UTILISATION DU PROTOCOLE SNMP POUR LA GESTION À DISTANCE D'UNE INTERFACE RADIO PAR PAQUETS

par

Haithem Hmida

mémoire présenté au Département de mathématiques
et d'informatique en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, février 1998

Sommaire

Ce mémoire aborde le problème de la gestion à distance d'un dispositif particulier de télécommunication appelé *Terminal Node Controller* (TNC). Ce dernier permet de transformer les paquets de données provenant de l'ordinateur auquel il est attaché en un signal audio qui est transmis à un radio branché au TNC (et vice versa).

La gestion à distance aide les analystes des réseaux de télécommunications à contrôler les dispositifs qui en font partie sans être obligés de se déplacer pour être à proximité de ces dispositifs. Jusqu'à maintenant, il n'existait aucun moyen pour effectuer la gestion d'un TNC à distance.

Dans le cadre de ce mémoire, nous avons développé un outil logiciel qui permet la gestion à distance d'un tel dispositif. Cet outil permet, entre autres, de contrôler et de gérer simultanément différents paramètres TNC à distance. En plus, nous avons développé une base de données particulière, nommée *Management Information Base* (MIB). Les objets du MIB sont des abstractions des paramètres du TNC à gérer. La lecture et la modification de ces paramètres se fait via les objets du MIB. L'implantation de cet outil est faite sous le système d'exploitation Linux et utilise les services d'un outil, nommé CMU-SNMP, spécifique au développement des applications de gestion des réseaux.

Remerciements

D'abord, je tiens à remercier les membres de ma famille, particulièrement mes parents ainsi que mes deux grands frères, pour tout le soutien et toute l'aide qu'ils m'ont apportés durant ces deux dernières années.

Mes remerciements les plus distingués sont aussi accordés à mon directeur de recherche, le professeur Michel Barbeau, qui m'a beaucoup aidé et guidé tout au long de ma recherche. L'environnement de recherche qu'il m'avait offert et son suivi m'ont beaucoup aidé à poursuivre mes travaux de recherche et finaliser ce mémoire au bon moment.

Je tiens à remercier également les membres du jury qui ont accepté d'évaluer mon mémoire. Leurs observations et commentaires me furent très précieux.

Un grand merci aussi à tous mes professeurs et mes collègues du Département de mathématiques et d'informatique de l'Université de Sherbrooke, avec qui j'ai passé beaucoup de bons moments et complété ma formation universitaire.

Cette maîtrise est subventionnée par le Conseil de recherche en sciences naturelles et en génie du canada (CRSNG), ainsi que le Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR).

Table des matières

Liste des tableaux

Liste des figures

Introduction

La gestion d'un réseau est définie comme étant le processus de contrôle d'un réseau de données de façon à maximiser son efficacité et sa productivité [13]. Le processus de gestion des réseaux inclut généralement la collecte des données, faite automatiquement ou via l'analyste responsable du réseau, le traitement de ces données et, ensuite, leur présentation. Ce processus peut inclure aussi une phase d'analyse des données et une phase de prise de décision sans intervention de l'analyste. En plus, il peut inclure une phase de production de rapports dont l'analyste peut se servir pour l'administration du réseau.

Pour répondre à ces besoins, l'*International Organization for Standardization* (ISO) a défini cinq domaines fonctionnels de la gestion des réseaux [13]:

1. Gestion des pannes: c'est le processus de détection, d'isolation et de correction des problèmes dans un réseau;

2. Gestion de la configuration: c'est le processus permettant d'initialiser un réseau et de le mettre hors service. Cela inclut aussi l'ajout, la mise à jour et la maintenance des accessoires et des paramètres du réseau;

3. Gestion de la sécurité: c'est le processus de contrôle de l'accès aux informations dans le réseau. Cela inclut les mots de passe et les droits d'accès à certaines informations ou au réseau lui-même;

4. Gestion des performances: c'est le processus permettant d'évaluer la performance des composants matériels et logiciels du réseau;

5. Gestion des comptes: c'est le processus permettant de contrôler l'utilisation individuelle et de groupe des ressources du réseau, de façon à assurer la meilleure attribution des ressources dont les utilisateurs, ou les groupes d'utilisateurs, ont besoin.

Pour couvrir tous ces aspects de la gestion d'un réseau, différents protocoles ont été développés. On distingue principalement SNMP et CMIS/CMIP [19].

Dans le cadre de ce mémoire, nous nous intéressons à la gestion à distance d'un dispositif de télécommunication appelé *Terminal Node Controller* (TNC). Ce dernier est un accessoire utilisé pour les communications sans fils et particulièrement par la communauté des radios amateurs [12]. Il est placé entre un ordinateur et un appareil radio. Sa fonction principale est d'accepter chaque paquet d'information provenant de l'ordinateur auquel il est attaché et de l'utiliser pour moduler un signal audio qui est transmis à une radio branchée au TNC. Vice versa, il peut transformer les signaux audio provenant de l'appareil radio auquel il est attaché en un paquet de données qui est transmis à l'ordinateur.

La gestion que nous appliquons sur le TNC est du type gestion de la configuration. En effet, un TNC possède plusieurs paramètres qui, dépendamment de leurs valeurs, influent sur son fonctionnement. Il existe deux grandes familles d'outils permettant la configuration et la gestion des TNC: les logiciels d'émulation de terminal et des programmes utilitaires. Ces outils sont utiles uniquement si le gestionnaire du TNC est à proximité. Par conséquent, ils ne permettent pas la gestion à distance.

Ce mémoire présente un nouvel outil permettant la gestion d'un TNC à distance. L'outil développé est basé sur le protocole SNMP. Avec ce dernier, chaque paramètre à gérer est vu comme étant un objet. L'ensemble des objets à gérer est groupé dans une

base de données appelée *Management Information Base* (MIB). Chaque objet du MIB est défini en respectant une syntaxe connue sous le nom de *Structure of Management Information* (SMI).

Nous avons identifié les paramètres que nous désirons gérer. Ensuite, nous avons défini un MIB qui représente ces paramètres dans la syntaxe SMI. Le MIB est implanté sous le système d'exploitation Linux [2] et utilise l'outil CMU-SNMP [1]. Linux a été choisi parce qu'il offre des niveaux de sécurité semblables à ceux de Unix et qu'il supporte la gestion des réseaux. CMU-SNMP est l'outil gratuit le plus populaire pour le développement d'applications de gestion des réseaux. Aux commandes de gestion fournies avec CMU-SNMP, nous avons ajouté une nouvelle commande permettant la gestion simultanée et à distance de plusieurs paramètres. Toutes les commandes sont écrites dans le langage C.

La suite de ce mémoire comporte quatre chapitres et une conclusion. Le chapitre 1 présente les concepts de base de la gestion des réseaux. Deux grands thèmes sont présentés: les systèmes de gestion des réseaux et les protocoles de gestion des réseaux. Le chapitre 2 introduit les notions relatives à un MIB. Cela inclut la structure de l'information de gestion, le groupe expérimental et les compilateurs MIB. Le chapitre 3 aborde plus en détail le protocole SNMP. D'abord, les notions de base de ce protocole sont données. Ensuite, les spécifications du protocole sont expliquées. Enfin, les principales améliorations apportées avec la deuxième version (SNMPv2) sont expliquées. Le chapitre 4 présente le nouvel outil que nous avons développé pour le contrôle et la gestion à distance d'un TNC.

Chapitre 1

Concepts de gestion des réseaux

Ce chapitre introduit les concepts de base de la gestion des réseaux. La première section concerne les systèmes de gestion des réseaux. Elle met l'accent sur l'architecture de ces systèmes et l'architecture des logiciels de gestion. La deuxième et dernière section présente les différentes familles de protocoles de gestion des réseaux, à savoir SNMP, CMIS/CMIP, CMOT et LMMP [13].

1.1 Systèmes de gestion des réseaux

1.1.1 Architecture d'un système de gestion des réseaux

Un système de gestion des réseaux est une collection d'outils de contrôle et de surveillance utilisés pour assurer le bon fonctionnement et la meilleure productivité du réseau. Généralement, il est constitué d'un ensemble de composantes logicielles et matérielles ajoutées à celles constituant le réseau en question. Les composantes logicielles utilisées pour la gestion figurent dans des ordinateurs hôtes et des processeurs de communication (tels que les ponts et les routeurs).

Mettre au point un système de gestion qui offre toutes les fonctionnalités et les services

de gestion est une tâche complexe. Idéalement, on commence par définir l'architecture du système, puis on passe à la phase de développement des applications et des outils de gestion du réseau.

Il n'existe aucune règle à appliquer pour définir l'architecture du système de gestion. Cependant, en tenant compte des fonctions requises par ce système, les points suivants doivent être considérés lors de son développement:

- le système doit posséder une interface graphique permettant de visualiser la hiérarchie du réseau et d'établir les connexions logiques entre ces niveaux hiérarchiques;
- le système doit posséder une base de données relationnelle qui peut enregistrer n'importe quelle information requise par les applications de gestion;
- le système doit être capable de retirer des informations à partir de tous les dispositifs connectés au réseau;
- le système doit être facile à étendre et à personnaliser. En effet, il n'existe aucun système universel adapté à tous les types de réseau. C'est pour cette raison que le système doit permettre l'ajout de nouvelles applications;
- le système doit être en mesure de détecter et suivre les problèmes qui peuvent survenir.

En se basant sur ces aspects, trois architectures de système de gestion des réseaux ont été établies [13]:

1. Architecture centralisée: cette architecture est appliquée pour les grands systèmes sur lesquels sont installées la grande majorité des applications. Chaque application doit sauvegarder ses données sur un support de stockage commun de la machine centrale (figure 1);

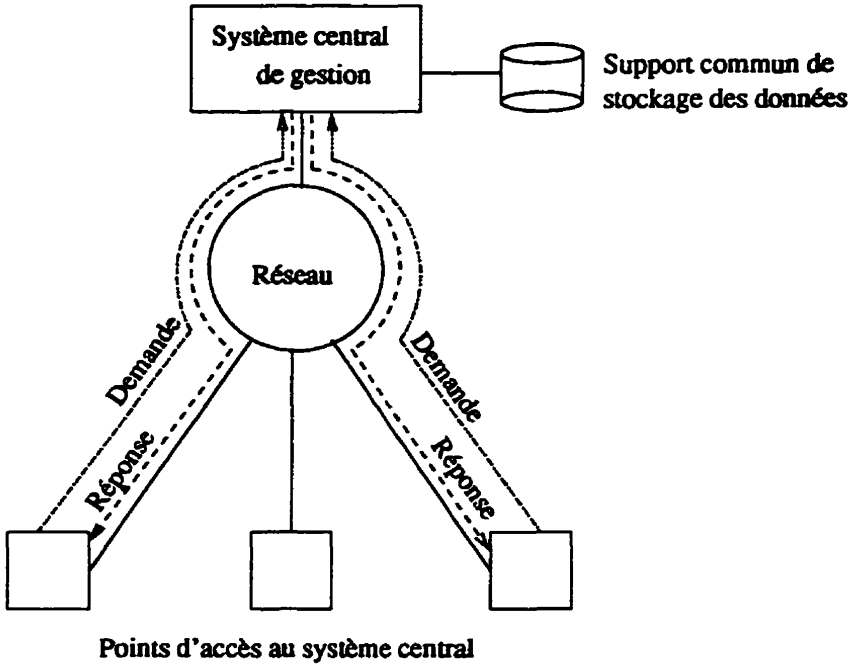


FIG. 1: *Architecture centralisée d'un système de gestion des réseaux.*

2. Architecture distribuée: elle regroupe plusieurs systèmes de gestion qui tournent simultanément sur le réseau (figure 2). Dans ce cas, chaque système gère uniquement une partie du réseau et stocke ses données dans son propre support. Un exemple d'architecture distribuée est le réseau Internet;

3. Architecture hiérarchique: c'est la combinaison des deux architectures susmentionnées. Le système central correspond à la racine de la hiérarchie. Il permet de sauvegarder quelques données dans l'unité centrale de stockage des données et contrôle l'accès aux différentes parties du réseau (figure 3). Les systèmes de bas niveau (systèmes fils) correspondent aux autres branches de la hiérarchie. Chaque système fils gère à sa façon sa part du réseau, sauvegarde des données dans une base de données locale et d'autres informations dans l'unité centrale de stockage des données. Cette architecture combinée est la plus flexible.

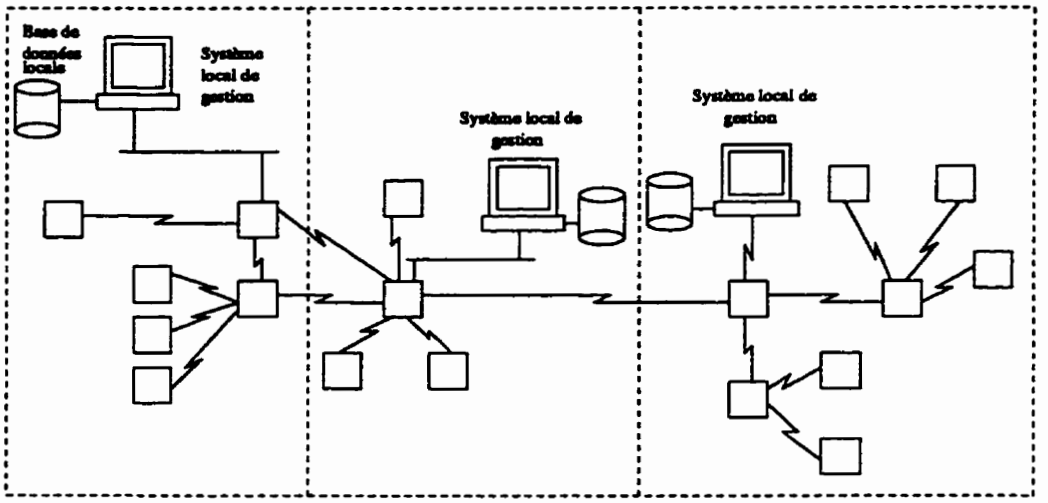


FIG. 2: *Architecture distribuée d'un système de gestion des réseaux.*

1.1.2 Architecture des logiciels de gestion des réseaux

L'architecture d'un logiciel de gestion d'un réseau varie en fonction des services fournis par le logiciel et de la plate-forme sur laquelle il tourne. La figure 4 présente une architecture générale.

Le logiciel de gestion peut être décomposé en trois parties:

1. Une interface utilisateur: c'est la partie du logiciel qui permet le dialogue entre l'utilisateur (généralement il s'agit du gestionnaire du réseau) et le logiciel de gestion. Cette interface est nécessaire pour toutes les stations de gestion dans le but de contrôler et surveiller le réseau. Elle peut être nécessaire pour les tests, le déverminage et parfois pour la lecture et la modification de quelques paramètres.

2. La gestion du réseau: cette partie du logiciel offre tous les services de gestion. Elle peut être simple, comme dans le cas de SNMP, ou complexe, comme dans le cas des systèmes OSI (Open Systems Interconnection) de gestion [19]. Cette composante

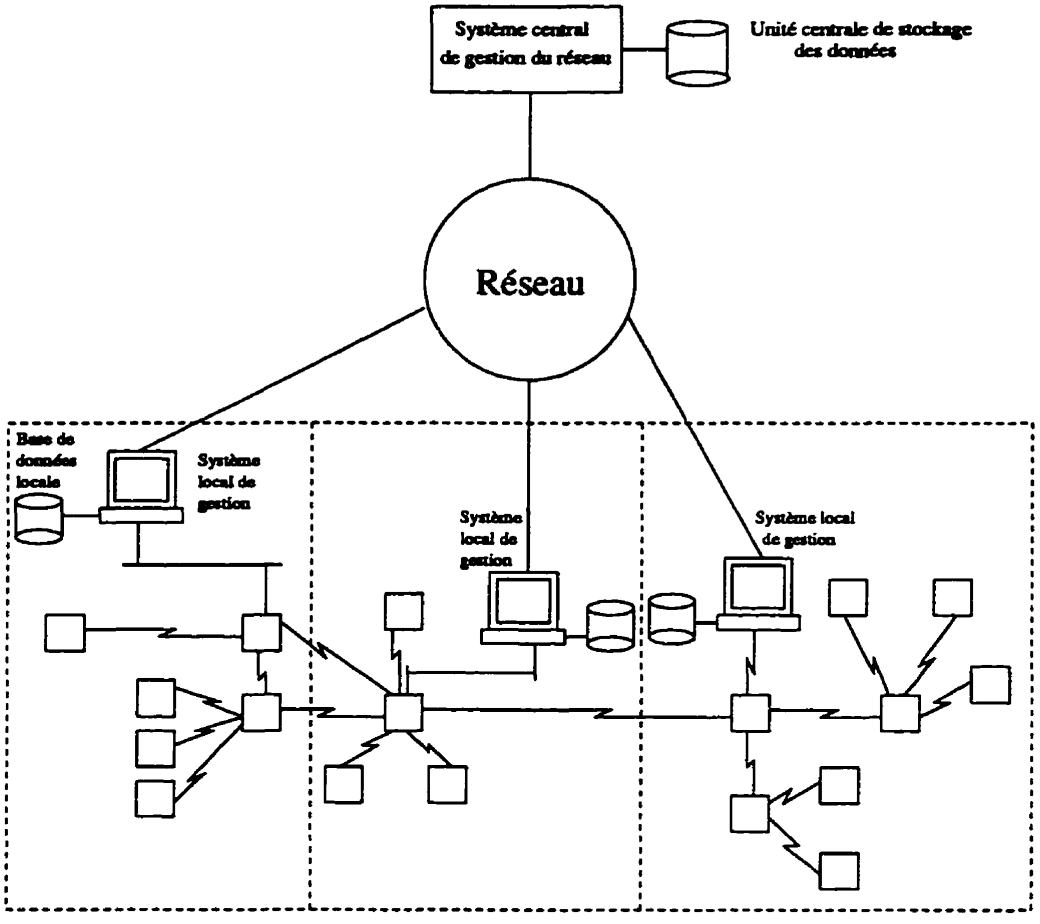


FIG. 3: *Architecture hiérarchique d'un système de gestion des réseaux.*

logicielle est subdivisée en trois niveaux:

- le niveau supérieur est constitué d'un ensemble d'applications qui fournissent les services requis par les utilisateurs. Ces applications peuvent correspondre par exemple aux types suivants de gestion: gestion des pannes, gestion de la configuration, gestion de la sécurité, gestion de la performance et gestion des comptes. Chaque application couvre un certain domaine de gestion et peut dépendre de la configuration du réseau.
- le deuxième niveau correspond aux différents modules et entités appelés par

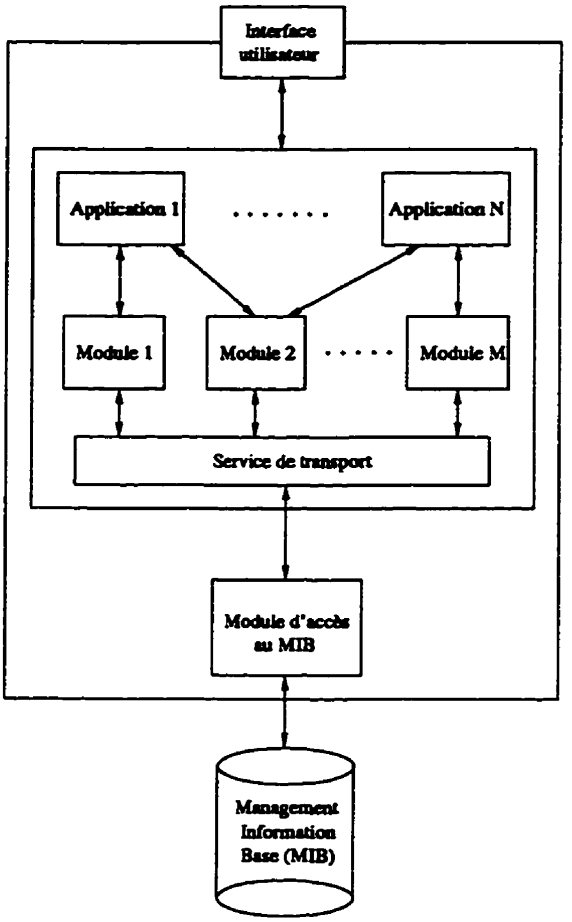


FIG. 4: *Architecture d'un logiciel de gestion des réseaux.*

le niveau supérieur. Ces modules implantent des fonctionnalités plus spécifiques, telles que la gestion des alarmes et la collecte des données, et peuvent être utilisés par plusieurs applications. L'organisation du logiciel en des applications et des modules permet une meilleure implantation et la réutilisation des modules pour développer d'autres logiciels ou pour ajouter de nouveaux services.

- Le niveau inférieur correspond au service de transport. Il représente le protocole utilisé pour échanger des données entre les stations de gestion et les

agents. Il offre, typiquement, des services primitifs tels que la lecture d'une information, la modification d'un paramètre et la production d'une information de notification.

3. Un module d'accès aux bases de données: pour accomplir ces tâches, le logiciel de gestion a besoin d'accéder à une base de données, appelée MIB, et de communiquer avec les agents et les stations. Le MIB local d'un agent contient des informations utiles pour la gestion de sa part du réseau, telles que les informations relatives à la configuration et l'état de ce noeud, et d'autres paramètres pour le contrôle de ce noeud. Le MIB local d'une station de gestion contient aussi bien les informations relatives aux différents noeuds que d'autres informations à propos des agents sous son contrôle. En plus, le noeud d'accès peut contenir des services pour la conversion du format du MIB local à un autre format standard utilisé par le système global de gestion.

1.2 Protocoles de gestion des réseaux

Pour lire des informations à partir de différents dispositifs appartenant à un certain réseau, les analystes doivent employer plusieurs méthodes d'accès. En effet, pour un même type de dispositifs (un modem par exemple) produits par différents manufacturiers, la façon de communiquer avec le dispositif varie. Un manufacturier propose, par exemple, un ensemble de commandes pour lire quelques informations relatives à ses dispositifs, alors qu'un autre fournit une interface de type menus. Pour de grands réseaux hétérogènes, l'apprentissage de différentes méthodes d'accès à un type de dispositifs rend le travail des gestionnaires inefficace. Les analystes ont besoin d'outils qui leur permettent de communiquer avec de tels dispositifs de façon homogène.

Jusqu'à la fin des années 70, il n'y avait aucun protocole utilisé spécifiquement pour la gestion des réseaux et de leurs dispositifs. Le seul outil de gestion utilisé jusqu'à ce

moment est *Internet Control Message Protocol* (ICMP). Ce dernier est implanté au-dessus d'*Internet Protocol* (IP) et permet d'envoyer des messages de contrôle à des routeurs et des stations.

Le service le plus employé de ICMP est le service *echo/echo-reply*. Ce service fournit un mécanisme pour tester, entre autres, si la communication entre deux entités est possible. Le récepteur d'un message *echo* doit retourner le contenu du message reçu sous la forme d'un message *echo-reply*. Ces messages ICMP, combinés avec des en-têtes IP, peuvent être utilisés comme des outils de gestion simples et efficaces. L'exemple le plus remarquable est le programme PING (Packet Internet Groper). Il peut servir à déterminer si un dispositif peut être atteint physiquement ou non, à vérifier si un réseau est accessible et à vérifier si un serveur ou une station est bien fonctionnel. PING a servi, durant des années, comme moyen de gestion. Cependant, vers la fin des années 80, lorsque la taille de l'Internet débuta sa croissance exponentielle [14], une attention a été portée sur le développement d'un autre moyen de gestion, plus sophistiqué. Pour répondre à cet urgent besoin, l'*Internet Activities Board* (IAB), l'organisme qui supervise les efforts d'interconnexion des réseaux et de développement des protocoles pour la communauté TCP/IP, a décidé de prendre en charge la coordination des efforts pour adopter un protocole standard de gestion des réseaux. Trois protocoles ont été proposés:

1. High-Level Entity Management System (HEMS): il correspond à la généralisation du protocole HMP (Host-Monitoring Protocol);

2. Simple Network Management Protocol (SNMP): c'est une version améliorée du protocole Simple Gateway Management Protocol (SGMP);

3. Common Management Information Protocol (CMIP) over TCP/IP (CMOT): c'est une intention d'incorporer, autant que possible, le protocole CMIP, les services CMIS et la structure de la base de données, standardisés par l'ISO pour la gestion des réseaux.

Au début des années 80, l'IAB a révisé ces protocoles et a recommandé l'implantation immédiate de SNMP comme solution à court terme et le protocole CMOT comme solution à long terme [7]. Il faut mentionner à ce stade que l'IAB et l'ISO ne sont pas les seules organisations impliquées dans le développement des protocoles standard. L'ITU-T et l'IEEE ont aussi participé au développement de cette catégorie de protocoles (LAN Man Management Protocol (LMMP) pour l'IEEE).

Les principes de base des protocoles SNMP, CMIP et LMMP sont résumés dans la suite de cette section.

1.2.1 SNMP

Actuellement, le protocole le plus utilisé pour la gestion des réseaux de données est SNMP [6]. SNMP est basé sur le modèle *agent/station de gestion*. Un agent est un veilleur capable de répondre à des requêtes provenant d'une station de gestion. Chaque dispositif qui fournit des informations à une station de gestion doit avoir son propre agent SNMP. Les agents et les stations communiquent entre eux en utilisant des messages standard. Chaque message est représenté sous forme de paquets. SNMP utilise UDP comme protocole de transport. Ce dernier a été retenu parce qu'il ne nécessite pas la maintenance d'une connexion permanente entre les agents et les stations, ce qui exige peu de ressources pour l'échange des messages. On distingue cinq types de messages: *GetRequest*, *GetResponse*, *GetNextRequest*, *SetRequest* et *Trap*.

GetRequest et GetNextRequest sont utilisés par une station pour extraire des informations à partir d'un dispositif connecté au réseau et ayant un agent SNMP. L'agent répond à cette requête en envoyant un GetResponse. SetRequest permet la configuration des paramètres d'un dispositif. Trap est un message non sollicité envoyé par un agent à une station pour l'informer d'un événement qui est survenu. Par exemple, un Trap peut être généré pour informer le système de gestion qu'un certain circuit n'est plus valide, que l'espace disque d'un dispositif est près de sa limite ou qu'un nouvel utilisateur s'est

connecté à la machine.

Le protocole SNMP ne permet pas l'accès à des informations, ni le changement des paramètres de configuration des dispositifs connectés au réseau, sans une certaine forme de sécurité. L'agent peut demander à chaque station voulant accéder à son MIB un mot de passe au sein de chaque message. De cette façon, l'agent peut vérifier les droits d'accès avant de fournir les informations demandées. Ces mots de passe sont connus sous l'appellation de *nom de communauté*. Les noms de communautés sont envoyés dans les paquets sous forme de caractères ASCII. Ceci présente malheureusement des problèmes de sécurité. L'IETF ne cesse d'étudier ce problème.

Bien que SNMP soit un protocole assez reconnu et utilisé pour la gestion des réseaux de données, quelques problèmes majeurs ne sont pas encore résolus:

- SNMP est utilisé uniquement pour les réseaux supportant IP;
- SNMP est inefficace pour l'accès à de très grandes tables de données;
- la réception des Trap n'est pas confirmée;
- la communication entre stations de gestion n'est pas possible;

En effet, SNMP fonctionne au-dessus du protocole IP. Bien que ce dernier soit largement utilisé, ce n'est pas tous les réseaux qui l'emploient. C'est pour cette raison que certains pensent que SNMP n'est qu'une solution temporaire pour la gestion des réseaux [16]. Une solution pour remédier aux problèmes de la non universalité de SNMP est l'utilisation d'agents mandataires (section 3.1.2, page 48). L'inconvénient de cette approche est qu'il faut écrire un programme de traduction de protocoles spécifique à chaque dispositif du réseau. Une autre solution consiste à implanter SNMP au-dessus d'autres protocoles de niveau transport pour lui permettre de fonctionner avec des réseaux qui ne supportent pas le protocole IP.

L'autre problème de SNMP est qu'il n'est pas efficace pour accéder au contenu des tables de données de grande taille. Par exemple, pour une table ayant 2 000 entrées (rangées) dont chacune possède quatre colonnes, il faut appliquer GetRequest et GetNextRequest 2 x 4 x 2 000 fois, c'est-à-dire 16 000 fois.

Le troisième problème de SNMP est qu'il ne confirme pas la réception des Trap. Un agent, par exemple, ne peut être certain qu'un Trap émis pour signaler un événement critique a été bien reçu par la destination.

Finalement, SNMP ne permet pas la communication entre stations de gestion. Par exemple, il n'y a aucun mécanisme permettant à un système de gestion de connaître les dispositifs connectés à un autre système. Malgré tous ces problèmes, SNMP est la solution la plus répandue pour la gestion des réseaux.

1.2.2 CMIS/CMIP

Le service pour échanger des informations de gestion entre stations de gestion et agents, au sein des systèmes OSI, est connu sous le nom de *Common Management Information Service Element* (CMISE). CMISE est spécifié en deux parties:

1. les services fournis par chaque élément du réseau pour des buts de gestion. C'est ce qu'on appelle *Common Management Information Service* (CMIS);

2. le protocole spécifiant le format des unités de données (PDU) utilisées et les services qui leur sont associés. C'est le *Common Management Information Protocol* (CMIP).

CMIS fournit sept services pour accomplir les opérations de gestion: M-EVENT-REPORT, M-GET, M-SET, M-ACTION, M-CREATE, M-DELETE et M-CANCEL-GET. En plus, les utilisateurs ont besoin d'établir des associations d'applications (c'est-à-dire des connexions avec les opérations A-Associate, A-Release et A-Abort) dans le but d'échanger des opérations de gestion. Ces derniers services sont fournis par l'*Association Control Service*

Element (ACSE). CMIP se base sur les services fournis par le *Remote Operations Service Element* (ROSE) pour échanger des PDU entre stations de gestion et agents. La figure 5 illustre ces notions de façon schématique. Les services de CMIS sont spécifiés en terme de primitives qui peuvent être vues comme étant des commandes ou des appels à des procédures avec paramètres. Ces services peuvent être confirmés ou non confirmés. Les services fournis par CMIS sont classés en trois catégories:

1. services d'association: pour que deux utilisateurs puissent utiliser CMIS pour effectuer des opérations de gestion, ils doivent en premier lieu établir une application d'association. C'est à base des services de ACSE que les associations sont établies. Ces services sont les suivants:

   - A-Associate: c'est le service permettant d'initialiser une connexion;
   - A-Release: c'est le service utilisé pour mettre fin à une connexion;
   - A-Abort: c'est le service utilisé pour mettre fin à une connexion anormale.

   Deux fonctions sont effectuées par ACSE en faveur de CMIS. La première est l'établissement d'une application d'association qui est utilisée pour échanger les primitives de CMIS. La deuxième, à l'établissement de l'association, est le choix que les deux utilisateurs font des services de CMIS qui seront utilisés sur cette association. Ces services sont exprimés en terme d'unités fonctionnelles. Une unité fonctionnelle définit un ensemble de fonctions fournies par un service. La figure 6 mentionne les unités fonctionnelles de CMIS. Le service fourni par défaut est l'unité fonctionnelle Kernel. Cette dernière est constituée de toutes les primitives de CMIS sauf M-CANCEL-GET et excluant les paramètres *portée*, *synchronisation*, *filtre* et *identification-liée* (réponses multiples). Ces limitations sont appliquées sur M-GET, M-SET et M-ACTION. Les deux utilisateurs peuvent négocier pour ajouter d'autres unités fonctionnelles à l'unité fonctionnelle Kernel.

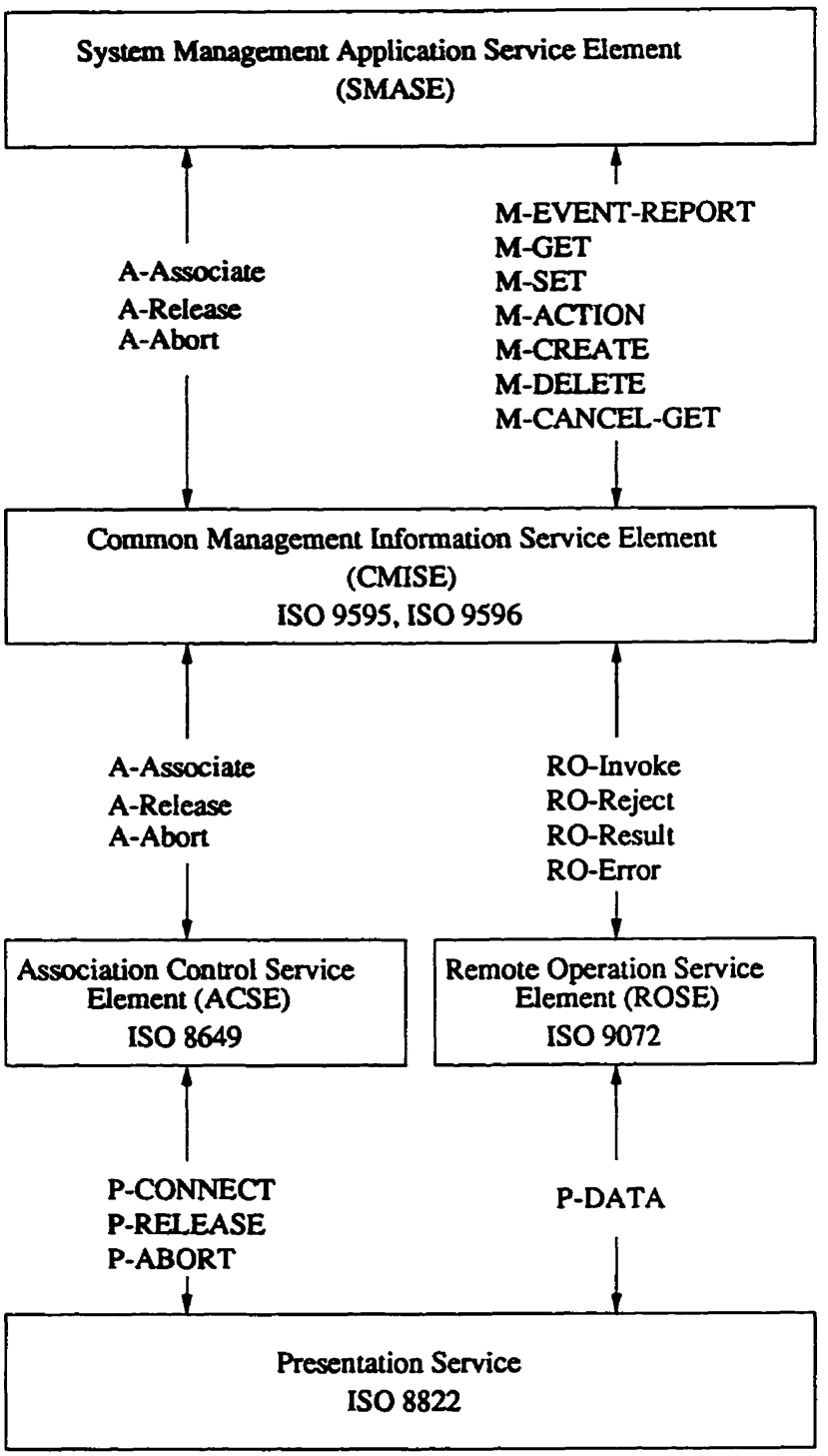


FIG. 5: *Les services fournis par CMIS et utilisés par CMISE.*

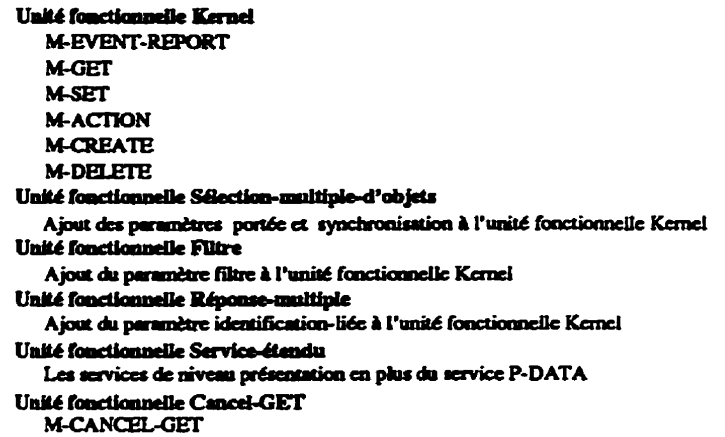


FIG. 6: *Les unités fonctionnelles de CMIS.*

La portée, la filtration et la synchronisation correspondent à trois paramètres utilisés par CMIS. Ils permettent la sélection d'un ou de plusieurs objets sur lesquels les opérations de gestion seront appliquées:

- la portée: elle réfère à l'identification d'un ou de plusieurs objets sur lesquels un filtre sera appliqué. Un objet de base est utilisé comme référence à tous les objets sur lesquels les opérations de gestion seront appliquées;
- la filtration: c'est une expression booléenne constituée d'une ou de plusieurs assertions relatives à la présence ou à la valeur des attributs d'un objet à gérer. Une assertion peut correspondre à une égalité, un ordre, une présence ou une comparaison;
- la synchronisation: lorsque plusieurs objets sont sélectionnés par filtration ou par portée, la notion de synchronisation est appliquée pour donner un ordre au traitement des objets sélectionnés.

2. service de gestion de notification: ce service fournit la même fonctionnalité que celle du Trap employé par SNMP dans le but d'informer la station de gestion à propos d'événements critiques qui se sont produits. L'unique primitive fournissant

ce service est nommée M-EVENT-REPORT.

3. services des opérations de gestion: ces services sont utilisés pour effectuer les opérations de gestion. Ils sont au nombre de six:

   - M-GET: il permet de retirer une donnée à partir d'un MIB. Un processus de gestion, chez une station de gestion, envoie une requête GET à un autre processus, actif chez un agent. Cette requête peut demander des informations relatives à un seul objet ou un ensemble d'objets. Pour chaque objet en question, les valeurs d'un ou plusieurs attributs peuvent être demandées. Le service M-GET est un service avec confirmation de réception;
   - M-SET: il permet de modifier les valeurs d'un ou plusieurs attributs d'un ou plusieurs objets. Contrairement au SetRequest de SNMP (qui est atomique), l'effet d'un M-SET peut être partiel parce que certaines modifications, parmi celles demandées, ne peuvent être faites;
   - M-ACTION: ça appelle une procédure d'action prédéfinie et spécifiée comme une partie d'un objet géré. Cette requête spécifie le type d'action ainsi que les paramètres d'entrée. Le service M-ACTION peut être demandé avec confirmation ou pas. La réponse retournée lors d'une action accomplie avec succès peut contenir la classe de l'objet géré et son instance, confirmant ainsi l'action. Elle peut aussi contenir le type de l'action et d'autres paramètres qui spécifient avec plus de détail l'action prise et les résultats de l'action. Si jamais une erreur se produit, une notification d'échec sera générée. Un exemple de M-ACTION est l'émission d'échos ICMP à différentes adresses dans le but de tester l'accessibilité IP entre quelques machines;
   - M-CREATE: ce service est utilisé pour la création d'une autre instance d'un certain objet. Avec CMIS, chaque objet géré possède ses propres instances. Parmi les utilisations de ce service, on distingue la permission aux objets gérés

de s'informer entre eux à propos de nouveaux objets qui viennent de s'ajouter. Par exemple, dans un système de gestion, il peut y avoir une définition d'un pont Ethernet. Chaque fois qu'un nouveau pont est ajouté, le système de gestion peut créer et utiliser une nouvelle instance de l'objet correspondant à cette définition;

- M-DELETE: il est utilisé pour supprimer une ou plusieurs instances d'une classe d'objets. Ce service est toujours demandé avec confirmation.
- M-CANCEL-GET: il permet d'abandonner un service GET déjà appelé. La seule raison pour laquelle le service GET peut être abandonné, c'est qu'il est difficile d'assurer la cohérence du MIB suite à l'abandon de l'une des autres opérations. M-CANCEL-GET est un service avec confirmation.

Parallèlement à CMIS, qui définit les services utilisés par les opérations de gestion, CMIP définit les procédures nécessaires à la transmission des informations de gestion et la syntaxe pour les services de CMIS. CMIP est défini en terme de PDU échangés entre des éléments CMISE fournissant les services de CMIS. Plus de détails sont donnés dans la référence [19].

Enfin, le protocole CMIS/CMIP ayant été conçu pour gérer différents types de réseaux supportant différentes familles de protocoles, il va de soi qu'il devient un protocole assez complexe. D'une part, il est vu comme étant le plus complet des protocoles de gestion des réseaux. D'autre part, il est difficile à implanter et demande beaucoup de ressources qui ne sont pas disponibles dans tous les systèmes.

1.2.3 CMOT

Le protocole *Common Management Information Service Over TCP/IP* (CMOT) propose d'implanter les services de CMIS au-dessus de TCP/IP. Cette idée a été proposée comme une solution intermédiaire, en attendant que l'ensemble des protocoles supportés

par CMIP soient implantés. Le document RFC 1189 [5] définit la structure du protocole CMOT. La figure 7 illustre le protocole CMOT dans le modèle ISO.

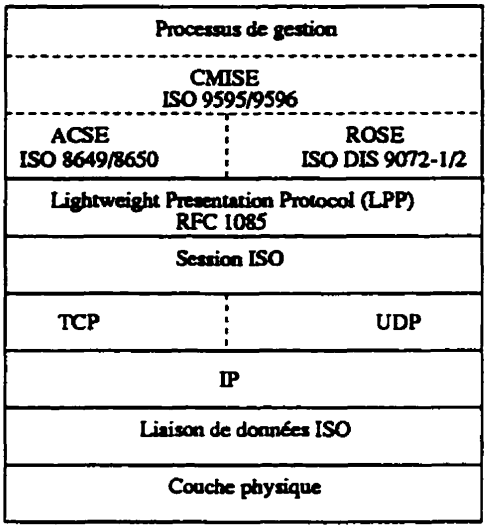


FIG. 7: *Le protocole CMOT dans le modèle de référence de l'ISO.*

Les protocoles de niveau application utilisés par CMIS restent inchangés avec CMOT. Ce dernier est basé sur les protocoles CMISE, ACSE et ROSE, comme décrit dans les paragraphes précédents. Cependant, au lieu d'attendre l'implantation du protocole de niveau transport de l'ISO, CMOT utilise un autre protocole qui lui est semblable et de même niveau, appelé *Lightweight Presentation Protocol* (LPP). Ce protocole fournit une interface à l'un des protocoles de niveau transport, UDP ou TCP, qui utilisent le protocole IP.

Le problème posé par l'utilisation de CMOT, c'est que plusieurs vendeurs d'outils et de dispositifs de gestion des réseaux ne veulent pas perdre de temps à développer des solutions temporaires. Par contre, plusieurs d'entre eux ont consacré à SNMP beaucoup d'efforts et de travaux à son développement. Ainsi, bien que CMOT soit bien défini et documenté, il est loin d'être une solution pratique et de figurer sur le marché de la gestion des réseaux.

1.2.4 LMMP

LAN Man Management Protocol (LMMP), le standard IEEE 802.1b, a été proposé comme solution pour les réseaux locaux (LAN). Il est aussi connu sous le nom de *Common Management Information Services and Protocol Over IEEE 802 Logical Link* (CMOL). Ce protocole, développé par 3Com Corp. et IBM, élimine le besoin des protocoles OSI pour implanter les services de CMIS. Vu que LMMP est implanté au-dessus de IEEE 802 Logical Link Layer (LLC), il n'a pas besoin d'être placé au-dessus d'autres protocoles de niveau réseau, tel que IP. De cette façon, LMMP est plus facile à implanter que CMIS/CMIP et CMOT. Cependant, il ne peut pas communiquer avec les routeurs pour livrer les paquets de données en dehors d'un réseau local.

1.2.5 Conclusion

Avant d'appliquer les protocoles standard de gestion des réseaux, les gestionnaires se servaient de méthodes simples pour le contrôle et la gestion des dispositifs connectés au réseau. Le besoin de méthodes qui peuvent s'appliquer à différents dispositifs a poussé la communauté responsable à développer des protocoles standard. Plusieurs protocoles ont été proposés.

SNMP est le protocole le plus utilisé. Les stations et les agents SNMP communiquent en utilisant un protocole bien défini pour lire, écrire et modifier les données d'un MIB regroupant l'ensemble des dispositifs à gérer. SNMP est cependant exploité avec les réseaux fonctionnant à base de TCP/IP seulement.

Le protocole CMIS/CMIP, proposé par l'OSI, fournit les différents services nécessaires pour une gestion plus générale et plus complète. CMIS/CMIP a besoin de la pile de protocoles de l'OSI, chose qui n'est pas encore largement disponible.

CMOT est la solution intermédiaire. Ce protocole permet d'avoir les services de CMIS au-dessus de la suite de protocoles TCP/IP. Malgré cette proposition, rares sont ceux

qui se sont basés sur un tel protocole pour la gestion des réseaux.

Au lieu de développer une solution intermédiaire, IBM et 3Com ont proposé le protocole LMMP qui offre les services de CMIS directement dans la couche LLC. LMMP reste quand même limité, vu qu'il a été conçu seulement pour les réseaux locaux.

Chapitre 2

Le MIB de SNMP

Dans ce chapitre, nous abordons le sujet des MIB employés par les systèmes de gestion des réseaux. Au début, nous présentons la structure de l'information de gestion. Nous commençons par présenter les notions de base, puis celles de plus haut niveau. Cela inclut la structure générale d'un MIB, des objets, des tables et des modules. Ensuite, nous présentons un groupe particulier d'objets nommé *experimental*. Enfin, nous abordons le sujet des compilateurs MIB. Deux types de compilateurs sont discutés: *SMIC* et *MOSY*.

2.1 Introduction

Comme n'importe quel système de gestion, les systèmes de gestion des réseaux, basés sur la suite de protocoles TCP/IP, possèdent une base de données contenant des informations relatives aux éléments à gérer. Avec les architectures TCP/IP et OSI, cette base de données est connue sous le nom de *Management Information Base* (MIB). Chaque ressource à gérer (par exemple une imprimante, un *Hub*, un routeur ou un serveur e-mail) est représentée sous forme d'un objet. La notion d'objet dans ce contexte est différente de celle utilisée dans la programmation orientée objet. En effet, un objet dans ce dernier contexte possède des attributs et des méthodes et peut supporter les notions d'héritage et

de polymorphisme. Cependant, un objet au sens des MIB ne reconnaît aucune de ces notions. Vis-à-vis des spécifications ASN.1, le MIB est défini comme étant un fichier ASCII constitué d'un ensemble de définitions d'objets. Les règles sémantiques et les méthodes d'exploitation sont spécifiées par ces objets. Du côté des agents et des applications de gestion, un MIB est constitué par des instances d'objets, qui ne sont que des variables. Ces dernières correspondent aux abstractions des différentes ressources à gérer.

Chaque noeud du système maintient un MIB qui reflète l'état des ressources gérées qu'il possède. Une entité de gestion (application chez une station de gestion) peut contrôler les ressources d'un noeud en lisant les valeurs des objets du MIB et les gère en modifiant et en mettant à jour ces mêmes valeurs.

Pour que le MIB réponde à tous les besoins d'un tel système de gestion des réseaux, il faut tenir compte de deux points:

1. avoir une structure d'objet commune pour présenter un certain type de ressource au niveau de tous les noeuds;

2. avoir un schéma commun de représentation des objets pour supporter l'inter-opérabilité.

Le premier point est solutionné en utilisant une définition commune à tous les objets. Le deuxième point est solutionné en définissant un unique et commun modèle d'information de gestion (*Structure of Management Information*, SMI). Tout objet défini au sein du MIB doit l'être au moyen de SMI pour que les MIB soient inter-opérables.

Dans la section suivante, nous allons examiner la structure de l'information de gestion des points de vue syntaxique et sémantique. Seules les versions SMIv2 et SNMPv2 seront mentionnées. En effet, ce sont celles qui ont été exploitées durant notre projet de maîtrise.

2.2 Structure de l'information de gestion

La structure de l'information de gestion, SMI, définit les règles à suivre pour spécifier et implanter un MIB. SMI identifie les types de données qui peuvent être utilisés dans un MIB ainsi que la façon dont les ressources à gérer sont représentées et nommées. SMI comporte deux techniques de standardisation pour:

1. définir la structure générale d'un MIB et

2. définir les objets (leur syntaxe et leur valeur).

Voyons chacun de ces aspects de plus près.

2.2.1 Structure du MIB

On sait maintenant que le MIB est l'équivalent d'une base de données, constituée d'un ensemble d'objets. À chaque objet est associé un identificateur unique nommé OBJECT IDENTIFIER (OID) dont la valeur est constituée d'une séquence d'entiers non négatifs. L'ensemble des objets définis au sein d'un MIB est structuré à la façon d'un arbre. Les objets sont aux feuilles de l'arbre. À chaque noeud de l'arbre, sauf à la racine, est associé un entier. La concaténation des entiers sur un chemin de l'arbre, de la racine à une feuille, constitue l'identificateur de l'objet qui est à la feuille. Trois noeuds sont situés en dessous de la racine: *iso*, *ccitt* et *joint-iso-ccitt* (figure 9). Les principaux noeuds situés en dessous du noeud *iso* sont représentés dans la figure 8. L'identificateur du noeud *internet* est le suivant:

```
internet OBJECT IDENTIFIER ::= {iso(1) org(3) dod(6) 1}
```

Ainsi, cet identificateur a la valeur 1.3.6.1. Cette dernière sert comme préfixe pour tous les noeuds figurant en dessous du noeud *internet*. Généralement, un identificateur sert à identifier le noeud ainsi que tous les objets qui sont situés en dessous de lui.

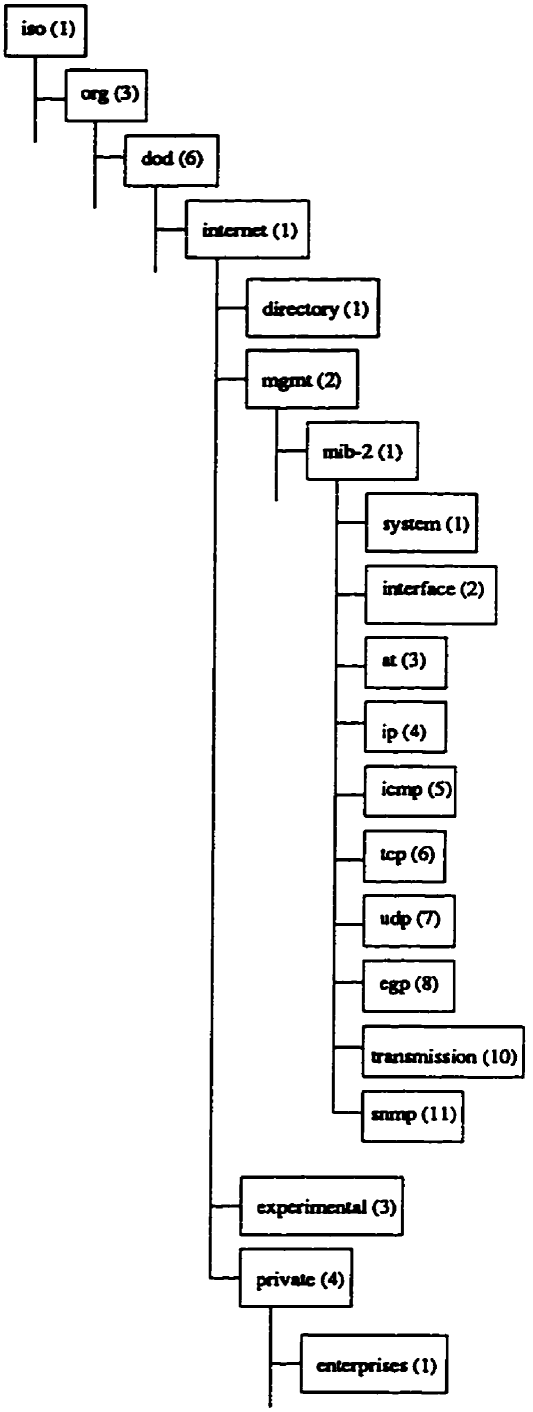


FIG. 8: *Les groupes d'objets du MIB-II.*

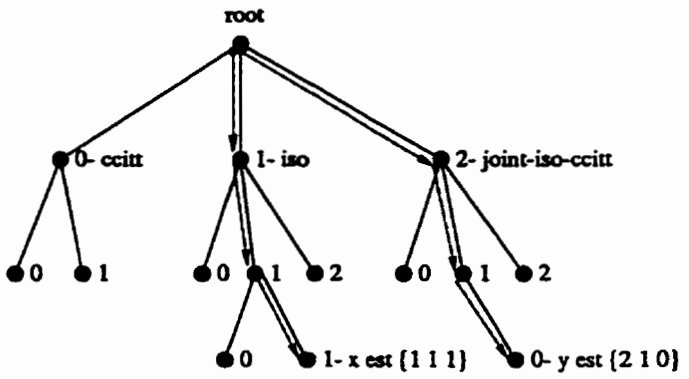


FIG. 9: *Structure arborescente des identificateurs des objets gérés.*

2.2.2 Structure des objets

Les objets, au sein d'un MIB, sont définis en utilisant une syntaxe appelée *Abstract Syntax Notation One, ASN.1* [19]. ASN.1 est un langage formel qui a été développé et standardisé par la ITU-T et l'ISO. Il est utilisé pour:

- définir une syntaxe abstraite pour les applications;
- définir la structure des unités de données (*Protocol Data Unit*, PDU) des couches application et présentation;
- définir la structure des MIB spécifiques aux systèmes de gestion des réseaux de type SNMP et OSI.

Dans le cadre de la gestion des réseaux, bien que d'autres langages formels puissent être employés pour définir ces mêmes aspects, en pratique ASN.1 est le plus utilisé.

La liste complète des types de données SNMPv2 est donnée dans le tableau 1:

Integer32 et *Unsigned32* permettent de spécifier des valeurs pouvant être négatives ou positives. *Gauge32* est utilisé pour spécifier des types de données dont les valeurs ne peuvent dépasser des bornes prédéfinies. Il peut par exemple servir à indiquer la valeur de la température mesurée par un thermomètre. *Counter32* ainsi que *Counter64* sont utilisés

Nom
Integer32
Unsigned32
Gauge32
Counter32
Counter64
TimeTicks
OCTET STRING
Opaque
OBJECT IDENTIFIER
IpAddress
BITS

TAB. 1: *Types de données utilisés avec les informations de gestion.*

pour compter la production d'un événement, tel que l'émission d'un paquet de données par une interface d'un réseau, ou pour mesurer un flux de données, tel que le nombre de bons paquets de données reçus par une interface. *TimeTicks* permet de spécifier des unités de données exprimées en centièmes de secondes. *OCTET STRING* est utilisé pour spécifier des octets pouvant contenir des informations binaires ou textuelles. Par contre, *Opaque* permet de ne spécifier que des octets de valeur binaire. *OBJECT IDENTIFIER* est utilisé pour identifier un objet. Les adresses réseaux sont spécifiées grâce à *IpAddress*. Enfin, *BITS* sert à regrouper un ensemble de bits prénommés.

Il faut noter que l'ensemble de ces types de données est divisé en trois catégories:

- types universels: cette catégorie regroupe les types suivants: INTEGER, OCTET STRING et OBJECT IDENTIFIER;
- types application: regroupe les types suivants: Integer32, Unsigned32, Gauge32, Counter32, Counter64, TimeTicks, IpAddress et Opaque.
- pseudo-types: cela inclut uniquement le type BITS.

Après avoir vu tous les types de données définis par SMI, nous pouvons maintenant définir un objet.

2.2.3 Définition des objets MIB

La définition d'un objet au sein d'un MIB est effectuée à base de macros. La macro utilisée pour définir un objet MIB est nommée *OBJECT-TYPE*. Cette dernière est reconnue dans les deux versions de SMI. Par contre, dans SMIv2 quelques champs ont été renommés et d'autres ont été ajoutés. Les différents champs utilisés par OBJECT-TYPE sont les suivants:

- *SYNTAX*: il permet de spécifier la syntaxe d'un objet. La syntaxe est construite en utilisant les types universels et les types application;
- *MAX-ACCESS*: il définit le mode d'accès permis à une instance d'un objet. Les valeurs possibles sont: *read-only*, *read-write*, *read-create*, *not-accessible* et *accessible-for-notify*;
- *STATUS*: désigne la validité de la définition de l'objet. Les valeurs possibles de ce champ sont *current* (valide), *deprecated* (remplacée par une autre) et *obsolete* (non valide et ne peut plus être appliquée);
- *DESCRIPTION*: il correspond à une description textuelle (chaîne de caractères) de la sémantique du type d'objets en question;
- *UNITS*: contient une définition textuelle des unités associées à un objet (par exemple, seconde pour le cas du temps);
- *REFERENCE*: contient une référence textuelle à un objet défini dans d'autres modules MIB;

- *INDEX*: définit comment les rangées d'une table sont indexées. Ce champ illustre de façon ordonnée les différents champs qui entrent dans la composition de l'index d'une table. Il n'y a aucune restriction sur le nombre d'index. Typiquement, ils correspondent à des objets colonnes de la table;

- *AUGMENTS*: permet d'étendre le nombre de colonnes d'une table sans toucher à sa structure (sans redéfinir la table). AUGMENTS reçoit comme paramètre l'identificateur d'une rangée d'une autre table. Cette dernière est appelée *table de base*; par contre, la table utilisant le champ AUGMENTS est appelée *table d'augmentation*. Une table de base peut être étendue par plusieurs tables d'augmentation. Cet aspect est semblable à la notion d'héritage;

- *DEFVAL*: spécifie la valeur à affecter à une instance d'un objet colonne lors de sa création. Aucune valeur initiale n'est attribuée aux objets correspondant aux index d'une table, ni à un objet scalaire. Seuls les objets colonnes accessibles en mode *read-create* peuvent avoir un champ DEFVAL.

Il faut noter que les champs SYNTAX, ACCESS (ou MAX-ACCESS) et STATUS (INDEX ou AUGMENTS pour les tables) sont obligatoires lors de la définition d'un objet. Par contre, les champs UNITS, REFERENCE, DESCRIPTION et DEFVAL sont optionnels.

2.2.4 Définition des tables

La macro OBJECT-TYPE est utilisée pour définir une table d'objets. Une table SNMP est composée d'un ensemble de rangées et de colonnes. L'argument du champ SYNTAX pour une table doit être *SEQUENCE OF <sequence>*. Enfin, la valeur du champ ACCESS ou du champ MAX-ACCESS d'une table doit être *not-accessible* (vu qu'une table n'est pas directement accessible avec les opérations de SNMP). Voici un

exemple de définition d'une table:

```
ifTable OBJECT-TYPE
   SYNTAX       SEQUENCE OF IfEntry
   MAX-ACCESS   not-accessible
   STATUS       current
   DESCRIPTION  "..."
   ::= { interfaces 2 }
```

Suite à la définition d'une table, vient la définition de ses rangées. Le constructeur OBJECT-TYPE est aussi utilisé dans ce but. L'argument du champ SYNTAX d'une rangée doit être un identificateur pour une séquence (l'identificateur de la rangée). La valeur des champs ACCESS ou MAX-ACCESS doit être aussi *not-accessible*. La valeur du champ STATUS d'une rangée donnée doit être la même que celle de la table. Le champ INDEX ou AUGMENTS spécifie comment les instances des objets colonnes de la table sont identifiées.

SMI mentionne que la valeur OID attribuée à une rangée doit être la même que celle de la table dont elle fait partie, tout en ajoutant la valeur 1 à la fin de cette même valeur OID. Par exemple, si la valeur OID d'une table nommée *printerTable* est x, alors la valeur OID de sa rangée (*printerEntry*) sera $x.1$. Voici un exemple de définition de rangées:

```
ifEntry OBJECT-TYPE
   SYNTAX       IfEntry
   MAX-ACCESS   not-accessible
   STATUS       current
   DESCRIPTION "..."
   INDEX    { ifIndex }
   ::= { ifTable 1 }

IfEntry ::= SEQUENCE {
              ifIndex           INTEGER,
              ifDescr           OCTET STRING,
```

```
        ifType              INTEGER,
        ifMtu               INTEGER,
        ifSpeed             Gauge,
        ifPhysAddress       OCTET STRING,
        ifAdminStatus       INTEGER,
        ifOperStatus        INTEGER,
        ifLastChange        TimeTicks,
        ifInOctets          Counter,
        ifInUcastPkts       Counter,
        ifInNUcastPkts      Counter,
        ifInDiscards        Counter,
        ifInErrors          Counter,
        ifInUnknownProtos   Counter,
        ifOutOctets         Counter,
        ifOutUcastPkts      Counter,
        ifOutNUcastPkts     Counter,
        ifOutDiscards       Counter,
        ifOutErrors         Counter,
        ifOutQLen           Gauge
}
```

2.2.5 Définition d'un module MIB

Dans les paragraphes précédents, nous avons présenté et défini les objets MIB utilisés pour représenter les ressources d'une entité gérée par un système de gestion des réseaux. En plus, nous avons défini les tables MIB qui servent à regrouper un ensemble d'objets. Avec ces deux notions (objet et table), on peut commencer à écrire un MIB. Tout est structuré en groupes d'objets. La façon dont cette structure est représentée est l'objet de cette sous-section.

Une spécification d'un MIB SNMP est structurée en trois sections:

1. Description des thèmes: cette section fait la correspondance entre les ressources

gérées et les définitions des informations de gestion contenues dans le(s) module(s) MIB. Elle permet aussi d'expliquer l'interaction entre les objets d'un MIB complexe.

2. Module(s) du MIB: cela constitue la partie principale d'un MIB. Un module est écrit dans la syntaxe ASN.1.

3. Références: cette section fournit la liste de toutes les sources pouvant aider à mieux comprendre les objets gérés.

Un module MIB est constitué des sections suivantes (figure 10):

1. Délimiteurs (<*moduleName*> et END): ils permettent de nommer le module et de le délimiter des autres modules.

2. Section de liaison(<*importedItems*>): dans cette section sont spécifiés les éléments définis dans d'autres modules et utilisés dans le module courant.

3. Identité du module (<*moduleIdentityDefinition*>): cette section identifie les modules MIB.

4. Définitions(<*definitions*>): elle contient les définitions de tous les objets et groupes d'objets à gérer, faites au moyen des macros. L'ensemble de ces macros est le suivant:

- *IMPORTS*: elle est utilisée pour spécifier des éléments définis dans d'autres modules MIB et qui sont utilisés par le module courant. Voici un exemple d'utilisation de cette macro:

```
IMPORTS
   Counter, Gauge        -- éléments à importer
      FROM RFC1155-SMI   -- source d'importation
   OBJECT-TYPE           -- élément à importer
      FROM RFC-1212      -- source d'importation
```

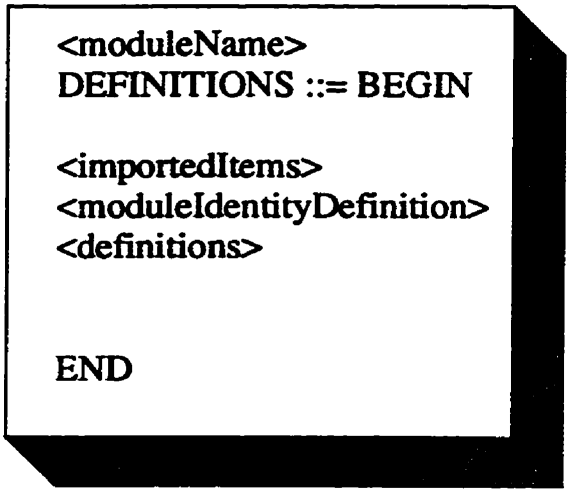


FIG. 10: *Principales sections constituant un module MIB.*

- *MODULE-IDENTITY*: elle est utilisée pour spécifier des informations relatives aux modules MIB SNMP. Ceci inclut l'historique de révision des modules, le nom de l'organisation qui a défini le module, l'adresse du groupe de support technique relative à ce module et une description de haut niveau des modules. Cette macro doit être utilisée une fois et doit être placée au début du module. Voici un exemple:

```
linuxMIB MODULE-IDENTITY
    LAST-UPDATED "9607182024Z"
    ORGANIZATION "TU Braunschweig"
    CONTACT-INFO
            "       Juergen Schoenwaelder
             Postal: TU Braunschweig
                     Bueltenweg 74/75
                     D-38108 Braunschweig
                     GERMANY
                Tel: +49 531 391 3249
                Fax: +49 531 391 5936
             E-mail: schoenw@ibr.cs.tu-bs.de"
    DESCRIPTION
```

```
            "Experimental MIB modules for the linux operating system."
    ::= { enterprises tubs(1575) ibr(1) 5 }
```

- *OBJECT-IDENTITY*: permet de spécifier un identificateur dans un module MIB. Une valeur OID est toujours affectée à cette macro. En voici un exemple:

```
mgmt OBJECT-IDENTITY
   STATUS       current
   DESCRIPTION
      "..."
   REFERENCE
      "..."
   ::= { iso org(3) dod(6) internet(1) 2 }
```

- *TEXTUAL-CONVENTION*: elle est utilisée pour créer un nouveau type de données. Ceci est fait en ajoutant d'autres règles d'utilisation à un type de données de base ou à un type de données déjà défini. Voici un exemple:

```
-- Ajout d'autres règles d'utilisation à un type
-- de données de base (OCTET STRING)
DisplayString ::= TEXTUAL-CONVENTION
   DISPLAY-HINT   "255a"   -- DisplayString ne doit contenir que l'un
   -- des 255 caractères ASCII.
   STATUS         current
   DESCRIPTION
      "..."
   SYNTAX         OCTET STRING (SIZE(0..255))

-- Ajout d'autres règles d'utilisation à un type
-- de données déjà défini (DisplayString)
UcDisplayString ::= TEXTUAL-CONVENTION
   DISPLAY-HINT   "255a"
   STATUS         current
   DESCRIPTION
```

```
      "..."
   SYNTAX          DisplayString

-- Ajout d'autres règles d'utilisation à un type
-- de données déjà défini (INTEGER)
TruthValue ::= TEXTUAL-CONVENTION
   STATUS          current
   DESCRIPTION
      "..."
   SYNTAX          INTEGER { true(1), false(2) }
```

- *OBJECT-TYPE*: elle est utilisée pour définir un objet. Plus de détails sur ce sujet sont donnés dans la section 2.2.3.

- *SEQUENCE* et *SEQUENCE OF*: la macro SEQUENCE OF est utilisée pour définir les rangées d'une table. Par contre, la macro SEQUENCE est utilisée pour spécifier les objets colonne d'une rangée d'une table. La section 2.2.4 aborde plus en détail le sujet des tables.

- *NOTIFICATION-TYPE*: elle est utilisée pour spécifier les événements qu'un agent peut rapporter à des stations de gestion. Voici un exemple d'utilisation:

```
coldStart NOTIFICATION-TYPE
   STATUS       current
   ENTERPRISE   snmp
   DESCRIPTION
      "..."
   ::= {snmpTraps 1}
```

- *OBJECT-GROUP*: elle est utilisée pour définir un ensemble de types d'objets interreliés. Ceci a pour but de mentionner qu'une relation existe entre un ensemble d'objets, ce qui permet une meilleure organisation des modules MIB. En voici un

exemple:

```
ipGroup OBJECT-GROUP
   OBJECTS { ipForwarding, ipDefaultTTL, ipInReceives,
             ipInHdrErrors, ipInAddrErrors,
             <le reste des éléments>
           }
   DESCRIPTION
      "..."
   ::= {ipMIBGroups 1}
```

- *NOTIFICATION-GROUP*: c'est l'équivalent de la macro OBJECT-GROUP, mais pour les événements émis par un agent. Voici un exemple d'utilisation:

```
bcoSprinklerEventGroup NOTIFICATION-GROUP
   NOTIFICATIONS { bcoSprBrEv, bcoSprPrEv }
   STATUS        current
   DESCRIPTION
      "..."
   ::= {bcoSprinklerEventGroups 7}
```

- *MODULE-COMPLIANCE*: cette macro permet de définir les aspects obligatoires lors de l'implantation d'un ou de plusieurs modules MIB. Voici un exemple de définition:

```
snmpMIBCompliance MODULE-COMPLIANCE
   STATUS        current
   DESCRIPTION   "..."
   MODULE        RFC1213-MIB
      MANDATORY-GROUPS { system }
   MODULE --This module
      MANDATORY-GROUPS { snmpStatsGroup, snmpORGroup,
                         snmpTrapGroup, snmpSetGroup }
   GROUP snmpV1Group
```

```
    DESCRIPTION  "..."
::= { snmpMIBCompliances 1 }
```

Le champ MODULE est utilisé une ou plusieurs fois pour mentionner chaque module requis. Chaque section MODULE spécifie les groupes d'objets nécessaires (MANDATORY-GROUPS) et les groupes d'objets optionnels pour l'implantation du module. Pour qu'une implantation soit conforme aux spécifications, elle doit implanter tous les objets appartenant aux groupes mentionnés dans le champ MANDATORY-GROUPS. Pour chaque groupe qui est conditionnellement obligatoire ou optionnel, il y a spécification d'un champ GROUP. Le champ DESCRIPTION décrit les conditions sous lesquelles un groupe d'objets est obligatoire (par exemple si un protocole particulier est implanté ou si un autre groupe est défini).

- *AGENT-CAPABILITIES*: cette macro est utilisée pour documenter les fonctionnalités d'un agent SNMP. La macro spécifie des raffinements ou des modifications relatifs aux macros OBJECT-TYPE définies dans les modules MIB. Voici un exemple d'utilisation de cette macro:

```
exemple-agent AGENT-CAPABILITIES
   PRODUCT-RELEASE      "ACME agent release 1.1 for 4BSD"
   STATUS               current
   DESCRIPTION          "agent ACME pour 4BSD"
   SUPPORTS             RFC1213-MIB
      INCLUDES          {system, interfaces, at, ip,
                         icmp, tcp, udp, snmp}

      VARIATION          ifAdminStatus
      SYNTAX             INTEGER {up(1), down(2)}
      DESCRIPTION        "..."

      VARIATION          ifOperStatus
```

```
        SYNTAX              INTEGER {up(1), down(2)}
        DESCRIPTION         "..."

        VARIATION           atEntry
        CREATION-REQUIRES   {atPhysAddress}
        DESCRIPTION         "..."

        VARIATION           ipDefaultTTL
        SYNTAX              INTEGER {255..255}
        DESCRIPTION         "..."

        VARIATION           ipInAddrErrors
        ACCESS              not-implemented
        DESCRIPTION         "..."

        VARIATION           ipRouteType
        SYNTAX              INTEGER {direct(3), indirect(4)}
        WRITE-SYNTAX        INTEGER {invalid(2), direct(3), indirect(4)}
        DESCRIPTION         "..."

        VARIATION           tcpConnState
        ACCESS              read-only
        DESCRIPTION         "..."

    SUPPORTS                EVAL-MIB
        INCLUDES            {functions, expressions}
        VARIATION           exprEntry
        CREATION-REQUIRES   {evalString}
        DESCRIPTION         "..."
::= { acme-agents 1 }
```

Le champ PRODUCT-RELEASE mentionne la version de l'agent décrit par le

champ DESCRIPTION. Le reste de la macro contient une section pour chaque module MIB pour lequel l'agent réclame une implémentation complète ou partielle. La description de chaque module commence par un champ SUPPORTS qui identifie le module. Ensuite, le champ INCLUDES spécifie la liste des groupes d'objets appartenant à ce module et qui sont implantés par l'agent. Enfin, pour chaque groupe supporté, est spécifiée la liste des objets implantés par l'agent, d'une façon différente de ce qui a été défini par la macro OBJECT-TYPE.

Par exemple, l'agent *exemple-agent* décrit ci-haut implante les modules MIB-II et EVAL-MIB. Le module MIB-II inclut tous les groupes sauf *egp*. Cependant, il y a un raffinement au niveau de la syntaxe pour les objets suivants: ifAdminStatus, ifOperStatus, atEntry, ipDefaultTTL, ipInAddrErrors, ipRouteType et tcpConnState. Enfin, on remarque que cet agent n'implante pas l'objet ipInAddrErrors.

2.3 Le groupe expérimental

Nous avons déjà mentionné qu'un MIB SNMP est constitué d'un ou de plusieurs modules. Chaque module est composé d'un ensemble de macros. Parmi ces macros, on distingue OBJECT-TYPE, qui permet de définir un objet correspondant à une ressource à gérer. La structure d'un MIB est bien organisée. Tous les objets ayant une relation particulière sont rassemblés en dessous d'un groupe d'objets particulier. Par exemple, les objets utilisés pour gérer les connexions TCP sont regroupés sous le groupe *TCP*. Pour tous ceux qui sont en phase d'expérimentation, les objets ainsi définis doivent être inscrits sous un groupe particulier, appelé *experimental*. Ainsi, la structure des groupes d'objets reste identique pour tous les MIB, chose qui permet à n'importe quelle application de gestion de pouvoir accéder aux objets avec précision (évidemment seulement si elle emploie le bon nom de communauté. La notion de nom de communauté est présentée dans le chapitre 3).

Dans le cadre de notre projet de maîtrise, c'est sous ce groupe d'objets que tous les objets que nous avons définis sont inscrits. Plus de précisions sur ces objets sont données dans le chapitre 4.

2.4 Les compilateurs MIB

Les compilateurs MIB sont équivalents à des traducteurs. Ils servent à produire des rapports, des structures de données ou des descriptions de MIB appropriées pour des stations de gestion.

Les compilateurs MIB s'adressent à deux catégories d'utilisateurs: les auteurs des MIB, qui ont besoin d'un outil leur permettant de s'assurer que leur MIB est syntaxiquement correct, et les programmeurs des applications de gestion et des agents, qui ont besoin d'outils pour leur simplifier la génération de code.

Un compilateur MIB a une structure semblable à celle des compilateurs des langages de programmation (tels que C et Pascal). Dans le cas de SNMP, les compilateurs sont classés en deux catégories: les compilateurs *front-end* et les compilateurs *back-end* (figure 11). La première catégorie traduit un fichier MIB en un format intermédiaire. Ce dernier est utilisé par la deuxième catégorie qui produit le résultat final.

Durant la phase de compilation *front-end*, deux étapes sont parcourues. Lors de l'analyse syntaxique, le compilateur effectue une vérification de la conformité aux structures standard des MIB. La génération du code intermédiaire produit un deuxième format du contenu du MIB. Actuellement, il n'existe aucun standard du format et du contenu de ce genre de fichier intermédiaire. Ce dernier peut être un simple fichier ASCII ou une autre forme de fichier plus complexe (tel que des enregistrements à la façon d'une base de données).

Les compilateurs *back-end* ont pour paramètre d'entrée le résultat de la phase précédente. Ils effectuent des tâches de haut niveau. Ceci inclut la présentation d'une structure

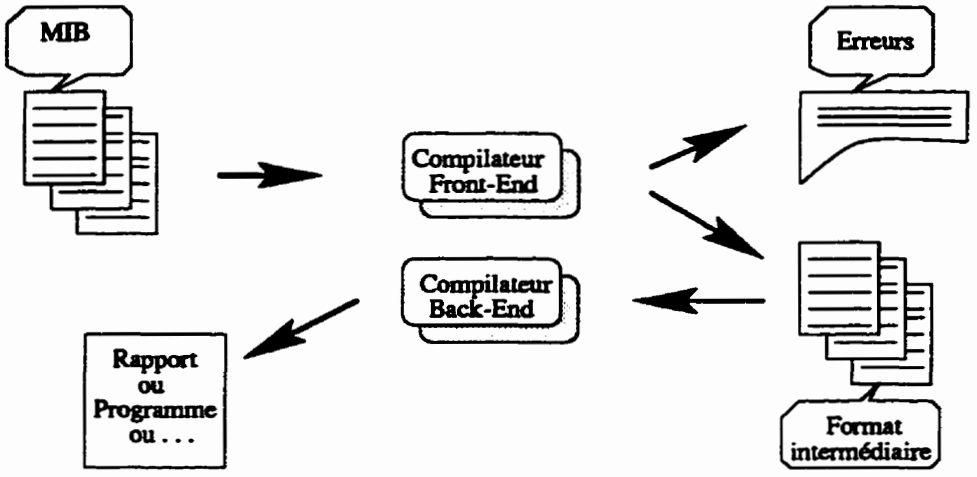


FIG. 11: *Structure d'un compilateur MIB.*

arborescente graphique du MIB, la création de structures en langage C (qui peuvent être utilisées pour écrire des applications de gestion ou des agents) et la production d'un code pour des applications de gestion et pour des agents.

Il existe plusieurs compilateurs de MIB. Parmi eux, on distingue *SMIC* et *MOSY*. SMIC est développé par Bay Networks [10]. Une version améliorée a été développée par David T. Perkins (dperkins@scruznet.com). Cette version est fonctionnelle sur les systèmes d'exploitation suivants: Sun Solaris1 et Sun Solaris2, IBM AIX et OS2, Hewlett-Packard HP-UX, Linux et Microsoft DOS et Windows NT. Ce compilateur vérifie la syntaxe et la grande majorité des règles sémantiques d'un module MIB. SMIC est aussi utilisé pour générer des formats de MIB accessibles en tant que pages WEB. De l'autre côté, MOSY reçoit des fichiers MIB et génère un fichier intermédiaire qui peut être utilisé par des compilateurs *back-end*. Cependant, il n'est presque plus utilisé depuis l'apparition de SMIC. La référence [18] contient un chapitre équivalent à un manuel d'utilisation de ces deux compilateurs.

2.5 Conclusion

Les systèmes de gestion des réseaux maintiennent une base de données, appelée MIB, pour enregistrer des informations relatives aux dispositifs à gérer. La compréhension de la structure des MIB est nécessaire pour pouvoir développer des applications de gestion basées sur SNMP. Dans ce chapitre, nous avons expliqué les notions de base d'une telle base de données. En premier lieu, nous avons présenté la structure de l'information de gestion. Cette dernière définit les règles à suivre pour spécifier et implanter un MIB. Ce dernier est une collection d'objets identifiés par un OID. En plus, chaque objet est défini au moyen d'une macro nommée OBJECT-TYPE. Les éléments complexes à spécifier sont définis par des tables. Une table comprend un ensemble de rangées constituées de groupes d'objets. L'accès à une rangée particulière est fait au moyen d'objet(s) index. Nous avons aussi vu comment écrire un fichier MIB. Cela est réalisé grâce à des modules MIB. Dans l'avant-dernière section, nous avons spécifié un groupe particulier d'objets (*experimental*) requis pour toutes les applications de gestion voulant accéder aux objets en cours d'expérimentation. Enfin, nous avons présenté deux compilateurs MIB (SMIC et MOSY) pouvant servir, par exemple, à effectuer une analyse syntaxique et sémantique d'un MIB.

Chapitre 3

Le protocole SNMP

Ce chapitre donne un aperçu du protocole SNMP. En premier lieu, nous présentons ses concepts de base. Ensuite, nous abordons ses spécifications. Le format ainsi que l'utilité de chacun des messages SNMP sont expliqués. Enfin, un aperçu de la version 2 du même protocole est donné. Seules les améliorations apportées avec cette nouvelle version seront mentionnées.

3.1 Concepts de base

Dans cette section, nous examinons les opérations de gestion supportées par SNMP. Ensuite, nous donnons un aperçu des noms de communautés utilisés avec chaque opération de gestion. Enfin, nous voyons comment les instances des objets gérés sont identifiées.

3.1.1 Opérations supportées par SNMP

Le protocole SNMP traite les informations de gestion comme des variables. Les opérations de SNMP extraient ou modifient les valeurs de ces variables. Les informations de gestion sont classées selon leur type. Chaque classe (ou type) d'information est nommée

objet ou *type d'objet*. Une instance d'une certaine classe est appelée *variable* ou *instance d'objet*. Un objet SNMP peut avoir une ou plusieurs instances. Deux vocables sont utilisés pour désigner ces deux types d'objets: un *objet scalaire* possède une seule instance, alors qu'un *objet colonne* possède zéro, une ou plusieurs instances (figure 12).

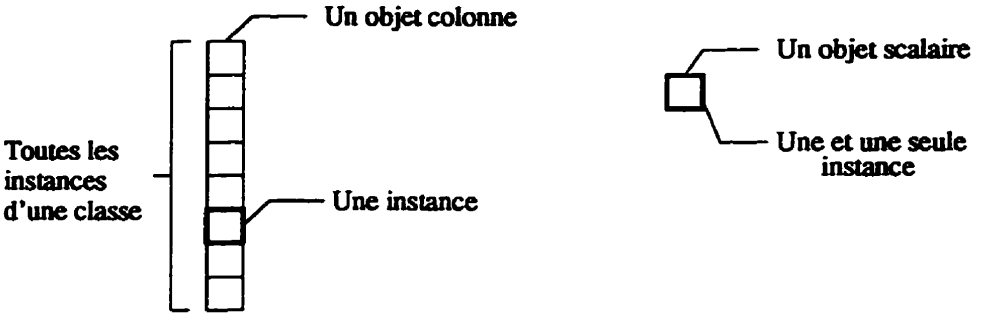


FIG. 12: *Types d'objets reconnus par SNMP.*

Un exemple d'objet scalaire est l'emplacement d'un système de gestion (vu qu'un tel système ne peut posséder qu'un seul emplacement à un moment donné). Un exemple d'objet colonne est l'état des connexions TCP. Il peut y avoir zéro, une ou plusieurs connexions à un système donné à n'importe quel moment.

Dans le cas de SNMP, les seules opérations applicables sur les objets scalaires sont les suivantes:

1. *Get*: permet à une station de gestion d'extraire la valeur d'un objet scalaire de chez un agent;
2. *Set*: permet à une station de gestion de modifier la valeur d'un objet scalaire chez un agent;
3. *Trap*: permet à un agent d'informer une station de gestion qu'un événement critique vient de se produire.

SMI propose d'organiser les objets colonnes dans des tables. Tous les objets colonnes d'une table utilisent un même mécanisme pour identifier les instances d'une certaine

classe d'objets. Ce mécanisme est appelé *schéma d'indexation.* Il n'y a aucune opération SNMP permettant d'avoir comme résultat une table entière, une colonne ou une rangée d'une table. Seules les valeurs des variables constituant les tables peuvent être retournées comme résultat.

3.1.2 Communautés et noms de communautés

La gestion des réseaux peut être vue comme étant une application distribuée. Elle implique une interaction entre un ensemble d'entités supportées par un protocole d'application. Dans le cas de SNMP, les entités correspondent aux applications de gestion au niveau des stations et au niveau des agents. Une application de gestion des réseaux requiert des interactions entre une station de gestion et un ensemble d'agents (figure 13). La station est capable d'extraire et de modifier la valeur des objets situés dans le MIB d'un agent et peut recevoir à son tour des réponses et des Trap.

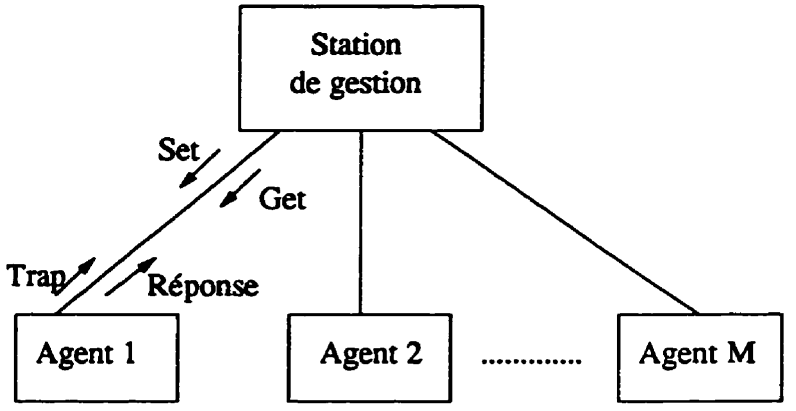


FIG. 13: *Messages échangés entre une station de gestion et un ensemble d'agents.*

On peut voir aussi la gestion des réseaux comme une relation entre un agent et un ensemble de systèmes de gestion (figure 14). Dans ce cas, chaque agent gère et contrôle l'accès à son MIB pour l'ensemble des stations de gestion. Ce contrôle possède trois aspects: un service d'authentification, une politique d'accès et un service de procuration.

Tous ces aspects sont reliés à la sécurité du système. Les agents doivent se protéger contre les accès non permis à leurs MIBs. SNMP fournit ce type de sécurité sous la forme de communauté. Une communauté SNMP est une relation entre un agent et un ensemble de stations de gestion, qui définit une authentification, un contrôle d'accès et des caractéristiques de procuration. Le concept de communauté est local, c'est-à-dire qu'il ne figure que du côté agent. Pour des buts d'identification, un nom est attribué à chaque communauté. En plus, chaque station faisant partie d'une communauté donnée doit utiliser ce nom lors des opérations Get et Set. Vu que les noms de communautés sont définis localement chez un agent, le même nom de communauté peut se retrouver chez différents agents. C'est pour cette raison qu'une station de gestion doit garder trace du ou des noms de communautés qui sont associés à chaque agent avec qui elle communique.

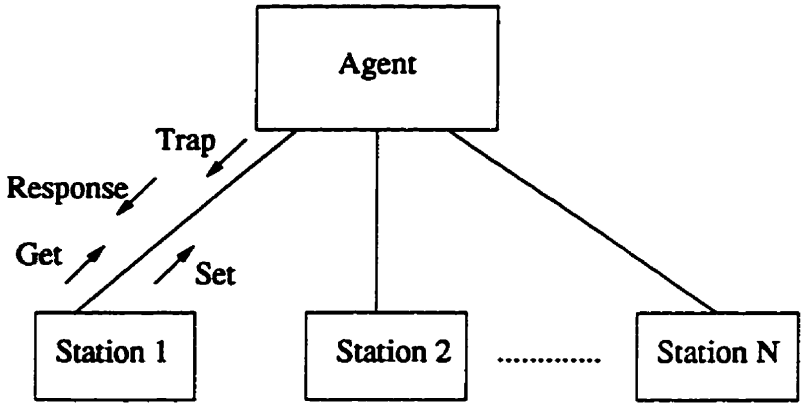


FIG. 14: *Communication entre un agent et plusieurs stations.*

Le rôle de chaque service utilisé par l'agent pour contrôler l'accès à son MIB est le suivant:

1. **Service d'authentification:** il permet à l'agent de donner accès à son MIB seulement aux stations autorisées. Chaque message provenant d'une station de gestion inclut un nom de communauté. Ce nom est semblable à un mot de passe et le message est considéré authentique seulement si la station de gestion utilise le bon

nom de communauté. Dans le cas contraire, l'accès au MIB est refusé.

2. **Politique d'accès:** c'est un service fourni par l'agent aux différentes stations de gestion, qui définit un ensemble de privilèges d'accès au MIB. Pour cela, nous introduisons les notions suivantes:

   - **Vue:** elle correspond à un sous-ensemble d'objets dans le MIB. Différentes vues sont définies pour chaque communauté;
   - **Mode d'accès:** on distingue deux modes: *read-only* et *read-write.*

   La combinaison d'une vue du MIB et d'un mode d'accès forme un *profil de communauté SNMP.* Ainsi, un profil de communauté définit le mode d'accès à un certain sous-ensemble d'objets figurant dans le MIB d'un agent. Un profil est associé à chaque communauté définie par un agent. La combinaison d'une communauté et d'un profil de communauté est connue sous le nom de politique d'accès.

3. **Service de mandatement:** l'utilisation de SNMP suggère que tous les agents, de même que les stations, supportent les protocoles UDP et IP. Alors que faire dans le cas des systèmes et des dispositifs n'utilisant ou ne supportant pas ces deux protocoles? La solution à employer consiste à passer à travers des agents mandataires. Un agent mandataire joue le rôle de traducteur de protocoles. Il reçoit toutes les requêtes provenant d'une station SNMP, puis les traduit en un autre format utilisé par l'agent destinataire. La figure 15 illustre ce mécanisme. La fonction de correspondance permet de traduire chaque paquet provenant d'une station SNMP dans le format utilisé par l'agent du dispositif. L'architecture des protocoles utilisés par des agents mandataires regroupe toutes les architectures des protocoles utilisés par les différents dispositifs ne supportant ni UDP ni IP.

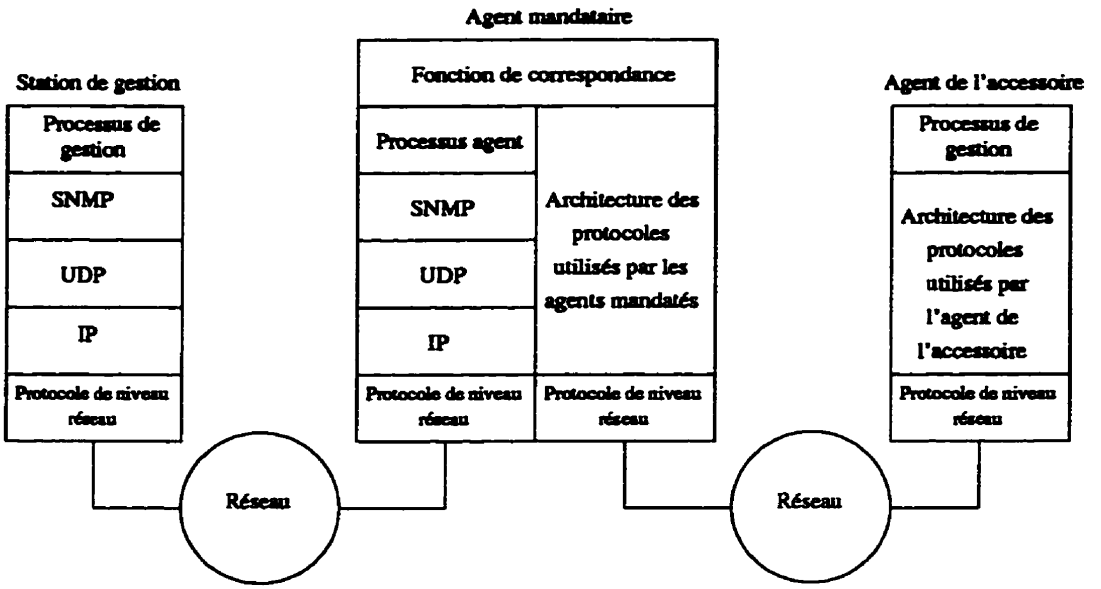


FIG. 15: *Configuration d'un agent mandataire.*

3.1.3 Identification des instances

Un schéma est utilisé par SNMP pour identifier les entités faisant partie du système de gestion. Un identificateur dans ce schéma est appelé *identificateur d'objet* (Object Identifier, OID) et l'identité d'une entité est déterminée par la valeur de son OID. L'affectation d'une valeur OID à une entité est appelée *enregistrement.* Une fois que l'enregistrement a été effectué, aucune autre entité ne peut être inscrite avec la même valeur d'OID. En plus, les caractéristiques de l'entité déjà enregistrée ne peuvent plus être changées et l'entité ne peut plus être supprimée.

Par définition, la valeur d'un OID est une séquence ordonnée d'entiers non négatifs, écrits de gauche à droite et contenant au moins deux éléments. Les valeurs des OID sont organisées de façon hiérarchique (comme un système de fichiers). Une structure arborescente est toujours utilisée pour illustrer les numéros qui apparaissent dans la séquence et qui correspondent aux valeurs des OID (figure 9, page 27). À chaque noeud de l'arbre correspond un numéro. Le noeud x de la figure 9 a comme valeur d'OID {1 1 1} . SNMP restreint la longueur de la valeur d'un OID à 128 numéros dans la séquence. Il

impose aussi une valeur maximale de $(2^{31}-1)$ pour chaque nombre utilisé. Voici quelques exemples de valeurs d'OID écrites dans un module MIB ainsi que les valeurs qui leur sont associées dans une application de gestion:

```
{ iso org(3) dod(6) internet(1) } est 1.3.6.1
{ 1 3 6 1 mgmt(2) mib(1) 2 } est 1.3.6.1.2.1.2
{ system 1 } est 1.3.6.1.2.1.1.1
{ internet 4 } est 1.3.6.1.4
```

Après avoir défini les identificateurs d'objets, voyons maintenant les variables SNMP. Nous avons déjà mentionné (section 3.1.1) qu'une instance d'une information de gestion correspond à une variable. L'identité d'une variable est basée sur l'identité de sa classe et de son identification au sein de la classe (figure 16). L'identité d'une classe d'objets est une valeur OID. Par exemple, l'identité de la classe *sysLocation*, un objet scalaire, est la valeur OID { 1 3 6 1 2 1 1 6 }. L'identité de la classe *tcpConnState*, un objet colonne, est la valeur OID { 1 3 6 1 2 1 6 13 1 1 }. Pour les objets appartenant à des tables, leur identificateur ne suffit pas pour identifier l'instance, vu qu'il y a une instance de chaque objet pour chaque rangée de la table.

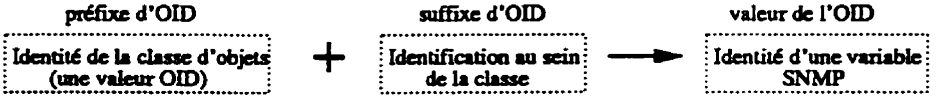


FIG. 16: *Identification d'une variable SNMP.*

La technique utilisée par SNMP pour identifier une instance spécifique à un objet figurant dans une table est nommée *schéma d'indexation.* En effet, une table est constituée d'un ensemble de rangées (zéro ou plus). Chaque rangée contient le même ensemble de types d'objets scalaires (ou objets colonnes). Chaque objet colonne possède un identificateur qui est le même dans chaque rangée. Par exemple, en examinant la figure 17, on remarque qu'il y a trois instances de *tcpConnEntry*, mais toutes de même identificateur d'objet: 1.3.6.1.2.1.6.13.1. La combinaison d'un identificateur d'objet pour un objet

colonne avec un ensemble de valeurs des objets index (utilisé pour distinguer une rangée d'une autre) permet de spécifier un objet scalaire particulier dans une rangée particulière de la table. Dans le cas de SNMP, la convention utilisée pour identifier un objet colonne d'une table est de joindre l'identificateur de l'objet scalaire avec les valeurs des objets index.

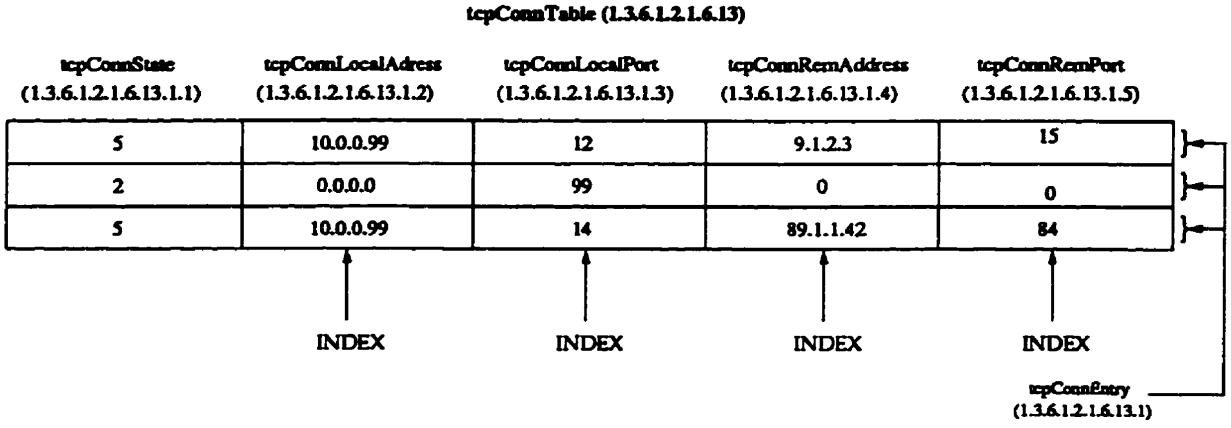


FIG. 17: *Les instances de la table des connexions TCP.*

Prenons un exemple simple. Soit la table *ifTable* appartenant au groupe *interface* (figure 19). Il n'y a qu'un seul objet index: *ifIndex*. Supposons qu'on veuille savoir le type de la deuxième interface d'un système donné. L'identificateur d'objet de *ifType* est 1.3.6.1.2.1.2.2.1.3. La valeur de *ifIndex* à laquelle on s'intéresse est 2. L'identificateur d'instance de *ifType* correspondant à la rangée contenant une valeur de *ifIndex* de 2 est: 1.3.6.1.2.1.2.2.1.3.2. Il suffit d'ajouter la valeur de *ifIndex* à la fin de l'identificateur d'instance.

Un autre exemple, plus complexe, est celui de *tcpConnTable* du groupe TCP (figure 18). Cette table possède quatre objets index. Un identificateur d'instance pour n'importe quelle colonne (au nombre de cinq) de la table est composé de l'identificateur de cet objet, combiné avec les valeurs d'une rangée particulière des quatre objets index.
Par exemple, tout identificateur d'instance pour tcpConnTable est de la forme suivante:

`x.i.(tcpConnLocalAddress).(tcpConnLocalPort).(tcpConnRemAddress).(tcpConnRemPort)`

où:

x = 1.3.6.1.2.1.6.13.1 = identificateur d'objets de tcpConnEntry;

i = le dernier entier de l'identificateur d'objet, pour un objet colonne.

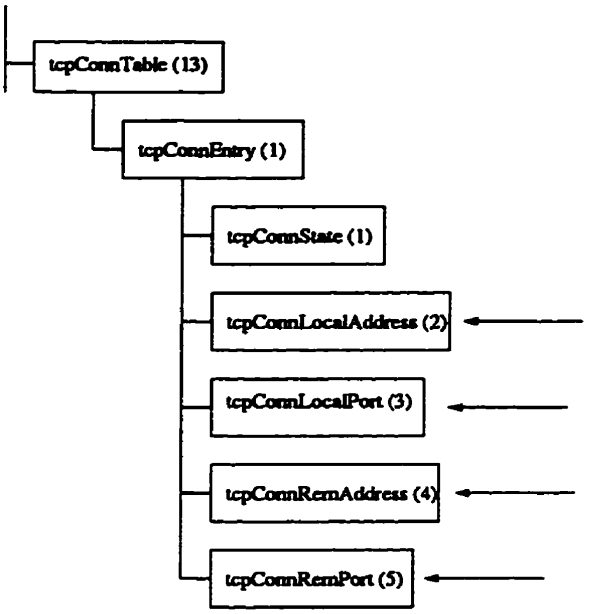


FIG. 18: *Le groupe TCP.*

En général, étant donné un objet dont l'identificateur est y, dans une table avec des objets index *i1, i2, ..., iN*, l'identificateur d'instance, pour une instance de l'objet y dans une rangée donnée est *y.(i1).(i2)...(iN).*

3.2 Spécifications du protocole

Dans cette section, nous nous proposons d'examiner le format des messages SNMP. Ensuite, nous allons décrire le PDU correspondant à chaque message.

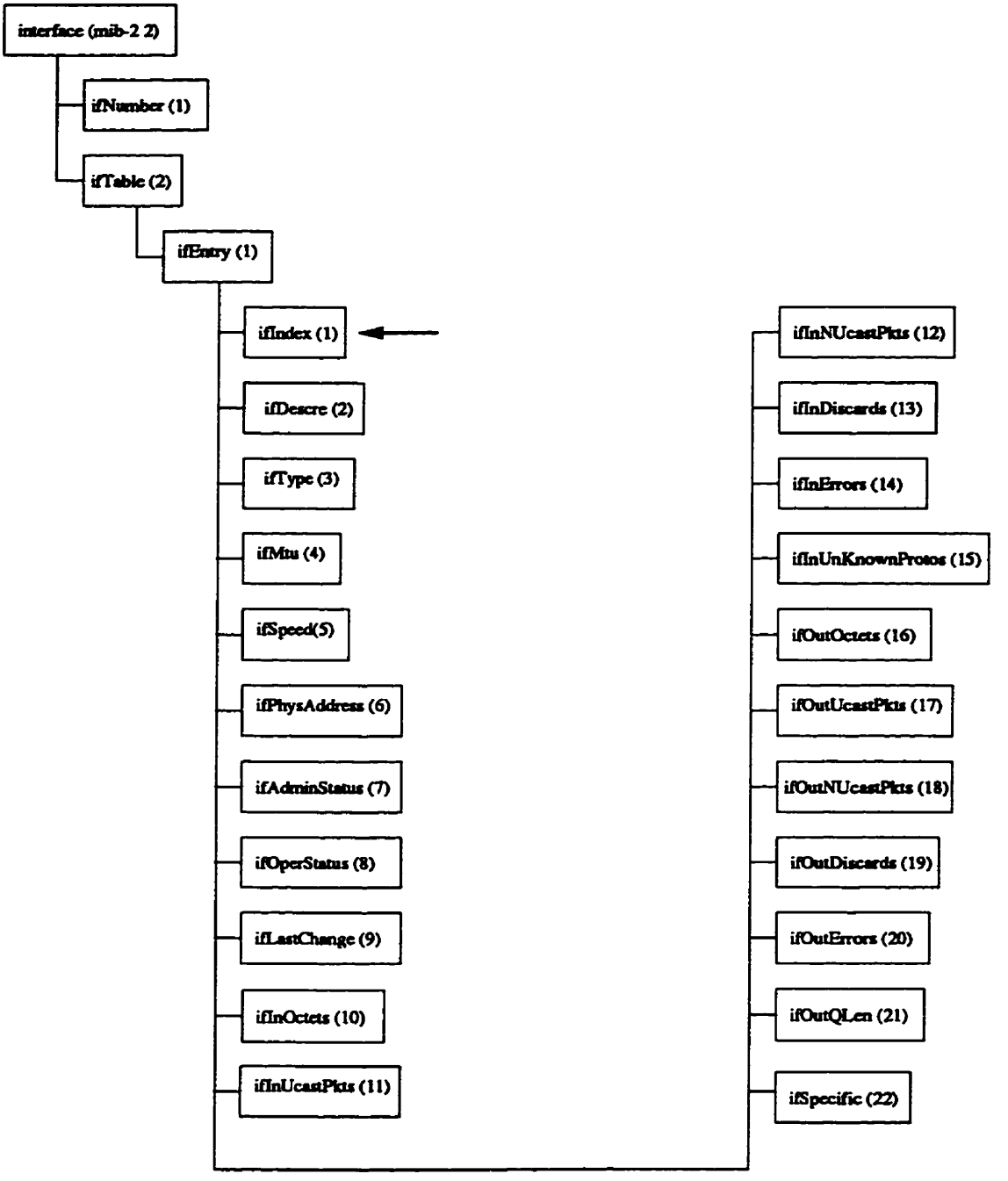


FIG. 19: *Groupe d'interfaces MIB-II.*

3.2.1 Le format des messages SNMP

Avec SNMP, les informations de gestion sont échangées entre une station de gestion et un agent (station gérée) sous la forme de messages. Chaque message (figure 20.a) intègre le numéro de version de SNMP, un nom de communauté à utiliser pour cet échange et un PDU parmi les cinq définis pour SNMP (figures 20.b, 20.c et 20.d). La signification de chaque champ est donnée dans la table 2.

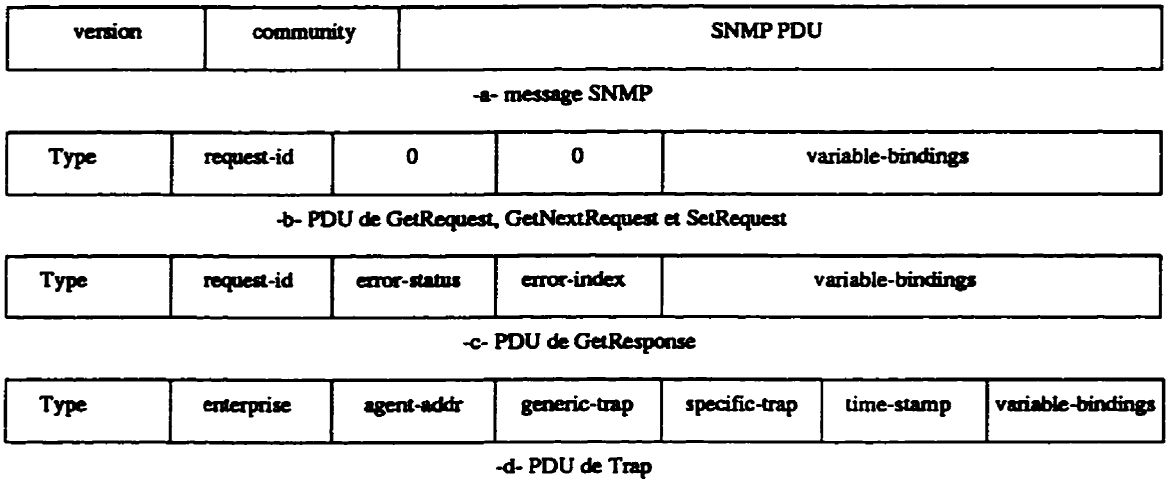


FIG. 20: *Format des messages SNMP.*

1. Transmission d'un message SNMP

   La transmission des PDU se fait entre entités (que ce soit d'une station vers un agent ou l'inverse). Les actions suivantes sont effectuées lors de la transmission d'un PDU:

   (a) le PDU est d'abord construit, en utilisant les règles de codage d'ASN.1;

   (b) ce PDU est ensuite passé à un service d'authentification, avec l'adresse transport de la source et de la destination et un nom de communauté. Le service

Champ	*Description*
version	version de SNMP.
community	il est utilisé comme un mot de passe pour les messages SNMP.
request-id	utilisé pour distinguer entre les demandes provenant d'une station de gestion.
error-status	utilisé pour indiquer qu'une anomalie est survenue lors d'une demande. Les valeurs de ce champ sont: noError (0), tooBig (1), noSuchName (2), badValue (3), readOnly (4) et genErr (5).
error-index	lorsque la valeur de error-status est non nulle, ce champ indique la variable qui a provoqué l'anomalie.
variable-bindings	la liste des noms de variables ainsi que leurs valeurs.
enterprise	le type d'objet qui a généré un Trap.
agent-addr	l'adresse de l'objet qui a généré un Trap.
generic-trap	type du Trap. Ses valeurs sont: coldStart (0), warmStart (1), linkDown (2), linkUp (3), authenticationFailure (4), egpNeighborLoss (5) et enterpriseSpecific (6).
specific-trap	code du Trap spécifique au fabricant du matériel.
time-stamp	il contient la valeur de l'objet sysUpTime

TAB. 2: *Les champs utilisés dans un message SNMP.*

d'authentification effectue les transformations nécessaires pour cet échange, telles que le chiffrement ou l'inclusion d'un code d'authentification, puis retourne le résultat;

(c) l'entité du protocole construit par la suite un message (figure 20.a) constitué d'un champ de version, d'un nom de communauté et du résultat de l'étape (b);

(d) ce nouveau message est ensuite codé puis passé au service de transport, qui

va le livrer à l'agent SNMP spécifié.

2. Réception d'un message SNMP

   À la suite de la réception d'un message, l'entité SNMP passe par les étapes suivantes:

   (a) elle vérifie la syntaxe du message et le détruit si l'analyse échoue;

   (b) elle vérifie le numéro de version et détruit le message s'il y a une erreur de compatibilité de version;

   (c) le nom de communauté et la portion PDU du message sont passés à un service d'authentification:

   - si l'authentification échoue, un Trap est généré et le message est détruit;
   - si l'authentification réussit, un PDU est retourné;

   (d) le PDU retourné est vérifié de façon syntaxique. Il est détruit dans le cas où la vérification échoue. Autrement, et en utilisant le nom de communauté, la politique d'accès appropriée est sélectionnée et le PDU est traité.

En pratique, le service d'authentification sert simplement à vérifier que le nom de communauté autorise la réception des messages provenant de l'entité SNMP source. C'est en répétant ces deux scénarios que la communication entre deux entités est établie et que l'échange des messages est effectué.

Voyons maintenant l'utilité de chaque type de message (ou PDU) ainsi que la signification de chaque champ utilisé dans chaque PDU.

1. **Le PDU GetRequest**

   Ce message est envoyé par une entité SNMP en faveur d'une station de gestion dans

le but d'extraire une information de chez le MIB d'un agent. L'entité émettrice inclut les champs suivants dans ce message:

- *type*: indique qu'il s'agit d'un GetRequest. Sa valeur est égale à 'a0'h.
- *request-id*: l'entité émettrice affecte des nombres à ce champ de façon à identifier chaque paquet émis au même agent. Ce champ permet à une application SNMP de corréler les réponses avec les demandes. Aussi, il permet à une entité SNMP de se défaire des PDU doubles générés par un service de transport non fiable.
- *variable-bindings*: il contient la liste des instances d'objet dont les valeurs sont demandées.

L'entité SNMP réceptrice répond à un GetRequest par un GetResponse contenant le même champ request-id. GetResponse est atomique. Deux cas de figure sont possibles: soit que toutes les valeurs demandées soient extraites, soit qu'aucune ne le soit. Si l'agent est capable de donner les valeurs de toutes les variables mentionnées dans le champ *variable-bindings* du message reçu (GetRequest), alors GetResponse contient les valeurs de chaque variable. Par contre, il suffit qu'une variable ne puisse pas être fournie pour qu'aucune valeur ne soit retournée. Voici les cas d'erreurs possibles:

(a) l'objet mentionné dans le champ *variable-bindings* ne coïncide avec aucun identificateur d'objet dans le MIB de l'agent ou l'objet nommé correspond à un type agrégé ne possédant pas d'instance associée. Dans ce cas, l'agent envoie un GetResponse dans lequel le champ *error-status* possède la valeur *noSuchName* et le champ *error-index* la valeur correspondant à l'index d'objet du champ *variable-bindings* qui est en cause.

(b) l'agent est capable d'extraire les valeurs de toutes les variables de la liste, sauf

que la taille du PDU résultant dépasse une contrainte locale. Dans ce cas, GetResponse contient la valeur *tooBig* dans le champ *error-status*.

(c) l'agent n'est pas en mesure de fournir la valeur pour au moins un objet pour d'autres raisons. Dans ce cas, GetResponse contient une valeur *genErr* dans le champ *error-status* et le champ *error-index* contient la valeur correspondant à l'index d'objet mentionné dans le champ *variable-bindings* qui est en cause.

2. **Le PDU GetNextRequest**

Du point de vue format, un message GetNextRequest est presque identique à un GetRequest. La seule différence entre eux est la suivante: pour un GetRequest, GetResponse retourne la valeur des instances des objets demandés. Pour un GetNextRequest, GetResponse retourne la valeur de la prochaine instance de chaque objet mentionné. De même que GetRequest, GetNextRequest retourne soit toutes les valeurs des variables mentionnées dans le champ *variable-bindings*, soit aucune. Puisque l'agent ne peut retourner que la valeur de l'instance d'un objet simple (et non d'un objet agrégé, tel qu'une branche de l'arbre ou une table entière), GetNextRequest réfère toujours à la prochaine instance et non au prochain objet.

GetRequest permet à la station de gestion de déterminer dynamiquement la structure d'une vue du MIB. En plus, il peut être utilisé pour découvrir une table dont les entrées sont inconnues.

Voici les principaux cas d'utilisation de GetRequest et de GetNextRequest:

(a) **Extraire la valeur d'un objet**: Supposons qu'une station de gestion veuille obtenir la valeur de tous les objets du groupe UDP chez un agent (figure 21). Pour cela, elle envoie un GetRequest de la façon suivante:

```
GetRequest(udpInDatagrams.0, udpNoPorts.0, udpInErrors.0, udpOutDatagrams.0)
```

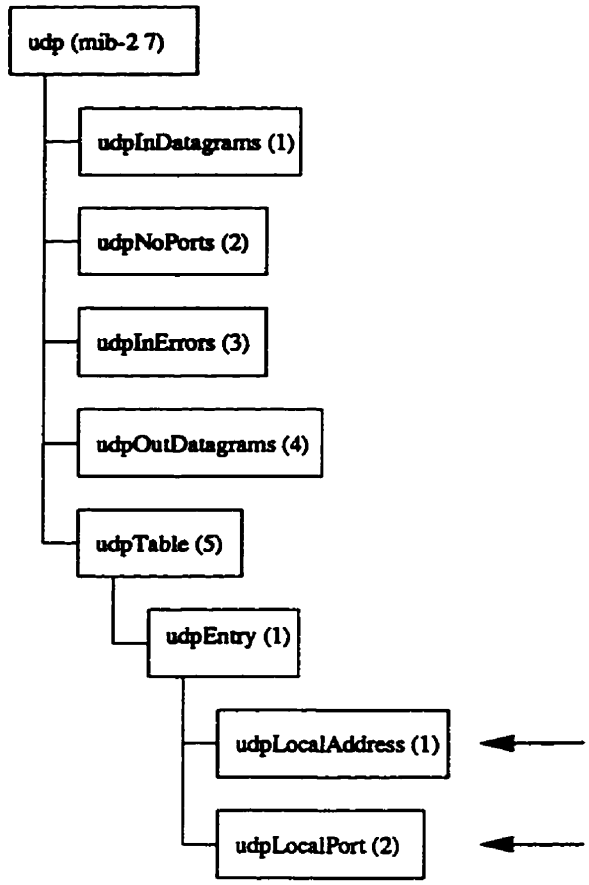


FIG. 21: *Le groupe UDP du MIB-II.*

Si la vue du MIB pour cette communauté comporte tous ces objets, alors un GetResponse est retourné avec la valeur des quatre objets en question, par exemple:

```
GetResponse((udpInDatagrams.0 = 100),(udpNoPorts.0 = 1),(udpInErrors.0 = 2),
        (udpOutDatagrams.0 = 200))
```

Cependant, si l'un des objets n'est pas supporté, alors un GetResponse sera retourné avec un code d'erreur. Pour s'assurer d'obtenir des réponses correctes (sous forme de valeurs valides), la station de gestion doit émettre séparément quatre GetRequest (pour *udpInDatagrams*, *udpNoPorts*, *udpInErrors* et *udpOutDatagrams*). Maintenant, si on applique GetNextRequest pour les mêmes

objets:

```
GetNextRequest(udpInDatagrams, udpNoPorts, udpInErrors, udpOutDatagrams),
```

alors l'agent retourne la valeur de la prochaine instance pour chaque identificateur de la liste. L'identificateur d'objet de udpInDatagrams est 1.3.4.1.2.1.7.1. Le prochain identificateur d'instance est udpInDatagrams.0 (ou 1.3.4.1.2.1.7.1.0). De la même façon, le prochain identificateur d'instance de udpNoPorts est udpNoPorts.0 (ou 1.3.4.1.2.1.7.2.0), celui de udpInErrors est udpInErrors.0 (ou 1.3.4.1.2.1.7.3.0) et celui de udpOutDatagrams est udpOutDatagrams.0 (ou 1.3.4.1.2.1.7.4.0). Si toutes les valeurs sont visibles, alors l'agent retourne le GetResponse suivant:

```
GetResponse((udpInDatagrams.0 = 100),(udpNoPorts.0 = 1),(udpInErrors.0 = 2),
            (udpOutDatagrams.0 = 200))
```

qui est le même que précédemment. Si on suppose que *udpNoPorts* n'est pas visible et que le même GetNextRequest est émis, alors la réponse est la suivante:

```
GetResponse((udpInDatagrams.0 = 100),(udpInErrors.0 = 2),(udpInErrors.0 = 2),
            (udpOutDatagrams.0 = 200))
```

Malgré que udpNoPorts.0 ne soit pas visible dans la vue du MIB, l'agent retourne la valeur de la prochaine instance, qui est dans ce cas *udpInErrors.0* (au lieu de *udpNoPorts.0*).

(b) **Extraire la valeur d'un objet inconnu**: GetNextRequest demande à un agent de retourner la valeur de la prochaine instance qui apparaît juste après l'identificateur d'objet mentionné comme paramètre. Il n'est cependant pas nécessaire que l'identificateur mentionné représente un objet existant ou une instance. Par exemple, puisque *udpInDatagrams* est un objet scalaire, alors il n'existe aucun objet dont l'identificateur est *udpInDatagrams.1*. Par ailleurs, si cet identificateur est mentionné dans un GetNextRequest, alors l'agent concerné va simplement chercher le prochain identificateur valide. La valeur

retournée sera *udpNoPorts.0* (ou 1.3.6.1.2.1.7.2.0). C'est de ce fait que GetNextRequest peut être utilisée pour découvrir la structure d'une table ou même d'un MIB entier (en appliquant répétitivement GetRequest et GetNextRequest).

Par exemple, si la station de gestion émet un GetNextRequest(udp), alors la réponse est un GetResponse(udpInDatagrams.0 = 100). Par la suite, la station sait que le prochain objet supporté dans cette vue du MIB est le *udpInDatagrams* et elle obtient en même temps la valeur courante de cet objet.

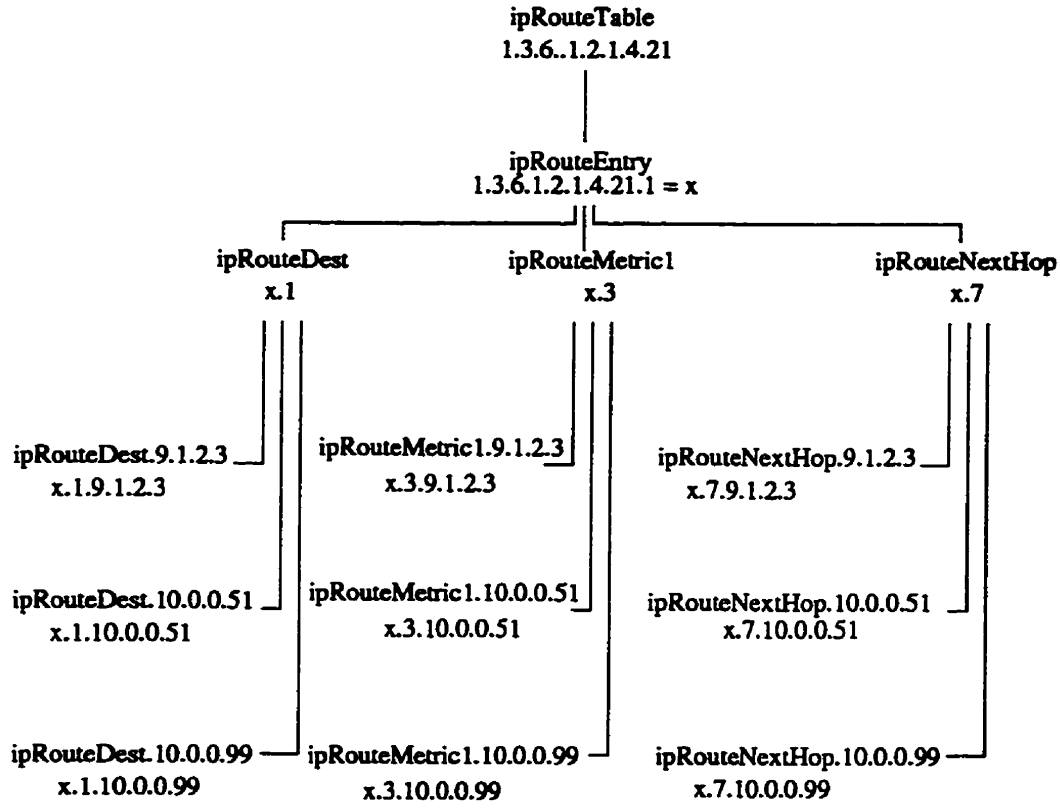


FIG. 22: *Les objets et les instances d'objets de la table ipRouteTable.*

(c) **Accéder aux valeurs d'une table**: Un GetNextRequest peut être utilisé

pour déterminer les entrées d'une table. Considérons l'exemple de la figure 22. La table est constituée de trois rangées (*ipRouteDest*, *ipRouteMetric1* et *ipRouteNextHop*). Supposons que les valeurs de chaque rangée soient celles du tableau 3.

ipRouteDest	ipRouteMetric1	ipRouteNextHop
9.1.2.3	3	99.0.0.3
10.0.0.51	5	89.1.1.42
10.0.0.99	5	89.1.1.42

TAB. 3: *Exemple de valeurs affectées aux instances de la table ipRouteTable.*

Supposons que la station de gestion veuille extraire la table entière et qu'elle ne connaisse ni son contenu ni le nombre de ses rangées. Dans ce cas, elle peut émettre un GetNextRequest contenant juste les noms des objets colonnes:

```
GetNextRequest(ipRouteDest, ipRouteMetric1, ipRouteNextHop)
```

L'agent répond avec les valeurs de la première rangée de la table:

```
GetResponse((ipRouteDest.9.1.2.3 = 9.1.2.3),(ipRouteMetric1.9.1.2.3 = 3),
            (ipRouteNextHop.9.1.2.3 = 99.0.0.3))
```

La station peut par la suite enregistrer ces valeurs (9.1.2.3, 3 et 99.0.0.3), puis extraire la prochaine rangée en passant comme argument à *GetNextRequest* le résultat du précédent GetResponse:

```
GetNextRequest(ipRouteDest.9.1.2.3, ipRouteMetric1.9.1.2.3,
               ipRouteNextHop.9.1.2.3)
```

L'agent SNMP répond par:

```
GetResponse((ipRouteDest.10.0.0.51 = 10.0.0.51),(ipRouteMetric1.10.0.0.51 = 5),
            (ipRouteNextHop.10.0.0.51 = 89.1.1.42))
```

Ensuite, à la requête:

```
GetNextRequest(ipRouteDest.10.0.0.51, ipRouteMetric1.10.0.0.51,
               ipRouteNextHop.10.0.0.51)
```

on a:

```
GetResponse((ipRouteDest.10.0.0.99 = 10.0.0.99),
            (ipRouteMetric1.10.0.0.99 = 5),
            (ipRouteNextHop.10.0.0.99 = 89.1.1.42))
```

La station de gestion ne sait pas qu'il s'agit maintenant de la dernière rangée de la table. Donc elle continue:

```
GetNextRequest(ipRouteDest.10.0.0.99, ipRouteMetric1.10.0.0.99,
               ipRouteNextHop.10.0.0.99)
```

Puisqu'il n'y a plus de rangée dans la table, l'agent répond avec les objets qui suivent dans la vue du *MIB*:

```
GetResponse((ipRouteDest.1.9.1.2.3 = 9.1.2.3),(ipRouteNextHop.9.1.2.3 = 3),
            (ipNetToMediaifIndex.1.3 = 1))
```

Puisque les noms des objets mentionnés dans la réponse ne sont pas tous les mêmes que ceux demandés, la station de gestion conclut qu'elle a atteint la fin de la table. Ainsi, GetNextRequest a pu servir à découvrir le nombre des rangées de la table et la valeur de chaque instance.

3. Le PDU de SetRequest

Un SetRequest est envoyé par une entité SNMP en faveur d'une station de gestion. Du point de vue format, ce PDU est semblable à celui de GetRequest, sauf que SetRequest est utilisé pour modifier la valeur d'un objet plutôt que de la lire. Le champ *variable-bindings* de SetRequest contient les identificateurs des instances d'objet et leurs valeurs correspondantes. L'entité SNMP réceptrice répond à un SetRequest par un GetResponse contenant le même champ *request-id*. Si l'agent est capable de modifier les valeurs de toutes les variables du champ *variable-bindings*, alors le GetResponse retourné indique la nouvelle valeur affectée à chaque variable modifiée. Par contre, si au moins une des valeurs des variables ne peut être modifiée,

alors aucune valeur n'est modifiée. Un SetRequest peut retourner les mêmes messages d'erreur qu'un GetRequest (*noSuchName*, *tooBig* et *genErr*). On peut avoir avec SetRequest un autre message d'erreur: *badValue*. Cette erreur est retournée si le SetRequest contient au moins une variable non cohérente. L'incohérence peut être dans le type, la longueur ou la valeur de l'objet.

Parmi les fréquentes applications de SetRequest, on distingue la mise à jour d'une table et la suppression d'une rangée d'une table:

(a) **Mise à jour d'une table**: prenons à nouveau l'exemple de la figure 22 et supposons que les valeurs de la table 3 existent. Si la station de gestion produit un SetRequest(ipRouteMetric1.9.1.2.3 = 9), alors la réponse suivante est obtenue: GetResponse(ipRouteMetric1.9.1.2.3 = 9). Cette requête a permis de modifier la valeur de *ipRouteMetric1* de la première rangée. Supposons maintenant que la station de gestion veuille ajouter une nouvelle rangée à la table dont les valeurs des objets *ipRouteDest*, *ipRouteMetric1* et *ipRouteNextHop* sont respectivement 11.3.3.12, 9 et 91.0.0.5. Pour cela, elle doit produire un:

```
SetRequest((ipRouteDest = 11.3.3.12), (ipRouteMetric1 = 9),
          (ipRouteNextHop = 91.0.0.5))
```

L'objet colonne *ipRouteDest* correspond à l'objet index de la table. La valeur de *ipRouteDest.x* est *x*, la valeur de l'index des identificateurs des objets colonnes. Maintenant, puisque la valeur 11.3.3.12 est affectée à *ipRouteDest*, le nom de l'objet est ipRouteDest.11.3.3.12. Il s'agit d'un identificateur d'instance non reconnu par l'agent. Dans ce cas, l'agent peut réagir de l'une des trois façons suivantes:

i. refuser l'opération et retourner un GetResponse avec une valeur *NoSuchName* dans le champ *errorStatus*.

ii. traiter cette opération comme une demande pour créer une nouvelle rangée, mais il s'aperçoit que l'une des valeurs à affecter est non appropriée.

Dans ce cas, il retourne une erreur de type *badValue*.

iii. accepter l'opération et créer une nouvelle rangée, ce qui donne une table à quatre rangées au lieu de trois. Dans ce cas, l'agent retourne:

```
GetResponse((ipRouteDest = 11.3.3.12), (ipRouteMetric1 = 9),
            (ipRouteNextHop = 91.0.0.5))
```

SNMP n'indique pas quelle décision l'agent doit prendre, mais laisse le choix au programmeur d'implanter le cas qui convient le mieux à son système.

(b) **Suppression d'une rangée d'une table**: on peut appliquer SetRequest pour la suppression d'une rangée d'une table. Dans le cas de *ipRouteDest*, on peut utiliser SetRequest(ipRouteDest.7.3.5.3 = invalid). Dans ce cas, on a une réponse de la forme:

```
GetResponse(ipRouteDest.7.3.5.3 = invalid)
```

L'effet de SetRequest dans ce cas est la suppression logique de la rangée de la table indexée par la valeur 7.3.5.3 de ipRouteDest.

4. Le PDU Trap

Un *Trap* est émis par une entité *SNMP* en faveur de la station de gestion du réseau. Il est utilisé pour signaler à la station une indication relative à un événement significatif. Les différents champs d'un Trap sont les suivants:

- *type*: il indique qu'il s'agit d'un Trap. Sa valeur est égale à 'a4'h;
- *enterprise*: il identifie l'agent qui a engendré le *Trap*. Sa valeur est prise de *sysObjectID* du groupe *system*;
- *agent-addr*: il s'agit de l'adresse de l'objet qui a généré le *Trap*;
- *generic-Trap*: c'est l'un des types prédéfinis des *Trap*, décrits ci-après;
- *specific-Trap*: un code qui indique plus spécifiquement la cause du *Trap*;

- *time-stamp*: le temps entre la dernière (re)initialisation de l'entité du réseau qui a déclenché le *Trap* et la génération du *Trap* courant;
- *variable-bindings*: des informations additionnelles reliées au *Trap*.

Le *generic-Trap* peut prendre l'une des valeurs suivantes:

(a) *coldStart (0)*: il s'agit d'un démarrage à froid. Ceci peut se produire lorsque la configuration de l'agent ou du protocole implanté est modifiée. Le démarrage à froid est non désirable car il peut affecter le contenu du MIB.

(b) *warmStart (1)*: l'entité SNMP émettrice du Trap se fait initialiser. Ceci peut se produire lorsque la configuration de l'agent ou de l'entité du protocole implanté est modifiée. Typiquement, il s'agit d'une routine de redémarrage.

(c) *linkDown (2)*: signale une panne dans l'un des liens de communication de l'agent. Le premier élément du champ *variable-bindings* représente le nom et la valeur de l'instance *ifIndex* de l'interface en question.

(d) *linkUp (3)*: signale qu'un lien de communication vient de se créer. Le premier élément du champ *variable-bindings* représente le nom et la valeur de l'instance *ifIndex* de l'interface référencée.

(e) *authenticationFailure (4)*: signale que l'entité émettrice du message a reçu un message qui n'a pu être authentifié.

(f) *egpNeighborLoss (5)*: signale qu'un protocole d'une passerelle externe (External Gateway Protocol, EGP) n'est plus visible et que la liaison avec elle n'existe plus.

(g) *enterpriseSpecific (6)*: signale que l'entité émettrice s'est aperçu qu'un événement spécifique à un manufacturier est survenu. Le champ *specific-Trap* indique le type d'événement.

3.3 Le protocole SNMPv2

SNMPv2 est une extension du protocole SNMP. La version 1 de SNMP est facile à implanter et ne demande pas beaucoup de ressources. Suite à l'expérience avec SNMP, plusieurs aspects ont fait l'objet d'améliorations. Ceci inclut les opérations nécessaires pour extraire de grandes quantités d'information à partir d'un MIB donné, des compteurs de grande capacité, des codes d'erreurs plus raffinés et un meilleur degré de sécurité. De tous ces aspects, celui de la sécurité était le plus attendu de la part des utilisateurs, sauf qu'il est le plus difficile à définir et que ce qui a été fait répond en partie seulement aux attentes [15].

Les améliorations apportées avec cette nouvelle version sont les suivantes:

1. le message GetBulk: il permet la lecture de grandes quantités d'informations, telles que les tables.
2. le message Inform: il permet la communication entre stations de gestion.
3. un nouveau format de Trap: les Trap de la version 2 sont semblables aux autres PDU, ce qui les rend flexibles.
4. de nouveaux types de données: six nouveaux types de données ont été ajoutés pour exploiter, principalement, les modes 32 bits et 64 bits.
5. des codes d'erreur améliorés: ceci élimine l'aspect atomique des requêtes de la première version.

Dans la suite de cette section, nous allons présenter ces nouveaux aspects plus en détail.

3.3.1 Le PDU GetBulkRequest

L'une des nouveautés apportées avec SNMPv2 est associée à ce type de messages. GetBulkRequest a pour but de minimiser le nombre de messages échangés pour lire un

grand nombre d'informations de gestion. Ce genre de PDU utilise le même principe de sélection que celui de GetNextRequest. C'est-à-dire que la sélection se fait dans un ordre lexicographique sur la prochaine instance d'objet. La différence entre eux, c'est qu'avec GetBulkRequest il est possible que plusieurs successeurs lexicographiques soient sélectionnés.

GetBulkRequest fonctionne de la manière suivante. Il inclut une liste de noms de variables, dans le champ *variable-bindings*, de longueur (N+R). Pour les N premiers noms de variables, la façon d'extraire leur valeur est identique à celle de GetNextRequest. Pour les R derniers noms de variables, plusieurs successeurs lexicographiques sont retournés. Dans un PDU de type GetBulkRequest, deux nouveaux champs ont été ajoutés: *nonrepeaters* et *max-repetitions*. Le champ *nonrepeaters* indique le nombre de variables de la liste *variable-bindings* pour lesquelles un unique successeur lexicographique est à retourner (la valeur N). Le champ *max-repetitions* spécifie le nombre de successeurs lexicographiques à retourner pour les variables appropriées de la liste *variable-bindings*. Pour éclaircir l'algorithme, nous introduisons les termes suivants:
L = le nombre de noms de variables dans la liste *variable-bindings*;
N = le nombre de variables, à partir de la première de la liste *variable-bindings*, pour lesquelles un unique successeur lexicographique est demandé;
R = le nombre de variables, juste après les N premières, pour lesquelles plusieurs successeurs lexicographiques sont demandés;
M = le nombre des successeurs lexicographiques requis pour chacune de ces R variables.
La relation entre ces différents paramètres est donnée dans la figure 23.

Voici un exemple d'application de GetBulkRequest. Supposons qu'on veuille extraire la valeur du premier successeur de l'objet *sysUpTime*, appartenant au groupe *system*, et la valeur des deux successeurs de chacun des objets *ipNetToMediaPhysAddress* et *ipNetToMediaType*, appartenant au groupe *ip*. Dans ce cas, on peut employer GetBulkRequest

avec les paramètres suivants:

```
GetBulkRequest [nonrepeaters=1, max-repetitions=2]
               (sysUpTime, ipNetToMediaPhysAddress, ipNetToMediaType)
```

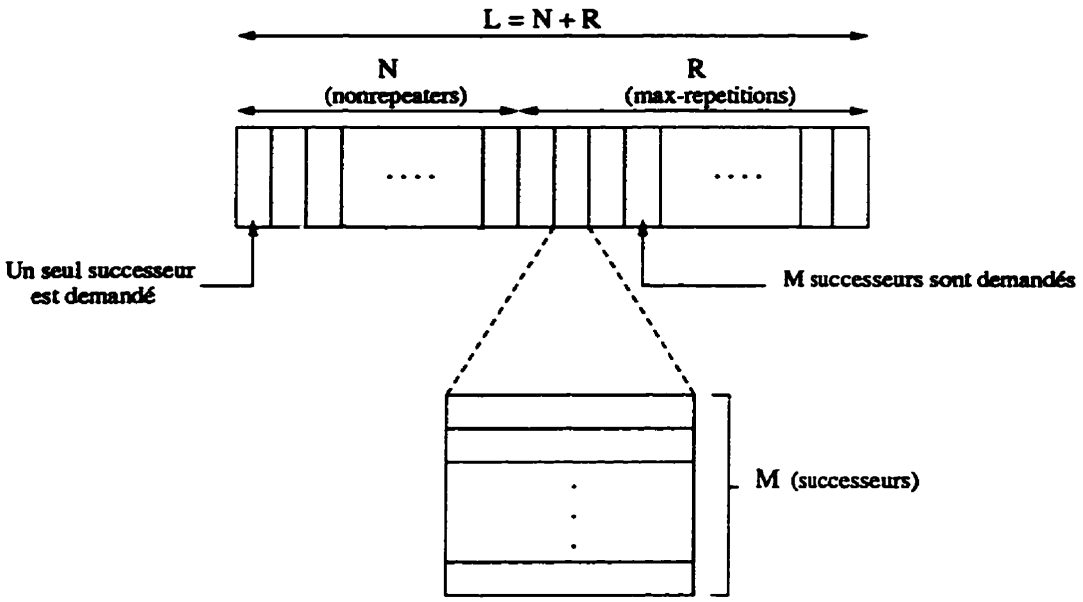


FIG. 23: *Relation entre les paramètres du champ variable-bindings du PDU GetBulkRequest.*

Ainsi, on a spécifié que N=1, M=2 et L=3. On peut déduire, par la suite, que R=2. Une fois reçue par l'agent, cette requête est traitée comme suit (tout en supposant que le MIB de l'agent contienne une vue des objets demandés): l'agent extrait la valeur de la première instance de l'objet *sysUpTime*. Ceci correspond à l'instance *sysUpTime.0*. Supposons que la valeur qui lui correspond est "123446". La partie de la requête concernant les N premières instances (dans ce cas N=1, donc une seule instance) est traitée. Par la suite, l'agent passe au traitement des R objets restant. Pour chacun d'eux (*ipNetToMediaPhysAddress*, *ipNetToMediaType*), les M premières instances (avec M=2) sont extraites. Voici un exemple de requête qui peut être échangée entre une station de gestion et un agent:

```
GetBulkRequest [nonrepeaters=1, max-repetitions=2]
               (sysUpTime, ipNetToMediaPhysAddress, ipNetToMediaType)
```

L'agent répond par:

```
GetResponse ((sysUpTime.0 = "123446"),
             (ipNetToMediaPhysAddress.1.9.2.3.4 = "000010543210"),
             (ipNetToMediaType.1.9.2.3.4 = "dynamic"),
             (ipNetToMediaPhysAddress.1.10.0.0.51 = "000010012345"),
             (ipNetToMediaType.1.10.0.0.51 = "static"))
```

Ensuite, la station peut émettre:

```
GetBulkRequest [nonrepeaters=1, max-repetitions=2]
               (sysUpTime, ipNetToMediaPhysAddress.1.10.0.0.51,
               ipNetToMediaType.1.10.0.0.51)
```

et l'agent répond par:

```
GetResponse ((sysUpTime.0 = "123446"),
             (ipNetToMediaPhysAddress.2.10.0.0.15 = "000010987654"),
             (ipNetToMediaType.2.10.0.0.15 = "dynamic"),
             (ipNetToMediaNetAddress.1.9.2.3.4 = "9.2.3.4"),
             (ipRoutingDiscards.0 = "2"))
```

La station de gestion conclut qu'elle a atteint la fin de la table à cause du changement des noms des instances des deux dernières instances retournées.

Trois remarques sont à noter. D'abord, si le processus de détermination de la valeur d'une variable échoue pour d'autres raisons que celle de *endOfMibView*, alors aucune valeur n'est retournée. Plutôt, un message d'erreur est envoyé, dans lequel est mentionné un *genError* dans le champ *error-status* et le champ *error-index* contient une valeur correspondant à l'index de l'objet en question. De plus, si la longueur du message à retourner dépasse une limite locale ou la taille maximale du message de la source, alors la réponse est envoyée avec une taille plus petite que celle attendue. Enfin, si au cours de la boucle décrite ci-haut toutes les *variable-bindings* ont une valeur de *endOfMibView*, alors la liste *variable-bindings* peut être réduite en ce point.

De cette façon, on peut appliquer GetBulkRequest séquentiellement pour extraire une table de grande taille. Il est évident que GetBulkRequest sera invoquée beaucoup moins de fois que GetNextRequest pour effectuer la même tâche.

Parmi les avantages de GetBulkRequest, on distingue principalement:

- une réduction de la taille et de la complexité des applications supportées par le protocole de gestion.

- de plus, si une demande est très grande du point de vue du nombre de blocs de données, l'agent retourne le plus grand nombre de données possible, au lieu d'envoyer un message d'erreur *tooBig*. Dans ce cas, la station n'a qu'à envoyer une autre demande GetBulkRequest pour les données dont la demande de lecture vient d'échouer.

3.3.2 Le PDU InformRequest

Ce PDU est échangé entre stations dans le but de partager des informations de gestion. Ce type de PDU est envoyé aux destinations spécifiées dans l'objet *SNMPv2EventNotifyTable*, défini dans le MIB concernant les gestionnaires (manager-to-manager), ou bien aux destinations spécifiées par l'application. Le format des PDU est décrit dans la figure 24. La liste de *variable-bindings* contient les éléments suivants:

1. la première variable est *sysUpTime.0*;

2. la deuxième variable est *SNMPv2EventID.i*, qui contient l'identificateur d'objet du type d'événement;

3. d'autres variables, optionnelles, spécifiées par l'application.

Lorsqu'un message InformRequest est reçu, l'entité SNMPv2 réceptrice (station réceptrice) détermine en premier lieu la taille du message à retourner. Si elle dépasse une

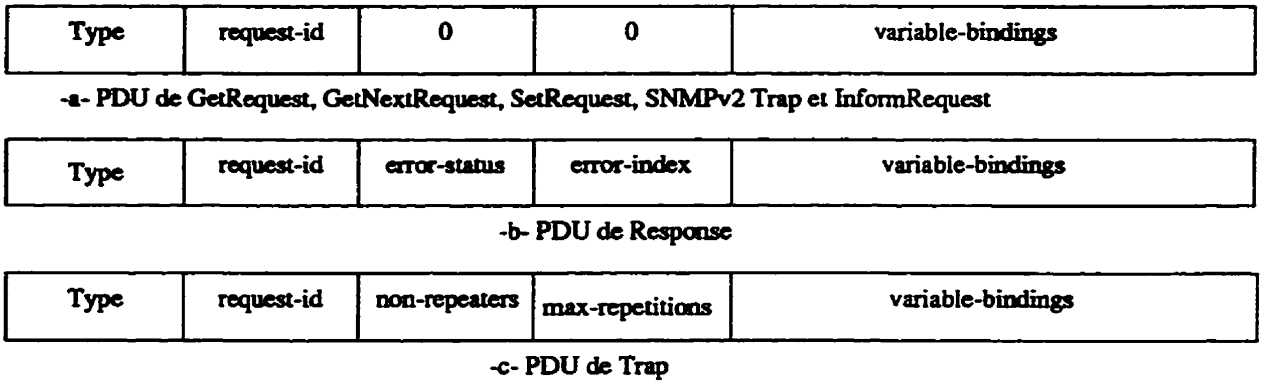


FIG. 24: *Le format des PDU de SNMPv2.*

limite locale ou la taille maximale d'un message, alors un PDU est émis avec une valeur *tooBig* dans le champ *error status*, une valeur nulle dans le champ *error-index* et un champ nul dans *variable-bindings*. Par contre, si le PDU reçu n'est pas trop grand, alors la station réceptrice fait passer son contenu à l'application destination, puis génère un PDU contenant les mêmes valeurs *request-id* et *variable-bindings* que celles du message InformRequest, avec un champ *error status* égal à *noError* et une valeur nulle dans le champ *error-index*.

3.3.3 Le PDU Trap

La syntaxe et la sémantique ne sont pas les mêmes entre les deux versions de SNMP. Dans la version SNMPv2, Trap a le même format que les autres PDU, sauf GetBulkRequest. Ceci a pour avantage de faciliter le traitement d'un message par le récepteur du message (la station de gestion). De même qu'avec SNMPv1, aucun accusé de réception n'est retourné à l'agent suite à l'émission d'un Trap.

3.3.4 Nouveaux types de données

SNMP utilise des types de données pour définir des modules MIB. Avec la version SNMPv2, quelques types ont été éliminés, d'autres ajoutés et d'autres figurent dans les deux versions. Les types ajoutés concernent principalement le support des données représentées sur 32 bits (*Integer32*, *Unsigned32*, *Gauge32*, *Counter32*), sur 64 bits (*Counter64*) et le pseudo-type BITS. Les nouveaux types de données définis par SNMPv2, ainsi que leurs descriptions, sont donnés dans la table 1 (section 2.2.2).

3.3.5 Codes d'erreur améliorés

Avec SNMP, si l'agent est capable de déterminer la valeur qui correspond à chaque variable de la liste *variable-bindings*, alors un GetResponse est émis. Cependant, si au moins une des valeurs n'a pu être identifiée, alors aucune valeur n'est retournée; c'est plutôt une erreur qui est envoyée. Avec SNMPv2, le champ *variable-bindings* est toujours rempli. S'il y a un problème avec l'une des variables, alors une indication est mentionnée dans le champ *value* de la variable en question, sans aucune influence sur les autres variables.

Par exemple, si une station veut extraire la valeur des objets x, y, z, alors elle émet un GetRequest(x, y, z). Supposons que l'agent concerné ne puisse pas retourner la valeur de l'objet y. Avec SNMPv1, si une valeur ne peut pas être identifiée (y), alors aucune valeur n'est retournée (ni x, ni z, bien que leurs valeurs puissent être extraites). Un GetResponse est retourné avec un code d'erreur. Dans le cas de SNMPv2, les valeurs des objets x et z sont retournées, alors que celle de y a un code d'erreur identifiant le type d'erreur qui a empêché l'extraction de la valeur du champ en question.

Avec SNMPv2, un GetRequest est construit selon les règles suivantes:

1. si la variable n'a pas un identificateur d'objet qui coïncide avec les variables accessibles par cette demande, alors son champ a la valeur *noSuchObject*;

2. sinon, si le nom de la variable ne peut coïncider exactement avec les variables accessibles par cette demande, alors la valeur de cette variable est *noSuchInstance*;

3. sinon, la valeur qui correspond au nom de la variable est retournée.

Les erreurs *genErr* et *tooBig* sont retournées dans les mêmes cas et de la même façon qu'avec SNMP (voir section 3).

3.3.6 Conclusion

Les systèmes de gestion des réseaux utilisent des protocoles standard qui permettent de fournir des services élémentaires servant à gérer à distance les ressources des réseaux. SNMP est le protocole le plus répandu dans ce domaine. Il fournit trois groupes d'opérations de gestion: Get, Set et Trap. Get permet d'extraire la valeur d'un objet. Deux opérations sont fournies dans ce but: GetRequest et GetNextRequest. L'emploi de ces deux opérations permet d'accéder à des objets scalaires et des objets colonnes (des tables). Le groupe d'opérations Set est employé pour modifier la valeur des objets. Ces deux groupes d'opérations (Get et Set) sont toujours employés par la station de gestion. Cependant, un agent ne peut invoquer que l'opération Trap. Celle-ci lui permet d'informer une station de gestion qu'un événement s'est produit.

En plus des opérations GetRequest, GetNextRequest, SetRequest et Trap, SNMPv2 fournit de nouveaux services pour résoudre les problèmes et les limitations posés par SNMPv1. D'abord, deux nouveaux messages ont été ajoutés: GetBulk, pour la lecture de grandes quantités d'informations, et Inform, qui permet la communication entre stations de gestion. De plus, six nouveaux types de données ont été ajoutés dans le but d'exploiter les modes 32 bits et 64 bits, et des codes d'erreurs améliorés ont été introduits pour éliminer le caractère atomique de la version précédente. Enfin, le format du message Trap a été modifié. Il est devenu semblable à celui des autres messages. Ceci donne de la flexibilité rendant le développement des applications de gestion moins complexe.

Chapitre 4

Partie expérimentale

Ce chapitre, qui comprend cinq sections, présente les résultats expérimentaux de notre projet de maîtrise. Dans la première section, nous décrivons brièvement l'environnement de notre application de gestion. Ceci comprend le matériel ainsi que le logiciel nécessaires pour gérer un *Terminal Node Controller* (TNC). Dans la deuxième section, nous identifions les paramètres tu TNC à gérer. La troisième section définit les objets à gérer (correspondant aux paramètres déjà identifiés). Dans la quatrième section, l'accès au MIB et la gestion du TNC sont abordés. Enfin, dans la dernière section, nous donnons une description générale des étapes à suivre pour implanter une application de gestion basée sur le système d'opération Linux et l'outil CMU-SNMP.

4.1 Introduction

Le but de notre projet de maîtrise est de présenter une nouvelle méthode pour la gestion à distance d'un dispositif de télécommunication appelé TNC. Ce dernier est un accessoire utilisé pour les communications sans fils et particulièrement par la communauté des radioamateurs. Cette dernière est constituée de personnes utilisant des fréquences radio spécifiques à des fins personnelles et pour l'expérimentation [11]. Comme le montre

la figure 25, un TNC est placé entre un ordinateur et un appareil radio. Sa fonction principale est d'accepter chaque paquet d'information provenant de l'ordinateur auquel il est attaché et de l'utiliser pour moduler un signal porteur audio qui est transmis à une radio branchée au TNC. Vice versa, il peut transformer les signaux audio provenant de l'appareil radio auquel il est attaché en des paquets de données qui sont transmis à l'ordinateur. Pour que l'échange des paquets de données entre stations radioamateurs soit fiable, il est nécessaire d'avoir un protocole de niveau liaison. Le protocole utilisé pour les communications radio par paquets est appelé AX.25 (Amateur X.25). Ce dernier est basé sur le protocole X.25, mais a été adapté pour répondre aux besoins des radioamateurs. Une description plus détaillée de ce protocole est donnée dans la référence [9].

Le système d'exploitation sur lequel est implanté notre outil de gestion est Linux. Il a été choisi vu qu'il offre des niveaux de sécurité semblables à ceux de Unix et qu'il supporte le protocole AX.25 et la gestion des réseaux. La référence [3] décrit la démarche à suivre pour installer sous Linux les modules de AX.25. CMU-SNMP est l'outil sur lequel nous nous sommes basé pour développer l'application de gestion du TNC. Cet outil est compatible avec SNMPv1 et supporte, en plus, quelques aspects de SNMPv2. Il est constitué principalement de quelques commandes de gestion et d'un agent. L'installation sous Linux de l'outil CMU-SNMP est décrite dans [1].

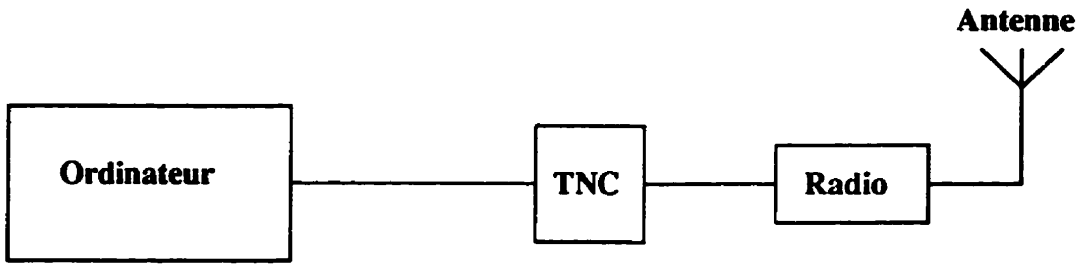


FIG. 25: *Composition d'un système de radio par paquets.*

4.2 Identification des paramètres du TNC

Le premier pas à faire, avant d'aborder la phase d'implantation des applications de gestion, est l'identification des paramètres du TNC à gérer. Celui que nous exploitons fonctionne en un mode appelé KISS [8]. Dans ce mode, les seuls paramètres qu'on peut gérer sont les suivants:

1. port: sous Linux, la communication avec le TNC se fait via un port bien défini. Ce paramètre permet d'identifier un port de communication parmi ceux qui sont configurés.

2. fullduplex: il permet de configurer le TNC en mode *fullduplex* ou *halfduplex*.

3. hardware: il permet de fixer quelques paramètres physiques du TNC.

4. persist: il permet de fixer la probabilité d'émettre une trame. Un nombre aléatoire variant entre 0 et 255 est généré chaque fois qu'une trame est prête à être émise. Si ce nombre est inférieur ou égal à la valeur de *persist*, alors la trame est émise. Sinon, la transmission est retardée.

5. slottime: il permet de choisir la durée du délai entre deux accès au canal de communication.

6. txDelay: c'est le délai entre l'activation de l'émetteur radio et l'émission des paquets de données. Le délai permet à la radio d'atteindre un état électroniquement stable avant de procéder à la transmission.

7. txtail: c'est le délai entre la fin de l'émission des paquets de données et la désactivation de l'émetteur radio.

4.3 Définition des objets correspondant aux paramètres du TNC

Les paramètres déjà identifiés peuvent être modifiés, mais ne peuvent être lus en mode KISS. Le MIB que nous développons permet aux applications de gestion de lire ces valeurs. Pour cela, nous avons associé à chaque paramètre un objet qui reflète sa valeur. En dessous du groupe *experimental* (figure 26), nous avons défini un sous-groupe appelé *internetExperim*. Ce dernier sert à grouper tous les objets expérimentaux relatifs à l'Internet. Ensuite, nous avons défini un autre sous-groupe d'objets appelé *tncControl*. Ce dernier fait partie du groupe *internetExperim* et permet de regrouper les objets associés aux paramètres du TNC à gérer. La définition de ces deux sous-groupes est la suivante:

```
experimental
   OBJECT IDENTIFIER ::= { internet 3 }
internetExperim
   OBJECT IDENTIFIER ::= { experimental 4711 }
tncControl
   OBJECT IDENTIFIER ::= { internetExperim 1 }
```

La définition d'un groupe d'objets est faite avec la macro OBJECT IDENTIFIER. On peut déduire que le groupe *experimental* est le troisième sous-groupe du groupe *internet*. De même, *internetExperim* est le 4711^e sous-groupe du groupe *experimental* et *tncControl* est le premier sous-groupe du groupe *internetExperim*. Ayant défini les différents sous-groupes d'objets nécessaires, nous définissons les objets en question. La macro OBJECT TYPE est utilisée dans ce but. Par exemple, la définition de l'objet tncTxTail correspondant au paramètre *txTail* est la suivante:

```
tncTxTail OBJECT-TYPE
   SYNTAX      INTEGER (0..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "L'objet tncTxTail est utilisé pour fixer la valeur du paramètre TxTail.
     Seules des valeurs multiples de 10 sont valides."
```

```
::= { tncControl 4 }
```

Cela permet de définir un objet, nommé *tncTxTail*, dont les valeurs sont de type INTEGER et pouvant varier entre 0 et 127. Une instance de cet objet est accessible en mode *read-write*. La définition de cet objet est obligatoire dans tout MIB de gestion d'un TNC. Enfin, cette définition mentionne que l'objet ainsi défini est le quatrième en dessous du groupe *tncControl*.

Le tableau 4 donne la liste des paramètres à gérer et des objets qui leur correspondent. La figure 26 illustre la façon dont les objets définis sont structurés au sein du MIB. La liste complète des définitions des objets est donnée en annexe A.

Paramètre	**Objet**
port	tncPort
fullduplex	tncFullDuplex
hardware	tncHardware
txtail	tncTxTail
persist	tncPersist
slottime	tncSlotTime
txdelay	TxDelay

TAB. 4: *Correspondance entre les paramètres du TNC et les objets qui leur sont associés.*

4.4 L'accès aux objets

Jusqu'à maintenant, nous avons identifié les paramètres du TNC et défini un MIB pouvant servir pour la gestion de ces paramètres. Il reste à déterminer comment accéder aux objets du MIB et effectuer des opérations de gestion.

L'outil CMU-SNMP est fourni avec différentes commandes de gestion. La liste des commandes que nous employons est donnée dans le tableau 5. En particulier, on distingue trois commandes pour extraire la valeur des objets à gérer: *snmpwalk*, *snmpget* et

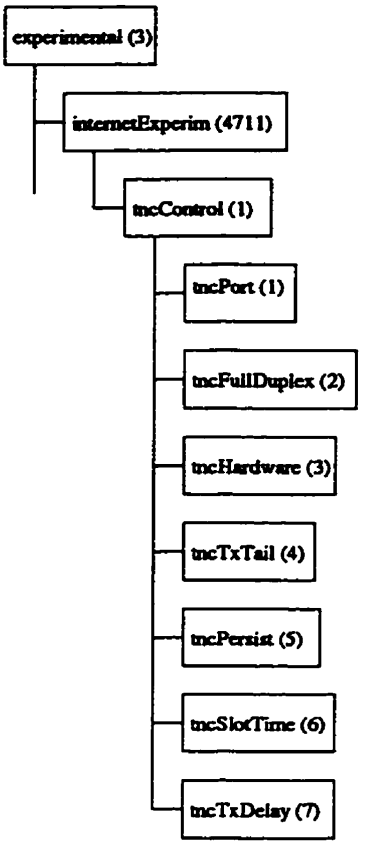


FIG. 26: *Organisation des objets correspondant aux paramètres du TNC au sein du MIB.*

snmpgetnext. D'abord, dans le cas de *snmpwalk*, trois paramètres sont nécessaires:

- nom de la machine hôte: cela correspond au nom de la machine contenant le MIB à accéder. La valeur de ce champ peut être *localhost*, s'il s'agit du MIB local de la machine de laquelle la commande est invoquée, ou bien le nom explicite de la machine distante. Ce nom peut être en format textuel (tel que *molly2.dmi.usherb.ca*) ou en un format numérique (tel que 132.210.48.189).
- nom de communauté: il s'agit de l'un des noms de communautés reconnus par la machine adressée. Dans notre cas, nous avons employé *public*.

- identificateur: il correspond à l'identificateur (OID) du groupe d'objets à accéder. Cet identificateur peut être exprimé en un format textuel (par exemple *system*), en un format numérique (par exemple .1.3.6.1.3.2.1.1) ou en une combinaison des deux (par exemple .1.3.6.1.3.2.1.system).

Commande	Description	CMU-SNMP
snmpwalk	retourne la liste des objets du groupe passé comme paramètre	oui
snmpget	retourne la valeur de l'objet mentionné	oui
snmpgetnext	retourne la valeur de l'objet suivant l'objet mentionné	oui
snmpset	modifie la valeur de l'objet mentionné	oui
snmpax25set	modifie simultanément différentes valeurs d'objets passés en paramètre	non

TAB. 5: *Liste des commandes de gestion.*

Par exemple, pour découvrir les différents objets, ainsi que leurs valeurs, situés en dessous du groupe *tncControl* du MIB local de l'ordinateur sur lequel on travaille, on peut employer la commande suivante:

```
snmpwalk localhost public .1.3.6.1.3.4711
```

Une réponse normale que l'agent peut retourner est:

```
internetExperim.tncControl.tncPort.0 = 1
internetExperim.tncControl.tncFullDuplex.0 = 1
internetExperim.tncControl.tncHardware.0 = 3
internetExperim.tncControl.tncTxTail.0 = 20
internetExperim.tncControl.tncPersist.0 = 10
internetExperim.tncControl.tncSlotTime.0 = 10
internetExperim.tncControl.tncTxDelay.0 = 10
```

Une autre façon d'obtenir la valeur d'un objet est la commande snmpget. Avant de l'employer, il est nécessaire de connaître l'identificateur explicite de l'objet à accéder. Pour cela, on peut employer la commande snmpwalk pour connaître les différents objets appartenant à un certain groupe, puis appliquer snmpget pour l'un (ou chacun) des objets retournés par la commande précédente. Pour savoir, à titre d'exemple, la valeur de l'objet *tncPort*, on peut employer la commande suivante:

```
snmpget localhost public .1.3.6.1.3.4711.tncControl.tncPort.0
```

Le résultat retourné par l'agent est:

```
internetExperim.tncControl.tncPort.0 = 1
```

Pour savoir la valeur de l'objet qui suit *tncPort*, par exemple, on applique la commande snmpgetnext de la façon suivante:

```
snmpgetnext localhost public .1.3.6.1.3.4711.tncControl.tncPort.0
```

Le résultat obtenu est:

```
internetExperim.tncControl.tncFullDuplex.0 = 1
```

Toutes les commandes que nous venons de citer ne permettent que d'extraire la valeur d'un objet figurant dans un MIB local ou distant. Pour modifier et mettre à jour la valeur d'un objet quelconque, il faut employer la commande snmpset. En plus des trois paramètres susmentionnés, cette commande possède un quatrième paramètre spécifiant le type de l'objet en question ainsi que la nouvelle valeur à lui affecter. Les types pouvant être utilisés sont: *i (INTEGER), s (STRING), x (HEX STRING), d (DECIMAL STRING), n (NULLOBJ), o (OBJID), t (TIMETICKS)* ou *a (IPADDRESS)*. Voici un exemple d'utilisation:

```
snmpset localhost public internetExperim.tncControl.tncSlotTime.0 i 20
```

De cette façon, on a affecté à l'objet *tncSlotTime* du MIB de l'ordinateur local, une valeur entière égale à 20. La commande snmpset ne permet de modifier qu'un seul paramètre. C'est pour éliminer cette contrainte et pour présenter aux gestionnaires une commande

permettant de modifier plus d'un paramètre à la fois que nous avons développé une nouvelle commande nommée *snmpax25set*. Sa syntaxe est la suivante:

```
snmpax25set hostname community [-p port][-f 0|1][-l txtail]
                               [-r persist][-s slot][-t txt]
```

Le paramètre [-p] spécifie le nom du port de communication avec le TNC; [-f] spécifie si le TNC doit fonctionner en mode *full-duplex* (valeur 1) ou *half-duplex* (valeur 0). Les paramètres [-l], [-r], [-s] et [-t] permettent de spécifier la valeur de *txTail*, *persist*, *slotTime* et *txDelay* respectivement.
Par exemple, la commande:

```
snmpax25set molly2.dmi.usherb.ca public -f1 -l10 -r20
```

permet d'accéder au MIB de l'agent situé dans une machine distante nommée *molly2.dmi.usherb.ca* et de définir la valeur des paramètres *fullduplex*, *txTail* et *persist* à fullduplex, 10 et 20 respectivement.

4.5 Implantation de l'application de gestion

Après avoir décrit la configuration du système de gestion que nous avons utilisé, décrit les différentes étapes que nous avons suivies pour écrire notre MIB (identification des paramètres de gestion et des objets qui leur correspondent) et présenté la façon dont nous gérons un TNC à distance, nous présentons dans cette section l'implantation en langage C de l'application. L'explication de ce code est nécessaire pour écrire des applications de gestion semblables à celles que nous présentons ou d'autres applications basées sur l'outil CMU-SNMP et fonctionnant sur le système Linux.

L'application de gestion, dans notre cas, utilise les fichiers suivants: *mib.txt*, *snmp_vars.h* et *snmp_vars.c*. Les relations entre ces trois fichiers, le MIB et le TNC sont données dans la figure 27.

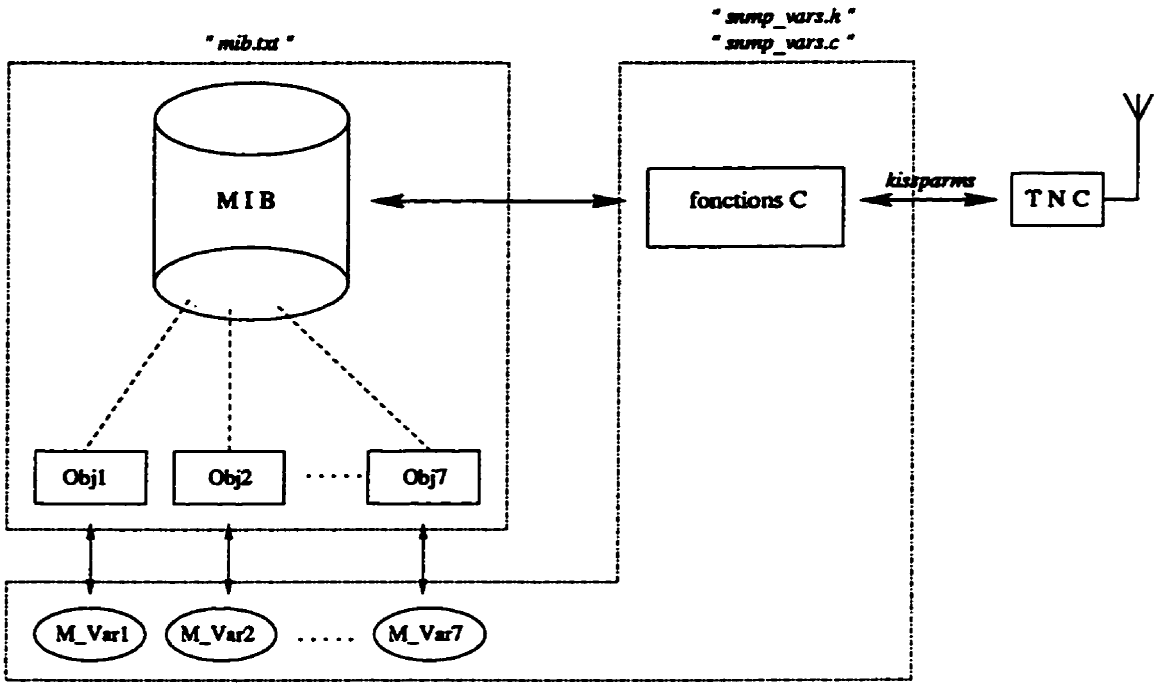


FIG. 27: *Éléments participant à l'implantation du système de gestion du TNC.*

Le fichier *mib.txt* correspond à la définition du MIB. Ce dernier est constitué de l'ensemble des objets à gérer (Obj1, Obj2, ..., Obj7). De l'autre côté, les deux fichiers *snmp_vars.h* et *snmp_vars.c* correspondent à la déclaration et à la définition de toutes les fonctions nécessaires pour manipuler les objets du MIB.

Nous avons associé à chaque objet du MIB une variable (*M_Var*) reflétant la valeur OID associée à cet objet. Ainsi, toute manipulation d'un objet (lecture ou modification) est reflétée sur la variable qui lui correspond. En plus, pour toutes les opérations permettant de modifier la valeur OID d'un objet (*snmpset* et *snmpax25set*), une commande particulière est exécutée dans le but d'affecter physiquement au paramètre approprié du TNC la nouvelle valeur OID de l'objet qui lui correspond. Cette commande est baptisée *kissparms*. Du côté programmation, trois étapes sont à effectuer:

1. définition des objets à gérer: les objets à gérer doivent être définis et ajoutés au

groupe d'objets approprié. Dans notre cas, les objets sont en dessous du groupe *tncControl*. Leur définition a été présentée à la section 4.3 et est donnée en annexe A.

2. enregistrement du nouveau groupe: le nouveau groupe d'objets ainsi défini (*tncControl*) doit être enregistré pour qu'il soit reconnu par l'agent. La fonction *snmp_vars_init* effectue cette tâche. Voici les lignes de code que nous avons ajoutées dans le fichier *snmp_vars.c* pour enregistrer le groupe *tncControl*:

```
mib_register (
   tnc_id_base,
   sizeof(tnc_id_base) / sizeof(oid),
   experimental_variables,
   sizeof(experimental_variables)/sizeof(*experimental_variables),
   sizeof(*experimental_variables));
```

tnc_id_base est un vecteur permettant de définir l'identificateur de groupe *internetExperim*.

```
#define INTERNET_EXPERIMENTAL 1, 3, 6, 1, 3
static oid tnc_id_base[] =
{
 INTERNET_EXPERIMENTAL, 4711
};
```

experimental_variables est un tableau permettant de définir tous les objets du groupe *tncControl*, leurs types, leurs modes d'accès, leurs fonctions de gestion et leurs identificateurs par rapport au groupe *internetExperim*. Ce tableau est défini comme suit:

```
struct variable2 experimental_variables[]={
{TNCPORT,       INTEGER, RWRITE, var_experimental, 2, {1, 1}},
{TNCFULLDUPLEX, INTEGER, RWRITE, var_experimental, 2, {1, 2}},
{TNCHARDWARE,   INTEGER, RWRITE, var_experimental, 2, {1, 3}},
```

```
{TNCTXTAIL,     INTEGER, RWRITE, var_experimental, 2, {1, 4}},
{TNCPERSIST,    INTEGER, RWRITE, var_experimental, 2, {1, 5}},
{TNCSLOTTIME,   INTEGER, RWRITE, var_experimental, 2, {1, 6}},
{TNCTXDELAY,    INTEGER, RWRITE, var_experimental, 2, {1, 7}}
};
```

TNCPORT, TNCFULLDUPLEX, TNCHARDWARE, TNCTXTAIL, TNCPERSIST, TNCSLOTTIME et TNCTXDELAY sont définis dans le fichier *snmp_vars.h* comme suit:

```
#define TNCPORT          1
#define TNCFULLDUPLEX    2
#define TNCHARDWARE      3
#define TNCTXTAIL        4
#define TNCPERSIST       5
#define TNCSLOTTIME      6
#define TNCTXDELAY       7
```

3. définition de la fonction d'accès aux objets: à ce stade, on doit définir l'ensemble des fonctions nécessaires pour accéder, lire et modifier la valeur des objets gérés. Cette fonction se nomme *var_experimental*. Elle est appelée pour chaque objet et est partiellement définie comme suit:

```
u_char *var_experimental(...)
{
    /* Déclaration des variables locales*/
    ...
    /* Allocation mémoire pour un objet */
    ...
    /* Traitement de l'objet en question */
    switch (...){
      case TNCPORT:
            *write_method = write_snmp_tnc_Port;     (1)
            long_return = snmp_tnc_port;             (2)
```

```
            break;
        /* les autres cas */
        ...
    }
    return ...;
}
```

Il y a un cas de traitement pour chaque objet enregistré. Par exemple, dans le cas de TNCPORT, il y a deux actions à effectuer. D'abord, la fonction responsable de l'enregistrement de l'objet est appelée. Il s'agit de la fonction *write_snmp_tnc_Port* (1). Ensuite, la valeur initiale de l'objet est enregistrée, ce qui correspond à l'étape (2). *snmp_tnc_port* est une variable que nous avons définie (dans le fichier *snmp_vars.h*) et qui contient la valeur courante de l'objet.
La fonction *write_snmp_tnc_Port* effectue les tâches suivantes:

```
static int write_snmp_tnc_Port (...)
{
   /* Déclaration des variables locales*/
   ...
   /* Vérifier que le paramètre est de type INTEGER */
   ...
   /* Vérifier que la valeur du paramètre est égale à zéro */
   ...
   if (action == COMMIT) {
         snmp_tnc_port = intval;
         execv("/usr/sbin/kissparms", argv);
   }
   return SNMP_ERR_NOERROR;
}
```

Si l'action à appliquer sur l'objet est une opération de modification (dans ce cas la

condition *action == COMMIT* est vraie), alors la variable globale *snmp_tnc_port* prend la nouvelle valeur de l'objet (*intval*), puis le programme *kissparms* est exécuté pour mettre à jour physiquement dans le TNC la valeur du paramètre en question. Cependant, si l'action correspond à une opération de lecture, alors l'agent retourne la valeur de la variable *snmp_tnc_port*.

Enfin, il reste à noter qu'un traitement similaire est effectué pour les autres paramètres. Le code complet de l'application de gestion est disponible à l'adresse suivante: http://www.dmi.usherb.ca/~hmida/project.html.

4.6 Conclusion

Ce chapitre a abordé le sujet de l'implantation de l'application de gestion. Nous avons commencé par identifier les paramètres du TNC à gérer. Sept paramètres ont été choisis: *port*, *fullduplex*, *hardware*, *txtail*, *persist*, *slottime* et *txdelay*. Avec ces paramètres, nous avons associé et défini les objets MIB nécessaires. L'ensemble de ces objets est groupé sous un groupe nommé *tncControl*. Ensuite, les commandes nécessaires pour le contrôle et la gestion du TNC ont été définies et présentées. Ces commandes (correspondant aux applications de gestion) sont divisées en deux groupes: les commandes de lecture des paramètres (*snmpwalk*, *snmpget* et *snmpgetnext*) et les commandes de modification des paramètres (*snmpset* et *snmpax25set*). C'est à base de ces commandes que la gestion à distance du TNC est effectuée. Il suffit de mentionner l'adresse de la machine hôte dans laquelle figure l'agent et le MIB approprié. En plus, la commande *snmpax25set* permet de modifier simultanément différents paramètres, chose qui facilite la tâche de gestion. Enfin, nous avons présenté les principales étapes à suivre pour développer une application de gestion similaire à la nôtre ou pour développer d'autres applications de gestion basées sur Linux et l'outil CMU-SNMP.

Conclusion

L'installation et l'accès aux réseaux de télécommunications croît d'une façon exponentielle. La mise au point de systèmes de gestion de tels réseaux ainsi que des dispositifs qui leur sont connectés est indispensable. Ceci permet de faciliter la tâche des analystes veillant au bon fonctionnement de leur réseau et de fournir un meilleur service aux utilisateurs.

Le travail effectué dans le cadre de ce mémoire présente une nouvelle méthode de gestion à distance d'un dispositif de télécommunications appelé TNC. La gestion à distance permet d'éliminer la nécessité d'être à proximité du dispositif à gérer, chose qui permet aux analystes de gérer plus d'un dispositif, au sein d'un même réseau ou de différents réseaux, à partir du même poste.

Les contributions apportées par ce mémoire se résument par un outil permettant de contrôler et de gérer simultanément et à distance plusieurs paramètres d'un TNC. Cet outil fait appel à des services fournis par un protocole standard de gestion du réseau Internet appelé SNMP et un outil, nommé CMU-SNMP, spécifique pour le développement des applications de gestion. Enfin, nous avons implanté une base de données, appelée MIB, utilisée par les services élémentaires de SNMP et fournissant des informations relatives aux paramètres du dispositif à gérer.

Deux problèmes majeurs ont été rencontrés durant ce mémoire. D'abord, nous avons remarqué qu'il n'existe aucun moyen pour lire ou vérifier la valeur des paramètres du TNC. En effet, ce dernier est exploité en un mode appelé KISS qui ne permet pas la

lecture directe des valeurs des paramètres du TNC. De ce fait, la validité de la valeur d'un paramètre, suite à sa modification, reste dépendante de la fiabilité d'un programme appelé *kissparms* que nous employons pour modifier physiquement les paramètres du TNC en mode KISS. Ce problème ne pourra pas être résolu tant que le TNC sera exploité en ce mode. Le second problème est qu'il peut y avoir une incohérence entre les variables correspondant aux paramètres du TNC et ce dernier. En effet, suite à un redémarrage de l'ordinateur auquel est attaché le TNC, les variables correspondant à ces paramètres prennent des valeurs initiales prédéfinies, qui ne correspondent pas nécessairement aux valeurs actuelles du TNC. Ce problème peut être résolu en attribuant à ces variables des valeurs initiales prises dans un fichier. Ce dernier sauvegarde les dernières valeurs attribuées à ces paramètres. Une autre solution consiste à lire directement ces valeurs à partir du TNC. Cette dernière solution n'est faisable que si on change le mode KISS. Ceci ne peut être fait facilement vu qu'actuellement l'unique mode d'exploitation du TNC qui corresponde à notre cas d'utilisation est le mode KISS.

Ce que nous avons pu conclure dans ce mémoire, c'est que les systèmes actuels de gestion des réseaux sont limités par les possibilités du protocole de gestion et par les objets utilisés pour représenter l'environnement géré. Le protocole SNMP, qui est à l'heure actuelle le principal protocole de gestion, est trop limité pour répondre aux besoins d'une gestion de systèmes complète. Récemment, deux efforts reliés à SNMP ont été réalisés. D'abord, une nouvelle version (SNMPv3) a été proposée [4]. Jusqu'à maintenant, aucun document officiel n'a été publié pour présenter les améliorations apportées avec cette version. Ensuite, on distingue SNMP++ [15]. Cette version a pour but de simplifier le développement d'applications de gestion SNMP basées sur le langage de programmation C++ et l'approche orientée objet. Les applications ainsi développées seront portables sur différents systèmes d'exploitation, incluant toutes les versions de Windows, HPUX et Solaris. SNMP++ semble être la meilleure version, sauf qu'elle reste quand même dépendante de l'architecture de base de SNMP. Un grand nombre des insuffisances de

SNMP sont prises en compte dans la gestion de réseaux OSI, à base du protocole CMIP. Cependant, ce remède (CMIP) est pire que la maladie, à cause de la complexité et de la taille de la gestion OSI. À présent que nous passons à une gestion globale de systèmes, il faut trouver d'autres techniques plus complètes et plus extensibles, tout en restant loin d'être complexes, pour gérer de tels réseaux. D'ailleurs, plusieurs sont ceux qui ont commencé à proposer de telles techniques. On distingue principalement DMI, X/OPEN et DME [17].

DMI (Desktop Management Interface) concerne la gestion des composants d'un PC, d'un Mac ou d'une station, c'est-à-dire le matériel, les systèmes d'opération, les applications, les unités de stockage et les périphériques. DMI reste quand même limité vu qu'il s'agit d'une gestion non répartie. De l'autre coté, X/Open fournit un ensemble d'interfaces de gestion qui isolent les applications des protocoles de gestion sous-jacents (tels que SNMP ou CMIP). Bien qu'il s'agisse ici d'une gestion répartie, X/Open reste étrange vu qu'on est dans une situation où nous avons besoin de standards pour nous isoler des autres standards. Enfin, DME (Distributed Management Environment) correspond à la gestion de systèmes et de réseaux dans des environnements hétérogènes multiconstructeurs. Il définit une architecture qui combine un canevas de gestion traditionnel (approche station de gestion/agent) avec un canevas orienté objet basé sur CORBA. Avec DME, une application de gestion est constituée de multiples objets qui coopèrent et se partagent l'information. Un objet peut à n'importe quel moment être un fournisseur de service (client) ou demandeur de service (station de gestion). L'incarnation commerciale de cette nouvelle architecture est l'environnement TME (Trivoli Management Environment) [17].

Pour la gestion des réseaux multifournisseurs complexes et de grande taille, nous avons besoin de logiciels de gestion de systèmes plus intelligents qui savent comment traiter des objets plus intelligents qui résident n'importe où sur le réseau.

Annexe A

Définition du groupe tncControl et de ses objets

```
experimental
   OBJECT IDENTIFIER ::= { internet 3 }
internetExperim
   OBJECT IDENTIFIER ::= { experimental 4711 }
tncControl
   OBJECT IDENTIFIER ::= { internetExperim 1 }

tncPort OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "The port name that is being configured."
   ::= { tncControl 1 }

tncFullDuplex OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
```

```
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "The TNC can be set into either full duplex
      or half duplex."
    ::= { tncControl 2 }

tncHardware OBJECT-TYPE
    SYNTAX      INTEGER (1..127)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "This is used to set the hardware specific
      parameters."
    ::= { tncControl 3 }

tncTxTail OBJECT-TYPE
    SYNTAX      INTEGER (1..127)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "This is used to set the Tx Tail time in
      milliseconds.
      Only values like 10, 20, etc are used."
    ::= { tncControl 4 }

tncPersist OBJECT-TYPE
    SYNTAX      INTEGER (1..127)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "This is used to set the persist value.
```

```
      This parameter is scaled to the range
      0 to 255."
    ::= { tncControl 5 }

tncSlotTime OBJECT-TYPE
    SYNTAX      INTEGER (1..127)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "This is used to set the slottime time
      in milliseconds. Only values like 10,
      20, etc are used."
    ::= { tncControl 6 }

tncTxDelay OBJECT-TYPE
    SYNTAX      INTEGER (1..127)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "This is used to set the TX Delay time
      in milliseconds. Only values like 10,
      20, etc are used."
    ::= { tncControl 7 }
```

Bibliographie

[1] CMU SNMP Library. Page web, http://www.net.cmu.edu/projects/snmp/.

[2] Linux Home Page. Page web, http://www.linux.org.

[3] Linux AX25-HOWTO, Amateur Radio. Page web, http://sunsite.unc.edu/LDP/HOWTO/AX25-HOWTO.html, 1995.

[4] SNMP Version 3. Page web, http://www.ietf.org/html.charters/snmpv3-charter.html, 1997.

[5] U. Warrierand L. Besaw, L. LaBarre, and B. Handspicker. The Common Management Information Services and Protocols for the Internet (CMOT and CMIP). Site ftp, ftp://nic.mil/rfc/rfc1189.txt, 1990.

[6] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol. Site ftp, ftp://nic.mil/rfc/rfc1067.txt, 1988.

[7] V. Cerf. IAB Recommendations for the development of Internet network management standards. Page web, ftp://nic.mil/rfc/rfc1052.txt, 1988.

[8] M. Chepponis and P. Karn. The KISSTNC: A Simple Host-to-Host communications protocol. *6TH Computer Networking Conference, Redondo Beach, California*, pages 38–43, 1987.

[9] T.L. Fox. *AX.25 Amateur Packet-radio Link-Layer Protocol, Version 2.0.* Amateur radio Relay League, Inc, 1984.

[10] Bay Networks Inc. Bay Networks Home Page. Page web, http://www.baynetworks.com.

[11] G. Jones. *Packet Radio: What? Why? How?* Tucson Amateur Packet Radio Corporation, 1996.

[12] P. R. Karn, H. E. Price, and R. J. Diersing. Packet Radio in the Amateur Service. *IEEE journal on selected areas in communications*, SAC-3:431–439, 1985.

[13] A. Leinwand and K. Fang. *Network Management, a practical perspective.* Addison Wesley, 1993.

[14] M. Lottor. Internet Growth. Site ftp, ftp://nic.mil/rfc/rfc1296.txt, 1992.

[15] P. E. Mellquist. *SNMP++, An Object-Oriented Approach to Developing Network Management Applications.* Prentice Hall, 1998.

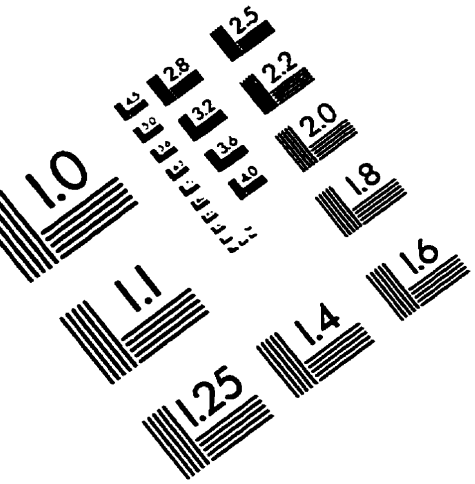
[16] N. Nicolaisen. I'm Failing and I Can't Boot Up! *Byte*, Vol.22, Number 10:113–119, 1997.

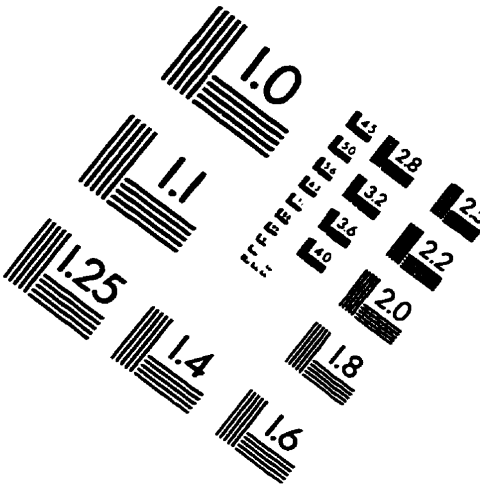
[17] R. Orfali, D. Harkey, and J. Edwards. *CLIENT/SERVEUR, Guide de survie.* International Thomson Publishing, 1997.

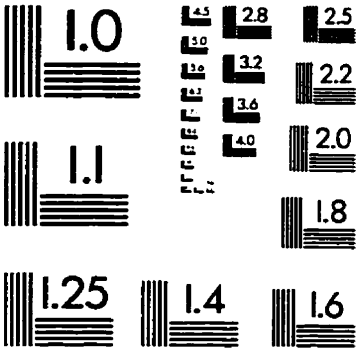
[18] D. Perkins and E. McGinnis. *Understanding SNMP MIBs.* Prentice Hall, 1997.

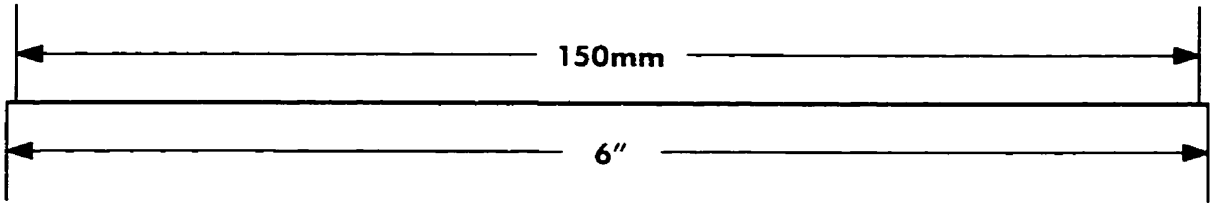
[19] W. Stallings. *SNMP, SNMPv2, and CMIP: the practical guide to network-management standards.* Addison Wesley, 1993.

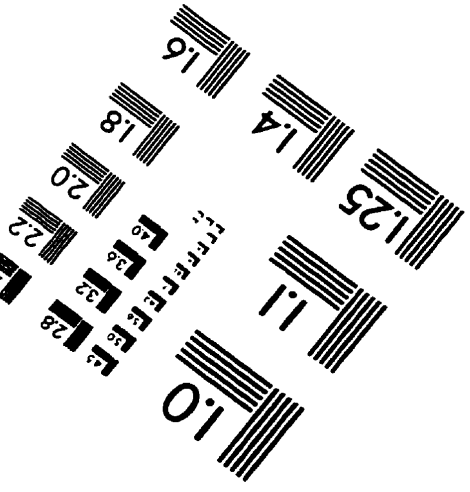
IMAGE EVALUATION TEST TARGET (QA-3)

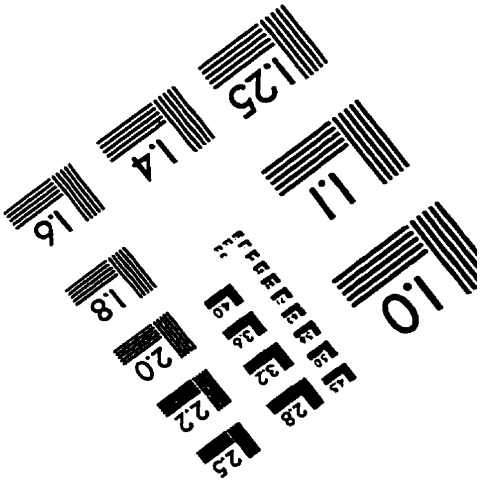












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989