphes suivants les différentes méthodes retenues pour mener à bien cette découverte.

#### 1. Détection initiale des équipements réseaux

En supposant qu'avant l'exécution de l'application, nous ne disposions d'aucune information sur les équipements constituant le réseau à étudier, il faut alors les trouver. Pour cela, la seule méthode simple et relativement fiable et de scanner une plage d'adresses IP préalablement fournie par l'utilisateur. Puisqu'à la base, nous ne considérons que les switchs supportant le protocole SNMP (nous n'utiliserons pas par exemple de *ping* ou *traceroute*), nous n'allons pas scanner le parc informatique avec des ping classiques mais avec des requêtes SNMP.

Les équipements qui répondront à celles-ci devront alors être triés afin d'obtenir au final une première liste d'éléments à interroger plus en détail. A noter que cette première requête doit être simple : elle ne doit récupérer qu'une seule information pour ne pas surcharger le réseau. Enfin, celle-ci doit être standard pour que tout équipement, switch ou simple serveur soit en mesure de répondre.

Scanner une plage d'adresses IP a également un autre avantage : puisque le serveur sur lequel va tourner le programme essaiera de communiquer vers toutes ces IP, il devra obtenir leurs adresses MAC et se constituera ainsi une table ARP relativement complète. Nous pourrons alors récupérer cette table afin de traduire les adresses MAC obtenues plus tard en adresses IP, plus pratique et parlant pour identifier un équipement. En effet, n'oublions pas que nous allons interroger principalement des switchs qui fonctionnent par défaut au niveau 2 du modèle OSI, autrement dit qui travaillent avec des adresses MAC et non des adresses IP. Certes les adresses MAC sont (normalement) uniques - et nous en abuserons afin d'éviter tous

doublons dans nos différentes listes d'équipements - mais un administrateur réseau identifiera bien plus rapidement une machine par son IP.

Enfin il faut noter que cette détection ne concerne pas les terminaux (puisqu'on se base sur le protocole SNMP et que même si des serveurs seraient ainsi détectés, ils seraient par la suite éliminés).

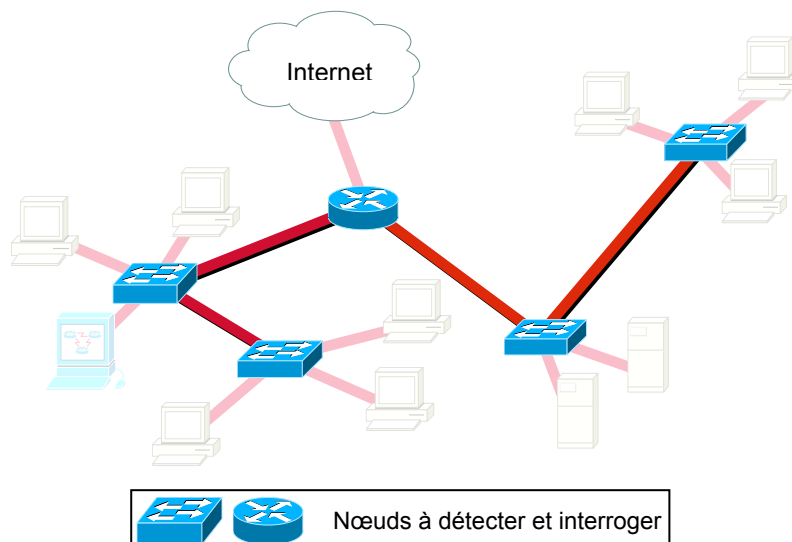


Figure 8 : Nœuds réseaux détectés initialement

Nous allons justement nous pencher maintenant sur ces éléments.

## 2. Détection des terminaux

Les « simples » ordinateurs ou imprimantes réseaux seront eux détectés depuis chaque switch interrogé : leurs adresses MAC se trouveront sur les ports des nœuds réseaux. Nous allons donc récupérer les correspondances *adresses MAC / ports* de chaque switch. Cette table, comme son nom l'indique, associe à chacun des ports des switchs une ou plusieurs adresses MAC, donc un ou plusieurs équipement (terminal ou nœud). De plus, si nous disposons d'une entrée dans la table ARP pour ces adresses MAC données, nous pourrions obtenir leurs adresses IP, et ainsi de suite plus d'information en les interrogeant par exemple par l'intermédiaire d'un protocole quelconque (http, ftp, smtp, etc.).

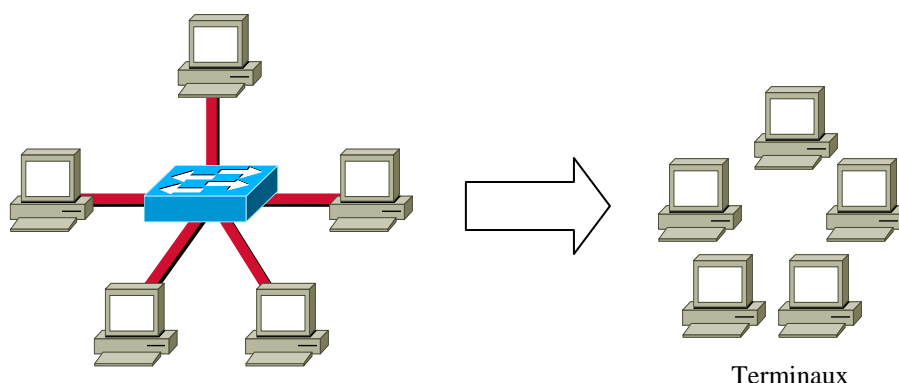


Figure 9 : Détection des terminaux



### 3. Détection des liaisons entre switches

Maintenant que nous avons toutes les correspondances *MAC/ports*, les liaisons entre les terminaux et les nœuds sont donc connues. Par contre, il reste à lier entre eux les switches. Or chaque interface réseau possède une adresse MAC, même les ports des switches, puisque ce sont en fait des cartes réseaux... Nous trouverons donc ces liaisons en cherchant, sur chaque switch, les MAC des ports des autres switches connus. Si une liaison potentielle est trouvée entre le switch courant (celui actuellement traité par l'application) et un switch distant, nous vérifierons alors la réciproque, c'est-à-dire que le switch distant voit bien le switch courant. Si tel est le cas, alors nous avons réellement une liaison entre ces deux équipements. Si la réciproque n'est pas trouvée, nous l'ignorons.

Il existe cependant une exception quant à la vérification de la réciprocity d'une liaison : il s'agit du cas particulier où le switch distant est en fait un routeur. En effet, un routeur ne dispose pas forcément des mêmes MIB qu'un switch classique et il se peut que nous n'arrivions pas à obtenir ses correspondances *MAC/Ports*. C'est pourquoi dans ce cas particulier, et dans ce cas uniquement, nous nous contenterons de la présence de l'adresse MAC d'un port du routeur sur un autre équipement pour considérer que la liaison existe. Afin de garantir un résultat cohérent, cette recherche de liaison sans tester la réciprocity ne sera faite qu'après avoir cherché les liaisons classiques et en ne prenant en considération que les adresses MAC des routeurs.

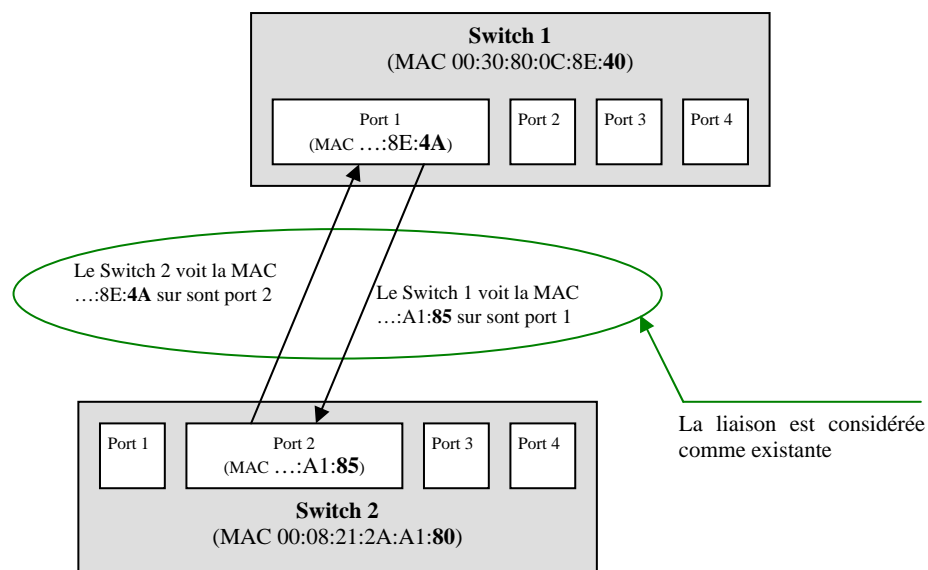
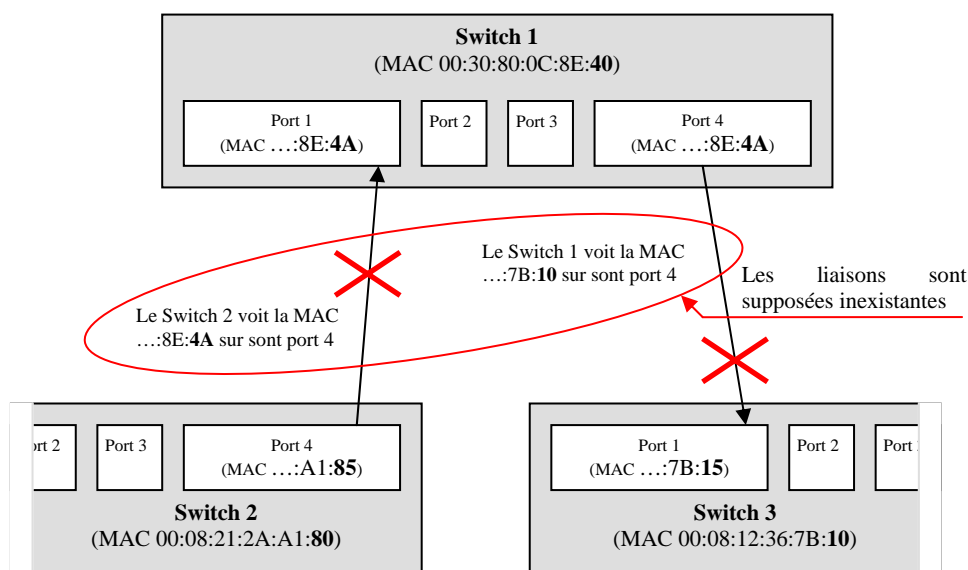


Figure 10 : Détection d'une liaison inter-switch



**Figure 11 : Non détection de liaisons inter-switchs**

La vérification de la réciprocité de la liaison est importante à cause d'un phénomène constaté : sur les interfaces reliant deux switchs entre eux se trouvent généralement des adresses MAC de terminaux qui pourtant n'y sont pas directement connectés. Or pour ces équipements en question, la condition de réciprocité n'est pas vérifiée. Par contre, cette même condition est vérifiée lorsqu'il s'agit de la liaison réelle entre les deux switchs... De plus, dès qu'une liaison est détectée, la correspondance en question est supprimée afin qu'elle ne puisse plus être prise en compte plus tard (nous pourrions alors détecter plusieurs liaisons différentes émanant dans la réalité d'une seule et unique liaison, pour les mêmes raisons qui ont justifié le test de la réciprocité des liaisons).

De plus, on prend soin de retirer l'adresse MAC de l'IP du switch (autrement dit son IP permettant son administration). Effectivement, dans le cas de clusters (cf. *Cas des clusters* page 20) nous pouvons par exemple retrouver l'adresse MAC du *commandeur* sur différents *membres* même si ceux-ci n'y sont pas reliés directement ! Ceci semble être causé - mais nous ne pouvons l'affirmer - par l'existence du cluster et que chaque membre connaît (forcement) le switch maître du groupe. Etant donné que seules les liaisons physiques nous intéressent nous nous contenterons donc uniquement des adresses MAC des interfaces.

Dans le même esprit, il faut également retirer les MAC des interfaces des switchs sur les correspondances de ces mêmes switchs. De fait, il arrive sur certain d'entre eux qu'une interface connaisse sa propre adresse MAC ! Afin d'éviter tout résultat erroné, nous les supprimons donc.

#### 4. Détection des switchs non gérables

Cette partie est relativement simple. En effet, si sur un port nous détectons plusieurs adresses MAC, il y a alors forcément un équipement réseau intercalé entre ce port et ces MAC ! Cet appareil pouvant être gérable ou non : il est nécessaire de traiter cette étape après avoir détecté les liaisons inter-switchs (donc entre éléments gérables !). Ainsi, toutes les liaisons entre équipements administrables seraient déjà trouvées, et ne resteraient que les périphériques non gérables ou les switchs administrables que nous n'aurions tout simplement pas détectés.

Néanmoins, le fait d'utiliser le nombre d'adresses MAC présentes sur un port donné implique deux limitations :

- Si un port est relié à un switch non administrable auquel aucun ordinateur n'est relié, il ne pourra être détecté. En effet, même si cet équipement possède une adresse MAC visible sur le port du switch gérable, nous n'en verrons qu'une seule et cet équipement sera alors confondu avec un terminal.
- Par ailleurs, nous ne pouvons qu'affirmer avec certitude qu'au moins un périphérique est intercalé entre un port du switch et les ordinateurs. Par contre, nous n'avons aucun moyen d'en connaître le nombre exact !

Maintenant, il faut bien comprendre qu'il n'est pas possible de pallier à ces défauts en utilisant uniquement le protocole SNMP. Or utiliser d'autres méthodes sortirait du cadre de ce projet !

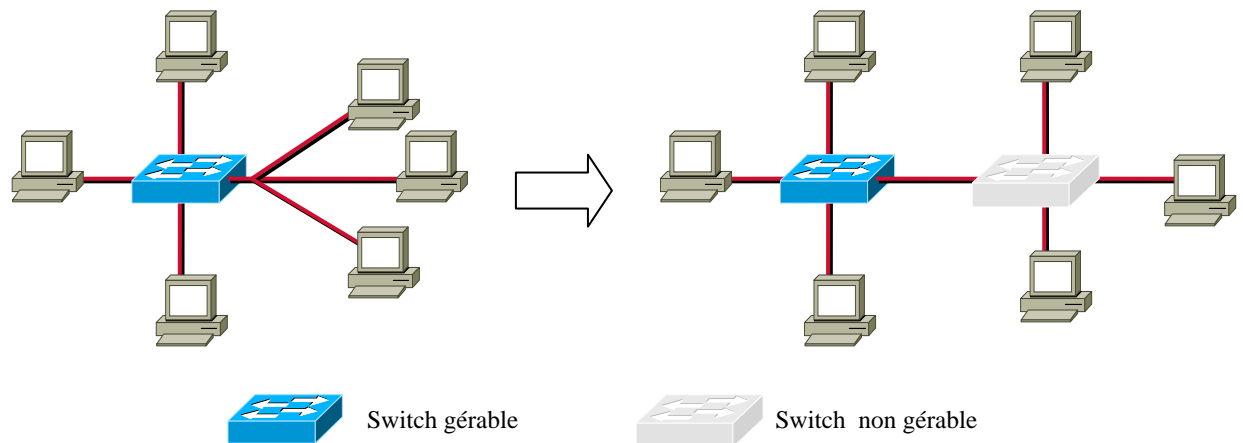


Figure 12 : Détection des switches non manageables

## 5. Détection des routeurs

Les routeurs sont très importants car tout ordinateur qui veut accéder à un réseau extérieur passe par lui. Les routeurs possèdent ainsi une table ARP conséquente et récente, d'où leur intérêt. Outre la difficulté de les détecter (des switches peuvent être de niveau 3 et donc avoir des fonctionnalités identiques aux routeurs classiques), il faut alors qu'ils soient interrogés le plus tôt possible. En effet, les correspondances *MAC/ports* obtenues ne donnent justement que des adresses MAC et il est préférable d'avoir leurs IP associées rapidement afin de pouvoir les exploiter tout au long de nos scripts. C'est pourquoi dès qu'un routeur est détecté, la liste des switches est remise à jour en incluant ce routeur s'il ne l'était pas déjà et surtout en changeant l'ordre de cette liste. Nous pouvons considérer cette liste comme une file, le premier élément étant le plus prioritaire puisque c'est lui qui sera traité en premier et ainsi de suite. Chaque nouveau routeur détecté (ou switch déjà connus mais dont on ne connaissait pas cette fonctionnalité) sera ainsi placé dans cette file de façon à être le prochain équipement qui sera interrogé, reléguant les autres switches en fin de liste.

Un problème peut par ailleurs survenir lorsque nous détectons de nouveaux routeurs : nous pourrions tomber sur les routeurs du fournisseur d'accès et notre programme essaierait tout naturellement de récupérer la table ARP par SNMP. Ceci n'est bien entendu pas envisageable. C'est pourquoi une liste de plages d'adresses IP autorisées est mise en place. Cette liste inclura automatiquement les plages d'IP à scanner puisque si nous voulons les scanner, c'est que nous autorisons justement l'envoi de requêtes SNMP vers ces IP ! Nous avons choisis d'implémenter une liste d'IP autorisées et non interdite pour plusieurs raisons. D'une part, nous ne connaissons pas systématiquement la plage d'IP de votre fournisseur d'accès ou même d'autres réseaux auxquels nous ne devrions pas accéder. Par contre, nous supposons connaître au moins les IP composant le réseau que nous voulons découvrir, ce qui

est d'autant plus vrai dans le cas où vous lancez un scan sur celles-ci ! D'autre part, et dans cette même logique, nous serions amenés à interdire toute plage d'adresses IP autres que celle que nous autoriserions. Il est donc plus simple, logique et concis de définir une plage autorisée plutôt qu'une plage interdite !

## **6. Auto-apprentissage du type des équipements**

Afin de fonctionner correctement, notre moteur de découverte se contente de trouver les équipements qu'il va pouvoir interroger. Néanmoins, nous aurons besoin de connaître plus précisément les types des équipements rencontrés lorsque nous voudrions les afficher. Nous pouvons obtenir deux informations par SNMP : un identifiant unique (ObjectID) par produit (il ne s'agit pas d'un numéro de série identifiant de manière unique un matériel, mais d'un numéro identifiant la marque du matériel, etc.), et d'autre part un numéro (sysServices) calculé à partir des couches du modèle OSI sur lesquelles notre matériel travaille. Nous pouvons disposer par ailleurs d'un fichier décrivant les ObjectID. Malheureusement, cette liste n'est pas formatée rigoureusement, et il est difficile d'obtenir automatiquement le type correspondant. De plus, cette liste étant longue, il n'était pas envisageable de la traiter manuellement en une seule étape. Enfin, le sysServices ne s'avère pas toujours exact pour identifier le type de matériel rencontré. Nous avons donc décidé de laisser l'utilisateur mettre à jour lui-même les informations selon le procédé suivant :

- Lorsqu'un équipement gérable est détecté, nous cherchons son ObjectID. Si nous par ailleurs nous avons déjà associé cet identifiant à un type d'équipement, alors nous connaissons son type et nous le prenons directement en compte.
- Si l'ObjectID obtenu n'est pas encore associé à un type, ou alors que nous ne le connaissons pas, nous récupérons également le sysServices et le moteur de découverte poursuit son travail sans renseigner le champ « type » de ce matériel.
- Une fois la découverte terminée, l'ensemble des équipements dont nous possédons un ObjectID mais où le champ « type » n'est pas complété sont présentés à l'utilisateur. Nous affichons les informations disponibles sur ce matériel, et nous lui signalons que celui-ci est certainement de tel type grâce au sysServices. L'utilisateur peut alors confirmer ce choix, ou renseigner un autre type. Une fois fait, les prochaines fois que nous rencontrerons ce matériel, nous saurons alors directement qu'il rentre dans tel catégorie.

## **7. Cas des clusters**

Les switches appartenant à un réseau sont par défaut indépendants et autonomes. Cependant, pour pouvoir être administrés à distance, ils possèdent chacun une adresse IP, ce qui peut devenir problématique pour les réseaux s'articulant autour d'une seule classe d'IP (typiquement une classe C, permettant de connecter 254 machines). Une solution consisterait alors à utiliser une classe locale pour les IP des switches. Les switches gérables haut de gamme permettent également d'avoir une seule adresse IP pour gérer un ensemble de switches. Nous parlons alors de *cluster*, le switch possédant sa propre IP est le *commandeur* (*commander*), les autres switches étant des *membres* (*members*). Afin de palier la défaillance du commandeur, des membres du cluster peuvent également être commandeurs potentiels et prendre ainsi le relais si le premier ne pouvait plus assurer ses fonctions. Il faut enfin noter que même s'ils appartiennent à un cluster, tous les switches peuvent garder leurs IP : il faudra donc à chaque fois qu'un membre est détecté, vérifier que nous ne l'avons pas déjà incorporé à notre liste de switches parce qu'il aurait déjà été scanné par exemple !

La figure suivant est une illustration d'un cluster constitué de 4 switches, dont un est le commandeur (et possède donc une adresse IP), un autre est commandeur potentiel (possède lui aussi une IP) et les deux derniers étant des membres.

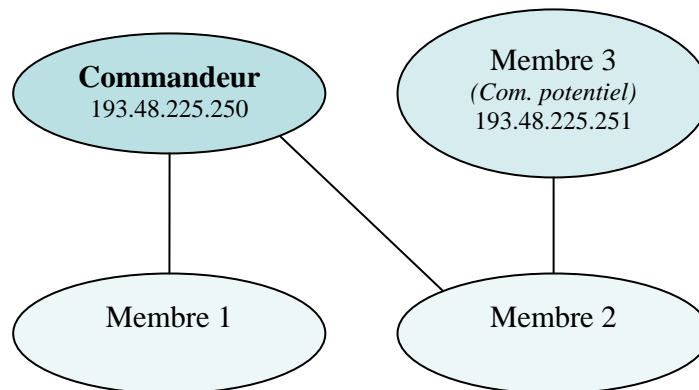


Figure 13 : Exemple d'un cluster

## 8. Cas des VLAN

Un VLAN (ou Virtual LAN) est un réseau physique virtuel. Ainsi, plusieurs réseaux virtuels peuvent être définis sur un même switch, rendant ainsi l'administration des réseaux indépendante de la géographie et accroissant la sécurité globale (les professeurs et les élèves d'une école ne sont plus sur le même réseau par exemple). Il sera nécessaire de les détecter car sur certains switches, les correspondances *MAC/Ports* doivent être demandées pour chaque VLAN. Si nous ne les récupérons pas toutes, il se peut qu'un switch appartenant totalement à un autre VLAN et détecté par un scan d'IP se retrouve isolé de toute autre équipement.

La figure suivante illustre un VLAN simple entre plusieurs terminaux. 8 Ordinateurs se partagent deux VLAN. Les Ordinateurs dans le même VLAN peuvent communiquer entre eux, mais deux ordinateurs dans deux VLAN différents ne le peuvent ! Notons malgré tout que tous les ordinateurs sont reliés aux mêmes switches, et que la liaison entre les deux switches n'est pas attribuée à un VLAN en particulier.

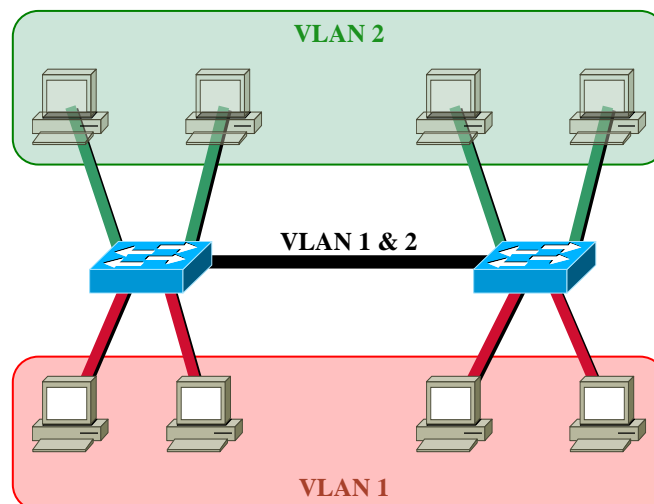
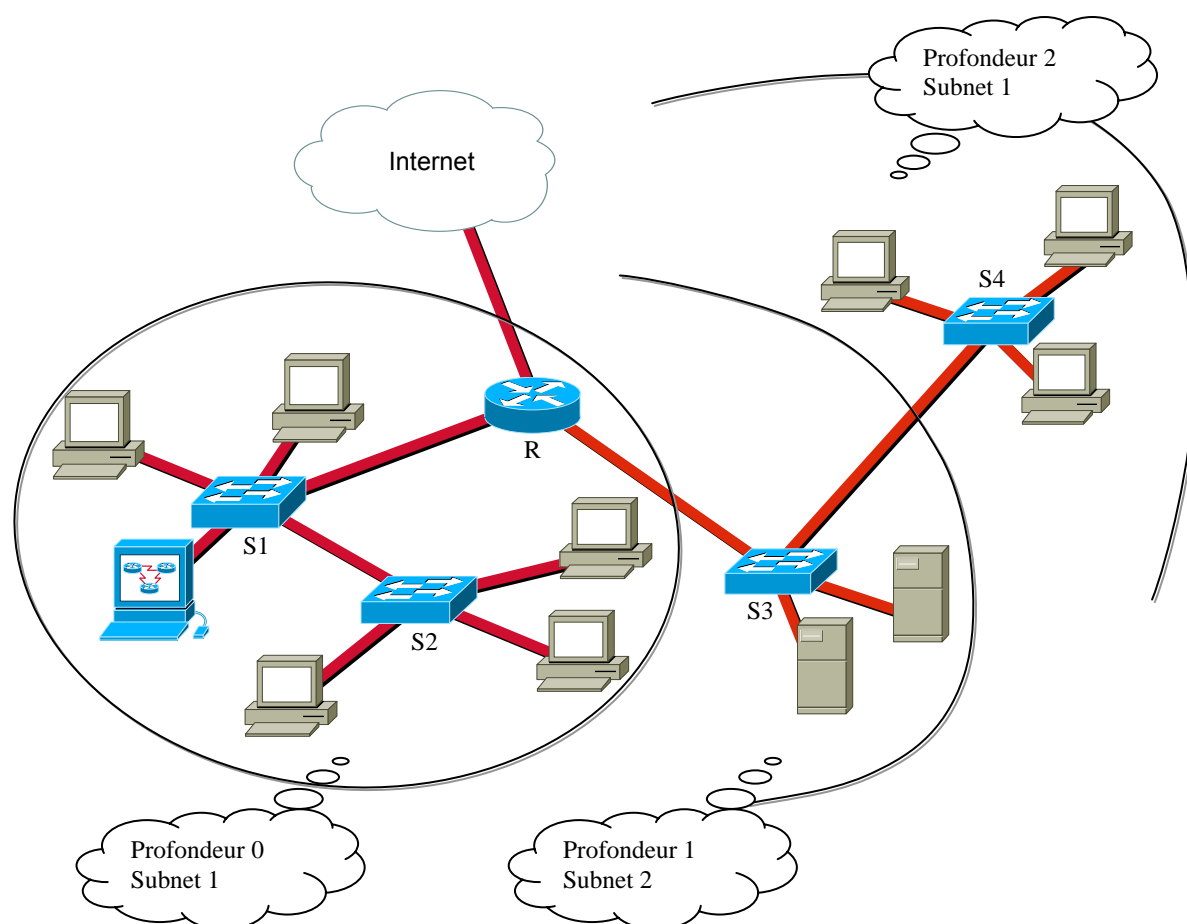


Figure 14 : Exemple de VLAN

## 9. Recherche en profondeur (Mode poursuite)

La recherche en profondeur, encore appelé mode poursuite, permet de détecter des switches à interroger de manière récursive. Par défaut, tous les nœuds réseaux sont détectés en premier, puis dans un second temps, nous les interrogeons et nous récupérons leurs correspondances *MAC/Ports*. Ceci impose donc de connaître tous les switches dès les premières étapes afin d'avoir une topologie complète. Or nous obtenons des adresses MAC. Si nous connaissons également leurs adresses IP, et que ces adresses n'ont pas encore été scannées ou indiquées par l'utilisateur, nous pouvons les scanner et éventuellement tomber sur des switches que nous aurions sinon oubliés. Une fois ces équipements détectés, ils sont ajoutés à la liste des switches déjà connus et seront traités plus tard, et ainsi de suite... Nous pourrions ainsi imaginer, d'un point de vue théorique, n'indiquer initialement l'adresse d'un seul switch et le moteur de recherche trouverait alors tout seul l'ensemble des équipements réseaux et tracerait sa topologie. Il faut néanmoins relativiser ce cas théorique puisque à moins que le switch initial soit un routeur, nous n'aurons pas de table ARP conséquente et ainsi beaucoup de nœuds réseaux risqueraient d'être ignorés.

Cette fonctionnalité est illustrée sur la figure suivante :



**Figure 15 : Recherche en profondeur d'équipement gérable**

Le routeur sépare le réseau Internet et deux sous réseaux, l'un constitué des switches S1 et S2, l'autre des switches S3 et S4. Dans un premier temps, seuls les switches S1 et S2, ainsi que le routeur ont été découverts. Lorsque nous allons interroger le routeur, nous allons trouver l'adresse MAC du switch S3 sur l'un de ses ports. Si nous avons la correspondance ARP adéquate, nous obtiendrons alors l'adresse IP du switch S3 et nous pourrions à son tour l'interroger. Nous répétons cette procédure afin de découvrir le switch S4, qui se trouve au

deuxième niveau de profondeur, en prenant comme référence (profondeur 0) le réseau que nous avons découvert initialement.

### 10. Modularité du script

Nous avons jusqu'à maintenant détaillé les différentes étapes nécessaires pour découvrir la topologie d'un réseau. Ces étapes sont autant de fonctions appelées par le moteur de recherche lui-même. Ainsi, un module sera chargé de scanner les plages d'IP, un autre de détecter les liaisons inter-switchs, etc. Il doit être également possible d'utiliser ou non tel ou tel module suivant les données fournies initialement et les limitations imposées par l'administrateur réseau.

Ainsi nous établissons plusieurs options de fonctionnement :

- Utilisation ou non d'une liste de switchs déjà fournie,
- Scanner ou non des plages d'adresses IP,
- Prise en compte ou non des plages d'IP autorisées,
- Récupérer ou non la table ARP locale du serveur sur lequel tourne l'application,
- Rechercher ou non les membres de cluster n'ayant pas d'IP,
- Activer ou non le mode poursuite (et si oui à quel niveau de profondeur s'arrêter),
- Chercher des informations complémentaires sur les ordinateurs trouvées.

La combinaison des options « *Utilisation ou non d'une liste de switchs déjà fournie* » et « *Scanner ou non des plages d'adresses IP* » permet de ne pas scanner le réseau mais nécessite alors de connaître les switchs. Ce cas doit être considéré avec intérêt puisque dans l'hypothèse où les nœuds du réseau restent inchangés mais où les liaisons sont modifiées, nous pourrions alors redessiner la topologie du réseau sans le charger inutilement avec des requêtes superflues ! Par contre, la première fois que le script est utilisé, il est préférable et surtout plus commode d'activer le scan afin de ne pas avoir à remplir la liste des switchs à la main.

La récupération de la table ARP locale est optionnelle car elle implique une configuration particulière du serveur sur lequel est exécuté le script. En effet, notre programme tourne sur un serveur web. Or pour récupérer cette table locale, il faut exécuter, depuis une page web (script PHP pour être plus exact), une commande système. Certain administrateur pourrait le considérer comme étant une faille de sécurité. De plus, la commande utilisée ne se trouvant pas forcément dans un dossier auquel a accès le script, il faut alors créer un lien symbolique sous linux, ou changer la configuration du serveur web. Enfin, pour terminer, les manipulations à faire ne sont pas les mêmes sous Windows ou Linux. Tout ceci pourrait amener l'utilisateur à ne pas vouloir utiliser cette fonction.

## 11. Processus de découverte

Voici une vue d'ensemble du processus de découverte du réseau :

1. Récupération d'une **liste de switches** préétablie (*si autorisation et existence*),
2. **Scan** des différentes plages d'IP (*si autorisation*),
3. Récupération de la **table ARP** (*si autorisation*),
4. **Confrontation les listes de switches** (ancienne et nouvelle) pour éliminer les doublons,
5. Pour chaque switch connu :
  - a. Recherche des membres d'un éventuel **cluster** (*si autorisation*) et confrontation avec les switches déjà connus. Ajout si nécessaire des nouveaux switches,
  - b. Récupération de la **table ARP** du switch,
  - c. Recherche de nouveaux **routeurs** et confrontation avec ceux déjà connus. Si nécessaire, ajout des nouveaux routeurs. Mise en ordre de la liste des switches afin que les routeurs soient prioritaires. Les routeurs d'adresses IP n'appartenant pas aux plages autorisées – si toutefois elles sont déclarées – ne sont pas pris en compte,
  - d. Recherche des VLAN,
  - e. Récupération des **adresses MAC des interfaces du switch**,
  - f. Pour chaque VLAN du switch courant :
    - i. Récupération des **correspondances adresses MAC/Ports**
    - ii. **Mise à jour des correspondances** du switch courant pour supprimer les adresses MAC des interfaces et l'IP de ce même switch ! On supprime par la même occasion les ports où il n'y a aucune MAC pour optimiser le temps d'exécution global du script,
  - g. **Mode poursuite** : scan des adresses IP issues des adresses MAC obtenues. Si détections de switches, confrontation avec ceux déjà connus et ajout des nouveaux. Vérification que le niveau maximal de profondeur à explorer n'est pas atteint, sinon, cette étape est ignorée...
  - h. **Fusion des correspondances** de chaque VLAN pour avoir une seule et unique correspondance pour ce switch,
6. Recherche des **liaisons inter-switchs** (avec test de réciprocité),
7. Recherche des **liaisons switches-routeurs** (sans test de réciprocité),
8. Recherche des **switchs non gérables**,
9. **Exportation des résultats** sous forme de deux tableaux : un pour les équipements, un pour les liaisons entre ces derniers.



## ***B. Suivie des modifications sur le réseau***

Le but de cette application est de découvrir la topologie d'un réseau, mais également de suivre son évolution et de dresser un historique des modifications survenues. Pour cela, le moteur de découverte sera exécuté régulièrement. Suivant la taille du réseau étudié et la fréquence d'exécution du moteur, les données à stocker peuvent devenir vite importante. C'est pour cela que la première fois qu'une découverte est faite sur le réseau, l'ensemble des informations sont stockés, tandis que pour les prochaines découvertes, seules les modifications seront enregistrées. C'est la seule solution optimale pour posséder un historique complet et néanmoins économiser le maximum de place possible sur les volumes de stockage. Par contre, ce choix impose des traitements supplémentaires à effectuer sur les données obtenues. Nous distinguons quatre types de modification :

- Ajout d'un nouvel équipement / d'une nouvelle liaison,
- Suppression d'un équipement / d'une liaison,
- Modification de paramètres d'un équipement / d'une liaison,
- Réapparition d'un équipement / d'une liaison après sa suppression.

Il faut cependant garder à l'esprit que ce moteur, tel quel, ne permet pas vraiment de faire de la surveillance temps réel : le protocole SNMP n'a pas été prévu pour. Actuellement, seule la méthode consistant à exécuter régulièrement le moteur de recherche est utilisée. Par contre, les options de fonctionnement du moteur peuvent varier : en effet nous pouvons imaginer le cas suivant :

- La découverte initiale se fait avec toutes les options actives, incluant les scans de plages d'IP.
- Comme le cœur du réseau ne doit pas évoluer souvent, une telle découverte est effectuée une fois par jour par exemple
- Par contre, une découverte partielle sans scan d'IP, est effectuée beaucoup plus régulièrement, par exemple tous les quarts d'heures ou toutes les heures suivant que le but soit de surveiller des utilisateurs ou de vérifier que des modifications à long terme non souhaitées soient apportées au réseau.

Dans tous les cas, le moteur ne doit pas être appelé de tel manière qu'il tourne en continue, ponctionnant dès lors une partie de la bande passante du réseau et du temps processeur des nœuds interrogés pour ses requêtes.

## ***C. Stockage des topologies***

Comme nous venons de le voir, il faut stocker d'une part la topologie du réseau initiale, et d'autre part les modifications qui lui sont apportées. Enfin, afin de manipuler facilement ces données, nous avons opté pour l'utilisation d'une base de données, en l'occurrence MySQL. Le moteur de découverte renvoyant deux tableaux PHP, l'un pour les équipements et l'autre pour leurs liaisons, nous les transférons dans deux tables 'ListeEquipements' et 'ListeLiens' qui assurent le même rôle que les tableaux initiaux. Les modifications sont quant à elles enregistrées dans les tables 'ChangementEquipements' et 'ChangementLiens'. Deux tables supplémentaires, 'ListeTypes' et 'ListeOid' sont créés afin d'indiquer le type des équipements pour la première et d'associer l'ObjectID à un type de matériel pour le seconde. Pour plus d'information, se reporter au paragraphe Auto-apprentissage du type des équipements. La figure suivante représente le modèle entités-associations correspondant.

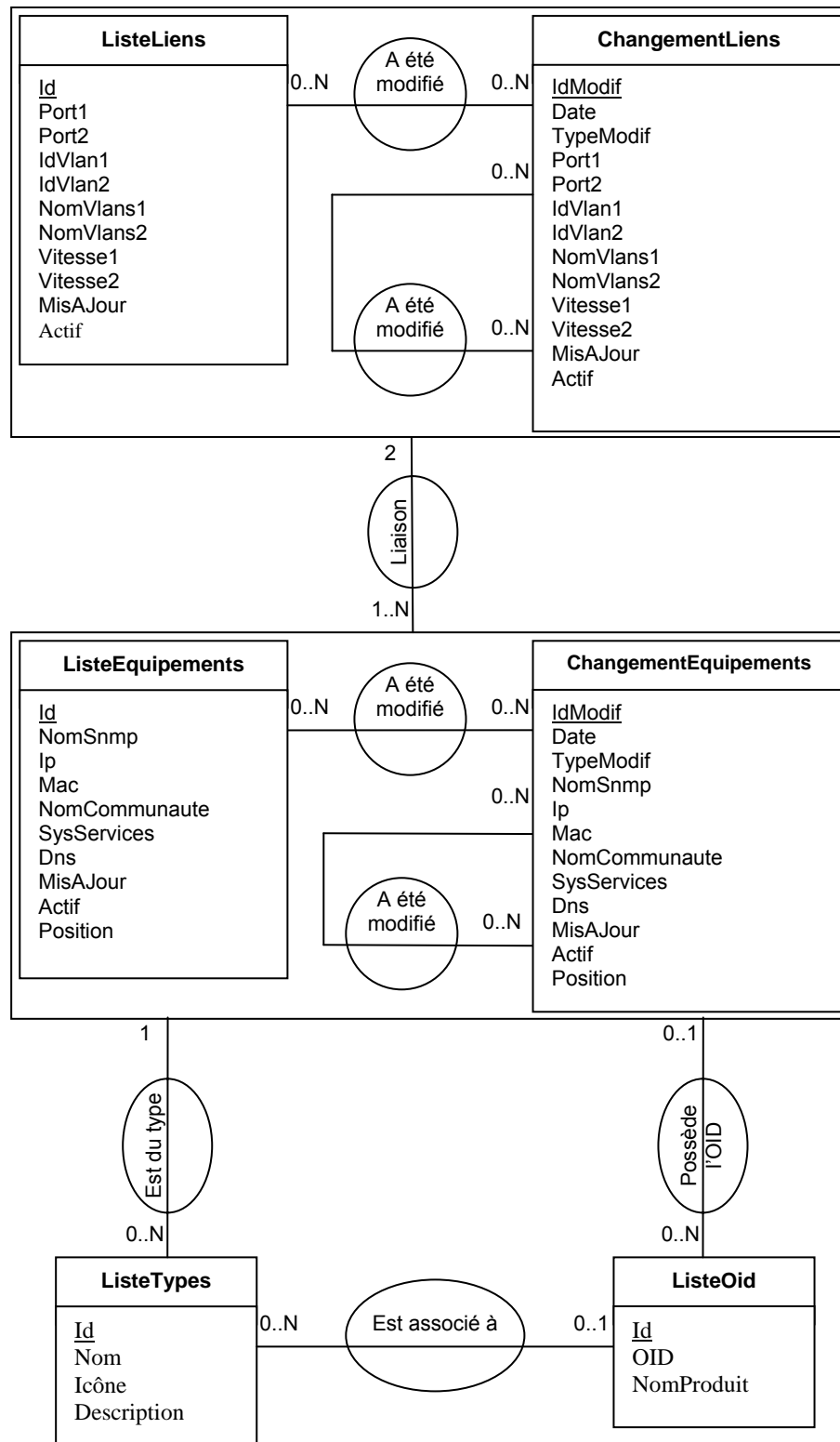


Figure 16 : Schéma entités-associations employé

Ce graphe ne prend pas en compte le fait que plusieurs réseaux à surveiller peuvent avoir été définis. En effet, l'utilisateur peut créer plusieurs réseaux, chacun étant alors caractérisé par un identificateur \$ID. Les jeux de tables décrivant la topologie d'un réseau sont ainsi toutes préfixés par cet identificateur. Par exemple la table 'ListeLiens' est en fait la table '\$ID\_ListeLiens'.

La table 'ListeEquipements' possède les attributs suivant :

- **Id** : Identificateur de l'équipement (Clé primaire),
- **NomSnmpp** : Nom du matériel obtenu par SNMP,
- **Ip** : Son adresse IP s'il en possède une,
- **Mac** : Son adresse MAC,
- **NomCommunaute** : Le nom de communauté s'il supporte le SNMP,
- **SysServices** : Cf. Fonction GetTypeServicesBySNMP page 35,
- **Type** : Type de matériel (Clé étrangère, de 'ListeTypes'),
- **OidProduit** : Cf. Fonction GetObjectIDBySNMP page 36,
- **Dns** : Nom DNS s'il possède une IP et que les noms DNS ont été résolus,
- **MisAJour** : Utile pour la Fonction ModificationEtatReseau page 44,
- **Actif** : Indique si l'équipement est actuellement présent sur le réseau ou non,
- **Position** : Permet de spécifier la position de l'équipement lors de son affichage.

La table 'ListeLiens' possède les attributs suivant :

- **Id** : Numéro d'indentification de la liaison
- **IdEq1** : Id de l'équipement 1 (Clé étrangère, de 'ListeEquipements' ou 'ChangementEquipements'),
- **IdEq2** : Id de l'équipement 2 (Clé étrangère, de 'ListeEquipements' ou 'ChangementEquipements'),
- **Port1** : Port sur lequel se fait la liaison, côté équipement 1,
- **Port2** : Idem côté 2,
- **IdVlan1** : Numéro du VLAN du port côté équipement 1,
- **IdVlan2** : Idem côté 2,
- **NomVlans1** : Nom du VLAN du port côté équipement 1,
- **NomVlans2** : Idem côté 2,
- **Vitesse1** : Vitesse de fonctionnement du port côté équipement 1,
- **Vitesse2** : Idem côté 2,
- **MisAJour** : Utile pour la Fonction ModificationEtatReseau page 44,
- **Actif** : Indique si la liaison existe actuellement.

Les tables 'ChangementEquipements' et 'ChangementLiens' reprennent les mêmes attributs ainsi que les suivants qui sont ajoutés :

- **IdModif** : Clé primaire de ces tables,
- **Date** : Date à laquelle la modification a été découverte,
- **TypeModif** : Numéro identifiant le type de modification :
  - **0** : *Suppression*
  - **1** : *Ajout*,
  - **2** : *Modification*,
  - **3** : *Réapparition*.

La table 'ListeTypes' possède les attributs suivant :

- **Id** : Clé primaire de cette table, identifie un type particulier,
- **Nom** : nom donné au type en question,
- **Icône** : Icône pour affichage,
- **Description** : Description éventuelle du type.

La table 'ListeOid' possède les attributs suivant :

- **Id** : Clé primaire de cette table,
- **OID** : ObjectID d'un produit,
- **IdType** : Type associé à cet ObjectID (clé étrangère, de 'ListeTypes'),
- **NomProduit** : Description éventuelle du produit.

## ***D. Implémentation***

L'implémentation a été faite en PHP (Hypertext Preprocessor), d'une part parce que ce langage dispose par défaut de fonctions SNMP, et d'autre part parce que ce programme pourra être ainsi facilement intégré sur un serveur web. Du fait de l'utilisation de ce langage de programmation, nous parlerons désormais plutôt de script que de programme. Il nous arrivera cependant d'utiliser le terme programme ou application sans que cela ne doive porter réellement à confusion. Une autre raison quant à l'utilisation de PHP est sa grande aisance pour toute manipulation de tableau. Or nos fonctions utilisent comme arguments et fournissent comme résultat principalement des tableaux. Il est donc agréable de pouvoir faire des fusions, intersection, soustraction et autres manipulations en appelant des fonctions optimisées, rapides et fiables conçues par des développeurs professionnels, plutôt que de devoir les recoder soit même (ce qui aurait été le cas en utilisant le langage Java).

De plus, bien que ce langage permette la programmation orientée objet, nous avons plutôt utilisé des bibliothèques de fonctions et un script principal. En effet, afin d'avoir un code simple, clair et facilement maintenable, nous avons divisé chaque partie élémentaire en autant de fonctions. Le script principal, se contente alors d'appeler ces différentes fonctions. Nous avons par ailleurs regroupé ces fonctions en plusieurs catégories. Celles-ci pourraient s'apparenter à des classes puisqu'elles regroupent des fonctions de même nature, mais ce ne sont pas des objets en tant que tel et par exemple l'héritage n'est pas utilisé. C'est pourquoi nous parlerons plus de *bibliothèques* comme bibliothèques de fonctions de même nature. Enfin, un avantage de développer des bibliothèques (tout comme ce serait le cas avec des classes) et d'une part de pouvoir les réutiliser pour d'autres applications, et d'autre part de pouvoir rajouter des fonctions sans devoir modifier l'ensemble des fichiers.

## **1. Débogage**

Il est important de prévoir dès le début des fonctions de débogage dans le code afin de faciliter la recherche et la correction des erreurs de programmation, et également de valider certains algorithmes ainsi que leurs optimisations. Etant donné que ce code est réalisé en PHP et est donc destiné à être exécuté à partir d'un serveur web, nous avons opté pour l'affichage de message dans la page web au cours de l'exécution des scripts et si besoin la création de page HTML spécifiques.

Par ailleurs, chaque fonction, à deux ou trois exceptions près, disposent d'un paramètre supplémentaire *\$niv\_debug* qui correspond au niveau de débogage ou encore de verbosité, anglicisme représentant la quantité d'information (sur le déroulement des fonctions) affichée selon notamment leurs importances. Cet argument peut prendre trois valeurs :

- **0** : qui revient à désactiver le mode débogage : aucune information non indispensable ne sera affichée.
- **1** : un certain nombre de messages sont affichés, montrant le déroulement du script. Par exemple, les appels des différentes fonctions sont signalés. Ce mode est utile pour suivre l'exécution des scripts et non pour déboguer une fonction en particulier
- **2** : dans ce mode, le déroulement interne de chaque fonction sera détaillé. C'est le mode le plus « coûteux » en termes d'affichage puisque de nombreux messages seront affichés et il restera à la charge du développeur de les trier. Ce mode est utile pour suivre le déroulement d'une fonction en particulier, connaître ces arguments, son résultat, etc.

Dans chaque fonction, un test est nécessaire à chaque fois qu'un message de débogage peut potentiellement être affiché. Afin d'optimiser la vitesse d'exécution des scripts, nous pouvons supprimer ces tests une fois que l'application sera finalisée.

Dans les paragraphes qui suivent sur les fonctions que nous avons développé, cet argument (*\$niv\_debug*) ne sera plus indiqué.

## 2. Gestion des erreurs

De plus, ces fonctions ont été écrites de façon à ce qu'elle renvoie un résultat exploitable (pouvant être nul) afin de ne pas interrompre l'exécution du script. Ainsi, même si par exemple un switch ne répond pas lorsqu'il est interrogé, le script continuera de traiter les autres switches. Au pire, ce manque d'information entraînera l'affichage d'une topologie incomplète voire partiellement erronée, au mieux ceci n'aura aucune incidence sur la découverte de la topologie du réseau ! Il faut aussi remarquer que dans le cadre de réseau de taille conséquente, par exemple celui de l'école, on ne peut garantir un résultat totalement juste, et il n'est pas anormal qu'un switch ne réponde pas, c'est pourquoi il est important que nos fonctions tiennent compte de ces possibilités (non réponse d'un équipement, etc.).

Enfin, toutes les fonctions SNMP pouvant ne pas répondre à cause d'un problème non lié au programme lui même mais à l'environnement logiciel ou matériel, sont codées de façon à ne pas remonter à l'utilisateur de message d'erreur. Ainsi, ce dernier ne verra pas de messages incompréhensibles, et si ces fonctions n'aboutissent pas, elles renverront comme nous l'avons vu précédemment un résultat nul ne bloquant pas la suite du script.

## 3. Dépendance et organisation des différentes librairies

Voici un diagramme représentant les librairies, les fonctions qu'elles contiennent et leurs dépendances (représenté par une flèche orienté dans le sens de la dépendance de la même manière que les diagrammes UML) :

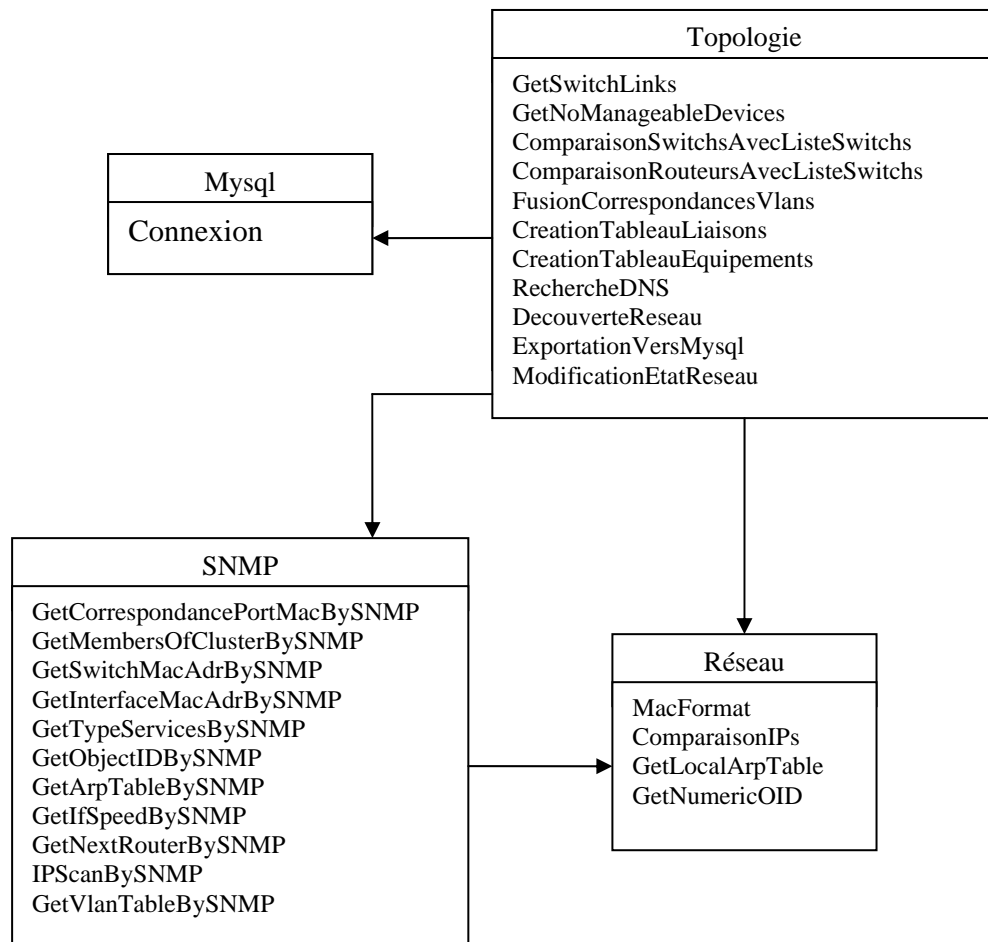


Figure 17 : Dépendances des différentes librairies

## 4. Librairie SNMP

Cette librairie inclue l'ensemble des fonctions permettant de récupérer les informations des switchs par l'intermédiaire du protocole SNMP.

Voici les fonctions qui y sont incluses :

- **GetCorrespondancePortMacBySNMP** : Récupère les correspondances *adresses MAC/Ports* du switch,
- **GetMembersOfClusterBySNMP** : Permet de trouver les membres d'un cluster de switchs. Ne s'applique qu'aux switchs de marque Cisco,
- **GetSwitchMacAdrBySNMP** : Donne l'adresse MAC du switch,
- **GetInterfaceMacAdrBySNMP** : Donne les adresses MAC des interfaces du switch,
- **GetTypeServicesBySNMP** : Permet d'indiquer le type d'équipement (switch, switch de niveau 3, routeur, etc.),
- **GetObjectIDBySNMP** : Permet de récupérer l'OID de l'entreprise commercialisant chaque nœud détecté du réseau,
- **GetArpTableBySNMP** : Récupère la table ARP,
- **GetIfSpeedBySNMP** : Obtient la vitesse des interfaces du switch,
- **GetNextRouterBySNMP** : Indique les routeurs que le switch connaît,
- **IPScanBySNMP** : Scanne par SNMP une plage d'adresses IP afin de découvrir des switchs potentiels,
- **GetVlanTableBySNMP** : Récupère la table des VLAN du switch. Ne s'applique qu'aux switchs de marque Cisco.

### a) Emploi des MIB et des OID

Avant de détailler chacune de ces fonctions, commençons par quelques remarques d'ordre général. Tout d'abord, dans la mesure du possible, seules les MIB standards ont été utilisées. Néanmoins, certains résultats nécessaires n'ont pu être obtenus qu'avec l'emploi de MIB propriétaires. Quand ce fut le cas, nous n'avons utilisé que les MIB Cisco qui ont le mérite d'être généralement supportée par les autres constructeurs, par exemple HP.

Qui est plus est, nous avons systématiquement utilisé des OID numériques et non leurs expressions alphanumériques. En effet, désigner les objets par des noms impose d'avoir installé les MIB sur le serveur que nous allons utiliser. Par contre, employé les termes numériques fonctionne sans que le daemon (ou service) SNMP ne connaisse les MIB mises en jeu. L'**Annexe 4 : OID et MIB** utilisées récapitule l'ensemble des OID utilisées dans cette librairie.

### b) Arguments des fonctions SNMP

Afin de créer une requête SNMP, il est nécessaire de connaître d'une part l'adresse IP ou le nom DNS du switch à interroger, et d'autre part leurs noms de communauté. Ces deux éléments constituent les arguments systématiques de nos fonctions. Ce dernier paramètre est par défaut à la valeur « *public* » (si on ne le précise pas, il prendra cette valeur).

Par ailleurs, et suivant les cas, d'autres arguments peuvent être nécessaire, par exemple l'argument optimisation, suivant qu'il est vrai ou faux, permet d'utiliser les optimisations prévues dans l'algorithme de la fonction en question, ou non. Ces arguments spécifiques seront détaillés pour chaque fonction.

En Annexe 1 : Librairie SNMP et fonctions (page 64) de ce rapport se trouve un récapitulatif des fonctions, incluant les arguments, leurs formats, ainsi que celui des résultats de ces fonctions.

### c) Fonction GetCorrespondancePortMacBySNMP

Il s'agit de l'une des principales fonctions de cette librairie. En effet, c'est elle qui va récupérer les correspondances *MAC/Ports* du switch, ou dit autrement, la liste des adresses MAC que le switch connaît sur chacun de ses ports.

Pour obtenir ces informations, nous utilisons quatre tables issues des MIB *BRIDGE-MIB* (rfc1493) et *ifMIB* (rfc2863). Nous n'utilisons ainsi que des MIB standard, qui plus est « *mandatory for all bridges* », autrement dit obligatoire pour tout switch. Voici les OID utilisées :

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.2.1.17.4.3.1.1	dot1dTpFdbAddress	Adresses MAC apprises par le switch en fonction de l'OID du numéro du <i>bridge</i> associé au port correspondant
.1.3.6.1.2.1.17.4.3.1.2	dot1dTpFdbPort	OID correspondant aux numéros des <i>bridges</i> associés à chacun des ports
.1.3.6.1.2.1.17.1.4.1.2	dot1dBasePortIfIndex	Correspondance entre les OID des ports et les numéros des <i>bridges</i>
.1.3.6.1.2.1.31.1.1.1.1	ifName	Nom du port en fonction de son OID

**Tableau 1 : OID utilisées (Fonction GetCorrespondancePortMacBySNMP)**

Nous récupérons à partir de ces OID quatre tableaux, respectivement *\$Mac\_Learned*, *\$Bridge\_Port\_Number*, *\$Bridge\_Port\_ifIndex* et enfin *\$Port\_ifIndex\_Name*.

Pour plus d'information, se reporter à l'Annexe 5 : Récupération des correspondances MAC/Ports page 81.

Pour obtenir les correspondances souhaitées, nous utilisons l'algorithme suivant :



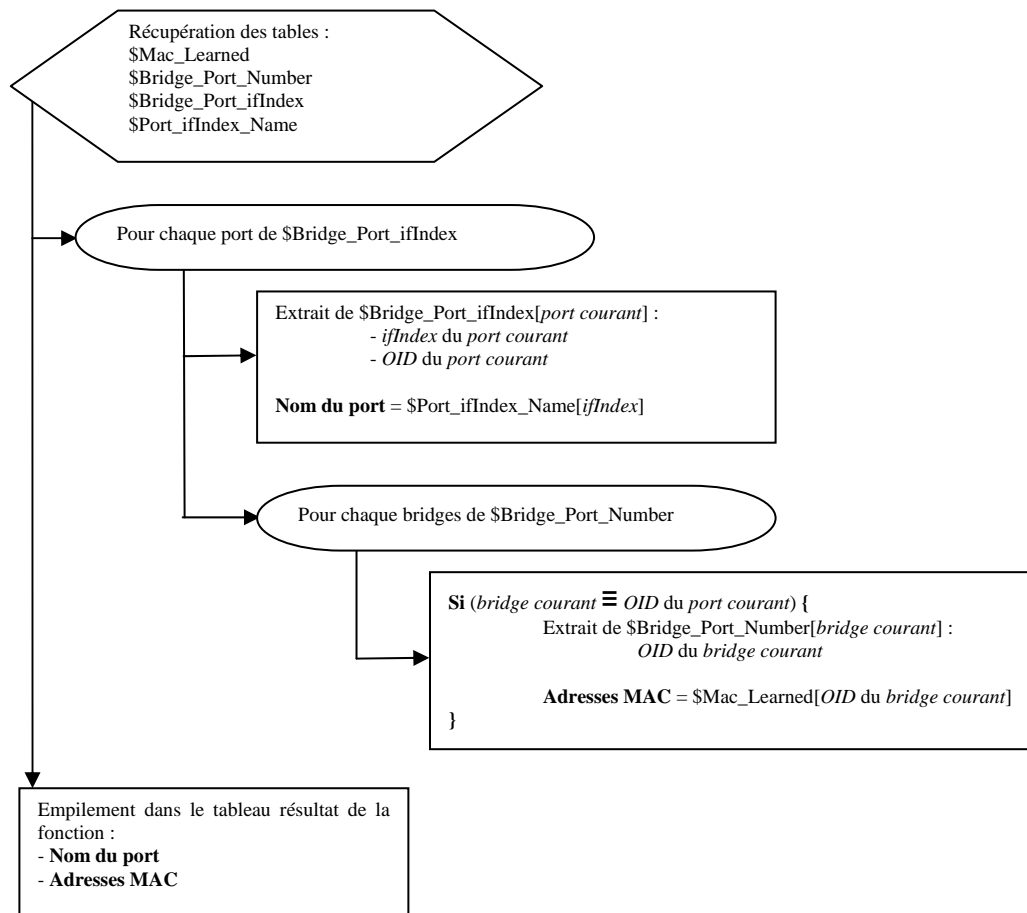


Figure 18 : Algorithme de la Fonction `GetCorrespondancePortMacBySNMP`

Après l'exécution d'une boucle sur le port courant et avant de passer au port suivant, nous pouvons supprimer les éléments du tableau `$Bridge_Port_Number` déjà parcourus (en mettant à « vrai » l'argument « *Optimisation* » de la fonction). En effet, les informations qui s'y trouvent ne concernent qu'un port à la fois et comme la première boucle se fait justement sur les ports du switchs, nous ne repassons jamais sur un même port. Cette optimisation est utile ici car pour chaque port, nous bouclons sur cette table, alors que les autres tableaux sont utilisés directement en indiquant l'index qui nous intéresse. A noter enfin que cette optimisation ne fait pas gagner du temps si peu d'éléments sont concernés (il faudra alors autant de temps pour supprimer ces éléments que pour les parcourir à la recherche des numéros des bridges d'autres ports). C'est pourquoi la suppression ne se fait que s'il y a un nombre minimum (fixé expérimentalement) d'éléments concernés !

Comme certains switchs peuvent avoir beaucoup de ports (50 par exemple), les quatre tables sont obtenus en utilisant la fonction `snmpwalkoid` (similaire à `snmpwalk`) qui récupère toutes les informations dont les OID sont issus de l'OID initiale passée en paramètre. (Par exemple, appeler `snmpwalk` avec comme OID initiale `.1.3.6.1.2.1.1` renverra les informations ayant pour OID `.1.3.6.1.2.1.1.x`). Cette méthode est en effet beaucoup plus rapide que d'utiliser `snmpget` qui ne renvoie les informations correspondants qu'à une seule OID à la fois (et donc si on obtient 50 valeurs avec `snmpwalk`, il faudrait appeler 50 fois `snmpget` pour obtenir le même résultat !). Cette dernière remarque est d'ailleurs valable pour les autres fonctions.

### d) Fonction GetMembersOfClusterBySNMP

La fonction `GetMembersOfClusterBySNMP` permet de vérifier à partir d'un switch possédant une adresse IP (donc un switch interrogeable !) s'il est membre d'un cluster. Etant donné que ce type de configuration n'est présent que sur la résidence des élèves avec des switches de marque Cisco, cette fonction n'a été développée que pour ce cas particulier. Nous avons par ailleurs constaté que celle-ci ne fonctionnait pas avec les switches HP.

Nous accédons aux membres du cluster soit par leurs adresses IP s'ils en possèdent une (et alors l'avantage du cluster n'est pas de minimiser le nombre d'IP nécessaire pour les gérer mais que cette administration se fasse sous une unique interface commune), soit par l'IP du *commandeur* en utilisant un nom de communauté spécial. En effet, en imaginant que le nom de la communauté du switch principal soit *public*, les membres auront pour communauté *public@es1*, *public@es2*, *public@es3*, etc. dans le cadre de switch Cisco. Il semble qu'utiliser des noms de communauté spécifique soit la solution adoptée par les différents constructeurs, ce qui est logique puisqu'il peut n'y avoir qu'une seule IP pour les administrer et que les seuls paramètres indispensables pour envoyer une requête SNMP sont justement ces deux arguments. Néanmoins, nous constatons que la façon d'utiliser les communautés est propre à chaque constructeur. HP utilise par exemple une autre convention : les membres auront pour communautés *public@sw0*, *public@sw1*, etc.

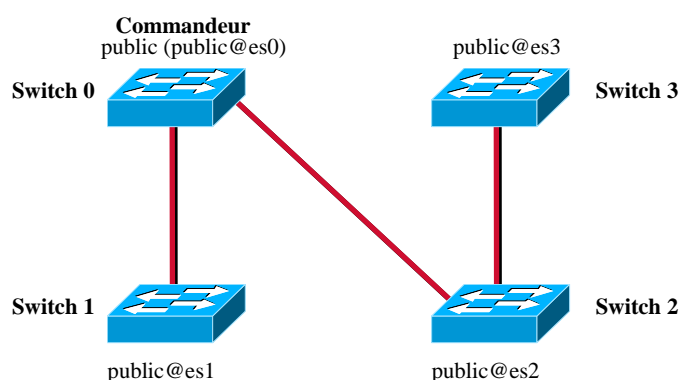


Figure 19 : Nom de communauté des switches d'un cluster

Enfin, cette fonction devant trouver les membres, faut il encore pouvoir détecter parmi les informations SNMP obtenues ces noms de communauté. Il s'est avéré que sur les équipements Cisco, ces informations sont situées dans la MIB *ENTITY-MIB* (rfc2737), à partir de l'OID suivante :

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.2.1.47.1.2.1.1.4	<i>entLogicalCommunity</i>	Nom de la communauté permettant d'obtenir des informations sur l'entité logique correspondante

Tableau 2 : OID utilisée (Fonction `GetMembersOfClusterBySNMP`)

Cette branche de la MIB correspond comme son nom l'indique aux entités logiques et non physiques (par exemple les VLAN s'y trouvent) et plus particulièrement à tous ce qui à trait aux communautés... Nous relevons dans cette branche tous les noms de communautés disponibles. Comme généralement on y trouve également ceux qui permettent d'accéder aux différents VLAN (cf. Fonction `GetVlanTableBySNMP` page 38) ainsi que la communauté par défaut (celle que l'on utilise en argument de cette fonction). Afin de ne pas retenir celles qui pointent vers le même switch, nous récupérons pour chaque nouveau membre potentiel leur

adresse MAC et nous les comparons avec celle du switch qui serait alors commandeur. Les MAC devant normalement être uniques, nous éliminons ainsi tout doublon !

Finalement, cette fonction renvoie une liste de switches contenant tous les membres détectés. Il faudra ensuite comparer cette liste avec celle déjà existante et ajouter uniquement ceux qui n'y figurent pas encore. C'est le rôle de la Fonction *ComparaisonSwitchsAvecListeSwitchs* (page 43). Dans le cas où il n'y aurait pas de cluster, cette fonction renvoie une liste vide.

### e) Fonction GetSwitchMacAdrBySNMP

Cette fonction renvoie l'adresse MAC du switch formaté par la Fonction *MacFormat* (page 39). Cette adresse correspond à l'IP du switch, autrement dit l'IP que nous utilisons pour pouvoir l'administrer. Elle est obtenue à partir de la MIB *BRIDGE-MIB* (rfc1493). Cette OID est par ailleurs obligatoire pour tout switch.

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.2.1.17.1.1.0	<i>dot1dBaseBridgeAddress</i>	Adresse MAC, unique, du bridge regroupant un ensemble de port

Tableau 3 : OID utilisée (Fonction GetSwitchMacAdrBySNMP)

### f) Fonction GetInterfaceMacAdrBySNMP

Cette fois ci, le but est de récupérer l'ensemble des adresses MAC des interfaces du switch courant. Nous l'obtenons à partir de la MIB *ifMIB* (rfc2863).

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.2.1.2.2.1.6	<i>ifPhysAddress</i>	Adresse MAC de toutes les interfaces du switch

Tableau 4 : OID utilisée (Fonction GetInterfaceMacAdrBySNMP)

Comme pour la fonction précédente, les adresses sont formatées grâce à la Fonction *MacFormat* (page 39). Nous avons remarqué que cet OID nous donnait également l'adresse MAC de l'IP du switch, sans que nous n'ayons un moyen simple de la supprimer. Il faudra donc faire la différence des tableaux issus de cette fonction et de la précédente.

### g) Fonction GetTypeServicesBySNMP

Cette fonction permet de connaître les types de services supportés sur le switch interrogé. Pour cela, nous récupérons la valeur *sysServices* située dans la MIB *MIB-II* (rfc1213).

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.2.1.1.7.0	<i>sysServices</i>	Cf. explications ci-dessous

Tableau 5 : OID utilisée (Fonction GetTypeServicesBySNMP)

Cette valeur est obtenue par un OU binaire à partir des éléments suivants :

<i>Bit</i>	<i>Valeur décimale</i>	<i>Fonction</i>	<i>Couche OSI</i>
0x01	1	« Physique »	1
0x02	2	« Liaison »	2
0x04	4	« Réseau »	3
0x08	8	« Transport »	4
0x10	16	« Session »	5
0x20	32	« Présentation »	6
0x40	64	« Application »	7

**Tableau 6 : Couches OSI et valeurs de sysServices correspondantes**

Nous utiliserons plus tard cette valeur pour essayer de proposer à l'utilisateur le type de nœud, dans le cas où nous ne l'aurions encore jamais rencontré. Pour plus d'informations, se reporter au paragraphe Auto-apprentissage du type des équipements page 20.

### h) Fonction GetObjectIDBySNMP

Cette fonction utilise la MIB *MIB-II* (rfc1213).

<b>OID numérique</b>	<b>Nom de l'objet</b>	<b>Description de l'objet</b>
.1.3.6.1.2.1.1.2.0	<i>sysObjectID</i>	OID du nœud (entreprise, marque, référence du produit) en question

**Tableau 7 : OID utilisée (Fonction GetObjectIDBySNMP)**

Cette fonction permet donc de récupérer cette OID afin d'être utilisée plus tard pour identifier le nœud détecté. Sachant qu'il existe une liste complète des OID des constructeurs, il suffit en effet de recouper l'information obtenue avec cette liste. La seule difficulté vient du fait que cette liste est longue et peut difficilement être exploitée directement, par exemple pour indiquer s'il s'agit d'un switch ou d'un routeur ! De plus, il serait long, fastidieux voir inutile de prétraiter cette liste afin d'associer un produit à un type de nœud ; la solution réside alors dans l'auto-apprentissage. Ainsi, le script récupère d'une part l'OID du produit grâce à cette fonction, d'autre part prédit le type de matériel grâce à la fonction précédente. Enfin, ces informations seront présentées à l'utilisateur pour qu'il les vérifie et les corrige le cas échéant. Cette association produit-type de l'équipement sera alors mémorisée par le script pour les prochaines découvertes.

### i) Fonction GetArpTableBySNMP

Cette fonction permet de retourner la table ARP du switch (il ne s'agit donc pas de la table ARP locale ; celle-ci sera récupérée par la Fonction *GetLocalArpTable*, page 40). Cette table se trouve dans la MIB *IP-MIB* (rfc1213).

<b>OID numérique</b>	<b>Nom de l'objet</b>	<b>Description de l'objet</b>
.1.3.6.1.2.1.4.22.1.2	<i>ipNetToMediaPhysAddress</i>	Adresses MAC en fonction des adresses IP

**Tableau 8 : OID utilisée (Fonction GetArpTableBySNMP)**

Nous pouvons ainsi extraire depuis l'OID l'adresse IP et la valeur retournée est l'adresse MAC qui y est associée. Les adresses MAC obtenues sont mises en forme grâce à la Fonction *MacFormat* (page 39). Cette OID devrait être présente pour tout type de switch. Néanmoins, nous avons constaté que certains ne répondaient pas sans pour autant que cela porte préjudice par la suite : nous avons donc ignoré ce problème.

### j) Fonction GetIfSpeedBySNMP

Cette fonction permet de connaître la vitesse de fonctionnement théorique de chaque port. Attention, il s'agit bien de la vitesse théorique réelle du port en question et non pas de sa vitesse théorique maximale de fonctionnement ! Ces informations sont tirées des MIB *IP-MIB* (rfc1213) et *ifMIB* (rfc2863).

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.2.1.2.2.1.5	<i>ifSpeed</i>	Vitesse théorique actuelle des ports du switch. Exprimée en bits par seconde.
.1.3.6.1.2.1.31.1.1.1.1	<i>ifName</i>	Nom du port en fonction de son OID

Tableau 9 : OID utilisées (Fonction GetIfSpeedBySNMP)

La première OID donne une table indiquant la vitesse des ports en fonction de leurs OID, qui est alors recoupée avec le nom de ces mêmes ports grâce à la table obtenue à partir de la deuxième OID.

### k) Fonction GetNextRouterBySNMP

Il s'agit ici de récupérer les IP des routeurs que le switch connaît. On peut les obtenir depuis la MIB *IP-MIB* (rfc1213).

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.2.1.4.21.1.7	<i>ipRouteNextHop</i>	Adresses IP des routeurs connus par ce switch

Tableau 10 : OID utilisée (Fonction GetNextRouterBySNMP)

Nous obtenons juste une liste d'adresses IP. Pour être précis, ces adresses correspondent aux prochains routeurs présents sur une classe d'IP donnée. Nous pouvons par ailleurs savoir de quelle classe il s'agit en interrogeant la même MIB à l'adresse .1.3.6.1.2.1.4.21.1.1 (*ipRouteDest*). Cette dernière information n'est pas récupérée car nous voulons juste trouver les routeurs et non, dans cette version du projet, savoir vers quelles routes ils pointent. A noter que des switches de niveau 3 sont parfois présent dans cette liste même s'ils n'ont pas de fonction de routage activée. Toutefois, cette liste étant forcément vérifiée puis triée par la suite, ceci ne pose pas de problème.

### l) Fonction IPScanBySNMP

Le but de cette méthode, comme son nom l'indique, est de scanner une plage d'IP en recherchant les équipements supportant le protocole SNMP. Pour cela, celle-ci prend en argument une liste d'adresses IP ainsi qu'une liste de noms de communauté à tester. Chaque équipement est interrogé sur une OID basique (.1.3.6.1.2.1.1.5.0 : *sysName*) afin d'une part de ne pas utiliser une MIB qui pourrait ne pas être présente sur tous les matériels que nous souhaitons détecter, et d'autre part pour que ce scan soit le plus rapide et le moins lourd possible (en ce qui concerne la charge du réseau). Cette fonction élimine également les doublons parmi l'ensemble des switches qu'elle a trouvés (utile par exemple si nous scanons une plage d'IP comportant plusieurs classes C et que des routeurs aient une IP dans chaque classe). Remarquons enfin que les switches pour lesquels nous n'aurions pas d'adresse MAC ne sont pas pris en compte : puisque tous notre moteur de détection de la topologie du réseau se base avant tout sur les adresses MAC, ces équipements n'ont en effet aucun intérêt pour nous !

### m) Fonction GetVlanTableBySNMP

Cette fonction est importante car elle permet de récupérer la liste des VLAN présent sur un switch. De fait, il faut savoir que lorsque nous interrogeons un switch pour obtenir ces correspondances *MAC/Ports*, nous ne récupérons que celles dont les ports appartiennent au VLAN par défaut, ou dit autrement au VLAN utilisé justement pour administrer le switch. Il existe cependant (au moins) une méthode permettant de récupérer ces informations pour les autres VLAN. Puisque les requêtes SNMP se base sur l'IP (fixe), le nom de communauté et les MIB (également fixes), certains constructeurs utilisent les noms de communauté (comme dans le cas des clusters) pour atteindre chaque VLAN.

Signalons tout d'abord qu'étant donné le parc informatique utilisé pour nos tests, nous avons étudiés ce problème uniquement avec les switchs de marque Cisco et cette fonction n'est malheureusement pas valable avec ceux de marque HP !

Afin d'obtenir la liste des VLAN, nous avons donc utilisé la MIB propriétaire *CISCO-VTP-MIB* (VTP : VLAN Trunk Protocol) avec les OID suivantes :

OID numérique	Nom de l'objet	Description de l'objet
.1.3.6.1.4.1.9.9.46.1.3.1.1.3	<i>vtpVlanType</i>	Type du VLAN considéré
.1.3.6.1.4.1.9.9.46.1.3.1.1.4	<i>vtpVlanName</i>	Nom du VLAN considéré

Tableau 11: OID utilisées (Fonction GetVlanTableBySNMP)

Enfin, les numéros des VLAN (qui servent d'identifiant et qui sont donc uniques) se trouvent à la fin de l'OID. Par exemple nous trouverons les informations désirées sur le VLAN 51 en interrogeant les OID *.1.3.6.1.4.1.9.9.46.1.3.1.1.3.1.51* et *.1.3.6.1.4.1.9.9.46.1.3.1.1.4.1.51*. La première information (*vtpVlanType* indique naturellement le type de VLAN. Le seul type qui nous intéresse est dans notre cas le type « *Ethernet* » et tous les autres ne seront alors pas pris en compte. Une fois ce premier tri effectué, nous récupérons les noms des VLAN en les associant avec leurs numéros. Enfin, nous avons remarqué que les VLAN de management avait pour nom « *default* » et que celui-ci semble ne pas être modifiable, contrairement aux autres VLAN créés par l'utilisateur. Une fois ces informations obtenues, nous pouvons créer les noms de communauté qui seront utilisés par les autres fonctions. Ceux-ci sont obtenus de manière similaire au cas des clusters de switch (cf. Fonction *GetMembersOfClusterBySNMP* page 34) : on ajoute au nom de communauté du switch le numéro du VLAN précédé par l'arobase '@'. Voici deux exemples de noms de communauté : *public@51* et *public@es4@51*. Dans le premier cas, le switch n'est pas membre d'un cluster alors qu'il l'est dans le deuxième !

Pour finir, nous avons signalé le fait que cette fonction n'était pas compatible avec des switchs autres que des Cisco. Néanmoins, cette fonction sera appelée pour chaque switch est son résultat remplacera les noms de communauté précédemment définies afin de tenir compte des VLAN. Si jamais on obtient aucune information en interrogeant la MIB indiquée un peu plus haut, cette fonction renvoie malgré tout le nom de communauté initial pour ne pas bloquer la suite des scripts...

## 5. Librairie Réseau

Cette librairie comporte des fonctions utiles pour les applications réseaux :

- **MacFormat** : Permet de formater correctement une adresse MAC,
- **ComparaisonIPs** : Compare deux IP en indiquant laquelle est la plus grande ou éventuellement si elles sont égales,
- **GetLocalArpTable** : Récupère la table ARP local,
- **GetNumericOID** : transforme une OID quelconque en une OID numérique.

### a) Fonction MacFormat

Cette fonction prend pour argument une adresse MAC dont le séparateur est « : ». Le résultat sera cette adresse formatée afin que chaque terme hexadécimal soit en majuscule et sur deux chiffres. Par exemple, l'adresse '0:0:10:a1:0:ef' deviendra '00:00:10:A1:00:EF'. Ce formatage est important car ces adresses MAC sont considérées comme des chaînes de caractères et nous serons emmené à les comparer entre elles !

Si jamais l'argument est incorrect, principalement que les séparateurs des valeurs hexadécimal ne sont pas des « : », ou que l'adresse ne comporte pas six éléments (exemple '0:0:10:a1:0:ef' ou encore '0:0:10:a1:0'), le résultat sera une chaîne de caractère vide.

### b) Fonction ComparaisonIPs

Cette fonction compare deux adresses IP. Elle est utilisée pour classer les IP des tables ARP ou encore vérifier qu'une adresse IP appartient bien à une plage autorisée. Sa particularité est d'utiliser une fonction PHP *ip2long* qui convertit, de manière unique, une adresse IP numérique en un entier (généralement négatif). Cette fonction est également employée - en conjonction de la fonction réciproque *long2ip* - pour générer une liste d'IP à partir d'une plage donnée.

Il faut noter que cette fonction *ip2long* est également pratique pour vérifier qu'une IP est correcte ou non, et dans certain cas elle peut même corriger automatiquement cette IP (par exemple l'IP 193.48.225 deviendra 193.48.226.0), même si cette correction n'est pas forcément pertinente.

Dans le cas où cette fonction ne serait pas disponible pour une raison ou une autre, on peut trouver dans l'aide en ligne des fonctions PHP son algorithme parmi les commentaires d'utilisateurs (<http://fr2.php.net/manual/fr/function.ip2long.php>, commentaire du 24 Avril 2004), et ainsi la recoder. Voici le code reproduit ci-dessous :

```
<?php
if (!function_exists("ip2long")) {
    function ip2long($ip) {
        $ex = explode(".", $ip);
        if (count($ex)!=4) return -1;
        list($a, $b, $c, $d) = $ex;
        $a = $a*16777216;
        $b = $b*65536;
        $c = $c*256;
        return $a+$b+$c+$d;
    }
}
?>
```

### c) Fonction GetLocalArpTable

Cette fonction récupère la table ARP locale, autrement dit celle du serveur sur lequel l'application est exécutée. Pour cela, le script PHP va exécuter la fonction système '*arp -an*'. Cette commande permet bien entendu d'obtenir la table ARP, et les arguments '*a*' et '*n*' permettent respectivement d'obtenir la table complète et de ne pas faire la résolution DNS des adresses IP. La particularité de cette fonction est justement qu'elle fait appel à une fonction système et non à une fonction PHP. De fait, ceci implique plusieurs restrictions : cette fonction ne peut être utilisée telle qu'elle sous Windows. Pour la faire fonctionner, il faudrait supprimer l'option '*n*' et prendre en compte que la première ligne du résultat est la légende du tableau retourné ! Par ailleurs, PHP ne permet d'accéder par défaut qu'aux commandes dont les fichiers binaires se trouvent dans les dossiers « */bin* », « */usr/bin* » et « */usr/local/bin* » (sous Linux). Par exemple, le binaire se trouvait dans le dossier « */usr/sbin* » sur notre serveur de test à la résidence. Trois solutions sont alors possibles :

- Nous créons un raccourci ('*ln -s*' sous Linux) de « */usr/sbin/arp* » dans « */usr/bin* »,
- Nous créons un raccourci du binaire dans le dossier où se trouve la librairie puisque si la commande '*arp -na*' ne fonctionne pas, la fonction essaye alors d'exécuter '*./arp -na*',
- Ou alors tout simplement de ne pas utiliser cette fonction en utilisant l'option adéquate.

Une évolution possible de cette fonction serait de constituer une table de correspondance IP/DNS en enlevant l'option '*n*' de la commande utilisée, pour finalement ne demander le nom DNS des IP que si nous ne l'avons pas encore dans notre cache. Cette particularité n'est pas implémentée à l'heure actuelle.

### d) Fonction GetNumericOID

Cette fonction a pour but de transformer une OID quelconque en une OID numérique. Par exemple, l'OID '*CISCO-VTP-MIB::vtpVlanType.1.1*' devient l'OID '*.1.3.6.1.4.1.9.9.46.1.3.1.1.3.1.1*'. Cette fonction peut être utilisée pour obtenir un résultat homogène quelque soit la plate forme utilisée et quelques soient les MIB qui y sont installées. C'est particulièrement le cas de la Fonction *GetObjectIDBySNMP* (page 36).

Nous obtenons cette traduction en faisant appel à une fonction système puisqu'il n'y pas d'équivalent parmi les implémentations SNMP de PHP. La commande système est la suivante : '*snmptranslate -On \$OID*' où *\$OID* est remplacée par l'OID à traduire. Notons tout de suite que si cette OID est déjà numérique, cette commande renverra alors la même OID (nous sommes donc sûr d'obtenir un résultat). Sinon, les mêmes remarques de la fonction précédente s'appliquent ici puisque nous effectuons encore un appel à une commande système.



## 6. Librairie Topologie

La librairie « Topologie » contient l'ensemble des fonctions permettant la découverte de la topologie du réseau. Voici les principales :

- **GetSwitchLinks** : Détecte les liaisons inter-switchs,
- **GetNoManageableDevices** : Détecte les switchs non gérables,
- **ComparaisonSwitchsAvecListeSwitchs** : Compare une ancienne liste de switch avec une nouvelle et retourne les éléments de cette dernière qui ne font pas doublons,
- **ComparaisonRouteursAvecListeSwitchs** : Même fonction que la précédente sauf qu'il s'agit cette fois ci des routeurs (le critère de comparaison n'est pas le même),
- **FusionCorrespondancesVlans** : Fusionne les correspondances *MAC/Ports* de chaque VLAN d'un même switch en une seule correspondance,
- **CreationTableauLiaisons** : Crée le tableau final matérialisant les liaisons entre équipements,
- **CreationTableauEquipements** : Crée le tableau final détaillant chaque équipement,
- **RechercheDNS** : Recherche le nom DNS correspondant à une adresse IP,
- **DecouverteReseau** : moteur de découverte de la topologie du réseau proprement dit. Il fait appel directement ou indirectement à toutes les autres fonctions détaillées jusqu'ici,
- **ExportationVersMysql** : Exporte les tableaux des liaisons et des équipements vers une base de données MySQL,
- **ModificationEtatReseau** : Enregistre dans deux tables spécifiques d'une base de données les modifications apportées au réseau depuis une première découverte de sa topologie.

### a) Fonction GetSwitchLinks

Découvrir la topologie d'un réseau revient notamment à découvrir les liaisons entre les différents nœuds. Remarquons tout d'abord que nous ne le ferons que pour les switchs administrables, c'est-à-dire pour ceux dont nous disposons des correspondances *MAC/Ports* et dans la mesure où nous connaissons les adresses MAC de tous les ports des switchs (gérables). Ces deux tableaux (correspondances et adresses MAC des interfaces) sont les arguments de cette fonction. Un dernier argument permet ou non de vérifier la réciprocité des liaisons (cf. Détection des liaisons entre switchs page 17). Nous allons alors comparer ces tableaux suivant l'algorithme suivant :

Arguments de la fonction : \$ListeSwitchs : Liste des switchs \$AdrMacPort : Adresses MAC des ports de chaque switchs \$CrspdcesMacPort : Correspondances <i>MAC/Ports</i> de chaque switch \$test_reciprocite = true/false ...
--

```

...
Pour chaque switch $i de $ListeSwitchs {
    Pour chaque port $p de $CrspdcesMacPort[$i] {
        Si ($test_reciprocite=true) Alors
            $Boucle = « chaque switch $j suivant de $ListeSwitchs »
        Sinon
            $Boucle = « chaque switch $j de $ListeSwitchs »
        Fin si

        Pour $Boucle "autre que le switch courant" {
            $Intersection = Intersection($CrspdcesMacPort[$i][$p], $AdrMacPort[$j])
            Si ($Intersection non null) {
                $reciprocite_confirmee = false
                Pour chaque port $k de ($CrspdcesMacPort[$j] {
                    $Intersection = Intersection($CrspdcesMacPort[$j][$k],
$AdrMacPort[$i])

                    Si ($Intersection non null) {
                        $reciprocite_confirmee = true
                        Sortir de la boucle $k
                    }
                }

                Si (($test_reciprocite = true ET $reciprocite_confirmee = true)
OU ($test_reciprocite = false)) {
                    Liaison trouvée : mise à jour du tableau résultat
                    Suppression de $CrspdcesMacPort[$i][$p]
                    Eventuellement suppression de $ CrspdcesMacPort[$j][$k]
                }
            }
        }
    }
}

```

Comme vous pouvez le voir sur cet algorithme, nous commençons par boucler sur les ports de chaque switch (2 premières boucles), puis sur tous les autres switchs. Cette troisième boucle est différente suivant que la vérification de la réciprocité est faite ou non. En effet, si elle est faite, il est inutile de revérifier avec un switch sur lequel on serait déjà passé (si le switch 1 voit le switch 3, alors lorsque nous en serons au switch 3, la vérification 3-1 aura déjà été faite puisque la réciprocité de la liaison vérifiée !).

Ceci peut se traduire ainsi (les boucles portent sur le même tableau !) :

```

For ($i = 0, $i < X, $i++) {
    For ($j = $i+1, $j < X, $j++) {
        ...
    }
}

```

Par contre, si le test de réciprocité n'est pas effectué, il se peut que le switch 3 voit le switch 1 mais non l'inverse, et lorsque nous en serons au switch 3, il faudra donc prendre en compte le 1, d'où les boucles suivantes :

```

For ($i = 0, $i < X, $i++) {
    For ($j = 0, $j < X, $j++) {
        ...
    }
}

```

Les correspondances des liaisons trouvées sont supprimées afin d'améliorer le temps de recherche des prochaines boucles (inutile de perdre du temps sur un élément déjà utilisé et sur lequel nous n'avons plus à repasser).

Finalement, nous constatons que cette fonction comporte beaucoup de boucles imbriquées, mais il serait difficile de faire sans puisque de toute façon il faut comparer deux tableaux. Enfin, nous pouvons percevoir ici l'intérêt d'utiliser le langage PHP puisque nous nous servons de la fonction Intersection(Tableau 1,Tableau 2) qui renvoie un tableau constitué des éléments du tableau 1 présent dans le tableau 2. Il aurait été par exemple nécessaire de recoder cette fonction sous Java ! Utiliser une fonction fournie étant normalement gage d'efficacité et rapidité, ce choix est judicieux.

### **b) Fonction GetNoManageableDevices**

Cette fonction utilise simplement l'algorithme évoqué au paragraphe *Détection des switchs non gérables* en page 18. Comme la fonction précédente, le tableau `$CrspdcMacPort` ainsi que la liste des switchs sont mis à jours afin de tenir compte des nouveaux switchs et liaisons détectées.

### **c) Fonction ComparaisonSwitchsAvecListeSwitchs**

Cette fonction permet de supprimer les doublons entre deux listes de switchs. Elle prend comme arguments deux listes de switchs, une nouvelle et une ancienne. Tous les équipements de la nouvelle liste qui ne se trouvent pas dans l'ancienne seront renvoyés dans le tableau résultat de cette fonction. Les autres ne seront pas présentes dans ce résultat.

Afin de comparer les listes, nous utilisons les adresses MAC des switchs, adresses censées être uniques. Ainsi, tout équipement présent dans la nouvelle liste qui n'aurait pas d'adresse MAC sera d'emblé éliminé !

### **d) Fonction ComparaisonRouteursAvecListeSwitchs**

Cette fonction est la même que la précédente, excepté le fait que la comparaison ne se fait plus sur leurs adresses MAC des équipements mais sur leurs adresses IP. Par ailleurs, cette fonction renvoie deux tableaux : le premier est la liste des nouveaux routeurs détectés afin de les rajouter aux autres, et le deuxième est la liste des doublons afin de ré agencer la liste des équipements comme exposé au paragraphe *Détection des routeurs* (page 19).

### **e) Fonction FusionCorrespondancesVlans**

Cette fonction a pour but de constituer un tableau unique de correspondances *MAC/Ports* par switch et non un par VLAN. Chaque port de la correspondance finale sera associé à un ou plusieurs VLAN (s'ils sont connus !). En effet, il peut arriver que les liaisons entre switchs soient « en mode *trunk* » et permettent le passage des paquets appartenant à plusieurs VLAN sur le même câble. Dans ce cas de figure, l'ensemble des noms des VLAN sont signalés sur ce port, et le numéro de VLAN est mis à une valeur arbitraire normalement impossible, dans notre cas « -1 ».

### **f) Fonction CreationTableauLiaisons**

Cette fonction utilise trois tableaux comme arguments : les correspondances *MAC/Ports*, le tableau des liaisons entre switchs et enfin le tableau donnant les vitesses des ports des switchs. Le résultat sera quant à lui un seul et unique tableau, compilation des deux précédents. En effet, les liaisons entre ordinateurs et switchs se déduisent immédiatement des correspondances *MAC/Ports* sur les mêmes switchs. Cette fonction renvoie par ailleurs un tableau des nouveaux équipements (ordinateurs, etc.) issu de ces correspondances.

### **g) Fonction CreationTableauEquipements**

Comme pour la fonction précédente, le résultat sera ici un tableau, combinaison du tableau des nouveaux équipements couplés avec la liste des switches.

### **h) Fonction RechercheDNS**

Cette fonction traduit une adresse IP en DNS (Domain Name Server). Pour cela, nous utilisons la méthode *gethostbyaddr*(Adresse IP) de PHP. Si le nom DNS ne peut être trouvé, nous renvoyons la valeur *null*.

### **i) Fonction ExportationVersMysql**

Après avoir obtenu les deux tableaux PHP représentant les équipements et leurs liaisons (résultat de la Fonction DecouverteReseau), il faut les enregistrer dans la base de données. C'est le rôle de cette fonction. Elle prend donc comme argument les deux tableaux précités, ainsi que l'identifiant du réseau correspondant. En effet, plusieurs réseaux peuvent être surveillés en même temps et chaque topologie est stocké dans un jeu de tables préfixées par cet identifiant. Pour connaître le format des tables utilisées, se reporter au paragraphe Stockage des topologies page 25.

### **j) Fonction ModificationEtatReseau**

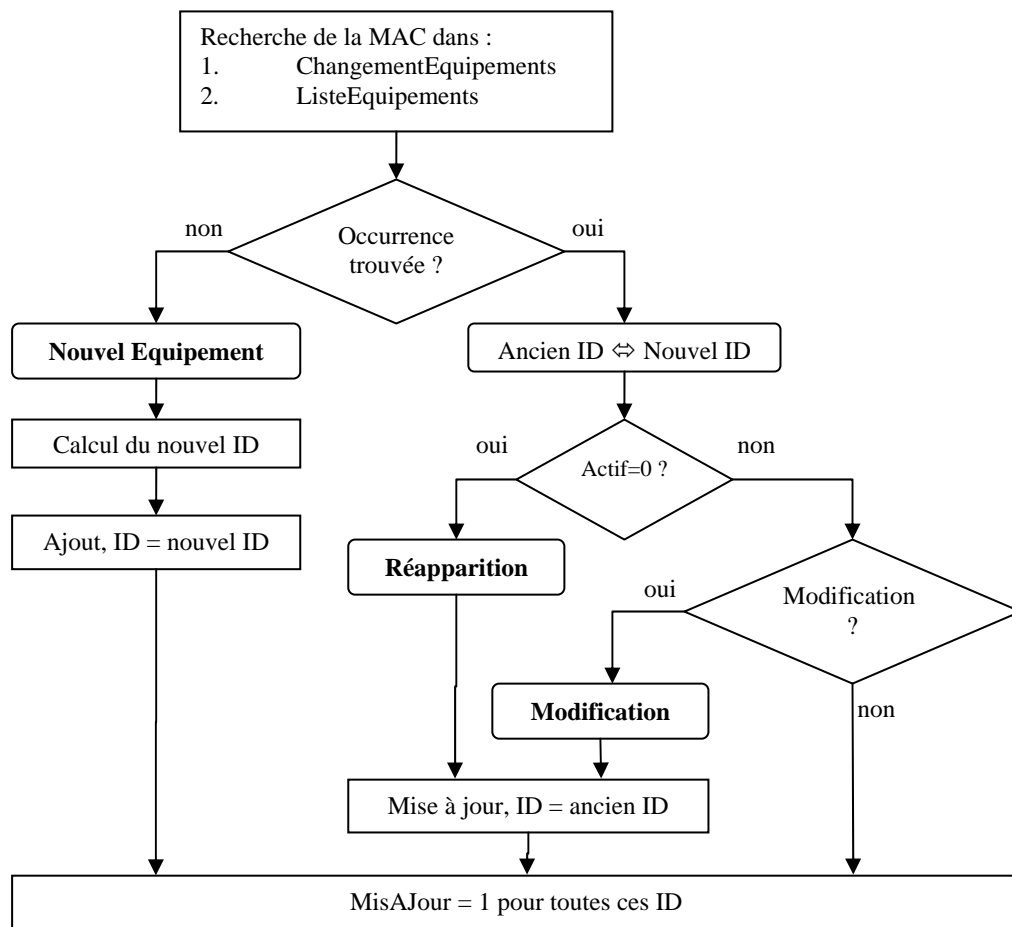
Cette fonction permet de comparer l'état précédent d'un réseau avec l'état actuel de celui-ci. Pour commencer, l'état précédent est enregistré dans les tables MySQL, et s'obtient à partir des tables 'ListeX' et 'ChangementX', où X est 'Equipements' ou 'Liens'. L'état actuel est quant à lui représenté par les deux tableaux PHP résultat de la Fonction DecouverteReseau. Nous partons chaque fois de l'état actuel, et nous cherchons à le corréler avec l'état précédent. Ainsi la fonction peut se décomposer en trois processus :

- Traitement des équipements possédant une adresse MAC,
- Traitement des équipements ne possédant pas d'adresse MAC (concerne en fait uniquement les switches/hubs non administrables détectés grâce à la Fonction GetNoManageableDevices.
- Traitement des liaisons.

Nous commençons par les équipements. Nous regardons pour chacun de ceux qui sont actuellement présent sur le réseau, s'il a déjà été détecté auparavant. Chaque matériel est identifié de manière unique par un numéro 'Id'. Celui-ci est malheureusement arbitraire, est dépend de la découverte effectuée. Nous corrérons alors dans un premier temps les Id des équipements actuels avec ceux que nous avons déjà détecté (ceux qui se trouvent dans les tables MySQL). Comme un équipement est censé normalement posséder une adresse MAC unique et invariable sur le réseau, nous les utilisons afin de faire cette identification. Cependant, les switches non gérable n'en possèdent pas. Le seul moyen de les identifier est d'utiliser leurs liaisons, c'est pour cela que nous les traitons dans un deuxième temps. Enfin, dès que tous les équipements sont corrélés avec ceux que nous avons déjà croisés, et que nous avons éventuellement ajouté les nouveaux, nous pouvons nous pencher sur les liaisons puisque celles-ci se font entre ces mêmes équipements et que nous avons maintenant leurs Id respectifs.

### 1. Traitement des équipements possédant une adresse MAC :

Pour chaque équipement actuellement détecté et ayant une adresse MAC, nous exécutons l'algorithme suivant :



**Figure 20 : Algorithme de traitement des équipements avec MAC**

La recherche initiale se fait d'abord dans la table 'ChangementEquipements' puis si nous n'y trouvons pas la MAC, dans 'ListeEquipements'. Cet ordre nous permet d'obtenir, si l'équipement existait déjà auparavant, son dernier état.

L'attribut MisAJour est donc mis à « 1 » dès qu'un ancien équipement a été traité. Ainsi à la fin, ceux qui auront cet attribut à « 0 » pourront être « supprimés » car cela signifie qu'il n'existe plus actuellement. A noter que toute modification se traduit par l'ajout d'une ligne dans la table 'ChangementX' et non pas par l'édition voire la suppression concrète de l'enregistrement incriminé. Ainsi il sera possible à tout moment de reconstituer la topologie du réseau à un instant déterminé, et non de connaître uniquement le dernier état du réseau.

### 2. Traitement des équipements ne possédant pas d'adresse MAC :

Nous avons vu que le traitement précédent utilisé les adresses MAC. Ceci est malheureusement impossible pour les nœuds réseaux non administrables : nous ne connaissons alors pas leurs adresses MAC ! Nous allons essayer d'identifier ces nœuds actuels avec les anciens grâce à leurs liaisons. Pour cela, nous commençons donc par constituer les deux tableaux suivant :

- AnciensEquipementsNonGerables[ID ancien équipement] = liste des équipements auquel il est connecté,
- NouveauxEquipementsNonGerables[ID nouveau équipement] = liste des équipements auquel il est connecté.

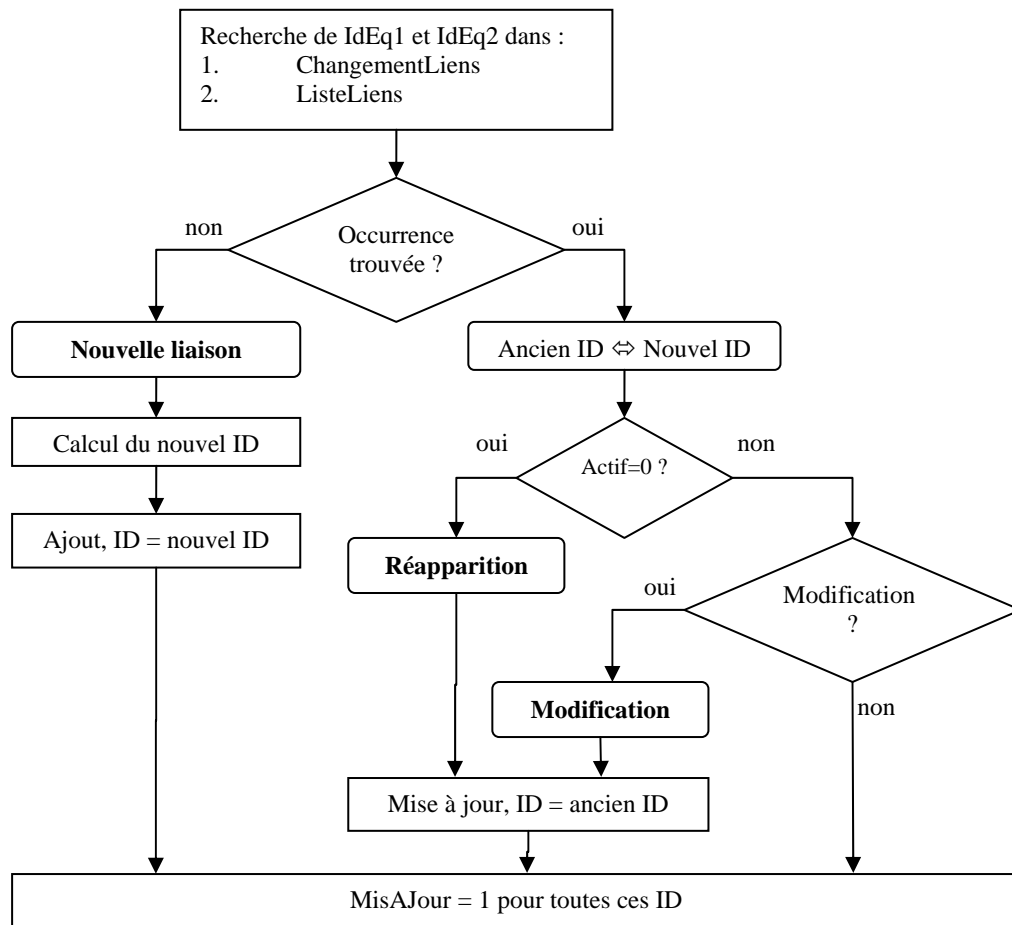
Enfin nous allons comparer, pour chaque entrée du tableau NouveauxEquipementsNonGerables, la liste associée à celle correspondante aux anciens équipements, et ce pour tous les anciens équipements non gérable : nous faisons l'intersection de ces deux tableaux. Au final, nous retenons l'ID de l'ancien équipement où l'intersection a donnée le plus de résultat : nous choisissons ainsi l'ancien équipement qui correspond le plus vraisemblablement à l'équipement que nous avons actuellement. Si aucun résultat n'est trouvé, alors l'équipement actuel est nouveau. Nous constituons ainsi un tableau associant équipements actuels et anciens équipements. Une fois cette corrélation effectués entre les ID des nœuds non administrables, nous utilisons le même algorithme que précédemment, sauf que nous n'avons pas à rechercher les adresses MAC : nous connaissons déjà les ID des équipements. Remarquons que nous ne sommes pas assurés de retrouver un ancien switch correspondant à celui que nous sommes entrain de tester, même s'il existe réellement. Par contre, le résultat final sera quand même correcte puisque dans le pire des cas, le switch actuel sera déclaré comme étant nouveau et l'ancien supprimé. Le résultat est donc toujours juste, mais nous gaspillons de la place mémoire pour stocker ces informations puisque nous rajoutons alors deux lignes là où nous aurions pu n'en mettre qu'une, voire aucune s'il n'y a pas eu de modifications.

Une fois que tous les équipements actuels (avec ou sans MAC) ont été traités, tous les anciens équipements qui ont pour attribut « MisAJour = 0 » n'existent plus est sont donc supprimés (ajout d'une ligne par équipement dans la table 'ChangementEquipements').

### 3. Traitement des liaisons :

Le traitement des liaisons est calqué sur celui des équipements.

Pour chaque liaison actuelle, l'algorithme suivant est exécuté :



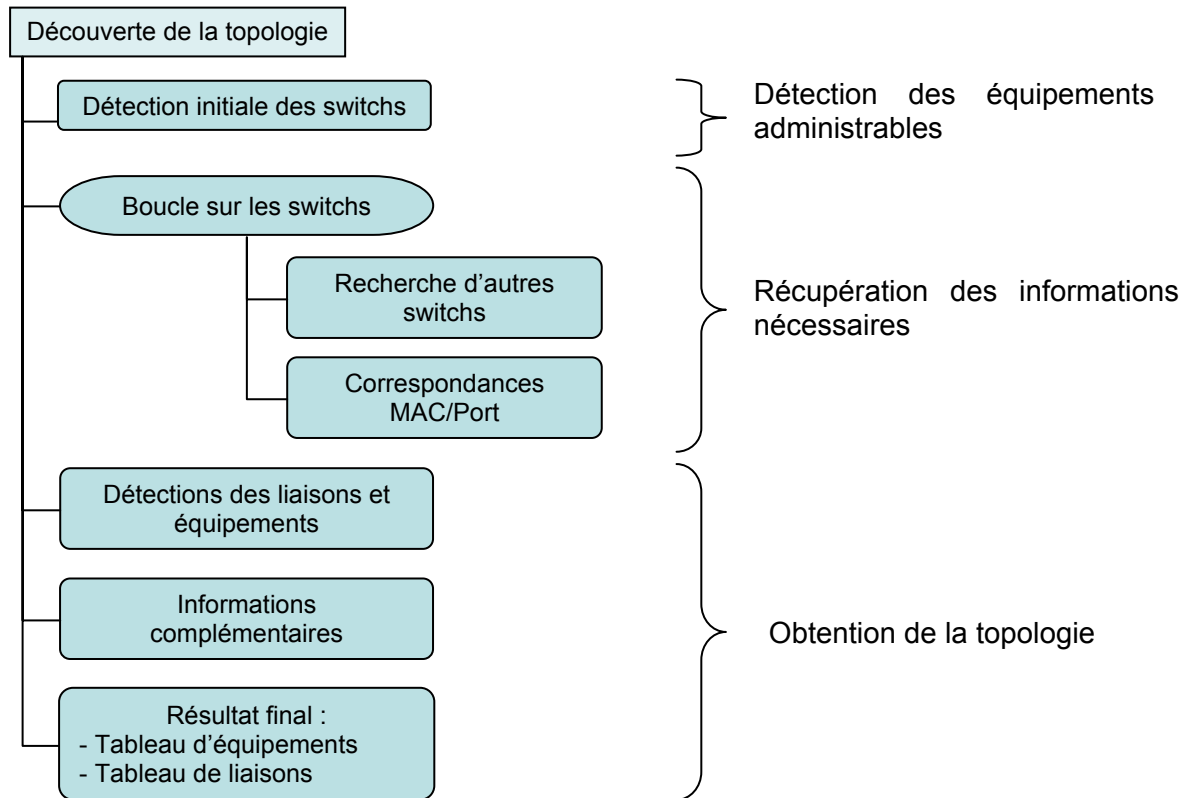
**Figure 21 : Algorithme de traitement des liaisons**

Là encore, toutes les anciennes liaisons qui au final ont l'attribut « MisAJour = 0 » sont supprimées.

### k) Fonction DecouverteReseau

Il s'agit de la fonction principale de ce programme puisque c'est elle qui doit être appelée pour découvrir automatiquement la topologie du réseau. Cette fonction ne prend comme argument une série de tableaux : options, liste initiale des switches, noms de communautés à tester, plages d'adresses IP à scanner et celles autorisées, etc. Elle suit l'agencement décrit au paragraphe

*Processus de découverte* (page 24). La Figure 22 donne une vue d'ensemble de cette fonction tandis que la Figure 23 représente son déroulement complet.



**Figure 22 : Vue simplifiée de la Fonction DecouverteReseau**



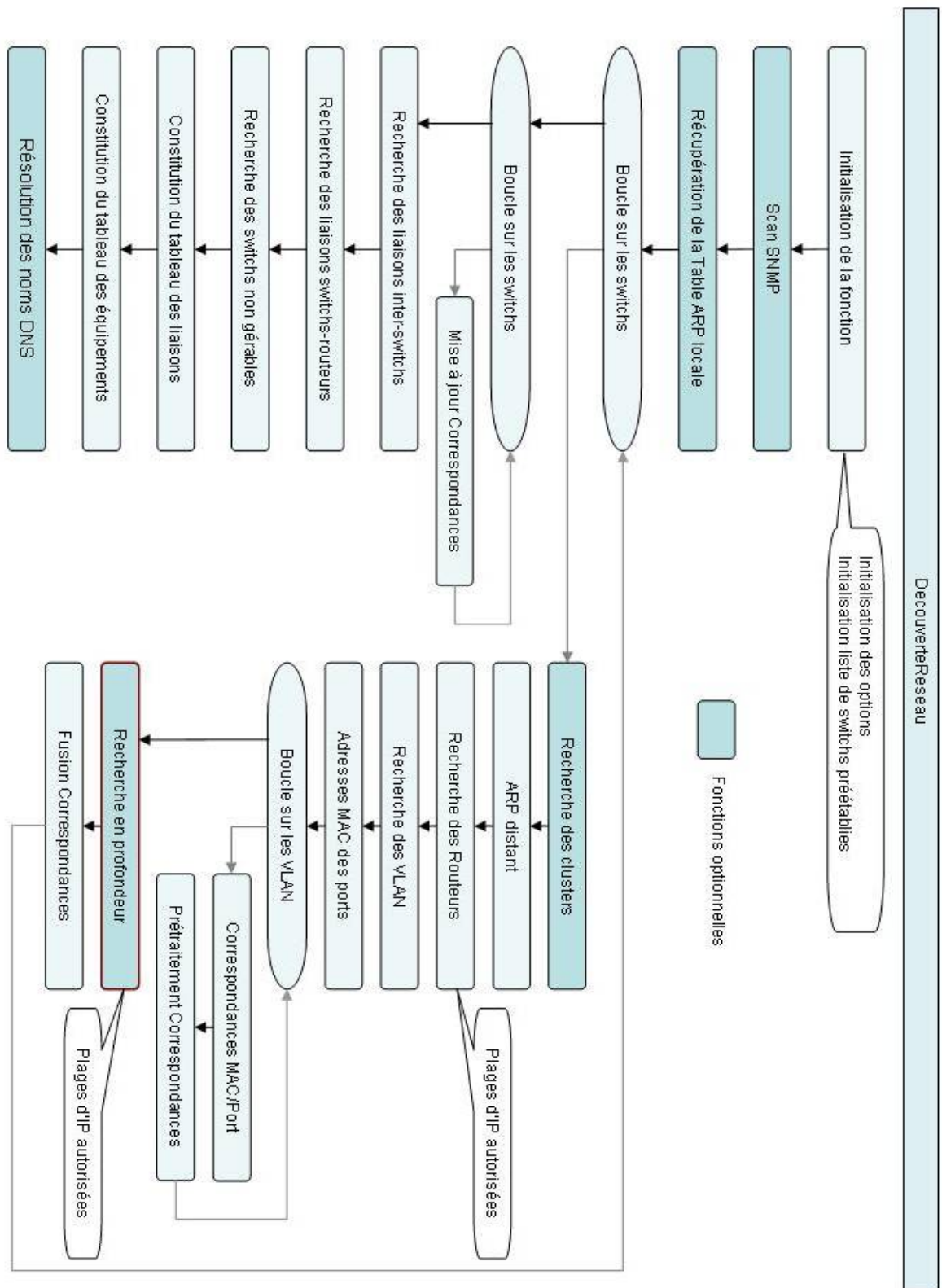


Figure 23 : Algorithme de la Fonction DecouverteReseau

Enfin le pseudo-code ci-dessous détaille cette fonction :

```

//Initialisation des différents tableaux utilisés :
Initialisation des Options
$ListeSwitchs = Initialisation de la liste des switchs préétablie
$ListeCommunities = Initialisation de la liste des noms de communauté à tester
$PlageIPs = Initialisation de la liste des plages d'IP à scanner
$PlageIPsAllowed = Initialisation de la liste plages d'IP autorisées
  
```

Récupération des adresses MAC et des ObjectID (OID du constructeur) des switches déjà connus

```
// Scanne des IP :
Si (autorisation_scan) {
    Pour chaque plage d'IP a scanner {
        Génération de la liste d'IP à scanner à partir de la plage définie

        IPScanBySNMP(liste d'IP a scanner, $ListeCommunities)

        Mise en cache des informations SNMP pour les IP qui le supporte dans $CacheInfoSNMP

        Si (autorisation_arp) $ArpTable = GetLocalArpTable()
    }

    Ajoute à $ListeSwitchs le résultat de la fonction
ComparaisonSwitchsAvecListeSwitchs($NouveauxSwitchs, $ListeSwitchs)
} Sinon si (autorisation_arp) {
    $ArpTable = GetLocalArpTable()
}

// Boucle générale sur chaque switch :
Pour chaque switch $i de $ListeSwitchs {
    // Recherche des clusters Cisco :
    Si (autorisation_recherche_cluster ET $i non encore vérifié) {
        GetMembersOfClusterBySNMP($i["IP"], $i["Community"])
        Ajoute à $ListeSwitchs le résultat de la fonction
ComparaisonSwitchsAvecListeSwitchs(nouveaux switchs, $ListeSwitchs)
    }

    // Récupération de la table ARP :
    Ajoute à $ArpTable le résultat de GetArpTableBySNMP($i["IP"], $i["Community"])

    // Recherche des routeurs :
    GetNextRouterBySNMP($i["IP"], $i["Community"])
    ComparaisonRouteursAvecListeSwitchs(routeurs, $ListeSwitchs, $PlageIPsAllowed)
    Récupère les informations sur les routeurs et élimine les doublons
    Si (routeur non encore traité par cette boucle et n'est pas un doublon) {
        Le switch $i+1 passe en fin de liste
        Le nouveau routeur est inséré à la place du switch $i+1
    }
    Si le routeur $j est un doublon : permutation du switch $i+1 avec ce routeur $j

    Récupération des vitesses des ports du switch $i

    // Détection des VLAN :
    GetVlanTableBySNMP($i['IP'], $i['Community'])

    // Récupération des correspondances MAC/Ports
    GetInterfaceMacAdrBySNMP($i["IP"], $i["Community"])
    ...
    ...
    Pour chaque VLAN $j du switch $i {
        GetCorrespondancePortMacBySNMP($i["IP"], nom de communauté du VLAN $j)
        Pour chaque Port des correspondances {
            Suppression des adresses MAC des ports du switch sur ses correspondances.
            Suppression des ports où il n'y a aucune adresse MAC
        }
        FusionCorrespondancesVlans(correspondances des VLAN du switch $i)
    }
}

Suppression des adresses MAC des IP des switchs sur chaque correspondance
...
```

```
...
// Recherche des liaisons inter-switchs :
Suppression des adresses MAC des switchs sur les MAC des ports des switchs ($AdrMacPort)
GetSwitchLinks($ListeSwitchs,$AdrMacPort,$CrspdceMacPort,avec test réciprocity)

// Recherche des liaisons switchs-routeurs :
GetSwitchLinks($ListeSwitchs,$AdrMacPort,sans test réciprocity)

// Recherche des équipements non gérable :
GetNoManageableDevices($ListeSwitchs,$CrspdceMacPort)

// Constitution des tableaux Liaisons et Equipements :
CreationTableauLiaisons($LiaisonSwitchs,$CrspdceMacPort,$VitessePort)
CreationTableauEquipements($ListeSwitchs,autres nœuds,$RarpTable,$CacheInfoSNMP)

Si (autorisation_recherche_dns) RechercheDNS(tableau Equipements)

// Résultats de la fonction :
- Tableau Liaisons
- Tableau Equipements
```

### E. Limitations

Plusieurs limitations vis-à-vis du moteur de découverte existent. Certaines sont inhérentes aux technologies employées, d'autres découlent d'un manque de temps afin de développer des solutions alternatives, et enfin certaines fonctionnalités peuvent être facilement ajoutées dans le futur.

#### 1. Détection des routeurs

Les routeurs sont actuellement détectés, mais comme nous n'obtenons pas généralement leurs correspondances MAC/Ports, ils apparaissent parfois en multiples exemplaires (un par subnet). Ainsi un routeur possédant trois pattes dans des sous réseaux de subnet différent risque au final d'apparaître trois fois dans nos données. Le seul moyen d'éviter ces doublons et d'obtenir les adresses MAC des interfaces du routeur, ce qui n'est pas toujours possible. Ensuite, il s'agit au moment de vérifier les doublons entre équipements, après leurs détections (scan d'une plage d'adresse IP, mode poursuite, etc.), de prendre en compte les adresses MAC des interfaces du matériel. Actuellement, seules les adresses dites d'administrations (celle associé à l'IP permettant de gérer le routeur) sont utilisées. De plus, pour les routeurs nous utilisons leurs IP et non leurs MAC. Il faudrait donc reprendre ces deux fonctions (ComparaisonSwitchsAvecListeSwitchs et ComparaisonRouteursAvecListeSwitchs).

#### 2. Détection des liaisons inter-switchs

Actuellement nous testons la réciprocity des liaisons afin de s'assurer qu'elles existent. Nous pourrions tenter de nous en affranchir. Il faudrait cependant cerner le problème qui nous a obligé à faire une telle vérification (cf. Fonction GetSwitchLinks page 41), puis le résoudre.

#### 3. Détection des nœuds non gérables

Les limitations inhérentes à l'utilisation du protocole SNMP sont exposées au paragraphe de la fonction correspondante (page 43). Il pourrait être intéressant de se renseigner sur d'éventuelles méthodes alternatives (ou complémentaires !). Néanmoins, ces méthodes seront certainement lourdes à mettre en place, si tant est qu'il en existe une efficace ! Il faudrait alors peser le pour et le contre pour savoir si se donner autant de peine est nécessaire.

## 4. Utilisation de MIB propriétaires

La Fonction GetVlanTableBySNMP utilise une MIB propriétaire Cisco. Bien que ces MIB soient généralement repris par les autres constructeurs, il serait intéressant de pouvoir s'en affranchir. Dans le même esprit, certaines fonctions ne sont opérationnels qu'avec des équipements de marque Cisco, même si elles utilisent des MIB standards. Concrètement, il faudrait disposer d'équipements d'autre marque afin de tester et recoder ces fonctions pour qu'elles puissent tourner avec ces nouveaux matériels.

## 5. Surveillance de l'état du réseau

La fonction permettant de trouver les modifications intervenues sur le réseau peut dans un premier temps être optimisée. Par ailleurs, il serait préférable de repenser le système de suivie des modifications. Pour le moment, le moteur est exécuté régulièrement avec des options variables suivant qu'il s'agisse d'une découverte initiale ou régulière. Le problème résulte du protocole SNMP. Très pratique pour une utilisation ponctuelle ou pour administrer un réseau, il n'est pas adapté à une surveillance « temps réel ». En effet, il se peut qu'un nœud ne réponde pas. En l'état actuel du moteur, cet équipement sera considéré comme ayant disparu du réseau et sera « supprimé ». Il faudrait donc mettre en place un système de réessaies successifs (ce qui veut aussi dire détecter lorsqu'il y a erreur) pour que l'équipement soit déclaré inactif (ou plus exactement « presque inactif ») après un certain nombre d'essais infructueux.

En ce qui concerne les terminaux, il serait préférable pour une surveillance très régulière (fréquence de l'ordre de la dizaine de minute par exemple) d'utiliser un simple ping. En effet, nous surchargerions alors moins le réseau. L'inconvénient de cette méthode est qu'elle nécessite de connaître potentiellement tous les terminaux possibles puisque seuls seraient pingés ceux qui auront été détectés par le moteur de découverte, donc indirectement par SNMP, une première fois. Nous ne pouvons effectivement surveiller ceux que nous ne connaissons pas encore (seul le moteur de découverte le permet) !

D'autre part, plutôt que d'exécuter le moteur pour l'ensemble des nœuds, nous pourrions le faire nœuds après nœuds dans un ordre défini. Le moteur le permet puisqu'il suffit de lui préciser de travailler uniquement sur une liste de switch que nous lui donnerions, et de constituer ainsi un certain nombre de listes de un ou deux nœuds chacune.

Enfin il faudrait se pencher sur la structure de la base de données pour savoir s'il ne serait pas possible de l'optimiser et ainsi rendre l'algorithme de la Fonction ModificationEtatReseau plus simple et optimal.

## 6. Bornes Wifi

Nous avons constaté que les bornes Wifi de l'école sont détectées comme étant des nœuds réseaux (ce qui est le cas) et alors interrogées. Malheureusement, les résultats semblent parfois aberrant, notamment lorsque les bornes sont connectées en Wifi avec les routeurs, ce qui n'est pas physiquement le cas ! Nous n'avons pas pu cerner exactement le problème faute d'affichage de la topologie à ce moment là. Néanmoins, ce problème vient peut-être du fait que les bornes utilisent rarement les MIB standards. Dans ce cas, nous avons deux possibilités : rajouter des fonctions permettant d'interroger ces équipements (donc grosso-modo une fonction par marque), ou alors les ignorer et ne pas les interroger. Dans tous les cas, ces méthodes nécessitent de connaître au moment où le moteur de recherche est exécuté le type de matériel rencontré et d'adapter ses requêtes en fonction.

## **7. Fonctionnalités supplémentaires**

Le moteur de découverte étant maintenant codé, il est facile de récupérer des informations supplémentaires sur les équipements, et obtenir pour ne citer qu'un exemple le type de liaison entre deux équipements (fibre optique, cuivre, etc.).

## IV. Interface utilisateur et représentation graphique

L'interface utilisateur sera la « partie émergée de l'iceberg » que constitue notre application. Elle doit permettre de gérer un maximum de paramètres de manière simple et conviviale et garantir un affichage de qualité sous de nombreuses plates-formes.

### A. Présentation de l'interface utilisateur

Dans cette partie, nous allons présenter de manière générale l'interface utilisateur ainsi que les technologies qui ont permis de la réaliser ainsi que son implémentation, afin que toute personne suivant la reprendre, la modifier, l'améliorer (ce qui est grandement possible...) puisse le faire aisément.

#### 1. Description de l'interface

L'interface de l'utilisateur se décompose en 5 parties distinctes. Comme dans ce domaine, une bonne illustration vaut mieux que de longs discours, un aperçu de cette interface est fourni en Figure 24.

The screenshot displays the snmpWatch web application interface. At the top is a blue header bar with the logo 'snmpWatch' and the title 'Découverte de topologie et monitoring de réseau par SNMP'. Below the header is a navigation menu with four tabs: 'Utilisateurs', 'Réseaux', 'Equipement', and 'Monitoring'. The 'Utilisateurs' tab is active, showing a sidebar with 'Utilisateur identifié' (Bonjour guiguix, Se déconnecter) and a main content area titled 'Paramètres des types de matériel'. This area contains a form for 'Modification des paramètres d'un type de matériel'. The form has a text input for 'Nom du type' (set to 'Routeur'), a radio button selection for 'Icône pour le matériel' (with 'routeur.png' selected), and a text area for 'Description/Détails' (containing 'Routeur générique'). At the bottom of the form are 'Valider' and 'Effacer' buttons. A footer bar at the bottom shows technical details: 'W3C XHTML 1.0', 'W3C CSS', 'PHP', and 'MYSQL'.

Figure 24 : Aperçu de l'interface graphique

Elle permet de faire apparaître de haut en bas et de gauche à droite :

- *un bandeau de titre* (en bleu) qui porte le nom que nous avons retenu pour notre application (à savoir « snmpWatch » : pourquoi ? pourquoi pas !)
- *un menu* (barre grise juste en-dessous) qui permet d'accéder à tous les éléments de l'interface ;
- *une barre latérale* qui permet de gérer l'authentification des utilisateurs et qui fournit également des informations et des liens utiles ;

- la zone de contenu où sont affichées les informations essentielles de la page
- un pied de page, qui renvoie vers des validateurs de code du W3C et vers les sites officiels de PHP et MySQL

## 2. Structure des menus

Voici résumé succinctement les différentes fonctionnalités que l'on retrouve au sein de chaque menu, elles représentent chacune des fonctions différentes de l'interface utilisateur :

### a) Gestion des utilisateurs

Elle repose sur l'utilisation de sessions de PHP, couplés à la table *User* qui stocke les logins et mots de passes des utilisateurs, ainsi que diverses informations les concernant.

Le principe en est le suivant : l'autorisation d'afficher le contenu des pages n'est donnée que si :

- Une session identifiant l'utilisateur existe
- Le bon couple login/mot de passe est fourni via un formulaire (on démarre alors une session identifiant l'utilisateur)

### b) Gestion des réseaux

L'application permet de surveiller différents réseaux. L'ajout/suppression des réseaux se fait via un formulaire et l'on utilise des tables préfixées par leur *Id* (identifiant numérique unique) pour distinguer les tables.

Grâce aux formulaires de cette partie, il est possible :

- On peut spécifier les paramètres par défaut de l'observation
- On peut uploader une image qui servira de fond pour la représentation de la topologie (typiquement un plan du lieu sur lequel est implanté le réseau)

Modification des paramètres d'un réseau

Nom du réseau :

Autoriser le scan des plages d'IP : ☒ Oui ☐ Non

Liste des plages d'IP à scanner : ☒ 10.0.0.1 - 10.0.0.254  
☐ 127.0.0.1 - 127.0.0.254  
☒ 192.168.0.1 - 192.168.0.254  
☐ 193.48.225.1 - 193.48.225.254

Noms des communautés à Scanner (séparés par ",") :

Autoriser le mode poursuite : ☒ Oui ☐ Non

Profondeur de la poursuite :

Utiliser les plages d'IP autorisées : ☒ Oui ☐ Non

Liste des plages d'IP autorisées : ☐ 10.0.0.1 - 10.0.0.254  
☐ 127.0.0.1 - 127.0.0.254  
☒ 192.168.0.1 - 192.168.0.254  
☒ 193.48.225.1 - 193.48.225.254

Utiliser l'ARP locale : ☒ Oui ☐ Non

Rechercher les membres des clusters : ☒ Oui ☐ Non

Rechercher les DNS : ☒ Oui ☐ Non

Utiliser une liste de switch prédéterminés : ☒ Oui ☐ Non

Délai de recherche automatique (min) :

Image de fond du réseau :

Valider

Figure 25 : Formulaire de création de réseau

### c) Gestion des équipements

Via l'interface utilisateur, il est possible de gérer le concept de « types d'équipements ». En effet, chaque élément du réseau est identifié par un type.

Il existe trois types par défaut : Routeur, Switch, Switch/Layer 3, Switch/Hub (qui représentent les types essentiels pour les nœuds du réseau : c'est-à-dire ceux qui sont reconnus automatiquement par SNMP).

Mais il est également possible d'ajouter d'autres types (pour identifier des équipements particuliers comme des serveurs, ...) Chaque type est représenté par une icône dans la représentation de la topologie du réseau (les icônes sont modifiables via un formulaire et de nouvelles icônes peuvent être uploadées).

Pour les équipements qui possèdent une OID, il est possible de réaliser un apprentissage de la nature des équipements. Pour ce faire, on leur assigne un type (estimé via le *SysServices* fourni par SNMP mais il peut être erroné) qui peut être modifié par la suite via un formulaire. Cette modification permet par la suite de les représenter correctement dans la topologie du réseau.

Figure 26 : Apprentissage du type d'équipement

### d) Affichage de la topologie et monitoring réseau

C'est le résultat à la fois du travail du moteur de recherche, mais aussi de tout le paramétrage réussi par l'utilisateur dans les diverses autres parties du site.

Un formulaire semblable à celui de la Figure 25 permet de personnaliser l'affichage de la topologie réseau, mais nous reviendrons sur cette fonction particulière dans une prochaine partie.

Il est également possible d'afficher les derniers événements réseaux (c'est ce que l'on nomme « monitoring ») sous forme tabulaire ou des codes couleurs (précisés dans la partie latérale du site) permettant d'identifier le type d'événement survenu.

Pour les équipements :

Date	Nom SNMP	IP	MAC	Nom Communauté	OID Produit	DNS
16/06/2005 à 10:31:01		192.168.4.1	00:03:47:B1:22:18			192.168.4.1
16/06/2005 à 10:31:01			00:02:2D:05:88:33			
16/06/2005 à 10:31:01			00:30:48:24:39:4B			
16/06/2005 à 10:31:01		193.48.224.246	00:20:DA:9B:41:E0			xylan2.metz.supelec.fr
16/06/2005 à 10:31:01			00:60:94:1A:E1:99			
16/06/2005 à 10:31:01			00:40:8C:69:1C:0E			
16/06/2005 à 10:31:01			00:0A:B7:29:1A:6D			
16/06/2005 à 10:31:01			00:0A:E6:06:03:68			
16/06/2005 à 10:31:01			00:30:48:23:FC:F9			
16/06/2005 à 10:31:01			00:40:8C:69:1C:0A			
16/06/2005 à 10:23:22			00:0A:E6:06:03:68			

Figure 27 : Monitoring des événements



Cette partie du site permet également de choisir la plage temporelle sur laquelle afficher les évènements du réseau en cours de surveillance.

### 3. Structures des fichiers

Les pages de l'interface utilisateurs sont des fichiers PHP4 (extension .php) et sont construites de la manière décrite dans les lignes suivantes, qui permet une grande flexibilité et une modification rapide du contenu des pages : chaque page qui s'affiche est constitué en fait de deux fichiers, un se situant à la racine du site, l'autre étant placé dans le sous répertoire *pages*, les deux fichiers portant exactement le même nom.

#### a) Description du fichier racine

Le fichier racine contient relativement peu de code mais remplit des fonctions essentielles, voici la manière dont il doit être renseigné :

```
<?php
// Inclusion automatique de toutes les librairies de fonctions
$RepertoireFonctions = dir("fonctions")
or die("<b>Erreur : </b> Merci de vérifier l'intégrité du script.");
while($item = $RepertoireFonctions->read()) {
    if (ereg("\.php$", $item)) include "fonctions/".$item;
}

// Début de la procédure d'utilisation
// Si le formulaire a été validé, on lance la procédure d'utilisation
if (isset($_POST['submit_identification'])) {
    $Autorisation = IdentifierUtilisateur ($_POST['login'], $_POST['mot_de_passe'], "User",
    "Id", "Login", "MotDePasse");
}
// Sinon, on vérifie si l'utilisateur est autorisé à afficher la page
else {
    $Autorisation = VerifierAutorisation("Id");
}

// Variables de la page
$page["Titre"] = "Titre de la page";
$page["Contenu"] = "chemin_vers/fichier_de_contenu.php";
$page["Lateral"] = "chemin_vers/fichier_encart_lateral.php";

// Affichage de la page
include("pages/layout.php");
?>
```

La première partie permet d'insérer dynamiquement tout les fichiers du répertoire *fonctions* dont l'extension est *.php*, ce qui revient à inclure toutes les librairies de fonctions nécessaires à l'interface utilisateur.

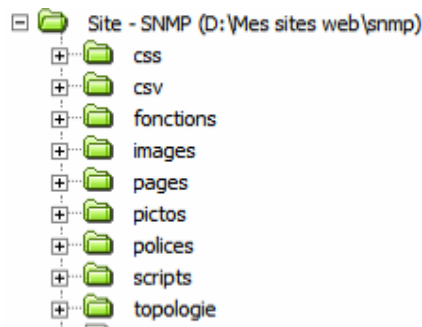


Figure 28 : Arborescence du site

Ensuite, on procède à l'authentification de l'utilisateur (qui repose sur l'utilisation de session PHP) dont le fonctionnement sera décrit plus bas dans ce document.

Enfin, il est nécessaire de renseigner trois champs du tableau *Page* :

- *Titre* : chaîne de caractère correspondant au titre de la page
- *Contenu* : chemin complet vers le fichier implémentant le contenu de la page (pour respecter la logique de l'application, ce fichier doit être placé dans le répertoire *pages* et porter le même nom que le fichier racine).
- *Lateral* : peut contenir le contenu de l'espace latéral gauche de la page (pour respecter la logique de l'application, si le nom du fichier racine est *ma\_page.php*, ce fichier doit être accessible depuis *pages/ma\_page\_lateral.php*).

Ensuite, la page *layout.php* est appelée, elle génère dynamiquement la mise en page du site.

Remarquons que l'accent a été mis sur le respect des normes du W3C (World Wide Web Consortium) et que le code de l'interface utilisateur est conforme (sauf erreur) aux normes XHTML 1.0 Strict et CSS 2.0. Lors d'éventuels futurs développements, il est primordial de veiller au respect de ces mêmes normes (accessibles via <http://www.w3.org/>).

Le contenu des pages doit lui aussi être structuré, l'architecture de base est la suivante :

```
<h3>
<?php echo $Page["Titre"]; ?></h3>
<p>Texte de la page.</p>
```

Qui permet d'afficher en tant que titre la chaîne de caractères qui a été définie dans le fichier racine et d'insérer du texte reconnu comme un paragraphe solidaire.

Pour respecter la mise en page, la structure des éléments de la partie latérale de l'interface doivent eux respecter la structure suivante :

```
<h3>Titre du panneau</h3>
<ul>
<li>Premier élément</li>
  <li>Deuxième élément</li>
  <li>Troisième élément</li>
</ul>
```

D'autre part, pour assurer le respect des normes du W3C lors de l'insertion d'images, il est préférable d'utiliser la fonction suivante, disponible dans la librairie *Images*.

```
AfficherImage ("chemin_vers/le_fichier_image.ext");
```

## B. Gestion des formulaires

Dans la plupart des pages du site, le contenu se résume à des formulaires, plus ou moins complexes, qui permettent d'interagir avec le moteur de découverte

Là encore, le respect des normes du W3C est assuré par l'utilisation de la librairie *Formulaires* qui implémente et facilite la création de formulaire dont le code est valide XHTML 1.0 et CSS 2.0.

Le code suivant permet de créer simplement un formulaire complexe :

```
// Contenu du formulaire
// Champ pour l'OID du matériel
$Contenu[0][0] = "text";
$Contenu[0][1] = "OID du produit";
```

```
$Contenu[0][2] = array("Modif[OID]", 30, 255, $Modif['OID'], "text");
// Champ pour le type de matériel
$Contenu[1][0] = "select";
$Contenu[1][1] = "Type du produit";
$Contenu[1][2] = array("Modif[IdType]", "ListeTypes", "Nom", "Id", "Nom ASC", $Modif['IdType']);
// Champ pour le nom du matériel
$Contenu[2][0] = "text";
$Contenu[2][1] = "Nom du produit";
$Contenu[2][2] = array("Modif[NomProduit]", 30, 255, $Modif['NomProduit'], "text");
// Fin du contenu pour le formulaire

echo Formulaire("modifier", $_SERVER['PHP_SELF']."?e=".$_GET['e'], $Contenu, "Modification des
paramètres d'un matériel");
```

En fait, il suffit de définir un tableau *Contenu* qui est un tableau à deux dimensions. La première dimension sert à indexer les différents champs présents dans le formulaire. Elle compte autant d'éléments que le nombre de champs dans le formulaire.

La seconde dimension comporte toujours trois éléments :

- le premier sert à définir le type de champ ;
- le second est le texte qui sera affiché en vis-à-vis du champ ;
- le troisième est un tableau contenant les paramètres du champ qui varient suivant le type de champ.

Les types de champs permettent de balayer les types classiques définis en XHTML plus certains types spéciaux que nous avons jugés utile d'implémenter :

- *text* : crée un champ texte classique
- *password* : crée un champ mot de passe classique
- *checkbox* : crée une liste de cases à cocher à partir des éléments d'une table MySQL
- *checkbox\_tableau* : crée une liste de cases à cocher à partir des éléments d'un tableau
- *select* : crée une liste déroulante à partir des éléments d'une table MySQL
- *radio* : crée une liste de boutons radio à partir des éléments d'un tableau
- *textarea* : crée une zone de texte multi ligne classique
- *file* : crée un champ classique permettant de télécharger une image sur le serveur
- *repertoire\_images* : crée une liste de boutons radio listant toutes les images (JPEG et PNG) d'un répertoire
- *grille* : crée un champ texte dont la valeur est modifiée lors du clic sur une image (utilisé pour le placement des équipements réseau)

Les paramètres diffèrent suivant le type de champ choisi, pour les connaître, il est nécessaire de consulter la documentation interne du fichier *formulaires.php* à laquelle une attention toute particulière a été portée.

Une fois le tableau à deux dimensions *Contenu* convenablement renseigné, il suffit d'appeler la fonction *Formulaire* avec les arguments suivants :

```
echo Formulaire("nom_formulaire", "page_destination.php", $Contenu, "Titre du formulaire",
$afficher_fieldset, $type_formulaire);
```

Avec les particularités suivantes :

- *\$afficher\_fieldset* : est un booléen et permet d'afficher ou non un fieldset (sorte de bordure regroupant les champs)
- *\$type\_formulaire* : est une chaîne de caractère précisant le type d'encodage du formulaire. Il peut ne pas être précisé (argument facultatif) sauf pour les formulaires permettant l'upload de fichier ou il doit valoir "mixed-form/data".

### C. Affichage de la topologie du réseau

Cette fonction est à la fois le plus intéressante du point de vue de l'utilisateur et la plus complexe de la part du programmeur. Elle a été réalisée grâce à la librairie GD de PHP qui permet de générer de manière dynamique des images. Elle a été testée avec succès avec la version 2.0.28 de GD.

Le but de cette fonction est d'afficher tout ou partie de la topologie réseau avec les paramètres que l'utilisateur fournit dans un formulaire. Chaque équipement affiché peut être cliqué afin de faire apparaître ses caractéristiques et de le positionner sur la représentation topologique.

Les icônes représentant les nœuds sont celles correspondant au type du nœud, elles sont personnalisables, comme nous l'avons vu précédemment. Les couleurs de liens ont également leur importance puisqu'ils reprennent le même code couleur que l'affichage des événements réseau pour signifier le type de modification observée.



Figure 29 : Représentation de la topologie d'un réseau

#### 1. Fonction d'affichage de la topographie

L'affichage de la topographie est réalisée par le fichier *pages/topographie.php* qui fait appel notamment aux bibliothèques de fonctions *Affichage* et *Images*. Comme les appels de fonctions sont assez complexes, nous indiquons ici en pseudo code la succession des étapes permettant cet affichage.

Récupération de l'Id du réseau en cours d'utilisation

Récupération de la taille de l'image à afficher (fonction de la résolution d'écran)

```
SI (une image de fond existe) {  
    On utilise l'image de fond redimensionnée  
}  
SINON {  
    On crée une image blanche de la bonne taille  
}
```

On définit le nombre de cases de l'image  
On calcule la taille des cases  
On définit la longueur des liaisons en pixels

On génère une clause de recherche des équipements fonction des paramètres saisis par l'utilisateur  
On récupère la liste des équipements concernés avec leurs infos que ce soit dans la liste ou dans les changements

On réalise la clause de recherche des liaisons fonction des paramètres saisis par l'utilisateur  
Recherche des liaisons impliquant les éléments sélectionnés

Equipements qui ont été visités = 0  
Liaisons qui ont été visitées = 0  
\$LiaisonsVisitees = array();

On trouve un point de départ du réseau : le(s) routeur(s)  
Si aucun routeur n'a été trouvé dans la liste des équipements, on l'ajoute de force

```
Pour chacun des routeurs trouvés {  
    On positionne l'équipement  
    On note la position de cette équipement  
    On appelle la fonction récursive AfficherLiaisons  
}
```

On appelle la fonction d'affichage des icones

On crée la map qui servira à cliquer sur les images

Enfin, on affiche l'image du réseau

On détaille de même le fonctionnement de *AfficherLiaisons*

```
AfficherLiaisons ($Image, $DimensionsImage, $Equipements, $EquipementsVisites,  
$EquipementPrecedent, $Liaisons, $LiaisonsVisitees, $LongueurLiaisons, $Coef, $Position, $Niveau,  
$AfficherVitesse) {
```

```
    Pour chacune des liaisons {  
        SI (la liaisons s'effectue avec l'équipement et si elle n'a pas déjà été visitée) {  
            On récupère l'id de l'équipement à l'autre bout de la liaison  
            On recherche le nouvel équipement dans la liste  
            Si a ce stade le nouvel équipement n'a pas été trouvé, c'est qu'il s'agit d'un équipement que  
nous ne souhaitons pas afficher au départ mais que nous afficherons quand même (c'est un noeud  
terminal du graphe que nous affichons)  
        }  
    }
```

On positionne le nouvel équipement  
On trace la liaison en cours d'étude

```
    On sélectionne la couleur de la liaison fonction du type d'événement trouvé  
    AjouterLiaison ($Image, $Position[$EquipementPrecedent['Id']],  
$Position[$nouvel_equipement['Id']], max($liaison['Vitesse1'], $liaison['Vitesse2']), $couleur,  
$AfficherVitesse);
```

On marque la liaison comme visitée

```
    Si (le noeud n'est pas terminal) {  
        on explore les liaisons qui le concernent  
    }
```

```

    AfficherLiaisons($Image, $DimensionsImage, $Equipements, $EquipementsVisites,
    $nouvel_equipement, $Liaisons, $LiaisonsVisitees, $LongueurLiaisons, $Coef, $Position,
    $Niveau+1, $AfficherVitesse);
}
    On a parcouru toutes les liaisons pour cet équipement, on peut le marquer comme visité
}
}

```

## 2. Positionnement des images

Le positionnement des images est par défaut partiellement aléatoire, mais, afin d'offrir une vue tout de même hiérarchisée, il répond quand même à deux critères :

- tous les équipements sont à la même distance de l'équipement auquel ils sont reliés
- la distance entre les équipements diminue avec la « profondeur » de l'équipement : c'est-à-dire le nombre de nœuds traversés depuis un des routeurs du réseau

Le positionnement des équipements est réalisé par la fonction suivante :

```

function PositionnerEquipement($DimensionsImage, $LongueurLiaisons, $Coef, $Niveau,
$Equipement, $EquipementPrecedent = "") {
    SI (aucune position prédéfinie) {
        SI (pas d'équipement précédent) {
            On positionne l'équipement dans un coin de l'image
        }
        SINON {
            Angle de rotation $alpha = pris au hasard entre 0 et 2*pi
            Nouvelle Longueur = $LongueurLiaisons/($Niveau*$Coef);
            Variation longueur = cos($alpha)*Nouvelle Longueur;
            Variation hauteur = sin($alpha)*$nouvelle_longueur;
        }
    }
    SINON {
        On renvoie la position enregistrée
    }
}

```

### Conclusion

Ce projet nous a permis de consolider nos connaissances sur les réseaux informatiques, connaissances « empirique » mais aussi issu du cours de « Systèmes d'Information » dispensé durant la même séquence.

Au niveau programmation, il nous a également permis d'approfondir nos connaissances sur les interfaces graphiques et la gestion des options de visualisation.

Comme nous nous y attendions, ce projet n'a pas été de toute simplicité mais malgré les difficultés rencontrées, le résultat est relativement satisfaisant, même s'il est toujours possible de l'améliorer.

## Annexes

### Préambule

Les trois premières annexes récapitulent l'ensemble des fonctions présentes dans les trois librairies développées pour ce projet. Chacune des fonctions est suivie d'une courte description et les arguments et résultats des fonctions sont détaillés, notamment en ce qui concerne le format des tableaux (PHP) utilisés.

La quatrième annexe liste l'ensemble des objets SNMP utilisés avec les fonctions de la librairie SNMP.

La cinquième annexe explicite la méthode utilisée pour récupérer les correspondances MAC/Ports des switches.

### Annexe 1 : Librairie SNMP et fonctions

Cette librairie est constituée de l'ensemble des fonctions utilisant le protocole SNMP.

Fonctions	Description																		
GetCorrespondancePortMacBySNMP	Récupère les correspondances <i>MAC/Ports</i>																		
Arguments :																			
<ul style="list-style-type: none"><li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li><li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li><li>• Optimisation (<i>booléen</i>) : active ou non les optimisations de la fonction (cf. Fonction GetCorrespondancePortMacBySNMP page 32)</li></ul>																			
Fonction GetCorrespondancePortMacBySNMP page 32)																			
<ul style="list-style-type: none"><li>• Niveau de débogage : cf. Débogage page 28</li></ul>																			
Résultat :																			
<ul style="list-style-type: none"><li>• Tableau des correspondances de la forme suivante :</li></ul>																			
<table><tr><th>Index</th><th>Nom du port</th><th colspan="2">Adresses MAC sur ce port</th></tr><tr><td rowspan="3">0</td><td rowspan="3"><i>(string)</i></td><th>Index</th><th>Adresse MAC</th></tr><tr><td>0</td><td><i>(string)</i><sup>1</sup></td></tr><tr><td>...</td><td>...</td></tr><tr><td rowspan="2">...</td><td rowspan="2">...</td><th>Index</th><th>Adresse MAC</th></tr><tr><td>...</td><td>...</td></tr></table>		Index	Nom du port	Adresses MAC sur ce port		0	<i>(string)</i>	Index	Adresse MAC	0	<i>(string)</i> <sup>1</sup>	...	...	...	...	Index	Adresse MAC	...	...
Index	Nom du port	Adresses MAC sur ce port																	
0	<i>(string)</i>	Index	Adresse MAC																
		0	<i>(string)</i> <sup>1</sup>																
		...	...																
...	...	Index	Adresse MAC																
		...	...																

<sup>1</sup> Toutes les adresses MAC sont au format 'XX:XX:XX:XX:XX:XX', ce qui signifie que les chiffres hexadécimaux sont systématiquement sur deux chiffres ou lettres, et en majuscules pour ces dernières (exemple : '0A' au lieu de 'a'). Cf. la Fonction MacFormat page 39.



GetMembersOfClusterBySNMP	Trouve les membres d'un cluster																								
<p>Arguments :</p> <ul style="list-style-type: none"><li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li><li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li><li>• Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>• Tableau des nouveaux switchs découvert :</li></ul> <table><tr><th>ID</th><th>Nom</th><th>IP</th><th>Community</th><th>sysServices</th><th>Type</th><th>ObjectID</th><th>MAC</th></tr><tr><td>0</td><td>(<i>string</i>)</td><td>(<i>string</i>)<sup>2</sup></td><td>(<i>string</i>)</td><td>(<i>int</i>)<sup>3</sup></td><td>0<sup>4</sup></td><td>(<i>string</i>)<sup>5</sup></td><td>(<i>string</i>)</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>0</td><td>...</td><td>...</td></tr></table>		ID	Nom	IP	Community	sysServices	Type	ObjectID	MAC	0	( <i>string</i> )	( <i>string</i> ) <sup>2</sup>	( <i>string</i> )	( <i>int</i> ) <sup>3</sup>	0 <sup>4</sup>	( <i>string</i> ) <sup>5</sup>	( <i>string</i> )	...	...	...	...	...	0	...	...
ID	Nom	IP	Community	sysServices	Type	ObjectID	MAC																		
0	( <i>string</i> )	( <i>string</i> ) <sup>2</sup>	( <i>string</i> )	( <i>int</i> ) <sup>3</sup>	0 <sup>4</sup>	( <i>string</i> ) <sup>5</sup>	( <i>string</i> )																		
...	...	...	...	...	0	...	...																		
GetSwitchMacAdrBySNMP	Donne l'adresse MAC du switch interrogé																								
<p>Arguments :</p> <ul style="list-style-type: none"><li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li><li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li><li>• Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>• Adresse MAC correspondant à l'IP utilisée pour gérer le switch : (<i>string</i>)</li></ul>																									
GetInterfaceMacAdrBySNMP	Donne les adresses MAC de toutes les interfaces du switch																								
<p>Arguments :</p> <ul style="list-style-type: none"><li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li><li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li><li>• Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>• Tableau listant les adresses MAC des interfaces (ports) du switch :</li></ul> <table><tr><th>Index</th><th>Adresse MAC</th></tr><tr><td>0</td><td>(<i>string</i>)</td></tr><tr><td>...</td><td>...</td></tr></table>		Index	Adresse MAC	0	( <i>string</i> )	...	...																		
Index	Adresse MAC																								
0	( <i>string</i> )																								
...	...																								
GetTypeServicesBySNMP	Indique le type de services fournis par le switch (cf. modèle OSI)																								
<p>Arguments :</p> <ul style="list-style-type: none"><li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li><li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li><li>• Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>• Valeur du sysServices (cf. la description de cette fonction page 35)</li></ul>																									

<sup>2</sup> Toutes les adresses IP ont la forme 'XXX.XXX.XXX.XXX', c'est-à-dire par exemple que le chiffre '1' sera traduit en '001'. Cette adresse IP peut techniquement être remplacée par le nom DNS correspondant.

<sup>3</sup> Cf. Fonction GetTypeServicesBySNMP page 35.

<sup>4</sup> Le champ type est automatiquement rempli à la valeur 0. Le champ est complété pour montrer qu'il existe, mais la valeur correcte sera cherchée qu'au moment de l'exécution de la Fonction CreationTableauEquipements.

<sup>5</sup> Cf. Fonction GetObjectIDBySNMP page 36.

GetObjectIDBySNMP	Donne l'objet ID de l'équipement						
<p>Arguments :</p> <ul style="list-style-type: none"> <li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li> <li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li> <li>• Niveau de débogage : cf. Débogage page 28</li> </ul> <p>Résultat :</p> <ul style="list-style-type: none"> <li>• OID du produit (fonction du constructeur, marque, type d'équipement, etc.)</li> </ul>							
GetArpTableBySNMP	Récupère la table ARP du switch						
<p>Arguments :</p> <ul style="list-style-type: none"> <li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li> <li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li> <li>• Niveau de débogage : cf. Débogage page 28</li> </ul> <p>Résultat :</p> <ul style="list-style-type: none"> <li>• Table ARP :</li> </ul> <table border="1"> <thead> <tr> <th>Adresse IP</th><th>Adresse MAC</th></tr> </thead> <tbody> <tr> <td>(<i>string</i>)</td><td>(<i>string</i>)</td></tr> <tr> <td>...</td><td>...</td></tr> </tbody> </table>		Adresse IP	Adresse MAC	( <i>string</i> )	( <i>string</i> )	...	...
Adresse IP	Adresse MAC						
( <i>string</i> )	( <i>string</i> )						
...	...						
GetIfSpeedBySNMP	Trouve les vitesses de fonctionnement des interfaces réseaux du switch						
<p>Arguments :</p> <ul style="list-style-type: none"> <li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li> <li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li> <li>• Niveau de débogage : cf. Débogage page 28</li> </ul> <p>Résultat :</p> <ul style="list-style-type: none"> <li>• Tableau associant les ports et leurs vitesses (en bits par seconde)</li> </ul> <table border="1"> <thead> <tr> <th>Nom du port</th><th>Vitesse du port</th></tr> </thead> <tbody> <tr> <td>(<i>string</i>)</td><td>(<i>int</i>)</td></tr> <tr> <td>...</td><td>...</td></tr> </tbody> </table>		Nom du port	Vitesse du port	( <i>string</i> )	( <i>int</i> )	...	...
Nom du port	Vitesse du port						
( <i>string</i> )	( <i>int</i> )						
...	...						
GetNextRouterBySNMP	Récupère les adresses IP des routeurs que le switch connaît						
<p>Arguments :</p> <ul style="list-style-type: none"> <li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li> <li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li> <li>• Niveau de débogage : cf. Débogage page 28</li> </ul> <p>Résultat :</p> <ul style="list-style-type: none"> <li>• Tableau listant les adresses IP des routeurs connus par le switch :</li> </ul> <table border="1"> <thead> <tr> <th>Index</th><th>Adresse IP</th></tr> </thead> <tbody> <tr> <td>0</td><td>(<i>string</i>)</td></tr> <tr> <td>...</td><td>...</td></tr> </tbody> </table>		Index	Adresse IP	0	( <i>string</i> )	...	...
Index	Adresse IP						
0	( <i>string</i> )						
...	...						

IPScanBySNMP	Scanne une liste d'IP par SNMP																								
<p>Arguments :</p> <ul style="list-style-type: none"><li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li><li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li><li>• Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>• Tableau des nouveaux switchs découverts :</li></ul> <table><tr><th>ID</th><th>Nom</th><th>IP</th><th>Community</th><th>sysServices</th><th>Type</th><th>ObjectID</th><th>MAC</th></tr><tr><td>0</td><td>(<i>string</i>)</td><td>(<i>string</i>)</td><td>(<i>string</i>)</td><td>(<i>int</i>)</td><td>0</td><td>(<i>string</i>)</td><td>(<i>string</i>)</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>0</td><td>...</td><td>...</td></tr></table>		ID	Nom	IP	Community	sysServices	Type	ObjectID	MAC	0	( <i>string</i> )	( <i>string</i> )	( <i>string</i> )	( <i>int</i> )	0	( <i>string</i> )	( <i>string</i> )	...	...	...	...	...	0	...	...
ID	Nom	IP	Community	sysServices	Type	ObjectID	MAC																		
0	( <i>string</i> )	( <i>string</i> )	( <i>string</i> )	( <i>int</i> )	0	( <i>string</i> )	( <i>string</i> )																		
...	...	...	...	...	0	...	...																		
GetVlanTableBySNMP	Recherche les différents VLAN présents sur le switch																								
<p>Arguments :</p> <ul style="list-style-type: none"><li>• Adresse IP ou nom DNS du switch à interroger (<i>string</i>)</li><li>• Nom de la communauté à utiliser. Par défaut « public » (<i>string</i>)</li><li>• Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>• Tableau des VLAN :</li></ul> <table><tr><th>Index</th><th>ID</th><th>Nom</th><th>DefaultVLAN</th><th>Community</th></tr><tr><td>0</td><td>(<i>int</i>)</td><td>(<i>string</i>)</td><td>(<i>boolean</i>)</td><td>(<i>string</i>)</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr></table>		Index	ID	Nom	DefaultVLAN	Community	0	( <i>int</i> )	( <i>string</i> )	( <i>boolean</i> )	( <i>string</i> )	...	...	...	...	...									
Index	ID	Nom	DefaultVLAN	Community																					
0	( <i>int</i> )	( <i>string</i> )	( <i>boolean</i> )	( <i>string</i> )																					
...	...	...	...	...																					

## Annexe 2 : Librairie Réseau et fonctions

Cette librairie regroupe les fonctions utiles pour toutes applications réseaux.

Fonctions	Description						
MacFormat	Vérifie qu'une adresse MAC a le bon format, et si besoin la reformate						
Arguments : <ul style="list-style-type: none"> <li>• Adresse MAC à vérifier (<i>string</i>)</li> <li>• Niveau de débogage : cf. Débogage page 28</li> </ul> Résultat : <ul style="list-style-type: none"> <li>• Adresse MAC formatée</li> </ul>							
ComparaisonIPs	Compare deux adresses IP en indiquant laquelle est plus grande que l'autre, ou si elles sont égales						
Arguments : <ul style="list-style-type: none"> <li>• Adresse IP 1 (<i>string</i>)</li> <li>• Adresse IP 2 (<i>string</i>)</li> </ul> Résultat : <ul style="list-style-type: none"> <li>• Cette fonction renvoie l'une des trois valeurs suivantes :               <ul style="list-style-type: none"> <li>○ 0 si les deux adresses IP sont égales,</li> <li>○ 1 si l'adresse IP 1 est plus grande que l'adresse IP 2,</li> <li>○ -1 dans le cas inverse.</li> </ul> </li> </ul>							
GetLocalArpTable	Obtient la table ARP locale du serveur						
Arguments : <ul style="list-style-type: none"> <li>• Niveau de débogage : cf. Débogage page 28</li> </ul> Résultat : <ul style="list-style-type: none"> <li>• Table ARP :               <table border="1"> <thead> <tr> <th>Adresse IP</th><th>Adresse MAC</th></tr> </thead> <tbody> <tr> <td>(<i>string</i>)</td><td>(<i>string</i>)</td></tr> <tr> <td>...</td><td>...</td></tr> </tbody> </table> </li> </ul>		Adresse IP	Adresse MAC	( <i>string</i> )	( <i>string</i> )	...	...
Adresse IP	Adresse MAC						
( <i>string</i> )	( <i>string</i> )						
...	...						
GetNumericOID	Traduit une OID quelconque en une OID numérique						
Arguments : <ul style="list-style-type: none"> <li>• OID à traduire (<i>string</i>)</li> </ul> Résultat : <ul style="list-style-type: none"> <li>• OID sous forme numérique (<i>string</i>)</li> </ul>							

### Annexe 3 : Librairie Topologie et fonctions

Cette librairie contient toutes les fonctions ne servant qu'à ce programme. Tous les tableaux qui seront représentés dans cette annexe le seront dans leurs formes minimales. Autrement dit, ces tableaux peuvent contenir des paires (clés, valeurs) supplémentaires, mais seront alors ignorées par les fonctions concernées. Par contre, si ces formes minimales ne sont pas respectées, les fonctions ne pourront pas opérer correctement et les résultats seront inexacts.

Fonctions	Description																																																																	
GetSwitchLinks	Construit la table des liaisons inter-switchs																																																																	
Arguments :																																																																		
<ul style="list-style-type: none"><li>Tableau listant les switches détectés :<table><tr><th>Index des switches</th></tr><tr><td>0</td></tr><tr><td>...</td></tr></table></li><li>Tableau contenant les adresses MAC des interfaces des switches :<table><tr><th>Index switches</th><th colspan="2">Adresses MAC des ports du switch</th></tr><tr><td rowspan="3">0</td><th>Index</th><th>Adresse MAC</th></tr><tr><td>0</td><td>(string)</td></tr><tr><td>...</td><td>...</td></tr><tr><td rowspan="2">...</td><th>Index</th><th>Adresse MAC</th></tr><tr><td>...</td><td>...</td></tr></table></li><li>Correspondances MAC/Ports des switches (tableau) :<table><tr><th>Index switches</th><th colspan="6">Correspondances du switch</th></tr><tr><td rowspan="6">0</td><th>Index ports</th><th>Nom du port</th><th colspan="2">Adresses MAC sur ce port</th><th>Numéro VLAN</th><th>Nom VLAN</th></tr><tr><td rowspan="3">0</td><td rowspan="3">(string)</td><th>Index</th><th>Adresse MAC</th><td rowspan="3">(int)</td><td rowspan="3">(string)</td></tr><tr><td>0</td><td>(string)</td></tr><tr><td>...</td><td>...</td></tr><tr><td rowspan="2">...</td><td rowspan="2">...</td><th>Index</th><th>Adresse MAC</th><td rowspan="2">...</td><td rowspan="2">...</td></tr><tr><td>...</td><td>...</td></tr><tr><td rowspan="3">...</td><th>Index ports</th><th>Nom du port</th><th colspan="2">Adresses MAC sur ce port</th><th>Numéro VLAN</th><th>Nom VLAN</th></tr><tr><td rowspan="2">...</td><td rowspan="2">...</td><th>Index</th><th>Adresse MAC</th><td rowspan="2">...</td><td rowspan="2">...</td></tr><tr><td>...</td><td>...</td></tr></table></li><li>Réciprocité (booléen) : active ou non la vérification de la réciprocité des liaisons (cf. Fonction GetSwitchLinks page 41)</li><li>Niveau de débogage : cf. Débogage page 28</li></ul>		Index des switches	0	...	Index switches	Adresses MAC des ports du switch		0	Index	Adresse MAC	0	(string)	...	...	...	Index	Adresse MAC	...	...	Index switches	Correspondances du switch						0	Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN	0	(string)	Index	Adresse MAC	(int)	(string)	0	(string)	...	...	...	...	Index	Adresse MAC	...	...	...	...	...	Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN	...	...	Index	Adresse MAC	...	...	...	...
Index des switches																																																																		
0																																																																		
...																																																																		
Index switches	Adresses MAC des ports du switch																																																																	
0	Index	Adresse MAC																																																																
	0	(string)																																																																
	...	...																																																																
...	Index	Adresse MAC																																																																
	...	...																																																																
Index switches	Correspondances du switch																																																																	
0	Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN																																																												
	0	(string)	Index	Adresse MAC	(int)	(string)																																																												
			0	(string)																																																														
			...	...																																																														
	...	...	Index	Adresse MAC	...	...																																																												
			...	...																																																														
...	Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN																																																												
	...	...	Index	Adresse MAC	...	...																																																												
			...	...																																																														

Résultat :

- Tableau suivant :

Index liaison	Liaison				
	Extrémité	Index switch	Port	N° VLAN	Nom VLAN
0	0 1 <sup>er</sup> extrémité	(int)	(string)	(int)	(string)
	1 2 <sup>e</sup> extrémité	...	...	...	...
	Extrémité	Index switch	Port	N° VLAN	Nom VLAN
	0 1 <sup>er</sup> extrémité	...	...	...	...
	1 2 <sup>e</sup> extrémité	...	...	...	...
	...	...	...	...	...
1	Correspondances MAC/Ports données en argument, où les ports relatifs aux liaisons nouvellement créées ont été supprimés. Même format que le tableau				

GetNoManageableDevices	Détection des nœuds non gérables et construction de la table des liaisons correspondante
------------------------	--

Arguments :

- Tableau listant les switches détectés :

Index des switches
0
...

- Correspondances MAC/Ports des switches (tableau) :

Index switches	Correspondances du switch					
0	Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN
	0	(string)	Index	Adresse MAC	(int)	(string)
			0	(string)		
			...	...		
...	...	...	Index	Adresse MAC	...	...
			...	...		
...	Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN
	...	...	Index	Adresse MAC	...	...
			...	...		
			...	...		

- Niveau de débogage : cf. Débogage page 28

Résultat :

- Tableau suivant :

	Index liaison	Liaison				
	0	Extrémité	Index switch	Port	N° VLAN	Nom VLAN
	0	0 1 <sup>er</sup> extrémité	(int)	(string)	(int)	(string)
		1 2 <sup>e</sup> extrémité	...	...	...	...
		Extrémité	Index switch	Port	N° VLAN	Nom VLAN
	...	0 1 <sup>er</sup> extrémité	...	...	...	...
		1 2 <sup>e</sup> extrémité	...	...	...	...
1	Liste des nouveaux switches détectés (même format que les autres listes de switches)					
2	Correspondances MAC/Ports données en argument, où les ports relatifs aux liaisons nouvellement créées ont été supprimés. Même format que le tableau					

ComparaisonSwitchsAvecListeSwitchs	Compare deux listes d'équipements et supprime les doublons. A utiliser avec tout équipement sauf routeurs																																				
<p>Arguments :</p> <ul style="list-style-type: none"><li>Tableau des nouveaux switchs à comparer :</li></ul> <table><tr><td>ID</td><td>Nom</td><td>MAC</td></tr><tr><td>0</td><td>(string)</td><td>(string)</td></tr><tr><td>...</td><td>...</td><td>...</td></tr></table> <ul style="list-style-type: none"><li>Tableau des switchs déjà détectés (qui vont servir pour la comparaison) :</li></ul> <table><tr><td>ID</td><td>Nom</td><td>MAC</td></tr><tr><td>0</td><td>(string)</td><td>(string)</td></tr><tr><td>...</td><td>...</td><td>...</td></tr></table> <ul style="list-style-type: none"><li>Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>Tableau listant les nouveaux switchs (donc issus du premier argument de cette fonction) qui ne sont pas présent dans les anciens switchs déjà détectés (donc issus du deuxième argument). Ce tableau à le même format que les tableaux donnés en argument (la forme exposé ici est la forme minimale, mais d'autres colonnes peuvent composer ces tableaux, et ces colonnes se retrouveront naturellement dans le tableau résultat !)</li></ul>		ID	Nom	MAC	0	(string)	(string)	...	...	...	ID	Nom	MAC	0	(string)	(string)	...	...	...																		
ID	Nom	MAC																																			
0	(string)	(string)																																			
...	...	...																																			
ID	Nom	MAC																																			
0	(string)	(string)																																			
...	...	...																																			
ComparaisonRouteursAvecListeSwitchs	Compare deux listes d'équipements et supprime les doublons. A utiliser avec les routeurs																																				
<p>Arguments :</p> <ul style="list-style-type: none"><li>Tableau listant les adresses IP des routeurs :</li></ul> <table><tr><td>Index</td><td>Adresse IP</td></tr><tr><td>0</td><td>(string)</td></tr><tr><td>...</td><td>...</td></tr></table> <ul style="list-style-type: none"><li>Liste des switchs déjà connus (tableau) :</li></ul> <table><tr><td>ID</td><td>Nom</td><td>IP</td></tr><tr><td>0</td><td>(string)</td><td>(string)</td></tr><tr><td>...</td><td>...</td><td>...</td></tr></table> <ul style="list-style-type: none"><li>Liste des plages d'adresses IP autorisées (tableau) :</li></ul> <table><tr><td>Index</td><td colspan="2">Plage d'adresses IP</td></tr><tr><td rowspan="3">0</td><td>Adresses limitant la plage</td><td>Adresse IP</td></tr><tr><td>Début</td><td>(string)</td></tr><tr><td>Fin</td><td>...</td></tr><tr><td rowspan="3">...</td><td>Adresses limitant la plage</td><td>Adresse IP</td></tr><tr><td>Début</td><td>...</td></tr><tr><td>Fin</td><td>...</td></tr></table> <ul style="list-style-type: none"><li>Niveau de débogage : cf. Débogage page 28</li></ul> <p>Résultat :</p> <ul style="list-style-type: none"><li>Tableau suivant :</li></ul> <table><tr><td>0</td><td>Liste des adresses IP des routeurs initiale où les IP apparaissant aussi dans la liste des switchs déjà connus ont été supprimées</td></tr><tr><td>1</td><td>Liste des index des switchs déjà connus apparaissant aussi dans la liste des adresses IP des routeurs.</td></tr></table>		Index	Adresse IP	0	(string)	...	...	ID	Nom	IP	0	(string)	(string)	...	...	...	Index	Plage d'adresses IP		0	Adresses limitant la plage	Adresse IP	Début	(string)	Fin	...	...	Adresses limitant la plage	Adresse IP	Début	...	Fin	...	0	Liste des adresses IP des routeurs initiale où les IP apparaissant aussi dans la liste des switchs déjà connus ont été supprimées	1	Liste des index des switchs déjà connus apparaissant aussi dans la liste des adresses IP des routeurs.
Index	Adresse IP																																				
0	(string)																																				
...	...																																				
ID	Nom	IP																																			
0	(string)	(string)																																			
...	...	...																																			
Index	Plage d'adresses IP																																				
0	Adresses limitant la plage	Adresse IP																																			
	Début	(string)																																			
	Fin	...																																			
...	Adresses limitant la plage	Adresse IP																																			
	Début	...																																			
	Fin	...																																			
0	Liste des adresses IP des routeurs initiale où les IP apparaissant aussi dans la liste des switchs déjà connus ont été supprimées																																				
1	Liste des index des switchs déjà connus apparaissant aussi dans la liste des adresses IP des routeurs.																																				



## FusionCorrespondancesVlans

Renvoie une correspondance MAC/Ports unique à partir des correspondances de chaque VLAN

Arguments :

- Correspondances MAC/Ports d'un switch (tableau) :

Index VLAN	Correspondances du switch pour les différents VLAN			
0	Index ports	Nom du port	Adresses MAC sur ce port	
	0	(string)	Index	Adresse MAC
			0	(string)
			...	...
...	...	...	Index	Adresse MAC
			...	...
	...	...	Index	Adresse MAC
			...	...

- Liste des VLAN des switchs (tableau) :

Index	VLAN			
0	ID	Nom	Défaut	Community
	(int)	(string)	(booléen)	(string)
...	ID	Nom	Défaut	Community
	...	...	...	...

- Optimisation (*booléen*) : active ou non les optimisations de la fonction
- Niveau de débogage : cf. Débogage page 28

Résultat :

- Correspondances MAC/Ports du switch (tableau) :

Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN
0	(string)	Index	Adresse MAC	(int)	(string)
		0	(string)		
		...	...		
...	...	Index	Adresse MAC	...	...
		...	...		
		...	...		

## CreationTableauLiaisons

Crée le tableau des liaisons final

Arguments :

- Tableau liant les vitesses des ports des switchs à leurs noms :

Index	Switch	
0	Nom port	Vitesse port
	(string)	(int)
...	Nom port	Vitesse port
	...	...

- Liste des liaisons inter-switchs (tableau) :

Index liaison	Liaison				
	Extrémité	Index switch	Port	N° VLAN	Nom VLAN
0	<b>0</b> 1 <sup>er</sup> extrémité	(int)	(string)	(int)	(string)
	<b>1</b> 2 <sup>e</sup> extrémité	...	...	...	...
...	<b>Extrémité</b>	<b>Index switch</b>	<b>Port</b>	<b>N° VLAN</b>	<b>Nom VLAN</b>
	<b>0</b> 1 <sup>er</sup> extrémité	...	...	...	...
	<b>1</b> 2 <sup>e</sup> extrémité	...	...	...	...

- Correspondances MAC/Ports des switches (tableau) :

Index switchs	Correspondances du switch					
	Index ports	Nom du port	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN
0	0	(string)	<b>Index</b>	<b>Adresse MAC</b>	(int)	(string)
			0	(string)		
			...	...		
	...	...	<b>Index</b>	<b>Adresse MAC</b>	...	...
			...	...		
...	...	...	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN
			<b>Index</b>	<b>Adresse MAC</b>		
			...	...		
	...	...	Adresses MAC sur ce port		Numéro VLAN	Nom VLAN
			<b>Index</b>	<b>Adresse MAC</b>		
			...	...		

- Niveau de débogage : cf. Débogage page 28

Résultat :

- Tableau suivant :

Liaisons	Index	Switch 1	Switch 2	Port 1	Port 2	VLAN 1	VLAN 2	Nom VLAN 1	Nom VLAN 2	Vitesse 1	Vitesse 2
	0	(int)	(int)	(string)	(string)	(int)	(int)	(string)	(string)	(int)	(int)
	...	...	...	...	...	...	...	...	...	...	...
Equipements	Index	ID de l'équipement					Adresse MAC				
	0	(int)					(string)				

CreationTableauEquipements

Crée le tableau des équipements final

Arguments :

- Tableau des switches :

ID	Nom	IP	Community	sysServices	Type	ObjectID	MAC
0	(string)	(string)	(string)	(int)	0	(string)	(string)
...	...	...	...	...	0	...	...
- Autres équipements détectés (tableau issu de la fonction précédente) :

Index	ID de l'équipement	Adresse MAC
0	(int)	(string)
...	...	...
- Table ARP inverse (table RARP) :

Adresse MAC	Adresse IP
(string)	(string)
...	...
- Tableau liant les ObjectID aux types d'équipements :

ObjectID (OID)	Type d'équipement
(string)	(int)
...	...
- Cache SNMP : cache contenant les informations des équipements ayant répondu au « ping » SNMP, mais étant des terminaux et non des nœuds réseaux (serveurs par exemple). Ce tableau à la forme suivante :

IP	Nom	Community	sysServices	ObjectID
(string)	(string)	(string)	(int)	(string)
...	...	...	...	...
- Niveau de débogage : cf. Débogage page 28

Résultat :

- Tableau des équipements :

Index	Equipement							
0	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID
	(int)	(string)	(string)	(string)	(string)	(int)	(int)	(string)
	...	...	...	...	...	...	...	...
...	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID
	...	...	...	...	...	...	...	...

RechercheDNS	Recherche le nom DNS correspondant à une IP							
Arguments :								
• Tableau des équipements :								
Index	Equipement							
0	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID
	(int)	(string)	(string)	(string)	(string)	(int)	(int)	(string)
	...	...	...	...	...	...	...	...
...	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID
	...	...	...	...	...	...	...	...

Résultat :

- Tableau des équipements, complété avec les noms DNS :

Index	Equipement								
0	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID	DNS
	(int)	(string)	(string)	(string)	(string)	(int)	(int)	(string)	(string)
	...	...	...	...	...	...	...	...	...
...	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID	DNS
	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...

DecouverteReseau

Moteur de découverte de la topologie du réseau

Arguments :

- Tableau récapitulant les options du moteur :

Options	Statut
UtilisationListeSwitch	(booléen)
ScanPlageIP	(booléen)
ScanIP_ModePoursuite	(booléen)
ScanIP_ModePoursuite_Profondeur	(int)
UsePlageIPsAllowed	(booléen)
CmdArp	(booléen)
SearchMembersOfCluster	(booléen)
RechercheDNS	(booléen)

- Liste initiale de switches à interroger :

ID	Nom	IP	Community
0	(string)	(string)	(string)
...	...	...	...

- Liste de noms de communautés à scanner :

Index	Nom de communauté
0	(string)
...	...

- Plages d'adresses IP à scanner :

Index	Plage d'adresses IP	
0	Adresses limitant la plage	Adresse IP
	Début	(string)
	Fin	...
...	Adresses limitant la plage	Adresse IP
	Début	...
	Fin	...

- Plages d'adresses IP autorisées :

Index	Plage d'adresses IP	
0	Adresses limitant la plage	Adresse IP
	Début	(string)
	Fin	...
...	Adresses limitant la plage	Adresse IP
	Début	...
	Fin	...

- Table associant OID est type de nœud :

ObjectID (OID)	Type d'équipement
(string)	(int)
...	...

- Niveau de débogage : cf. Débogage page 28

Résultat :

- Tableau suivant :

Liaisons	Index	Switch 1	Switch 2	Port 1	Port 2	VLAN 1	VLAN 2	Nom VLAN 1	Nom VLAN 2	Vitesse 1	Vitesse 2
	0	(int)	(int)	(string)	(string)	(int)	(int)	(string)	(string)	(int)	(int)
	...	...	...	...	...	...	...	...	...	...	...
Equipements	Index	Equipement									
	0	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID	DNS	
		(int)	(string)	(string)	(string)	(string)	(int)	(int)	(string)	(string)	
		...	...	...	...	...	...	...	...	...	
	...	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID	DNS	
	...	...	...	...	...	...	...	...	...	...	

ExportationVersMysql	Exporte les tableaux d'équipements et de liaisons vers une base de données MySQL									
Arguments :										
• Tableau des liaisons :										
Index	Switch 1	Switch 2	Port 1	Port 2	VLAN 1	VLAN 2	Nom VLAN 1	Nom VLAN 2	Vitesse 1	Vitesse 2
0	(int)	(int)	(string)	(string)	(int)	(int)	(string)	(string)	(int)	(int)
...	...	...	...	...	...	...	...	...	...	...
• Tableau des équipements :										
Index	Equipement									
0	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID	DNS	
	(int)	(string)	(string)	(string)	(string)	(int)	(int)	(string)	(string)	
	...	...	...	...	...	...	...	...	...	
...	ID	Nom	IP	MAC	Community	sysServices	Type	ObjectID	DNS	
	...	...	...	...	...	...	...	...	...	
• Préfixe des tables (préfixe identifiant le réseau). Cf. Fonction ExportationVersMysql page 44.										
Résultat :										
• Entrées dans la base de données										

ModificationEtatReseau					Enregistre dans une base de données MySQL les modifications survenues sur le réseau depuis un état de référence					
Arguments :										
<ul style="list-style-type: none"><li>Date (et heure) à laquelle le nouvel état du réseau à été obtenu</li><li>Tableau des liaisons :</li></ul>										
<b>Index</b>	<b>Switch 1</b>	<b>Switch 2</b>	<b>Port 1</b>	<b>Port 2</b>	<b>VLAN 1</b>	<b>VLAN 2</b>	<b>Nom VLAN 1</b>	<b>Nom VLAN 2</b>	<b>Vitesse 1</b>	<b>Vitesse 2</b>
0	(int)	(int)	(string)	(string)	(int)	(int)	(string)	(string)	(int)	(int)
...	...	...	...	...	...	...	...	...	...	...
<ul style="list-style-type: none"><li>Tableau des équipements :</li></ul>										
<b>Index</b>	<b>Equipement</b>									
0	<b>ID</b>	<b>Nom</b>	<b>IP</b>	<b>MAC</b>	<b>Community</b>	<b>sysServices</b>	<b>Type</b>	<b>ObjectID</b>	<b>DNS</b>	
	(int)	(string)	(string)	(string)	(string)	(int)	(int)	(string)	(string)	
	...	...	...	...	...	...	...	...	...	
...	<b>ID</b>	<b>Nom</b>	<b>IP</b>	<b>MAC</b>	<b>Community</b>	<b>sysServices</b>	<b>Type</b>	<b>ObjectID</b>	<b>DNS</b>	
	...	...	...	...	...	...	...	...	...	
<ul style="list-style-type: none"><li>Préfixe des tables (préfixe identifiant le réseau). Cf. Fonction ModificationEtatReseau page 44.</li><li>Niveau de débogage : cf. Débogage page 28</li></ul>										
Résultat :										
1. Entrées dans la base de données										

## Annexe 4 : OID et MIB utilisées

La Figure 30 ci-dessous représente l'ensemble des objets SNMP utilisés dans la librairie du même nom. Sont détaillés les OID complètes des objets (numérique et symbolique), les noms des objets utilisés, les MIB dans lesquelles on peut les trouver, ainsi que les fonctions qui s'y rapportent.

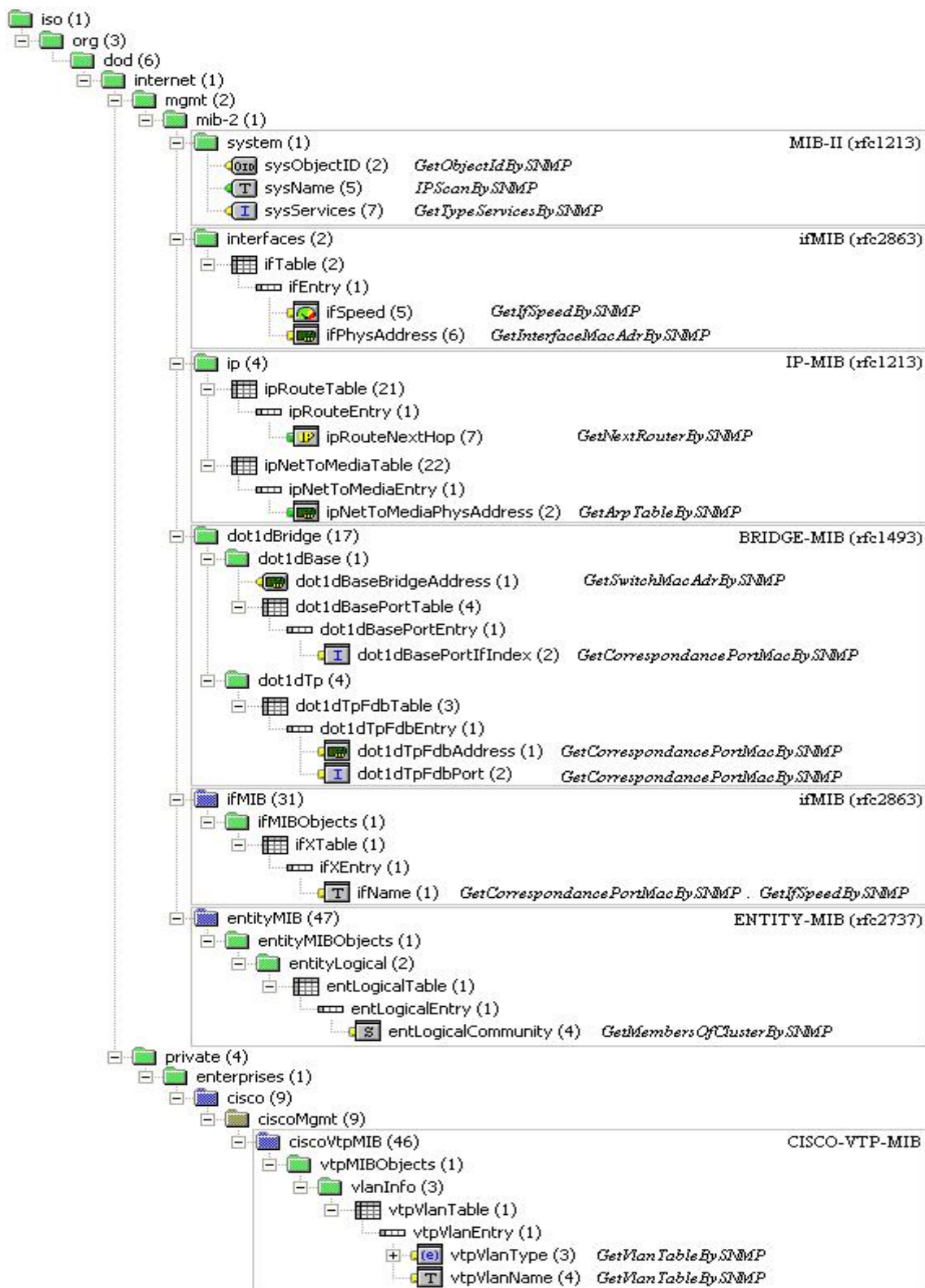


Figure 30 : Objets SNMP utilisés, OID et MIB correspondantes



## **Annexe 5 : Récupération des correspondances MAC/Ports**

La méthode explicitée ci-après est une généralisation du processus exposé dans une documentation Cisco disponibles parmi les documentations fournies avec les sources du programme, ou encore à l'adresse suivante : [http://cisco.com/en/US/tech/tk648/tk362/technologies\\_tech\\_note09186a00801c9199.shtml](http://cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a00801c9199.shtml).

Hypothèses :

- le nom de communauté en lecture est « public »,
- « @1 » correspond au VLAN 1 et est le suffixe à ajouter au nom de communauté,
- « xcisco » est le nom du nœud interrogé,
- nous recherchons l'adresse MAC '00:00:0C:07:AC:08'.

Remarque : Si nous cherchons une ou des adresses MAC sur le VLAN par défaut, il n'est pas nécessaire de préciser le VLAN dans le nom de communauté.

Instructions étape par étape :

1. Dans cet exemple, le VLAN 1 est utilisé pour obtenir la table d'adresses MAC apprises par le switch et appartenant au VLAN 1. Cette table correspond à l'objet dot1dTpFdbAddress (.1.3.6.1.2.1.17.4.3.1.1) :

```
snmpwalk -v2c -c public@1 xcisco '.1.3.6.1.2.1.17.4.3.1.1'
17.4.3.1.1.0.0.12.7.172.8 = Hex: 00 00 0C 07 AC 08
17.4.3.1.1.0.1.2.27.80.145 = Hex: 00 01 02 1B 50 91
17.4.3.1.1.0.1.3.72.77.90 = Hex: 00 01 03 48 4D 5A
...
```

2. Maintenant que nous avons trouvé l'adresse MAC cherchée, il faut trouver sur quel pont (bridge) elle est connectée. Nous récupérons donc dot1dTpFdbPort (.1.3.6.1.2.1.17.4.3.1.2) pour chercher le numéro du bridge :

```
snmpwalk -v2c -c public@1 xcisco '.1.3.6.1.2.1.17.4.3.1.2'
17.4.3.2.0.12.7.172.8 = 13
17.4.3.1.2.0.1.2.27.80.128 = 13
17.4.3.1.1.0.1.2.27.80.145 = 13
...
```

3. Une fois le numéro du pont connu pour le VLAN 1, il faut savoir à quel port il est relié, soit donc connaître l'ifIndex du port en question. Ceci s'obtient à partir de dot1dBasePortIfIndex (.1.3.6.1.2.1.17.1.4.1.2) :

```
snmpwalk -v2c -c public@1 xcisco '.1.3.6.1.2.1.17.1.4.1.2'
17.1.4.1.2.13 = 2
17.1.4.1.2.14 = 3
17.1.4.1.2.15 = 4
...
```

4. Connaissant enfin l'ifIndex du port, nous allons chercher son nom depuis l'objet ifName (.1.3.6.1.2.1.31.1.1.1.1) :

```
snmpwalk -v2c -c public@1 xcisco '.1.3.6.1.2.1.31.1.1.1.1'  
31.1.1.1.1.1 = VL1  
31.1.1.1.1.2 = Fa0/1  
31.1.1.1.1.3 = Fa0/2  
...
```

En résumé, nous avons trouvé à l'étape l'adresse MAC cherchée. L'étape deux nous permet de connaître le pont sur lequel elle apparaît, et à partir de là le port sur lequel elle se trouve grâce à l'étape trois. Enfin la dernière étape permet de connaître le nom du port : l'adresse MAC '00:00:0C:07:AC:08' a été apprise sur le port 'Fa0/1'.

Afin d'obtenir les correspondances MAC/Ports des switches, on commence par lister les ports (étape 3), on cherche en parallèle leurs noms (étape 4) et leur bridge sur lesquels ils sont connectés (étape 2) et enfin on regarde les MAC se trouvant sur ces ponts (étape 1). A chaque fois, nous récupérons d'abord les tables complètes, puis nous corrérons systématiquement les OID des objets jusqu'à trouver l'information désirée.