Projet de fin d'année
Logiciel de supervision réseau
Rapport de projet

Yves Gallen
<gallen@enseirb.fr>

Christophe Giaume
<giaume@enseirb.fr>

Samuel Guiberteau
<guiberte@enseirb.fr>

Marion Puyou
<puyou@enseirb.fr>

Matthieu Rozieres
<rozieres@enseirb.fr>

Gaël Tessier
<tessier@enseirb.fr>

ENSEIRB
2ème année informatique

27 mai 2002

Résumé

Le projet consistait à reprendre un logiciel de supervision de réseau dont les principales fonctionnalités sont la découverte de la topologie d'un réseau Ethernet à partir des informations récupérables en interrogeant les équipements réseau avec le protocole SNMP, et l'affichage graphique de cette topologie.

Table des matières

Introduction

Ce rapport est la conclusion d'un projet effectué dans le cadre de notre scolarité à l'ENSEIRB (Ecole Nationale Supérieure d'Electronique, d'Informatique et de Radiocommunicaton de Bordeaux). Il s'agit du projet de fin de deuxième année ; il dure un peu plus de trois mois pendant lesquels les cours sont poursuivis, mais de manière allégée. C'est le premier gros projet de notre scolarité, les autres – beaucoup plus liés à un enseignement précis – duraient moins d'un mois. D'autre part, c'est le premier véritable travail en équipe : les autres projets étaient réalisés en binôme alors que pour celui-ci les équipes étaient de l'ordre de 6 à 8 élèves. C'est aussi la première fois que nous avons été amenés à travailler sur un véritable projet, répondant à une attente d'un client ; tous nos précédents projets étaient des exercices à but purement pédagogique.

L'an passé, M GOUDAL, administrateur système du réseau informatique de l'école, avait proposé (également dans le cadre d'un projet de fin de deuxième année) la réalisation d'un logiciel de découverte de topologie d'un réseau Ethernet[1]. Le principe est le suivant : l'application collecte des informations sur les équipements réseau (commutateurs et routeurs) et en déduit la structure du réseau, c'est-à-dire qui est branché sur qui. La topologie est ensuite affichée dans une interface graphique. D'autres informations pratiques comme les débits peuvent ensuite être affichées sur la topologie.

Les ingénieurs systèmes ont commencé à mettre en place cette année des réseaux virtuels (VLAN au sens de la norme IEEE 802.1Q [1]). Le logiciel qui avait été réalisé l'année dernière n'était alors plus capable de fournir la topologie. Ainsi, M GOUDAL a décidé de proposer un sujet de projet de fin de deuxième année qui consistait à reprendre le résultat du projet de l'année dernière pour l'adapter à ce nouveau mode de fonctionnement du réseau. C'est ce sujet que nous avons choisi, plusieurs membres du groupe avaient déjà suivi l'évolution du projet l'année dernière et désiraient travailler dessus cette année.

Ce document est un *rapport de projet*, il ne présente que la partie déroulement du projet. La documentation technique fait l'objet de deux autres documents, *User manual* et *Maintainer manual*, qui sont livrés avec les sources.

[1] Réalisé par Samuel BRETAUDEAU, Vincent GLAUME, Baptiste MALGUY, Eric MORAND, Gilles SEGUIN et Florian TAVERNIER.

Chapitre 1

Cahier des charges

1.1 Présentation de la problématique

1.1.1 Cadre et besoins

Le réseau informatique de l'ENSEIRB est un réseau commuté de taille assez importante : il comprend environ 450 machines, serveurs, stations, PC et un grand nombre de terminaux.

C'est surtout un réseau "souple", en ce sens qu'il y a un brassage assez fréquent des branchements, pour accroître les performances du réseau avec une meilleure répartition des machines, ou parce qu'on a besoin de déplacer ou d'ajouter des machines, ou tout simplement parce qu'un équipement a une défaillance. Il devient alors très difficile de connaître l'organisation des machines.

Au cours de l'année 2001, des élèves de l'école ont implémenté un logiciel facilitant l'administration d'un réseau en calculant sa topologie et en fournissant diverses informations telles que les débits. Cette solution ne prend pas en compte la modification des branchements.

Depuis janvier 2002, le réseau a été découpé en plusieurs *VLAN*, afin d'optimiser l'utilisation des ressources et la sécurité en confinant les messages de broadcast dans des sous-réseaux.

Cette nouvelle configuration du réseau implique désormais des dysfonctionnements de la solution évoquée ci-dessus qui ne répond plus complètement aux besoins du client. De nouvelles attentes sont donc maintenant formulées : on désire notamment connaître l'appartenance d'un équipement à un VLAN. Ce sont ces considérations qui sont à l'origine de ce projet, qui a pour but la reprise du logiciel existant et son amélioration.

Pour diverses raisons, les ingénieurs systèmes peuvent être amenés à déterminer ponctuellement sur quel nœud du réseau est branché un ordinateur :

- cela permet tout d'abord de vérifier qu'une machine est effectivement rattachée au bon VLAN ;
- lorsqu'un utilisateur connecte au réseau une machine non autorisée, qui peut être détectée par son adresse IP, les administrateurs cherchent à localiser rapidement l'intruse ;
- quand certaines branches du réseaux sont saturées, il est intéressant de déterminer les machines responsables du trafic excessif ;

– un équipement peut avoir un comportement défectueux, il est alors bon d'en avertir l'administrateur du réseau.

Il est également intéressant pour les administrateurs, de suivre l'évolution de la charge du réseau dans ses différentes parties pour pouvoir apprécier les points critiques et adapter la répartition des machines en conséquence.

1.1.2 Conséquences

Il existe cependant des choix de la solution actuelle qui ne sont pas à remettre en cause (utilisation du système de gestion de bases de données MySQL et de l'API GTK pour l'interface graphique, présentation hiérarchique dans l'interface des équipements, utilisation de la bibliothèque *ucd-snmp*).

Nous devons donc reprendre la solution existante que l'on doit adapter (exigences nouvelles dues aux VLAN) et compléter (nouvelles attentes du client et nouvelles fonctionnalités). Il y a également un travail de normalisation et de structuration du code à réaliser (dans la mesure du possible).

1.1.3 Déroulement

L'environnement de déploiement est le réseau de l'ENSEIRB.

Les différents intervenants sont :
– Frédéric Goudal, le client ;
– François Pellegrini, le responsable pédagogique ;
– les réalisateurs :
Yves Gallen, Christophe Giaume, Samuel Guiberteau, Marion Puyou, Matthieu Rozieres et Gaël Tessier.

1.2 Evolutions et cycle de vie

1.2.1 Supposition fondamentale

La détermination de la topologie d'un réseau s'appuye sur l'interrogation de ses équipements de liaison. Ainsi est-il nécessaire de disposer de commutateurs administrables ou "manageable switches".

C'est une condition nécessaire mais pas suffisante : il faut également que les nœuds du réseau soient compatibles avec le protocole *SNMP*, ou plus précisément qu'ils implémentent le *BRIDGE-MIB*.

1.2.2 Evolutions prévisibles

Afin de permettre la reprise et la maintenance du logiciel, les sources et la documentation seront placés sur `http://www.sourceforge.net`, un site internet permettant le dépôt de projets libres.

Il existe un grand nombre d'équipements qui respectent ou implémentent à différents degrés les spécifications des protocoles (SNMP et *RMON*) permettant l'extraction des données.

Ceci signifie qu'une compatibilité totale n'est pas garantie avec tous les équipements, autres que ceux utilisés par le client. Il est envisageable que cette compatibilité soit apportée ultérieurement.

Il faut également préciser que nous sommes à priori limités pour nos tests et nos manipulations. En effet le seul environnement réseau utilisé sera celui de l'ENSEIRB (au moins dans un premier temps), sur lequel nous ne pouvons agir que de manière restreinte : on ne pourra pas modifier les cablages, les configurations, ou perturber son bon fonctionnment.

1.3 Besoins fonctionnels

Si certains besoins fonctionnels sont essentiels au logiciel, d'autres ne sont pas aussi importants mais restent vivement souhaités.

1.3.1 Etablissement de la topologie

Cela constitue la base du logiciel. Celle-ci doit être mise à jour régulièrement selon un intervalle de temps spécifié et mettre en évidence les correspondances entre équipements réseau et port de branchement. La topologie devra prendre en compte l'existence des VLAN.

On complète la topologie par des informations caractéristiques du réseau.

1.3.2 Récupération de données statiques

Mode de fonctionnement : Le logiciel pourra également fournir le mode de fonctionnement des différentes interfaces réseau. Ces renseignements permettront aux administrateurs d'identifier les causes de performances bien inférieures à celles attendues.

Un port peut ainsi être configuré en "full ou half duplex"; si les deux extrémités d'une liaison ne sont pas configurées de la même manière, des débits médiocres sont à prévoir...

1.3.3 Récupération de données dynamiques

Débits : La récupération des débits doit être effectuée dynamiquement sur chaque branche du réseau. Plutôt que de fournir les débits circulant dans les câbles, on préfèrera donner le rapport entre leur débit effectif et leur débit maximal.

Statistiques : Bien plus que des données instantanées, les données statistiques, moyennées sur une journée ou une semaine, sont significatives du fonctionnement du réseau.

La charge sur une branche du réseau peut varier énormément en fonction du nombre d'utilisateurs connectés, des applications s'exécutant... la charge moyenne traduit mieux le comportement du réseau : problème de configuration des ports ou topologie mal "équilibrée".

"Monitoring" d'une grandeur : Il peut être intéressant de suivre l'évolution d'un paramètre, indépendamment de ce qu'effectue le reste du programme.

1.4 Besoins non fonctionnels

1.4.1 Contraintes sur le produit

Concernant la portabilité, le programme doit impérativement fonctionner sur les systèmes Sun Solaris 7 et versions supérieures, et il serait fort souhaitable qu'il s'exécute aussi sous GNU/Linux.

La topologie du réseau et les informations complémentaires doivent être visualisées par une interface graphique. Celle-ci doit offrir d'une part une vue globale de l'arbre du réseau, et d'autre part une vue graphique centrée sur un équipement réseau. Sur cette vue, on pourra représenter certaines informations comme les débits ou les VLAN en coloriant les liens par exemple. Afin de rendre l'affichage plus lisible, on pourra également masquer certaines machines en fonction de critères comme l'appartenance aux VLAN. Les fils d'un équipement donné pourront aussi être triés suivant des critères similaires.

Dans le projet repris, l'affichage graphique ne permet de visualiser que les fils directs du boitier réseau sélectionné. Il serait souhaitable de rendre ce comportement plus souple.

Le comportement du programme doit être paramétrable par des fichiers texte de configuration. Par exemple, l'affichage à partir des données doit pouvoir se faire grâce à des méthodes génériques paramétrables.

Le projet que nous reprenons ne fonctionne qu'avec une base MySQL. Il serait souhaitable d'avoir une couche d'abstraction afin d'avoir la possibilité de stocker les informations dans un autre système de gestion de bases de données voire dans des fichiers textes.

D'autre part, il serait intéressant de pouvoir exporter certaines informations pour pouvoir les exploiter avec des scripts externes ou bien d'autres applications. La topologie devra en particulier être exportable dans un format texte. Quant aux mesures concernants les débits, un export dans un format compatible avec un grapheur externe comme RRDTOOL est envisagé.

Réciproquement, des données pourraient être importées par le logiciel pour effectuer un traitement : par exemple, chargement d'une sauvegarde.

La récupération de la liste des machines du réseau est faite par NIS dans le projet repris. Ceci pourrait être rendu plus souple en proposant un certain nombre de méthodes de collecte de listes de machines comme un transfert de zone DNS ou bien encore une interrogation LDAP. Le choix de la méthode de collecte pourrait être automatique en s'appuyant sur les fichiers de configuration du système.

1.4.2 Contraintes sur le développement

Plusieurs langages de programmation peuvent être utilisés : essentiellement le C/C++, mais aussi les langages de script sh et Perl.

Quant à la récupération des informations du réseau, elle devra s'appuyer sur le protocole SNMP.

L'utilisation de RMON[14] est envisagée pour récupérer l'historique des différents compteurs (nombre de paquets, nombre d'octets, nombre d'erreurs, etc.) afin de pouvoir grapher ces valeurs sans surcharger le réseau.

RMON tient également à jour une matrice dont les lignes et les colonnes sont des machines et dont chaque cellule contient un certain nombre de compteurs relatifs aux transferts entre les machines associées. Cette information pourra être utilisée pour la représentation des débits en particulier.

Chapitre 2

Ingénierie de la solution

2.1 Contexte de l'analyse

Nous disposions au début du projet des sources et du rapport de programmation du groupe ayant travaillé l'an dernier sur ce sujet. La première étape a été la familiarisation avec ce code.

Le code source initial était réparti dans deux répertoires, un par application :

- `treeb` : collecte des informations relatives aux équipements du réseau à l'aide du protocole SNMP, stockage de ces informations dans une base de données, construction de l'arbre de la topologie qui est lui-même stocké dans la base de données.
- `psnmp` : application graphique de présentation de la topologie (qui est lue dans la base de données), et d'interrogation dynamique (affichage des débits par exemple).

Nous allons donc décrire ici l'analyse réalisée sur ce code et les conséquences que nous en avons tirées et qui ont influencées le travail réalisé.

2.2 Code source repris

2.2.1 Premières impressions

Une des premières choses que l'on a remarquée est la duplication de plusieurs fichiers entre les deux répertoires. Malgré une partie commune relativement importante (mais pas toujours parfaitement identique), des fonctions avaient été ajoutées dans une des versions et pas dans l'autre. Visiblement, le manque de temps et de communication dans le groupe ayant travaillé sur ce projet l'année dernière est la cause de ce problème de structuration.

2.2.2 Organisation modulaire

En étudiant plus en détail le code, nous nous sommes aperçus qu'il n'y avait pas de frontière entre les différents modules. La facilité de reprise du code en souffre terriblement.

Afin de comprendre l'organisation des modules qui n'était pas documentée, nous avons utilisé l'outil *cxref* pour construire le graphe des appels entre les

différentes fonctions. Ainsi, pour chaque fonction, on sait d'une part quelles fonctions elle appelle, et d'autre part quelles fonctions l'appellent. Le rapport de cxref nous a beaucoup aidé dans l'analyse des dépendances. Une des conclusions de ce travail a été la mise en évidence de très nombreuses dépendances croisées : il n'y a pas d'organisation en couche.

Pour chaque requête SNMP, il y a un certain nombre d'opérations à faire autour de la requête elle-même si bien que le code nécessaire à une simple interrogation fait une trentaine de lignes. Les requêtes sont assez nombreuses et quasiment toutes du même type mais plutôt que de factoriser le code commun, celui-ci a été copié/collé un grand nombre de fois. Ceci pose plusieurs problèmes. Tout d'abord les fonctions perdent énormément en lisibilité. Par exemple, une simple fonction du type "récupérer A ; si A vérifie un test, récupérer B sinon récupérer C" prend quasiment cent lignes. D'autre part, les améliorations effectuées sur une interrogation au niveau du code commun ne profitent pas aux autres. Ainsi la maintenabilité est réduite.

Il y avait un problème semblable au niveau des requêtes MySQL bien que le code autour de chaque requête soit moins volumineux que celui des interrogations SNMP.

2.2.3 Qualité du code en lui-même

La qualité de la programmation était plutôt médiocre.

- Les valeurs de retour des `malloc()` n'étaient quasiment jamais testées.
- Beaucoup de choses étaient codées en dur (chemin vers des exécutables externes par exemple, avec des chemins très spécifiques à l'ENSEIRB comme `/opt/jumble`).
- Défauts de sécurité : dépassements de buffers possibles, ouverture de fichiers temporaires avec un nom constant.
- Les inclusions (`.h`) se faisaient manifestement sans réflexion mais avec des méthodes du type "ça marche pas, on en ajoute jusqu'à ce que ça compile".

Quant à la partie du code correspondant à l'interface graphique en GTK, elle avait été à l'origine développée avec un générateur de code visuel (Glade) puis retouchée à la main. Le code que génère Glade est très difficile à maintenir pour plusieurs raisons. Tout d'abord, les noms des différents composants graphiques (fenêtres, boutons, etc.) ne sont pas du tout explicites (`Windows3`, `Button34`, etc.). D'autre part, le code n'est pas divisé de manière logique : tout est groupé dans deux fichiers.

2.2.4 Synthèse

Le code initial que nous avons repris était très peu maintenable à cause de ses problèmes de structuration. Même si on peut être tenté de repartir de zéro devant un tel projet, nous avons choisi de ne pas le faire pour les raisons suivantes :

- le fait de repartir de zéro est dangereux, car il se peut que l'on perde sans le savoir des "connaissances" puisqu'il y a toujours des subtilités importantes mais difficiles à saisir lorsqu'elles ne sont pas documentées ;
- de plus, l'évaluation du temps de développement est une tâche très difficile, et il était hors de question de fournir à la fin du projet un produit moins fonctionnel que celui que nous reprenions ;

– enfin, on pouvait conserver en permanence une application qui tournait pour tester de petites améliorations.

Nous avons donc choisi de restructurer le code par étape en s'assurant de la non regression de l'application à chaque étape.

2.3 Analyse et stratégie

2.3.1 Restructuration

Le premier travail a été la fusion des fichiers communs entre les deux parties (`treeb` et `psnmp`, cf 2.2.1, page 9).

Ensuite nous avons décidé de mettre en place des scripts pour faciliter la compilation du logiciel à l'aide des outils classiques GNU `autoconf` et `automake`.

2.3.2 L'algorithme de construction de la topologie

Lors de notre analyse du code, nous avons constaté que l'algorithme de construction de la topologie était assez obscur et quasiment pas documenté. Nous avons donc décidé de l'étudier précisément afin d'en comprendre le fonctionnement. Ceci était d'autant plus nécessaire que nous avions à le modifier pour qu'il prenne en charge les réseaux virtuels. Le résultat de l'analyse de l'algorithme est présenté à l'annexe B, page 29.

Nous avons également effectué des recherches sur des algorithmes de ce type. Et nous avons trouvé un document de qualité qui s'intéressait précisément à la découverte des topologies de réseaux Ethernet : [18]. L'intérêt de ce document est qu'il présente le problème d'une manière formelle, en cherchant à le résoudre mathématiquement (avec des théorèmes démontrés). De plus, il cherche à trouver un algorithme fonctionnant le mieux possible même lorsque l'information est partielle. Ceci est très important dans le cadre de notre projet, car nous n'avons aucune garantie de disposer d'une information complète (voir A, page 27). On notera qu'il existe très peu d'informations à ce sujet ; la plupart des gens ayant travaillé sur des systèmes de découverte de topologie de réseaux se sont intéressés au cas des réseaux IP (découverte des routeurs), pour lesquels le problème est complètement différent.

Ayant alors deux algorithmes, celui du code repris d'une part – implémenté et fonctionnant *a priori* – et d'autre part celui de [18] – prouvé, nous avons analysé l'un par rapport à l'autre avec pour objectif l'amélioration de la solution implémentée dont on dispose.

2.3.3 Nouvelles fonctionnalités

Prise en charge des VLANs : La grande différence avec l'environnement de développement du projet de l'an dernier est l'utilisation des VLANs. Ce changement nous impose bien sûr de reconsidérer la définition des entités logiques du réseau, puisque cette donnée supplémentaire s'ajoute aux caractéristiques des équipements. Un de nos premiers objectifs lorsque nous avons commencé à réfléchir sur la gestion des réseaux virtuels a donc été de définir clairement la notion de réseau logique.

Ici, nous considérons qu'un réseau logique correspond à un VLAN. Une entité logique est donc un equipement réseau, plus particulièrement l'ensemble de ses

interfaces qui appartiennent à un VLAN donné. Un même équipement réseau peut donc être scindé en sous ensembles d'interfaces suivant les VLANs sur lesquels elles sont configurées.

De plus, si sur un réseau Ethernet classique nous avons la garantie qu'il n'y a pas de boucle, cela n'est plus vrai lorsque des réseaux virtuels sont définis. Nous avons alors analysé les conséquences de ceci sur l'algorithme de construction de la topologie. Aussi bien l'algorithme implémenté dans le projet de l'année dernière que celui du document [18] ne fonctionnent que si on suppose la non existence de boucle. Ainsi, nous nous sommes dirigés vers la solution suivante : les informations récupérées pour les différents VLANs sont isolées et l'algorithme est exécuté indépendamment sur chaque VLAN. Nous obtenons alors autant d'arbres que de VLANs. Pour conserver l'affichage global de l'ensemble du réseau (tous réseaux virtuels confondus), nous avons choisi alors de développer un module de fusion d'arbres. On notera que la fusion n'est pas toujours possible, dans le sens où son résultat n'est pas forcément un arbre. L'affichage d'un graphe général étant beaucoup plus délicat que celui d'un arbre, nous limiterons la fusion au cas où son résultat reste un arbre. Ainsi, on peut afficher simultanément plusieurs VLANs lorsque la fusion de leurs arbres respectifs ne crée pas de boucle.

Interface graphique : La topologie du réseau était représentée dans le projet initial de deux manières : d'un côté sous forme d'un arbre du type "explorateur", d'un autre côté sous forme d'une carte. La première représentation était tout à fait satisfaisante. Par contre la seconde était extrêmement limitée : elle n'affichait que le père et les fils directs du nœud sélectionné. Nous avons analysé la possibilité d'améliorer cette carte en affichant un nombre quelconque de machines. En effet, il est naturel de vouloir disposer de cette carte global. Un des logiciels de supervision que nous avions testé, Nagios, permettait une représentation graphique d'une topologie de réseau saisie à la main (dans des fichiers de configuration). Ce logiciel disposait de plusieurs méthodes de représentation (circulaire, en arbre, etc.) plutôt intéressantes car présentant un bon compromis entre la clarté de l'affichage et la quantité de machines à représenter. Comme il s'agit d'un logiciel libre, nous avons pu nous en inspirer pour l'implémentation du calcul des coordonnées des machines.

L'affichage graphique était très lent avec la représentation d'origine qui ne comportait que peu de machines ; nous avons donc analysé l'implémentation de l'affichage des éléments de la carte pour l'optimiser afin qu'une représentation plus complète soit possible sans décupler le temps nécessaire à son calcul.

Chapitre 3

Mise en oeuvre

3.1 Paquetage du produit

Le projet initial que nous reprenons était très loin d'un système logiciel livrable. Nous avons essayé de l'améliorer dans cette direction en le rendant plus portable, plus modulaire, et en le documentant.

3.1.1 Automatisation de la compilation

A l'origine la compilation était assez laborieuse dès qu'on sortait de l'environnement des développeurs. Afin de la simplifier, nous avons utilisé `autoconf` et `automake`. Le logiciel se compile et s'installe maintenant avec le classique `./configure && make && make install`.

3.1.2 Généralisation de l'implémentation

Données d'entrée Nous avons supprimé les spécificités au réseau actuel de l'ENSEIRB en particulier au niveau de la liste des machines à interroger et de la base initiale de correspondances entre les noms ou les IPs et les adresse MAC.

En effet, il existe différentes sources donnant accès à ces informations, mais elles ne sont pas toutes disponibles sur tous les systèmes. Par exemple, à l'ENSEIRB on peut se servir de la base NIS pour avoir d'une part la liste des machines du réseau et d'autre part leurs adresses physiques (MAC). A l'origine, la récupération était codée en dur dans le logiciel qui n'était pas utilisable sans modification sur un réseau où les tables NIS correspondantes ne sont pas présentes.

Pour permettre une utilisation de notre logiciel sur le plus grand nombre de réseau, nous avons choisi de fournir ces listes indispensables au fonctionnement du logiciel dans des fichiers dont les chemins sont passés sur la ligne de commande. Nous avons écrit des scripts `sh/perl` pour créer ces listes dans le cas de la base `NIS` de l'ENSEIRB. Nous avons aussi écrit des scripts semblables utilisables sur n'importe quel réseau faisant tourner `arpwatch`[1] Les deux jeux de scripts sont utilisables à l'ENSEIRB puisqu'`arpwatch` y tourne également.

[1] `arpwatch` est un logiciel basé sur la `libpcap` capturant tous les paquets réseaux afin de tenir à jour une table des correspondances entre adresses MAC et adresses IP ; cette table est enregistrée dans un format texte simple.

Nous avons choisi d'écrire des scripts `sh/perl` car ils sont bien adaptés pour ce travail. De plus, un utilisateur quelconque peut les modifier facilement pour les adapter à son réseau sans avoir à plonger dans le code de notre application.

Modularité Comme nous l'avons décrit dans l'analyse, le code repris présentait de graves problèmes de structuration qui nuisaient à la maintenabilité et à la lisibilité, nous avons donc essayé d'y remédier.

La quasi totalité du logiciel a été reprise afin de l'organiser en modules *logiques*. Cette refonte a été effectuée de manière incrémentielle afin de pouvoir faire des tests de non regression à chaque étape. Nous nous étions posé la question de l'éventuel intérêt de refaire complètement certaines parties à partir de zéro après l'analyse du code. Nous n'avions pas choisi cette possibilité en particulier parce qu'on risquait de ne pas pouvoir finir une application avec au moins autant de fonctionnalités que celle d'origine ; compte-tenu des gros dépassements sur nos prévisions de départ en particulier pour ce qui concerne la prise en charge des réseaux virtuels, ce choix s'est avéré judicieux.

3.1.3 Documentation du produit

Nous avons documenté le logiciel et pour l'utilisateur et pour le mainteneur. Ces deux documentations ont été rédigées en anglais (de même que l'ensemble des commentaires des sources et des noms de variable) afin que l'utilisation et l'évolution du logiciel ne soit pas limitée aux francophones ; ceci nous semblait assez important dans l'optique logiciel libre, surtout qu'il n'existe quasiment pas d'autres logiciels libres avec ses fonctionnalités.

Pour l'utilisateur Un *user manual* a été rédigé pour expliquer les fonctionnalités de l'application. Il donne les prérequis et explique comment compiler et utiliser les programmes c'est à dire l'application de collecte et l'interface graphique.

Pour le mainteneur D'autre part, nous avons rédigé un *maintainer manual* à l'usage des futures personnes qui pourraient être amenés à effectuer des développements sur l'application. Il décrit l'organisation des sources (la logique du découpage de nos modules). Il présente aussi certains points délicats comme l'algorithme de fusion des arbres ou l'heuristique de ventilation des *forwarding lists* par réseau virtuel.

Nous avons aussi documenté précisément la partie sur la récupération des informations concernant la configuration des réseaux virtuels ; en effet la localisation de ces informations nous a pris beaucoup de temps et mérite un récapitulatif.

3.2 Prise en charge VLANs

Manque de standardisation Si les informations générales que l'on peut récupérer par SNMP comme la liste des interfaces, leurs adresses, etc. sont très standards, les choses se gâtent un peu pour ce qui concerne les VLANs.

Les informations récupérables par SNMP sont organisées de façon arborescente. Chaque noeud de l'arborescence contient une valeur (qui peut être

de différents types). Les RFCs définissent la signification des différents nœuds. Cependant, les informations relatives aux VLANs ne sont pas toujours implémentées de la manière décrite dans la RFC correspondante [13].

En conséquence, il y a eu un travail de recherche de la localisation des informations assez important car il fallait le faire pour chaque type d'équipement. Une fois que l'on savait où trouver les différents éléments de la configuration des réseaux virtuels pour chaque équipement, nous avons implémenté leur récupération de la manière suivante : on commence par récupérer le type d'équipement (une chaîne décrivant l'équipement est récupérable par SNMP, elle comporte en général le constructeur et la référence de la machine), ensuite suivant le type de l'équipement, on récupère l'information à la bonne "adresse".

Comme il s'agit d'un élément du logiciel qui a de grande chance de poser des problèmes lors de l'utilisation d'équipements différents de ceux utilisés à l'ENSEIRB actuellement, nous avons veillé à bien documenter ceci pour que des tiers puissent facilement adapter le logiciel à d'autres marques ou modèles de commutateurs.

Défaut d'analyse Un autre point intéressant pédagogiquement parlant a été une mauvaise analyse de notre part des informations à notre disposition. Sur certains équipements, les *forwarding lists* sont récupérées ventilées par réseau virtuel. Sur d'autres elles sont récupérés globalement mais on peut aussi récupérer la configuration du matériel (c'est-à-dire à quel VLAN appartient chaque port, et sur quel VLAN il taggue les paquets non taggués provenant d'un port donné). Nous pensions que cette information était suffisante pour ventiler les différentes adresses des *forwarding lists* mais en réalité cela oblige à mettre en place une heuristique assez délicate : il faut savoir où les paquets sont taggués, autrement dit descendre dans la topologie mais celle-ci n'est pas encore construite puisque c'est précisément pendant sa construction que l'information est nécessaire.

On notera par ailleurs que le client n'avait pas non plus réalisé qu'il y avait un problème à ce niveau là. En fait nous pensons que nous nous sommes laissés abuser par le fait que la méthode permettant de récupérer les *forwarding lists* ventilés par VLAN n'était pas standard (c'est une méthode spécifique au constructeur 3Com). Aussi, pour nous, c'est inconsciemment la méthode normalisée (celle conforme aux RFCs) qui était la plus judicieuse...

3.3 Amélioration de l'interface

Nous nous sommes inspirés de l'implémentation des méthodes d'affichage de Nagios [20] d'arbres réseau sur plusieurs niveaux, dans le but de rester le plus compatible possible avec les versions futures de ce logiciel afin de profiter des corrections de bugs/améliorations dans leurs algorithmes de placement des machines.

Outre le rajout de menus pour appeler les fonctionnalités, nous avons réglé quelques problèmes d'affichage comme des scintillements dus à une méthode de rafraichissement peu judicieuse (lecture d'un fichier configurant une couleur pour chaque lien). De plus, un réaffichage était réalisé à chaque ajout sur la représentation graphique du réseau, ce qui ralentit l'affichage.

3.4 Autres améliorations

Quant à l'algorithme de topologie, nous l'avons amélioré après notre analyse du document [18] ; d'autre part nous avons pu voir grâce à cette étude qu'il n'existait pas de solution miracle. Certains cas mal traités dans l'algorithme d'origine ont été mis en évidence par les exemples de cas fournis dans le document. En outre, il propose un algorithme complet ; cependant celui-ci est difficile à implémenter et nous avons donc préféré améliorer celui qui existait et qui était assez proche de celui-ci plutôt que de le jeter.

D'autre part, un certain nombre de bugs mineurs comme des fuites mémoires ont été corrigés.

Il existait plusieurs fichiers de configuration pour lesquels le groupe de l'année dernière avait écrit un parser à la main. Celui-ci marchait moyennement ; nous avons donc écrit un parser de fichier de configuration avec `yacc` et `lex` afin de faciliter le paramétrage de l'application et surtout l'ajout de nouvelles directives de configuration.

Il fonctionne maintenant sur les plateformes suivantes :

- Solaris/Sparc
- Linux/x86
- FreeBSD/x86
- NetBSD/x86
- OpenBSD/x86
- OpenBSD/Sparc

Chapitre 4

Bilan

4.1 Méthodes de travail

Ce projet est un travail d'équipe pour lequel nous étions six. Pour la plupart d'entre nous, c'était la première fois que nous étions amenés à travailler dans un groupe non réduit à un binôme. Dès le départ, nous avons effectué un certain nombre de choix pour nous faciliter le travail en équipe :

- mise en place d'une liste de diffusion (*mailing list*) à usage interne.
- création du projet sur SourceForge [15] en particulier pour le service CVS qu'il propose.
- nomination d'un chef de projet qui fixe les objectifs, les étapes et les moyens, qui s'assure de l'harmonisation des liens inter-unités, et qui coordonne les actions.
- nomination d'un responsable communication qui s'assure que l'information circule correctement entre les différents membres de l'équipe, et qui a aussi en charge la rédaction de compte-rendus internes après chaque réunion (qu'elle soit interne, ou bien avec le client ou le responsable pédagogique).

CVS Afin d'éviter de perdre du temps sur le transfert des modifications des sources entre nous, nous avons donc utilisé le système CVS de partage des sources. Ces dernières étaient ainsi partagées sur une base commune (hébergée sur les serveurs SourceForge auxquels nous accédions avec `ssh`) appelée le *repository*. Chaque membre de l'équipe crée ensuite une copie privée des sources (opération *checkout*) sur laquelle il effectue son travail puis le diffuse en mettant à jour la base commune (opération *commit*). Les modifications de la base commune génèrent automatiquement un courrier électronique envoyé à tous les membres du groupe qui savent ainsi ce que les autres ont modifié sur les sources ; ce message comprends un rapide descriptif de l'objet de la mise à jour saisi par le développeur ainsi qu'un listing des différences induites par la modification (résultat d'un `diff -u`). Les autres développeurs peuvent ensuite, s'ils le souhaitent, mettre à jour leur copie privée (opération *update*). On notera que CVS gère très bien la modification simultanée du même fichier par plusieurs personnes (à condition qu'elles touchent des parties différentes du fichier bien sûr). En outre CVS archive toutes les versions de chaque fichier et il est ainsi possible

de récupérer les versions antérieures, d'afficher un historique des modifications d'un fichier, etc.

Bilan Nous n'avons pas regretté nos choix quant à nos méthodes de travail, bien au contraire. Il est d'ailleurs intéressant de constater que ces méthodes qui manquaient au groupe de l'année dernière nous ont permis d'éviter certaines erreurs qu'ils ont commises :

- Le fait d'avoir nommé un coordinateur nous a permis de faire des modules ayant une interface cohérente. Les modules de l'an passé étaient assez incohérents, sans doute parce que la logique de leur organisation et de leur convention (nommage, prototypage, etc.) était pensée par la première personne ayant besoin de ce module ; puis cette logique n'était pas communiquée aux autres membres. Lorsque d'autres avaient besoin d'une fonction manquante dans le module par exemple, il l'ajoutait sans chercher à la rendre cohérente avec l'existant.
- L'absence d'outil de partage des sources obligeait chacun à travailler de manière isolée ; la démarche pour savoir ce que les autres ont modifié dans le source était coûteuse en temps et donc rédibitoire. Avec le CVS, chacun est informé de toutes les modifications des source, et peut donc ensuite y jeter très facilement un œil, tout simplement en lisant les messages électroniques envoyés par le serveur à chaque *commit*. Souvent des bugs mineurs ont été ainsi découverts lors de la relecture par les autres membres. Il y a un exemple flagrant d'aberration qui aurait été évitée si le groupe de l'année dernière avait utilisé un outil de partage des sources. L'application est constituée de deux exécutables distincts mais qui utilisent des fonctions communes ; le source était donc à l'origine divisé en trois morceaux, la partie commune, la partie spécifique au premier exécutable et la partie spécifique au second exécutable. Dans l'urgence de la fin du projet, certains devaient ajouter des fonctions dans les modules communs mais comme ceux-ci étaient aussi en cours de modification par les autres, ils ont dupliqué les fichiers communs pour ajouter dans leur propre copie les routines manquantes. Et finalement, la fusion des modifications effectuée sur ces fichiers communs n'a pas eu lieu.

4.2 Répartition des tâches

La répartition des tâches a beaucoup évoluée au cours de la réalisation du projet. En effet, les divers problèmes que nous avons rencontrés obligeaient parfois à revoir les affectations des tâches. En particulier, la toute première répartition que nous avons effectuée s'est rapidement avérée complètement utopique : elle prévoyait un travail parallèle sur les nouvelles fonctionnalités que nous souhaitions ajouter à l'application : gestion des VLANs, création de systèmes de vues, etc. Si elle s'est avérée utopique, c'est parce que nous n'avions pas vraiment étudié le code avant de la faire, et le résultat de cette étude a mis en évidence de graves problèmes dans le projet repris. En effet, l'ensemble manquait énormément de cohérence globale. Et la logique que l'on pouvait espérer trouver dans l'organisation modulaire n'existait pas.

Ensuite, nous avons donc effectué une répartition différente prenant en compte de nouvelles tâches relatives à la réorganisation du code. Pendant quelques se-

maines, le travail a été d'une part la recherche des informations sur les VLANs pour préparer leur implémentation, et d'autre part la restructuration du code. Au cours de cette période, nous nous sommes familiarisés avec les sources.

Enfin, sur la dernière partie du projet, nous avons eu une répartition stable pour le développement des nouvelles fonctionnalités.

Coordination technique : Christophe GIAUME.

Communication et documentation : Yves GALLEN.

Prise en charge des VLANs : Gaël TESSIER et Yves GALLEN.

Algorithme de fusion (pour les VLANs) : Samuel GUIBERTEAU et Marion PUYOU.

Algorithme de topologie : Gaël TESSIER.

Interface graphique : Matthieu ROZIERES.

4.3 Valeur ajoutée

En reprenant la terminologie de Brooks dans un des premiers chapitres de [17], ce que nous avons repris ressemblait à un *logiciel*, alors que ce que nous fournissons s'apparente plus à un *système logiciel livrable*. En effet, pour ce qui concerne l'aspect *logiciel livrable* :

- Le logiciel a été documenté ; nous avons rédigé deux documentations techniques : le *User manual* et le *Maintainer manual*.
- Faute d'avoir accès à d'autres réseaux, les tests se sont limités au réseau de l'ENSEIRB. Cependant, nous avons gardé à l'esprit que l'objectif n'était pas de se contenter de faire un logiciel fonctionnant correctement sur le réseau actuel de l'école.
- Nous avons beaucoup amélioré la portabilité du logiciel. A l'origine, il ne fonctionnait correctement que sous Solaris 7/Sparc. Il tourne désormais également sous GNU/Linux et sous les *BSD.

Quant à l'aspect *système logiciel*, lors de notre travail de structuration du code, nous nous sommes efforcés de rendre l'ensemble le plus modulaire possible. Par exemple l'interface actuelle est en GTK, mais tout ce qui est relatif à l'interface sans être dépendant de GTK comme le calcul des coordonnées des machines fait l'objet d'un module à part. Ainsi, tout est prêt pour la création d'une interface avec un autre *toolkit*, QT par exemple.

4.4 Avancement

Au niveau de l'interface utilisateur,, un affichage sur plusieurs niveaux de profondeur a été réalisée avec la possibilité d'afficher tel quel l'arbre de topologie liée à un VLAN ou de le fusionner avec d'autres. Plusieurs méthodes de dessin ont été mise à disponibilité. Cependant, l'implémentation de l'affichage de la charge sur les liens du réseau ainsi que la coloration de ceux-ci n'ont pas pu être réalisés faute de temps.

4.5 Perspectives

On peut raisonnablement espérer que ce logiciel sera repris par la suite, étant donné qu'il propose des fonctionnalités peu répandues, et qu'il a été mis sur SourceForge [15].

La modularité introduite permet l'ajout de fonctions au niveau de l'interface et de la récupération. Néanmoins, la qualité de programmation de l'interface n'a pas été privilégiée, et un certain travail reste à faire. L'utilisation de Glade dans le projet repris a produit un code lourd et peu agréable à réutiliser. Nous ne nous sommes pas attardé sur ce point, car nous préfèrerions travailler sur la modularité des API de récupération de données.

De plus, l'utilisation de containers GTK pour l'affichage de la topologie est peu adaptée, et il serait préférable d'utiliser des zones de dessins (bien plus rapide lors de l'affichage).

Le programme repris s'appuyait complètement sur la base de donnée, ce qui est assez lourd en terme de communication. Après avoir étudié le volume d'informations de la base, nous avons remarqué qu'il serait plus judicieux de manipuler en mémoire les données recueillies, la base ne servant que d'appui pour le stockage, voire la remplacer par un systeme de sauvegarde dans un format de fichier.

Pour finir, notre programme se limite à l'étude d'un réseau découpé en VLAN dans lequel reseau IP et VLAN sont confondus. Cette distinction supplémentaire permettrait de lever de nombreuses ambiguités concernant le traitement des routeurs IP dans notre algorithme de topologie. Il est a noté que le logiciel Nomad [19] propose cette fonctionnalité.

4.6 Remerciement

Nous tenons à remercier M Goudal et M Pellegrini pour l'aide et les conseils qu'ils nous ont apportés tout au long du projet.

Conclusion

Ce projet était, comme nous l'avions précisé en introduction, le premier sur une période relativement longue d'une part, et le premier réalisé au service d'un client d'autre part. Pour ces raisons là, ce projet était bien plus intéressant et plus motivant que les autres puisque nous n'avions pas l'impression de travailler uniquement pour nous.

Aussi, nous avons pû constaté certaines réalités du génie logiciel : tout d'abord l'évaluation des temps de développement est très difficile et souvent optimiste. Par ailleurs, nous avons pû nous rendre compte à quel point le coût de la communication interne était élevée mais indispensable. Enfin, le temps de codage se révèle être très minime globalement par rapport aux autre tâches : planification, analyse, tests unitaires et tests d'intégration, dès que l'on veut faire un produit fini.

L'an passé, le logiciel avait été mis sous license GPL [16] mais il n'avait pas été diffusé. La première chose que nous avons fait a été de mettre le projet sur SourceForge [15]. Au cours du projet, nous avons été contacté par des gens qui étaient intéressés par ce logiciel et qui tentaient de l'utiliser (avec difficulté puisque la version initiale n'était pas du tout adaptée à une utilisation sur un réseau différent de celui de l'ENSEIRB) ; nous les avons invités à réessayer le logiciel à la fin de notre projet. Psnmp est un des premiers logiciels libres permettant la découverte de la topologie d'un réseau Ethernet.

Certains d'entre nous comptent maintenir le logiciel en le faisant évoluer en fonction des retours que nous auront des utilisateurs et en y ajoutant quelques fonctions manquantes qui pourraient être intéressantes.

Bibliographie

[1] Norme IEEE 802 *Local Area Network standards and Metropolitan Area Network standards (Ethernet, Token Ring, Wireless LAN, Bridging and Virtual Bridged LANs, etc.)*
`http ://www.ieee802.org/`

[2] RFC 1155 *Structure and Identification of Management Information for TCP/IP-based Internets*

[3] RFC 1156 *Management Information Base for Network Management of TCP/IP-based internets*

[4] RFC 1157 *A Simple Network Management Protocol (SNMP)*

[5] RFC 1212 *Concise MIB Definitions*

[6] RFC 1213 *Management Information Base for Network Management of TCP/IP-based internets : MIB-II*

[7] RFC 1442 *Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)*

[8] RFC 1493 *Definitions of Managed Objects for Bridges*

[9] RFC 1565 *Network Services Monitoring MIB*

[10] RFC 1573 *Evolution of the Interfaces Group of MIB-II*

[11] RFC 1643 *Definitions of Managed Objects for the Ethernet-like Interface Types*

[12] RFC 1905 *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*

[13] RFC 2674 *Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions* définition de la `QBRIDGE-MIB`

[14] RFC 2819 *Remote Network Monitoring Management Information Base*

[15] SourceForge : Site offrant des services d'aide au travail collaboratif sur les projets Open Source.
`http ://www.sourceforge.net/`

[16] The GNU General Public License (GPL).
`http ://www.gnu.org/licenses/gpl.html`

[17] *Le mythe du mois-homme, essais sur le génie logiciel*, Fredericks P. Brooks

[18] *Topology Discovery for Large Ethernet Networks*, Bruce Lowekamp, David R. O'Hallaron, Thomas R. Gross.
`http ://www.cs.wm.edu/ lowekamp/papers/lowekamp-topology-sigcomm01.pdf`

[19] *Nomad, a Network Monitoring and Mapping program*
http ://netmon.ncl.ac.uk/

[20] *Nagios, successor of NetSaint*
http ://www.nagios.org/

Glossaire

adresse IP (Internet Protocol) : C'est une addresse 32 bits qui est assignée aux machines qui utilisent le protocole IP par l'administrateur système. Les 32 bits contiennent un identifiant de réseau, et un identifiant de machine sur ce réseau, la répartition des 32 bits d'information étant variable suivant les classes d'adresses. C'est une adresse logique contrairement a l'adresse MAC qui est une adresse physique.

adresse MAC (Media Access Control) : Une adresse MAC est l'adresse physique d'une interface réseau, fixée par le constructeur et permet d'identifier de façon unique une machine sur un réseau local. Cette adresse MAC est utilisée dans la couche liaison de données d'un réseau.

Glade Interface pour la bibliothèque GTK permettant de dessiner des interfaces grahiques.

GTK (Gimp ToolKit) : GTK est un ensemble d'outils multiplateforme pour la création d'interfaces graphiques utilisateur et est à l'origine conçu pour le système de gestion de fenêtres X.

NIS (Network Information Services) : Network Information Services. Services d'information sur le réseau. Services donnant accès à des bases de données de réseau fournissant par exemples des adresses IP, Ethernet, des mots de passe ou des noms de serveur. Aussi connu sous le nom de Yellow Pages .

OSI (Open Systems Interconnect) C'est un modèle de référence qui spécifie les différentes couches au sein d'un réseau. Il a été défini par l'International Standards Organization.

OUI (Organizationally Unique Identifier) : Idendifiant codé sur 3 octets qui identifie de façon unique un constructeur.

Qt Boite à outils pour le développement d'interfaces graphiques. Il est également multiplateformes (Comme GTK).

Routage IP C'est la faculté pour une machine qui possède plusieurs interfaces d'envoyer les paquets IP reçus vers leur destination via l'interface la plus adaptée.

RPC (Remote Procedure Call) : C'est un appel qui permet l'exécution d'une procédure sur une machine distante.

SGBD (Système de gestion de base de données) : Un ensemble de programmes permettant aux utilisateurs de créer et d'utiliser des bases de données. Les activités supportées sont :
- la definition d'une BD, spécification des types de données à stocker.

- la construction d'une BD, stockage des donnees proprement dites.
- la manipulation de données, en particulier ajouter, supprimer et retrouver des données.

SNMP (Simple Network Management Protocol) : Protocole standard qui spécifie un format pour la collecte des données de gestion de réseau. Les services SNMP autorisent les noeuds de réseau à envoyer des informations de statut à un programme de gestion SNMP fonctionnant sur un réseau IPX ou TCP/IP. Ces services peuvent être gérés grâce à des outil standard de gestion SNMP faisant partie du système d'exploitation réseau ou grâce à d'autres systèmes de gestion de fournisseurs tiers.

Index

Annexe A

Description du logiciel

L'information de la topologie des réseaux locaux est très intéressante pour leur gestion (pour éviter les problèmes de congestion en repérant les goulots d'étranglement par exemple) ; cependant celle-ci est difficile à obtenir.

La plupart des outils relatifs à la topologie des réseaux s'appuient sur une découverte au niveau IP qui ignorent donc les équipements de niveau 2 (OSI).

Les fabricants de matériel réseau comme Cisco fournissent des logiciels permettant la découverte au niveau Ethernet mais ceux-ci sont difficilement utilisables dans un environnement hétérogène. Il existe aussi des logiciels commerciaux qui *a priori* sont capables de découvrir la topologie d'un réseau local mais ceux-ci ne sont bien sûr pas Open Source, et ils coûtent assez cher. Psnmp est donc un des premiers logiciels libres (le premier ?) capable d'effectuer cette tâche.

A.1 Découverte de la topologie

La complexité de la découverte de la topologie d'un réseau Ethernet vient de la transparence intrinsèque aux commutateurs : les différentes machines branchées au réseau ignorent que celui-ci comporte des commutateurs (*hubs* ou *switchs*).

Chaque machine envoie ses paquets sur son interface réseau ; le paquet comporte les adresses physiques de l'expéditeur et du destinataire. Les interfaces réseaux ignorent les paquets qui arrivent avec une adresse de destination qui n'est pas la leur. Ainsi, le fonctionnement le plus simple est celui du *hub* : il s'agit d'un boitier comportant un certain nombre de ports ; tout ce qui est reçu sur un port est retransmis sur tous les autres. Comme cette méthode n'est pas très efficace, la plupart du temps on utilise des *switchs* ; ceux-ci diffèrent des *hubs* parce qu'ils tentent d'envoyer les paquets que sur le bon port. Pour cela, ils tiennent à jour une table du type {adresse physique, port}. Lorsqu'il doit transmettre un paquet dont l'adresse est dans sa table, il l'envoie uniquement sur le port associé ; sinon il se comporte comme un *hub*, c'est-à-dire qu'il l'envoie sur tous les ports. La table est mise à jour à chaque paquet transitant par l'équipement qui en extrait les adresses. Cette table est un cache : toutes les informations sont datées et supprimées au bout d'un certain temps. La liste des adresse physiques correspondant à un port dans cette table est appelée *forwar-*

ding list.

Grâce au protocole SNMP [4], nous pouvons récupérer ces *forwarding lists* d'une manière standard (dans la *BRIDGE-MIB* [8]). A partir de là on sait sur quel port chaque équipement réseau "voit" une machine. En faisant des recoupements entre ces informations sur les différents commutateurs, on peut en déduire la topologie du réseau. On remarquera que les *forwarding lists* étant des caches, on ne peut en aucun cas considérer que ces listes sont complètes.

A.2 Notre application

On récupère des informations sur les *forwarding lists* des équipements du réseau, on peut déterminer la topologie du réseau grâce à un algorithme spécifique.

Une fois cette topologie déterminée, on utilise une interface graphique pour présenter le résultat à l'utilisateur.

Annexe B

Algorithme de la solution reprise

B.1 Description du fonctionnement

L'algorithme de construction suppose que le réseau dont on veut déterminer la topologie ne comporte pas de boucle. Pour cela, il effectue si nécessaire une scission des équipements responsables du problème.

Vient ensuite la construction de l'arbre représentant le réseau à partir des informations obtenues. Il s'agit de déterminer les liaisons directes entre équipements, du type `nœud1-port1` vers `nœud2-port2`. Les liaisons entre équipements ne sont normalement pas orientées et le sont juste pendant la construction de l'arbre pour les besoins de l'algorithme.

Dans un premier temps, seuls les nœuds internes de l'arbre sont pris en compte pour la construction, puis les équipements de type station qui sont les feuilles de l'arbre sont finalement placées. Ceci permet de réduire, souvent de manière significative, le nombre d'équipements lors de la construction de l'arbre.

B.2 Placement des équipements réseaux

Tout les équipements manipulés ici sont des nœuds internes de l'arbre, et on emploiera le terme *noeud* pour les désigner.

L'algorithme commence pqr positionner tous les équipements dans l'état *non testé*. Puis une racine est choisie pour l'arbre : il s'agit de la machine depuis laquelle est lancé le programme. C'est depuis cette racine qu'est ensuite lancée la construction.

D'un point de vue global, l'algorithme de construction place des équipements dans l'arbre puis s'appelle récursivement sur les fils directs du nœud sur lequel il vient d'être appelé, après modification de l'arbre due aux traitements effectués.

Plus précisément, il est appelé sur un équipement donné que l'on nomme N, et procède par étapes :

1. L'algorithme récupère la liste des équipements distants, visibles par le nœud N ou voyant N sur l'une de leurs interfaces, et se trouvant encore dans l'état non testé. Le nœud N, ainsi que tous les nœuds de la liste, sont mis dans l'état *testé*.
2. Pour chacun d'eux, il lance la procédure de placement dans l'arbre en cours de construction, grâce à la routine `host_place_son` décrite par la suite. L'arborescence est donc modifiée par l'insertion de nouveaux équipements.
3. Pour finir, l'algorithme est appelé récursivement sur les fils directs du nœud N dans l'arbre actuel après modification de l'arbre due aux traitements effectués lors de l'appel sur le nœud N.

La routine `host_place_son` prend en paramètre un nœud N et un autre nœud F et se charge de placer ce dernier dans l'arborescence enracinée en N.

1. Elle commence par récupérer la liste des fils directs de N, liés au port sur lequel est visible F si ce port est connu, sur tous les ports de N sinon. Cette liste peut éventuellement être vide.
2. Pour chacun des équipements F_i de la liste, la fonction tente de déterminer qui de F_i ou de F est le plus proche du nœud N. Pour cela, elle fait appel à la routine essentielle `more_direct_son`.
   - S'il s'agit de F, on met à jour la liaison `(N -> F`$_i$`)` en la remplaçant par `(F -> F`$_i$`)` et on crée une liaison `(N -> F)`.
   - S'il s'agit de F_i, alors F se trouve dans le sous-arbre enraciné en F_i et on appelle donc la fonction `host_place_son` avec pour arguments F_i et F.

     Ici, on positionne un marqueur permettant de s'assurer que l'on ne rencontre pas deux nœuds F_i qui sont plus proches de N que F : ceci est bien sûr impossible, car cela signifierait que F a deux pères alors que l'on s'est assuré d'avoir un arbre et non un graphe. Si malgré tout, on se retrouve dans ce cas avec le marqueur positionné, on n'effectue alors aucun traitement.
   - Si les informations dont on dispose sont insuffisantes pour déterminer les positions relatives des deux fils, on essaye résoudre le problème avec les ancêtres successifs de N. Pour finir, on effectue l'un des deux traitements précédents selon le cas, à moins que l'on n'ait toujours pas réussi à placer F par rapport à F_i, auquel cas on ne fait rien.
3. Finalement, si l'on n'a trouvé aucun fils F_i plus proche de N que F, alors F est placé comme fils direct du nœud N, en précisant les ports de la liaison s'ils sont connus. Une liaison `(N -> F)` est donc ajoutée à l'arbre.

En ce qui concerne le positionnement relatif de deux fils d'un même nœud, la fonction `more_direct_son` regarde sur quels ports chaque fils voit son père et l'autre fils. Si pour l'un deux les ports sont différents, alors cet équipements se trouve entre les deux autres dans l'arbre.

B.3 Placement des feuilles

Ce placement est plus aisé que le précédent et nettement moins critique : en effet, une erreur dans le placement d'une feuille n'affecte que cette dernière.

Une fois l'ossature de l'arbre réalisée, on récupère l'ensemble des nœuds internes de l'arbre, et on place les feuilles de chacun d'eux grâce à la fonction `host_place_all_leaf`.

Cette fonction prend pour seul paramètre le nœud N pour lequel elle doit placer les feuilles.

1. Elle récupère ensuite la liste de ses ports sur lesquels il voit d'autres nœuds de l'arbre, et la liste des feuilles visibles par N ou voyant N sur une de leurs interfaces.
2. Pour chacune des feuilles récupérées, on vérifie qu'elle est vue sur un port libre, et si c'est le cas on crée une liaison directe entre N et cette feuille.

Annexe C

Autres logiciels

C.1 Nagios (anciennement Netsaint)

Il semble que ce logiciel [20] soit l'un des plus aboutis en matière de supervision des réseaux IP, en tout cas en ce qui concerne les logiciels libres.

C.2 Le logiciel Nomad

C'est un des rares logiciels [19] de ce type que l'on a étudié en détail car il semblait proposer une solution au problème des VLANs.

Nomad est un logiciel de supervison réseau assez similaire au projet sur lequel on a travaillé. Nous l'avons trouvé que très tard mais nous regardé ce qu'il proposait comme fonctionnalité. Il travaille au niveau IP et Ethernet. La documentation précisait que l'application gérait les VLAN. Malheureusement, cette gestion est minimaliste.

- le critère d'un détection de VLAN est la présence du mot VLAN dans la chaine de descriptiton d'une interface (ce qui n'est pas fiable du tout ; par exemple aucune des interfaces des SuperStack du réseau de l'école ne réponde ce critère car les notions d'interface et de VLAN sont dissociées sur ces switchs).
- les OID utilisés sont les mêmes que les notres, nous n'avons rien apris de ce côté la.

La version étudiée succinctement était la première proposée depuis 8 mois, et la première à aborder le problèmes des VLANs. L'approche du problème de topologie réseau nous a cependant semblé interessante : distinction réseau IP Ethernet, le découpage logique des machines, etc.