UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MÉMOIRE

SOUMIS POUR L'OBTENTION D'UNE

MAÎTRISE EN INFORMATIQUE

SNMPV3-MODULAIRE :

UNE MÉTHODOLOGIE DE CONCEPTION

ET DE MISE EN ŒUVRE

D'UN PROTOCOLE DE GESTION DE RÉSEAU

PAR

YLIAN SAINT-HILAIRE

NOVEMBRE, 1998

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations, sigles et acronymes

API	« Application Programming Interface »
ASN.1	« Abstract Syntax Notation 1 » ITU TS X.208, ISO 8824.
BER	« Basic Encoding Rules »
CBC	« Cipher Block Chaining », utilisé avec DES.
CGI	« Common Gateway Interface »
CORBA	« Common Object Request Broker Architecture»
DES	« Data Encryption Standard »
HMAC	« Keyed-Hashing Message Authentication »
HTTP	« Hypertext Transfer Protocol »
IDL	« Interface Description Language »
IESG	« Internet Engineering Steering Group », une partie de IETF
IETF	« Internet Engineering Task Force », http://www.ietf.org
JAVA	Langage de programmation développé par Sun
JDK	« Java Development Kit », Compilateur Java de SUN.
JIT	« Juste In Time », Environnement rapide d'exécution des programmes Java.
JMAPI	« Java Management API »
MD5	« Message Digest » Version 5. Algorithme de hashage à une seule direction.
MIB	« Management Information Base »
NMS	« Network Management Station »
OID	« Object Identifier »
OMG	« Object Management Group » http://www.omg.org
OO	« Orienté Objet » ou « Object Oriented », Méthodologie de programmation
PDU	« Protocol Data Unit »
RFC	« Request For Comments »
SHA-1	« Standard Hashing Algorithm » Version 1. Algorithme de hashage à une seule direction.
SNMP	« Simple Network Management Protocol »

SUN	Compagnie qui a développée le langage Java http://www.sun.com
UDP	« User Datagram Protocol » RFC 768
USM	« User Security Model » Défini dans SNMPv3
VACM	« View-based Access Control Model »
VARBIND	« Variable Binding », Regroupement d'un OID et d'une valeur
WBEM	« Web Based Enterprise Management »

Le résumé

SNMPv3-MODULAIRE : Une méthodologie de conception et de mise en œuvre d'un protocole de gestion de réseau.

Depuis une vingtaine d'années, la gestion de réseaux se développe dans un contexte d'intranet. La récente explosion des interconnections entre réseaux appelle une révision des programmes de gestion, particulièrement au niveau de la sécurité et de la modularité. À partir des plus récents standards (toujours en évolution), on a développé un ensemble modulaire et sécurisé de programmes de gestion de réseaux SNMPv3 (Simple Network Management Protocole – Version 3). On propose aussi une méthodologie de tests systématiques. Au cours de l'année, ces programmes ont été éprouvés en ligne sur internet, et se sont avérés performants. La mise en œuvre comprends 25 modules principaux écrits en JAVA.

Abstract

SNMPv3-MODULAR: A conception methodology and an implementation of a network management protocol.

In the past 20 years, the network management techniques have been developed in the context of private Intranets. The recent explosion of interconnection calls for a review of management programs, especially at the level of security and modularity. Based on the most recent standards (still in evolution), a set of modular and secure network management programs SNMPv3 (Simple Network Management Protocol – Version 3) has been developed. A systematic test method is also proposed. Already in the past year, those programs have been proven on-line on Internet, by numerous users, and shown to perform adequately. The implementation implies the making of 25 main modules written in JAVA.

L’introduction

Les réseaux informatiques touchent de plus en plus notre vie courante. On compte sur les services offerts par les réseaux pour les transactions bancaires, les recherches web, la téléconférence, etc. Les services offerts par les réseaux sont donc rendus indispensables. Pour assurer que les services rendus par les réseaux soient convenables, il est nécessaire de surveiller le réseau et d’agir quand une erreur se produit. Pour ce faire, il faut obtenir les données de gestion des équipements de réseaux et, si nécessaire, contrôler ces équipements.

En 1988, la première version de SNMP (Simple Network Management Protocol) est adoptée comme standard par l’IETF (Internet Engineering Task Force). Ce protocole de gestion est incorporé dans beaucoup d’équipements construits jusqu’à aujourd’hui.

Depuis dix ans, les besoins en gestion de réseau ont beaucoup augmenté. Certaines organisations ont plusieurs milliers d’unités à gérer dans plusieurs domaines de l’administration, aussi la gestion d’équipements se fait maintenant par les réseaux publics et privés. Comme la première version de SNMP ne comporte pas de mécanisme de sécurité, elle ne pouvait pas être utilisée sur les réseaux publics.

Avec SNMPv2, l’IETF a voulu faire une mise à jour de SNMP. Après plusieurs années de recherche, le groupe de travail de l’IETF n’est pas parvenu à un consensus.

Une autre tentative de mise à jour de SNMP a été lancée à partir des conclusions du groupe de travail de SNMPv2. Le SNMPv3 est la toute dernière version de ce protocole qui introduit la sécurité dans la gestion de réseau. Cette version est très avancée dans son cheminement vers le standard.

Les objectifs de ce document

Ce document présente l’historique du protocole SNMP, il présente le nouveau standard SNMPv3, et il propose une architecture et une mise en œuvre de cette nouvelle version axée sur l’architecture modulaire et la testabilité du produit final.

Ce document aborde donc les thèmes suivants:

Le premier chapitre introduit : la gestion de réseaux, son fonctionnement et ses problèmes. Ce chapitre est une introduction au protocole SNMP et son utilisation.

Dans le chapitre 2, on retrouve une introduction à la méthodologie de conception orientée objet, et la discussion des techniques de conception et de test qui sont reliées à la construction de protocoles de communications.

Dans le chapitre 3, on retrouve la description détaillée du protocole SNMPv3. Ce chapitre décrit le découpage du protocole en modules. Pour chacun des modules, on décrit leur fonctionnement et leurs interfaces. On y explique comment est effectué le décodage des paquets, et comment fonctionne la sécurité.

Dans le cadre de cette recherche, une mise en œuvre de SNMPv3 a été construite en Java. Le chapitre 4 explique la mise en œuvre, la conception, les décisions qui ont du être prises, et le fonctionnement du programme.

Pour assurer un bon fonctionnement du logiciel, des tests doivent être faits. Le choix d'une architecture de tests est important pour assurer la qualité du logiciel, tout en minimisant le nombre de tests. Le chapitre 5 décrit donc une architecture de tests pour SNMP-Modulaire.

Chapitre I

Ce chapitre introduit la gestion de réseau, les problèmes qui se posent, et certaines de leurs solutions. On décrit le fonctionnement et l'utilisation de plusieurs protocoles de gestion de réseau, en plaçant l'accent sur le protocole SNMP.

1.1 Les besoins et implications de la gestion de réseau

1.1.1 La gestion de réseau

Une bonne gestion de réseau permet l'utilisation optimale de toutes les ressources offertes par le réseau. La gestion de réseau a trois fonctions : la cueillette des données de gestion, l'interprétation et le contrôle. Un administrateur doit pouvoir exercer un contrôle sur les éléments de son réseau pour offrir un meilleur service possible.

On désigne par « élément de réseau » chacun des équipements qui sont branchés au réseau. Des exemples d'éléments de réseaux sont : les routeurs, les concentrateurs, les postes de travail, les imprimantes, etc.

L'administrateur doit pouvoir faire la surveillance du réseau pour détecter les pannes et les mauvais fonctionnement du réseau. En cas de panne, il doit interpréter l'information reçue pour identifier la source du problème. Un protocole de gestion est nécessaire pour exercer les fonctions de gestion sur un réseau. Il doit être capable de dialoguer avec tous les éléments du réseau [WSTA 93].

1.1.2 Les solutions à la gestion de réseaux

Comme introduit précédemment, la fonction de gestion de réseaux est divisée en trois parties :

1.1.2.1 La cueillette des informations de gestion

Un logiciel qui fait de la gestion de réseau doit amasser toutes les informations de gestion dont il a besoin. Ces informations parviennent de chacun des éléments du réseau. Pour les obtenir, une étape de découverte des éléments de réseau est effectuée, puis les requêtes sont envoyées aux éléments un à un pour en tirer les informations de gestion.

Ces informations de gestion comprennent toutes sortes d'informations reliées à un élément de réseau : le type d'équipement, le nombre de paquets qui passent sur chaque port d'un routeur, l'usager qui utilise une station de travail...

Pour effectuer cette tâche, une technique pour démasquer les éléments de réseaux est nécessaire. Un protocole pour obtenir les informations de gestion est aussi nécessaire. SNMP est un protocole qui permet de réaliser ces fonctionnalités.

1.1.2.2 L'interprétation des données

Une fois que l'on a les informations de gestion, il est important de les interpréter correctement. Par exemple : 5000 paquets par seconde sur un port d'un routeur est peut-être normal ou peut-être le signe d'une surcharge et d'une dégradation de la qualité du service offert. Même si on connaît la charge exacte d'un routeur, cette information ne suffit pas pour prendre des décisions pertinentes de gestion.

Les manufacturiers d'équipements offrent quelquefois des logiciels qui savent interpréter les informations de gestion de leurs équipements de gestion. Toutefois, une interprétation automatique complète et correcte de l'état d'un réseau et de ses équipements est difficile à réaliser. L'intervention humaine est souvent requise pour interpréter les données ou pour placer des bornes acceptables.

1.1.2.3 Le contrôle des équipements

Quand des informations de gestion sont obtenues et comprises, il est quelquefois nécessaire d'agir. Il doit être possible de demander à un équipement, par exemple, de se ré-initialiser, de couper ou d'activer des services...

Pour ce faire, un protocole de gestion doit transmettre un ordre (requête) à l'équipement approprié pour déclencher l'action. En raison de l'absence de sécurité par le passé, cette fonctionnalité n'a que rarement été utilisée.

Si on considère ces fonctionnalités, une application de gestion ressemble beaucoup à une application client/serveur. L'application de gestion est le client, et les éléments de réseau sont les serveurs. Il y a plus d'une ressemblance, car la gestion de réseau emprunte beaucoup des méthodes développées dans le cadre de gestion client/serveur.

Plusieurs solutions sont déjà utilisées ou ont été proposées pour la gestion de réseau. Ces solutions seront analysées en détail à la section 1.6.

1.1.3 Le protocole SNMP

SNMP (Simple Network Management Protocol) est le protocole de gestion de réseaux proposé par l'IETF. Il est actuellement le protocole le plus utilisé pour la gestion des équipements de réseaux. SNMP est un protocole relativement simple; toutefois l'ensemble de ses fonctionnalités est suffisamment puissant pour permettre la gestion des réseaux hétérogènes complexes. Il est utilisé aussi pour la gestion à distance des applications: les bases de données, les serveurs, les logiciels, etc. [CASE 93]. L'usage du protocole SNMP peut aussi dépasser largement le cadre de la gestion des équipements de réseaux.

L'environnement de gestion SNMP est constitué de plusieurs composantes [OCHE 98]: une station de gestion de réseau (NMS- *Network Management Stations*), des éléments de réseaux (NE, *Network Elements*), des variables MIB et un protocole (voir Figure 1).

Les différentes composantes du protocole SNMP sont les suivantes:

- **Les éléments de réseaux** sont les équipements ou les logiciels que l'on cherche à gérer. Cela va d'une station de travail à un concentrateur, un routeur, un pont, etc. Chaque élément de réseau dispose d'une entité dite agent de réseaux qui répond aux requêtes de la plate-forme de gestion. L'élément de réseau dispose de l'instrumentation pour le repérage des variables de gestion de réseaux et des mécanismes de communication inter-couches.

- **Les agents** sont des modules qui résident dans les éléments de réseau. Ils vont chercher l'information de gestion tel que le nombre de paquets en erreur reçus par un élément de réseaux.
- **La plate-forme de gestion de réseaux** exécute les applications de gestion qui contrôlent les éléments de réseaux. Physiquement, la plate-forme est un poste de travail avec un processeur rapide, affichage couleur et nécessitant beaucoup de mémoire et d'espace sur disque.
- **Une MIB** (Management Information Base) est une collection d'objets résidant dans une base d'information virtuelle. Des collections d'objets reliés sont définies dans des modules MIB spécifiques.
- **Un protocole** permet à la plate-forme de gestion d'aller chercher les informations sur les éléments de réseaux et aussi de changer les paramètres sur ces derniers. De plus, il permet à la plate-forme de gestion de recevoir des alertes provenant des éléments de réseaux.

SNMP fournit quelques commandes de bases pour la recherche et la mise à jour des variables de la MIB.

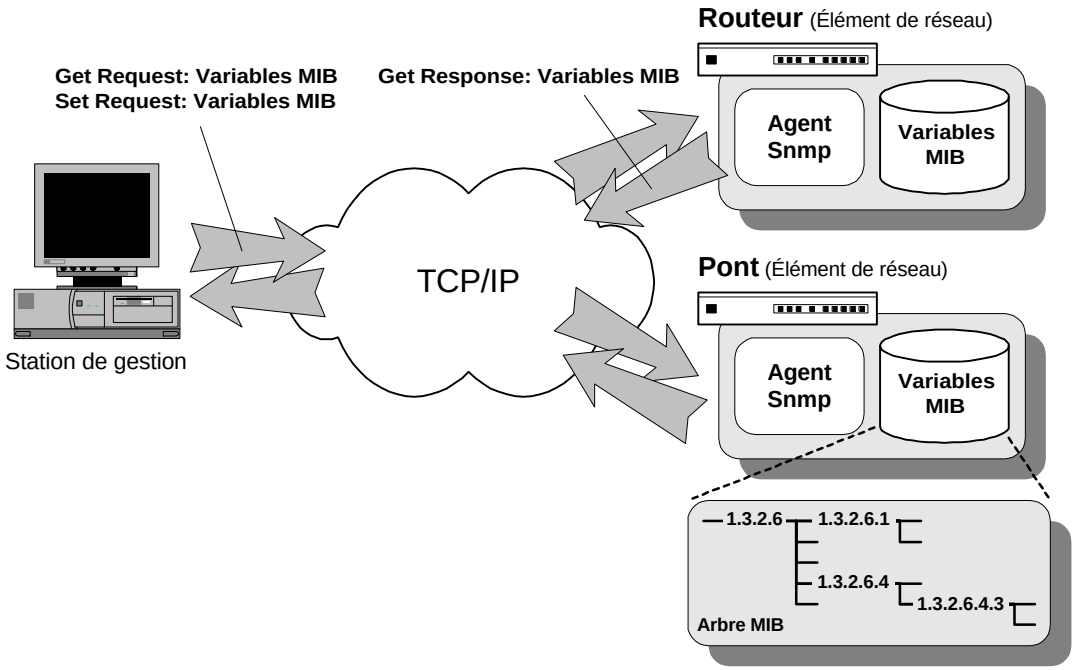


Figure 1 : L'environnement SNMP

1.1.4 Les requêtes et les réponses SNMP

Le fonctionnement de SNMP est asymétrique; il est constitué d'un ensemble de requêtes, de réponses et d'un nombre limité d'alertes. La station de gestion envoie des requêtes à l'agent, lequel retourne des réponses. Lorsqu'un événement anormal surgit sur l'élément de réseaux, l'agent envoie une alerte (*trap*) à la station de gestion de réseau. Les commandes et les données sont représentées dans le format ASN.1 [ISO 87]. SNMP utilise le protocole UDP [RFC 768]. Le port 161 est utilisé par l'agent pour recevoir les requêtes de la station de gestion. Le port 162 est réservé pour la station de gestion pour recevoir les alertes des agents.

1.1.4.1 Les requêtes de SNMP

Il existe quatre types de requêtes: *GetRequest, GetNextRequest, GetBulk, SetRequest.*

La requête *GetRequest* permet la recherche d'une variable sur un agent;
La requête *GetNextRequest* permet la recherche de la variable suivante;

La requête *GetBulk* permet la recherche d'un ensemble de variables regroupées;
La requête *SetRequest* permet de changer la valeur d'une variable sur un agent.

1.1.4.2 Les réponses de SNMP

À des requêtes, l'agent répond toujours par *GetResponse*. Toutefois si la variable demandée n'est pas disponible, le *GetResponse* sera accompagné d'une erreur *noSuchObject*.

1.1.4.3 Les alertes (Traps, Notifications)

Les alertes sont envoyées quand un événement se produit qui pourrait être d'importance à la station de gestion. Les alertes possibles sont: *ColdStart, WarmStart, LinkDown, LinkUp, AuthentificationFailure*. La nouvelle version de SNMP offre une autre forme d'alerte : la « notification ».

1.2 Les informations de gestions

Les informations de gestion peuvent être, par exemple : la quantité d'information transmise à chaque port d'un routeur, la température à l'intérieur d'un imprimante laser ou l'usager qui utilise présentement une station de travail.

Généralement, un équipement a plusieurs dizaines ou même des centaines d'informations de gestion. Pour chacune de ces informations de gestion, on doit s'entendre sur l'identification et la signification d'un objet de gestion. Cette entente permet à une plate-forme de gestion de demander des informations de gestion à un agent et de les interpréter correctement. Pour ce faire, des documents de standards décrivent les identificateurs d'objets placés dans des MIBs.

1.2.1 L'identificateur d'objet

Les variables que l'on cherche à administrer à distance doivent être identifiées de façon unique. Elles ne peuvent être choisies aléatoirement. Des règles normalisées ont été proposées pour autoriser l'utilisation de ces numéros d'identificateur. Un objet représente une variable ou un groupe de variables. L'identificateur d'un objet est une séquence d'entiers séparés par un

point (.). On dispose alors d'un structure hiérarchique représentée sous forme d'un arbre (voir Figure 2). La racine n'est pas numérotée. Chaque nœud de l'arbre décrit un identificateur d'objet. Le numéro de l'identificateur de l'objet MIB s'écrit 1.3.6.1.2.1. Par exemple, le nom de cet identificateur est iso.org.dod.internet.mgmt.mib.

Les feuilles de l'arbre correspondent alors aux instances de l'objet qui sont les variables. La valeur échangée entre l'agent et la plate-forme de gestion est le numéro d'identificateur.

Lorsqu'une entreprise veut définir son propre ensemble de variables de gestion, elle va enregistrer son numéro d'objet sous le nœud iso.org.dod.internet.private.entreprise. Ces MIB seront dites privées. Elles correspondent à la racine 1.3.6.1.4.1.

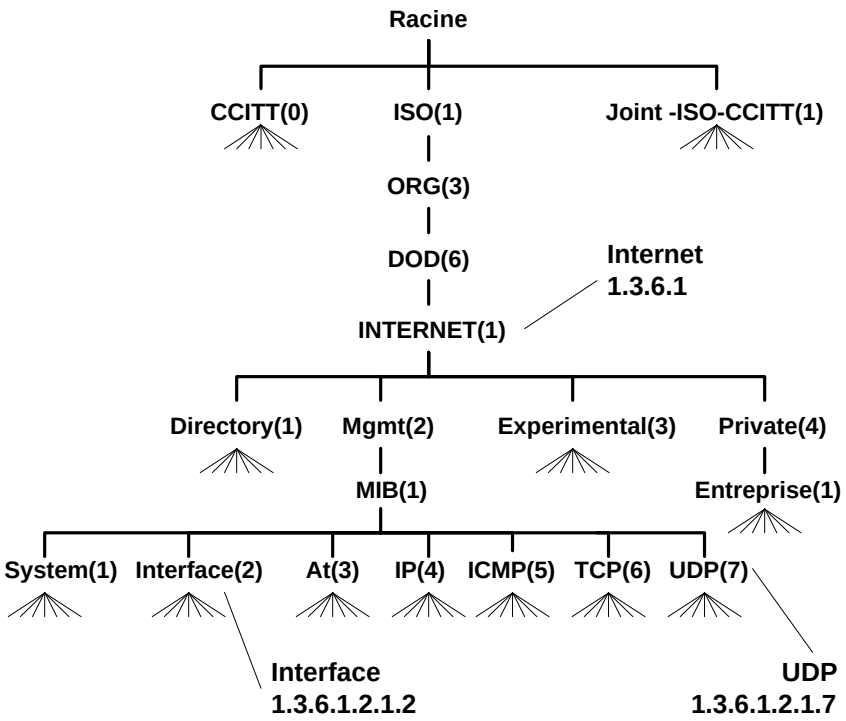


Figure 2: Les identificateurs d'objets de la MIB

1.2.2 Les MIB (Management Information Base)

La MIB est la base de données des informations de gestion maintenue par l'agent, à laquelle la plate-forme va venir pour s'informer. Cette base de données d'informations de gestion est

constituée d'objets (pas dans le sens d'orienté objet) qui représentent des variables. Deux MIB publics ont été normalisées: MIB I et MIB II (dite 1 et 2). Elles décrivent l'ensemble des variables TCP/IP. Le Tableau 1 décrit l'ensemble des groupes de variables qui sont décrit dans la MIB I. On forme généralement un groupe de variables pour chaque partie de l'instrumentation que l'on doit gérer.

La MIB II est utilisée par la version SNMPv2. Elle est riche en terme de variables. Les entreprises peuvent de leur côté définir leur propre MIB.

Le tableau suivant décrit la MIB I

NOM du groupe	Description
System	La description de toutes les entités gérées.
Interface	L'interface de données dynamiques ou statiques.
Address translation	La table d'adresses IP pour les correspondances avec les adresses physiques.
IP	Les statistiques du protocole IP, l'adresse cachée, la table de routage.
ICMP	Les statistiques du protocole ICMP.
TCP	Les paramètres TCP, les statistiques, la table de connexion.
UDP	Les statistiques UDP.
EGP	Les statistiques EGP, table d'accessibilité.
SNMP	Les statistiques du protocole SNMP.

Tableau 1: La description des composantes de la MIB 1

Dans ces groupes de variables, on retrouve les principaux paramètres de configuration et les statistiques relatives à la famille des protocoles TCP/IP.

1.3 Les différentes versions de SNMP

Il est important de comprendre que plusieurs versions SNMP ont été décrites et publiées dans des documents de standards ou dans des publications d'entreprises. Voici les différentes versions de SNMP (aussi voir « L'état des RFCs » plus bas) :

SNMPv1 (complet): Ceci est la première version du protocole, tel que définie dans le RFC 1157. Ce document remplace les documents plus vieux comme RFC 1067 et RFC 1098. On dit que la sécurité de cette version est triviale, car la seul vérification qui est faite est basée sur la chaîne de caractères « community ».

SNMPsec (historique): Cette version ajoute de la sécurité au protocole SNMPv1 et est définie par RFC 1351, RFC 1352 et RFC 1353. La sécurité est basée sur des groupes. Très peu ou aucun manufacturiers n'a utilisé cette version qui est maintenant largement oubliée.

SNMPv2p (historique): Beaucoup de travaux on été exécutés pour faire une mise à jour de SNMPv1. Ces travaux ne portaient pas seulement sur la sécurité. Le résultat est une mise à jour des opérations du protocole, des nouvelles opérations, des nouveaux types de données. La sécurité est basée sur les groupes de SNMPsec. Cette version est décrite par RFC 1441, RFC 1445, RFC 1446, RFC 1448 et RFC 1449.

SNMPv2c (expérimental): Cette version du protocole est appelé « community string-based SNMPv2 ». Ceci est une amélioration des opérations de protocole et des types d'opérations de SNMPv2p et utilise la sécurité par chaîne de caractères « community » de SNMPv1. Cette version est définie par RFC 1901, RFC 1905 et RFC 1906.

SNMPv2u (expérimental): Cette version du protocole utilise les opérations, les types de données de SNMPv2c et la sécurité basée sur les usagers. Cette version est décrite par RFC 1905, RFC 1906, RFC 1909 et RFC 1910.

SNMPv2* (expérimental): Cette version combine les meilleures parties de SNMPv2p et SNMPv2u. Les documents qui décrivent cette version n'ont jamais été publiés dans

les RFC. Des copies de ces documents peuvent être trouvées sur le site web et SNMP Research (un des premiers à défendre de cette version).

SNMPv3 (proposé): Cette version comprend une combinaison de la sécurité basée sur les usagers et les types et les opérations de SNMPv2p, avec en plus la capacité pour les « proxies ». La sécurité est basée sur ce qui se trouve dans SNMPv2u et SNMPv2*. Le standard SNMPv3 sera détaillé au chapitre III (page 48).

Quand on réfère à SNMPv2, il est important de savoir à quel document (RFC) on se rattache. [WSTA 93] traite en fait de la version SNMPv2p, quand il mentionne SNMPv2.

De toutes ces versions, le présent document traitera uniquement de SNMPv1, SNMPv2c et SNMPv3. Comme SNMPv1 et SNMPv2c sont très proches, plusieurs agents qui suivent la norme SNMPv1 peuvent passer à la norme SNMPv2c avec des changements mineurs (ajout du support de GETBULK et de quelques autres types).

Note :

Les RFC suivants ont trait à la définition de SNMPv2. Le groupe de l'IETF recommande des noms SNMPv2p, SNMPv2c, SNMPv2usec pour les distinguer. On peut toujours référer à une version spécifique de SNMPv2 en indiquant le numéro du document RFC concerné.

RFC 1441, RFC 1445, RFC 1446, RFC 1448 et RFC 1449 (SNMPv2p)
RFC 1901, RFC 1905 et RFC 1906 (SNMPv2c)
RFC 1905, RFC 1906, RFC 1909 et RFC 1910 (SNMPv2usec)

L'état des RFCs (documents de normalisation)

Chaque RFC passe par un cycle de vie. Généralement on écrit un RFC pour combler un besoin. Quand on écrit un première spécification, l'IETF place cette spécification dans

l'état expérimental. Suite à un consensus sur la maturité du document, on peut avancer le document à la phase de « standard proposé » , on effectue alors des tests d'inter-operabilité, on valide alors le document, mais pas des mises en œuvres. Si on a un consensus sur la viabilité des mises en œuvres, l'IETF avance le document comme une norme, un standard. Si un document est amélioré par un autre RFC, on placera le document dans le mode « Historique ». D'autres documents peuvent avoir un rôle informatif. Des documents (RFC) sur le fonctionnement de l'IETF sont sur le site Internet http://www.ietf.org.

1.4 La présentation des standards SNMPv1 et SNMPv2c

1.4.1 Le paquet SNMPv1

Le paquet SNMPv1 [RFC 1157] est complètement encodé en ASN.1 [ISO 87]. La dimension des champs est indiquée dans le paquet par l'encodage ASN.1 et BER (Basic Encoding Rules). Les requêtes et les réponses ont le même format.

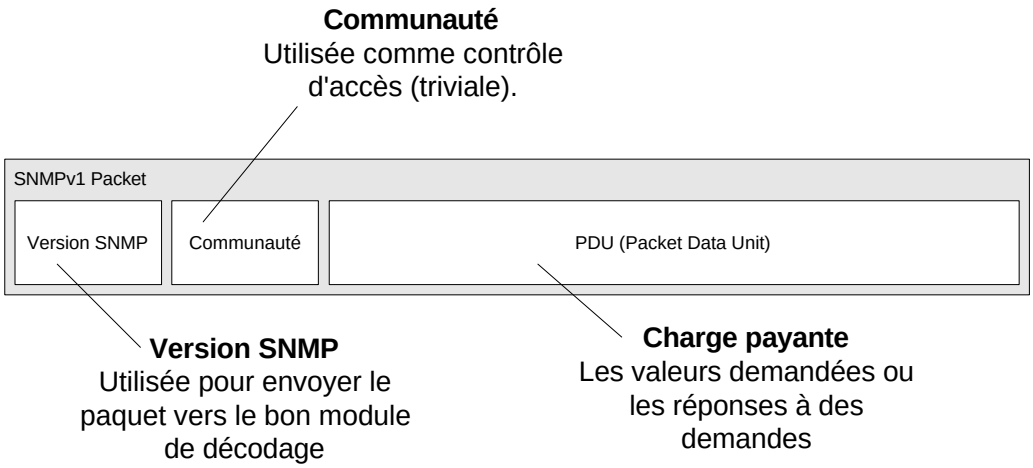


Figure 3 : Le paquet SNMPv1

- Le premier champ indique la version du protocole.
- La **version** la plus utilisée est encore la version 1. Plusieurs versions 2 ont été proposées par des documents de travail, mais malheureusement, aucune d'entre elles n'a jamais été adoptée comme standard. La version 3 est actuellement en voie d'être adoptée. On place la valeur zéro dans le champ version pour SNMPv1, et la valeur 3 pour SNMPv3.
- La **communauté** permet de créer des domaines d'administration. La communauté est décrite par une chaîne de caractères. Par défaut, la communauté est « PUBLIC ».

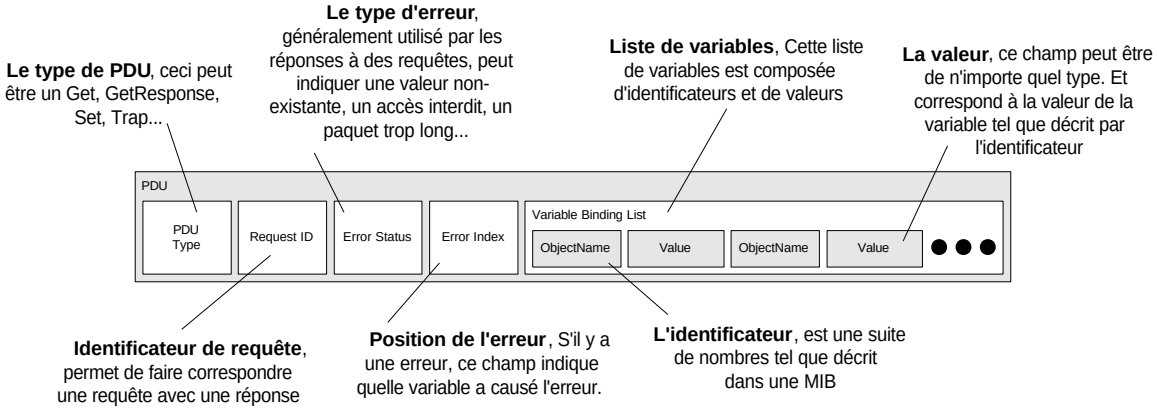


Figure 4 : Le PDU[1] (Protocol Data Unit) de SNMPv1

Le **« PDU type »** décrit le type de requête, de réponse ou d'alerte. Le Tableau 2 donne les valeurs associées à ces champs.

Type de PDU	Nom
0	Get-request
1	Get next-request

[1] On remarquera plus loin que le PDU de SNMPv1 reste inchangé dans SNMPv3.

2	Set-request
3	Get response
4	Trap

Tableau 2: Les types de PDU SNMP

Le **« Request ID »** permet à la station de gestion d'associer les réponses à ses requêtes.

Le **« Error Status »** est l'indicateur du type d'erreur. Si aucune erreur ne s'est produite, ce champ est mis à zéro. Les réponses négatives possibles sont décrites dans le tableau suivant :

Réponses	**Description**
NoAccess	Accès non permis
WrongLengh,	Erreur de longueur
WrongValue,	Valeur erronée
WrongType,	Type erroné
WrongEncoding	Erreur d'encodage
Nocreation	Objet non créé
ReadOnly[2]	Pas de permission d'écrire
NoWritable	Pas de permission d'écrire
AuthorisationError	Erreur d'autorisation

Tableau 3 : Les types de réponses erronées de SNMP

[2] Le message « readOnly » a été défini par les standards, mais ne doit jamais être utilisé. Voir [WSTA 93] page 155, « The curious Case of readOnly ».

1.5 Les faiblesses de SNMPv1-SNMPv2c

1.5.1 Les faiblesses du SNMPv1

Une des plus grandes faiblesses du protocole SNMPv1 est l'absence d'un mécanisme adéquat pour assurer la confidentialité et la sécurité des fonctions de gestion. Les faiblesses comprennent aussi l'authentification et l'encryption, en plus de l'absence d'un cadre administratif pour l'autorisation et le contrôle d'accès. Ce problème rend la sécurité sur SNMPv1 du type : « SHOW-AND-TELNET », c'est à dire qu'on utilise SNMP pour l'acquisition des données de gestion, mais pour effectuer le contrôle on utilise le protocole Telnet[3].

Le groupe de travail de l'IETF qui a œuvré sur SNMPv2 a voulu inclure la sécurité dans la nouvelle version. Malheureusement, le groupe n'a pas pu atteindre un consensus sur le fonctionnement du mécanisme de sécurité. Partant de là, deux propositions ont été développées (SNMPv2u et SNMPv2*). La plupart des experts s'entendent pour dire que deux standards SNMP ne peuvent pas coexister, et que ceci n'est pas une solution à long terme.

Tous les consensus du groupe de travail ont été rassemblés (uniquement les améliorations qui ne portaient pas sur la sécurité), et le groupe de travail SNMPv2 de l'IETF a terminé ses travaux en publiant une version de SNMPv2 (on l'appelle SNMPv2c, RFC 1901, RFC 1905 et RFC 1906) sans sécurité.

1.5.2 Les améliorations de SNMPv2c

SNMPv2c a introduit quelques nouveaux types, mais sa nouveauté majeure est l'opération GETBULK, qui permet à une plate forme de gestion, de demander en bloc plusieurs variables consécutives dans la MIB de l'agent. Généralement, on demande autant de variables que l'on

---

[3] Même si telnet, avec un login et mot de passe, est souvent utilisé pour effectuer le contrôle d'équipements, la sécurité reste souvent insuffisante. Le mot de passe et les données de contrôle sont envoyées sans encryption sur le réseau.

peut entrer dans un paquet SNMP. Ceci règle un problème majeur de performance dans SNMPv1. Avec la version 1, la plate forme est obligée de faire un GETNEXT et d'attendre la réponse pour chaque variable de gestion.

1.5.3 Les améliorations de SNMPv3

SNMPv3 résout le problème de la sécurité et la modularité. La sécurité est nécessaire si l'on désire faire la gestion de machines sensibles sur un réseau public. Avec l'Internet, ceci est souvent le cas.

La sécurité dans SNMP pose un problème : Comme aucun algorithme de sécurité n'est parfait et que la technologie de décodage évolue, la sécurité offerte par un algorithme est relative. Il sera donc nécessaire de changer le système de sécurité dans les logiciels SNMP.

Pour résoudre le problème de sécurité qui change, le standard SNMPv3 propose que les logiciels SNMP soient bâtis de façon modulaire, c'est-à-dire que certains modules peuvent être ajoutés et enlevés facilement par l'utilisateur du logiciel SNMP. L'architecture d'SNMP est aussi modulaire, ceci permet au standard SNMPv3 d'être mis à jour, sans entraîner des changements dans tous les documents du standard. Comme les documents du standard sont modulaires, on peut donc faire évoluer le standard SNMPv3 sans trop de difficultés.

1.6 Les nouvelles architectures

SNMP n'est pas seul dans la course pour avoir une part du très grand marché des logiciels de gestion de réseaux. Cette section est consacrée à un survol des autres solutions de gestion qui ont été proposées, leurs architectures et leurs caractéristiques.

On présente ces autres solutions à titre de comparaison. Certaines des solutions présentées peuvent être utilisées pour compléter SNMP. Par exemple en SNMP, on utilise souvent SNMP avec le Web ou une autre solution pour rendre l'expérience de gestion plus conviviale pour l'usager et pour répondre à d'autres besoins bien spécifiques. On pourra donc voir dans quel contexte SNMP peut être utilisé.

Voici les autres solutions de gestions proposées:

- Solution de gestion basée sur le Web
- Solution Web Management (WBEM)
- La solution JMAPI (Java Management API)

- La solution CORBA

Avant d'analyser chaque solution en détail, on doit prendre note que la gestion de réseaux ressemble aussi beaucoup à une grande application distribuée. Dans ce contexte, de nombreuses solutions nouvelles dans la gestion de réseaux découlent directement de techniques d'abord appliquées aux logiciels distribués.

1.6.1 Les solutions de gestion basées sur le Web

Le Web est un des services les plus utilisés sur l'Internet. On l'utilise pour chercher de l'information ou passer des commandes. Cette technologie peut être appliquée telle quelle à la gestion de réseaux, ceci est l'objectif de cette première solution.

Les architectures d'administration basées sur le Web ont comme intérêt principal la présentation des données de façon indépendante de la plate-forme. En effet, le langage « HyperText Markup Language Protocol » (HTML) permet de présenter les données de la même façon sur tous les navigateurs Web, il y a donc une uniformisation dans la représentation des données. La transmission des informations de gestion est effectuée par le protocole HTTP.

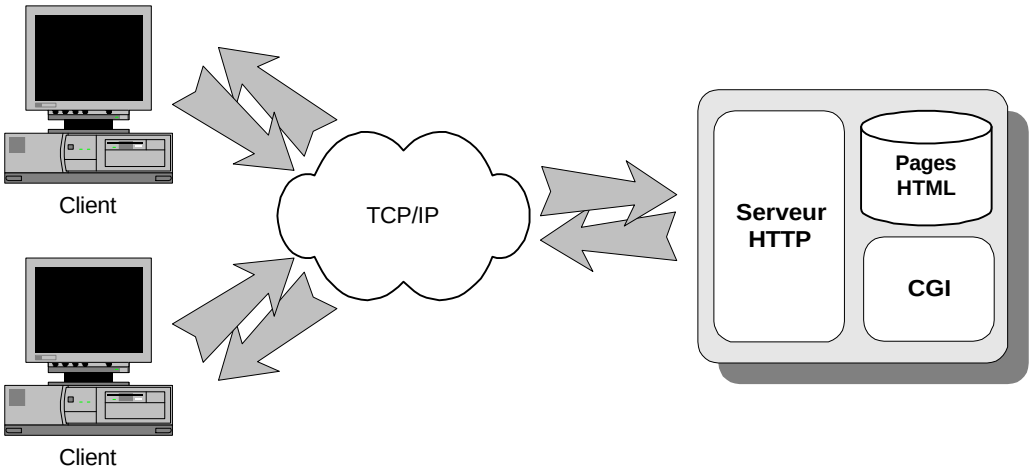


Figure 5 : Architecture du WEB

Pour retourner des informations aux serveurs et avoir un traitement plus dynamique, on peut utiliser les CGI (Common Gateway Interface, voir plus bas).

Dans l'approche de la gestion de réseau par HTTP et CGI, on observe que la seule exigence remplie par cette solution est celle de la représentation uniforme des données. Par contre, le modèle des ressources à gérer est ici totalement dépendant du choix du développeur. Rien n'est prévu pour uniformiser la modélisation des données.

Les aspects sécurité et intégration sont ici peu développés ou laissés à la guise du concepteur. Cependant, cette solution a permis de voir émerger les architectures suivantes qui sont également basées sur l'utilisation du Web.

Qu'est qu'un CGI?

> Un CGI (Common Gateway Interface) est une interface entre le serveur de page web et des logiciels capables de générer les pages web. Dans la majorité des cas, les pages web demandées sur l'Internet sont chargées d'un fichier disque et envoyées sur le réseau. Dans certaines circonstances, le serveur web peut demander à un logiciel la construction de la page web, ce logiciel pourra obtenir des informations dynamiques d'une base de données ou d'une autre source, et utiliser ces informations pour construire la page web.

Les avantages de la gestion par le WEB

Cette technique requiert des outils déjà utilisés pour fabriquer des sites web interactifs. Les CGIs et le HTML sont très utilisés et bien connus. Le CGI peut être conçu pour obtenir les données de gestion de n'importe quelle façon, l'abus de cette liberté peut toutefois mener à un système difficile à entretenir, et qui s'adapte mal lorsque le nombre d'éléments à gérer est grand.

1.6.2 La solution Web-Based Entreprise Management (WBEM)

La solution WBEM est née d'une initiative de CISCO, Microsoft, Intel, Compaq, BMC Software et bien d'autres industriels le 17 juillet 1997. Ceux-ci se sont retrouvés autour d'une charte commune insistant sur les efforts de l'industrie pour trouver des solutions d'administration interopérables, mais en intégrant les technologies déjà existantes.

L'objectif du projet WBEM est de consolider et d'unifier les données fournies par des technologies d'administration existantes.

Cet objectif peut être atteint par la réalisation de deux caractéristiques principales :

- La définition d'une description commune de données, extensible et indépendante de la plate-forme, ce qui permettra aux données d'origines diverses d'être décrites, instanciées et accessibles, quelle que soit la source originelle de ces données.
- La définition d'un protocole standard sur lequel ces données pourront être publiées et accessibles, ce qui permettra aux applications d'administration d'être indépendantes de la plate-forme et d'être physiquement distribuées dans l'entreprise. Cette définition comprend donc le développement d'un schéma unifié de gestion, qui permettra de faciliter l'accès à ces informations de gestion.

En fait WBEM ne remplacera jamais les protocoles d'administration existants (SNMP, CMIP, ...), mais il fournit un point d'intégration pour les données issues de ces technologies.

Un autre objectif de WBEM est d'utiliser le navigateurs Web comme principale interface usager, ce qui offre une indépendance de l'interface homme-machine par rapport à la plate-forme.

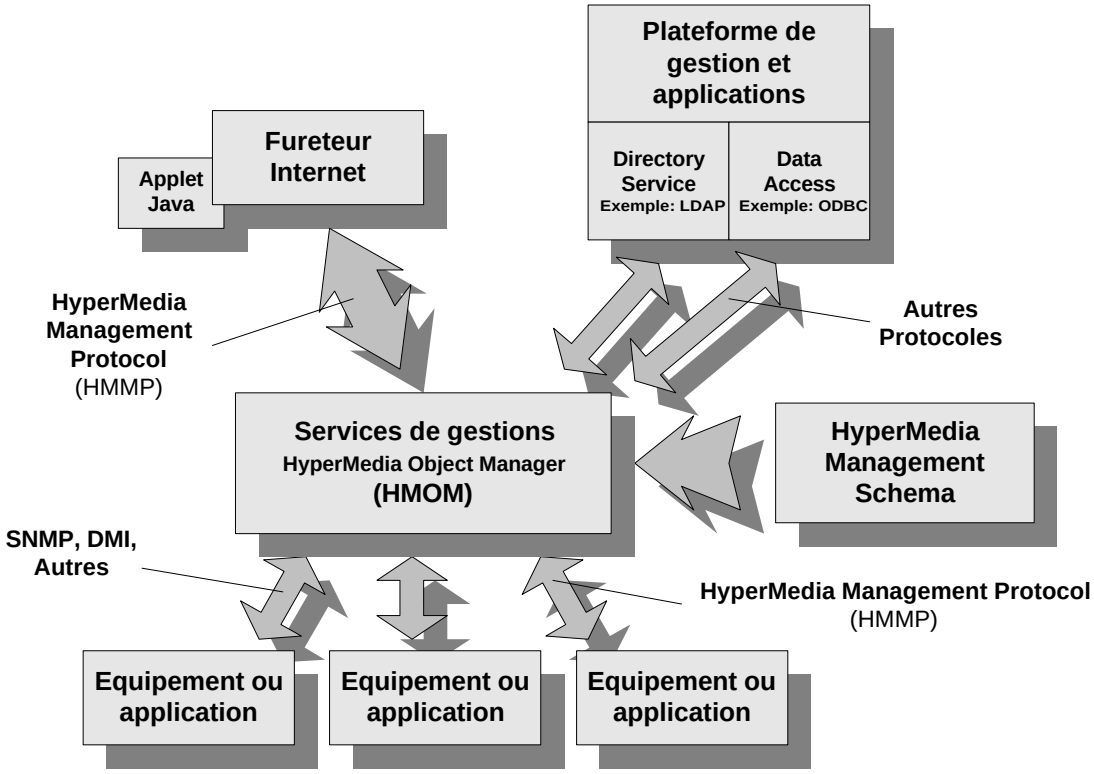


Figure 6 : Architecture de la solution « Web-Based Entreprise Management »

L'architecture de WEBM s'articule autour de trois composantes principales :

- **L'HyperMedia Management Schema** (HMMS) : c'est un schéma objet hérité du Common Information Model (CIM) [ISO 89] et utilisé pour modéliser les ressources à administrer;
- **L'HyperMedia Management Protocol** (HMMP) : il s'agit du protocole de communication qui sera utilisé pour accéder au HMMS;
- **L'HyperMedia Object Manager** (HMOM), qui gère les éléments du réseau comme des objets, et rend les services d'administration.

Le HMOM joue un rôle central : il est à la fois en relation avec les objets à gérer, et avec les applications de gestion. De plus, c'est par son intermédiaire que l'accès se fait au HMMS.

Les avantages de WBEM

WBEM permet de réduire la complexité des solutions d'administration, dans la mesure où elle offre une visualisation complète, et personnalisable par le biais d'un navigateur Web, des données d'administration. En effet, l'interface utilisateur est unique quel que soit le type de machine, grâce à l'utilisation de navigateurs Web.

Ensuite, WBEM fournit une uniformité dans la gestion des réseaux, des systèmes et des applications, puisqu'il permet l'accès et l'association de données de sources différentes, puis, l'architecture à plusieurs niveaux.

Cette architecture s'adapte donc aux problèmes de gestion en général. De plus, elle est basée sur des standards et elle est compatible avec les protocoles existants, on peut donc dire qu'elle utilise des technologies déjà éprouvées.

Par ailleurs, les frais d'investissements et d'entretiens sont faibles. En effet, il n'y a pas besoin d'une station de travail coûteuse dédiée ou d'une infrastructure compliquée, l'utilisation des navigateurs Web offre un coût faible et une utilisation indépendante des plates-formes bien que facilement extensible.

Enfin, cette architecture permet de créer des nouveaux services, puisque l'intégration de services élémentaires permet la construction de services spécialisés.

1.6.3 La solution JMAPI (Java Management API)

JMAPI (Java Management API) est née de l'initiative de Sun, CISCO, Novell, Bay Networks, CA, BMC, Tivoli et 3COM dès mars 1997.

JMAPI est un ensemble extensible de classes et d'interfaces Java qui permet de construire des applications de gestion de réseaux, de systèmes et de services, et de développer des applications d'administration orientée objet basées sur le Web, pour des environnements divers et variés. Ainsi, une application d'administration, dotée d'une GUI Java, utilisée dans un navigateur Web, peut interagir avec des instances d'objets administrés sur un serveur central (Admin Runtime Module).

Ainsi, on retrouve dans cette solution, la représentation et la modélisation uniforme des données.

L'architecture de cette API est constituée des trois composantes suivantes :

- **Le navigateur Web avec support Java** : il permet à l'administrateur d'effectuer les opérations d'administration sur n'importe quel élément de son système. Cependant, celui-ci peut également utiliser comme interface une application indépendante ou encore une interface en ligne de commande.
- **Le Admin Runtime Module** : c'est un serveur central qui fournit aux applications les instances des objets administrés. Il comprend les interfaces des objets agents, des agents SNMP, des données administrées et des interfaces pour JDBC (Java DataBase Connectivity). Par ailleurs, il contient également une base de données et un serveur HTTP pour télécharger les fichiers HTML contenant les applets JMAPI. En effet, la plupart des architectures classiques pour le Web Management utilisent un serveur Web et un serveur d'administration indépendants et n'étant pas obligatoirement localisés sur la même machine, c'est une architecture dite 4/3, alors que JMAPI réunit les deux serveurs sur le même hôte : architecture 3/3.
- **Les éléments de réseau** : ce sont les ressources à gérer (du terminal au serveur de noms, en passant par les comptes NT).

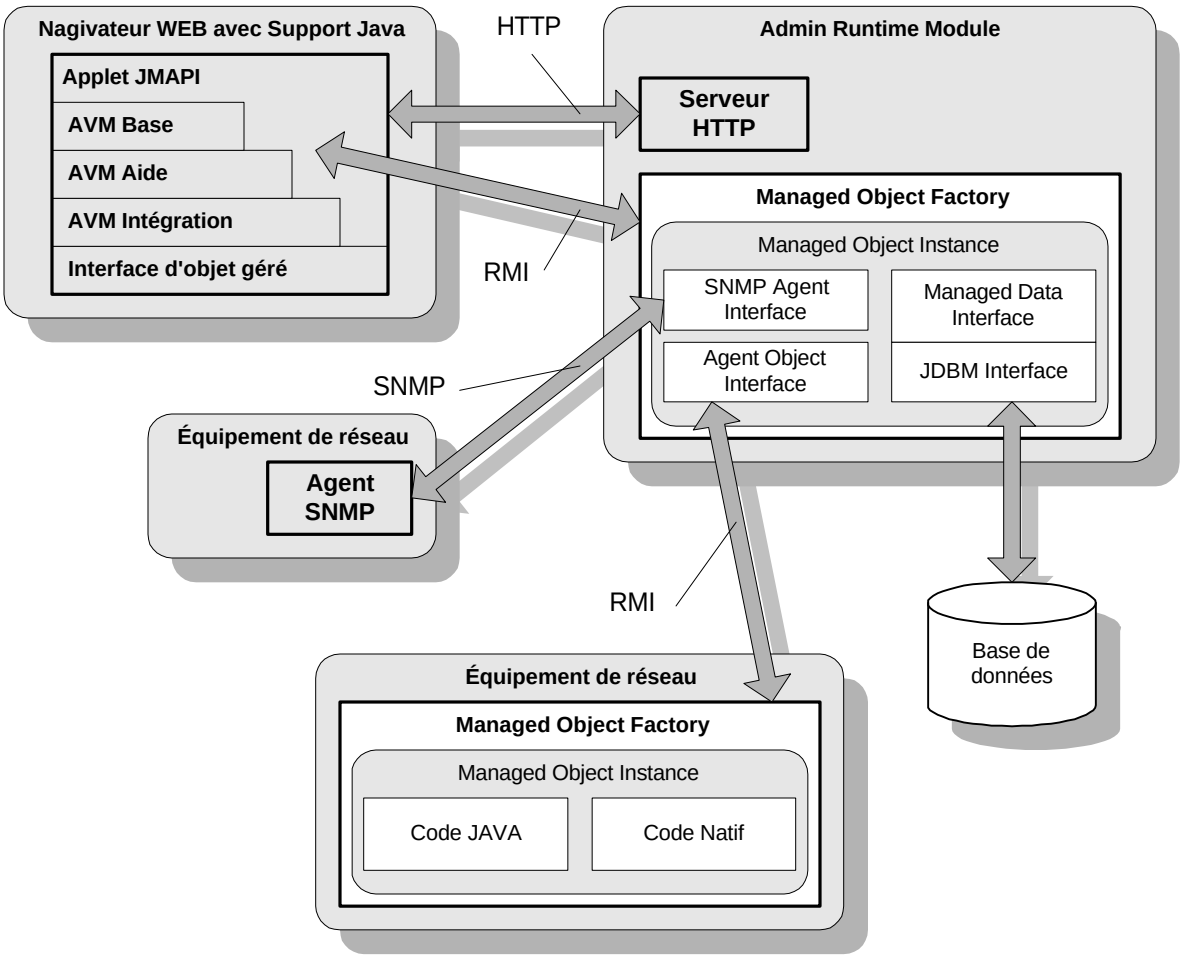


Figure 7 : Architecture de JMAPI

Les principales caractéristiques de ces architectures par rapport à leur ancêtre HTTP/CGI sont les facilités offertes pour une intégration d'autres technologies, notamment SNMP, mais aussi un modèle de données presque commun, qui met l'accent sur l'uniformité des données et de leur représentation.

Pour plus de détails sur WBEM, consulter la page web du consortium WBEM à : http://wbem.freerange.com/.

Les avantages de JMAPI

Comme JMAPI est basé sur Java, le langage et les programmes de gestion sont neutre. Ceci est un grand avantage quand on gère un réseau hétérogène. Aussi, les programmes compiler peuvent être charger à distance et exécuter, ce qui facilite la mise a jour.

JMAPI offre aussi une solution complète de gestion. Java est utiliser pour obtenir les données, pour faire les décisions et les afficher dans un navigateur.

1.6.4 La solution CORBA

L'Object Management Group (OMG) fondé dès 1989 à l'instigation de grandes sociétés telles que HP, Sun, Unisys, a répondu aux besoins actuels d'interopérabilité face à un parc de matériels et logiciels très divers, par la création d'une architecture distribuée hétérogène, basée sur un modèle orienté-objet totalement indépendant de l'implémentation des divers objets qui le composent.

Ainsi est née CORBA (Common Object Request Broker Architecture) en 1991.

CORBA définit un langage générique de définition d'interfaces : « Interface Definition Language » (IDL) et des APIs pour permettre à tout objet de communiquer avec des ORB.

CORBA est donc un bus logiciel pour les systèmes répartis. En effet, c'est un « middleware » solide qui permet une transparence des transactions locales et distantes entre les objets. Un atout de cette architecture est qu'elle n'est inféodée à aucune grande compagnie, mais à un ensemble d'entreprises.

Enfin, avec l'apparition du CORBA IIOP (Internet Inter ORB Protocol), on voit apparaître une alternative à HTTP pour le Web dans une architecture 3 tiers. La gestion de réseaux par CORBA se fait en plaçant des objets qui représentent les équipements à gérer sur le bus. On effectue des opérations de gestion en utilisant les méthodes offertes par ces objets.

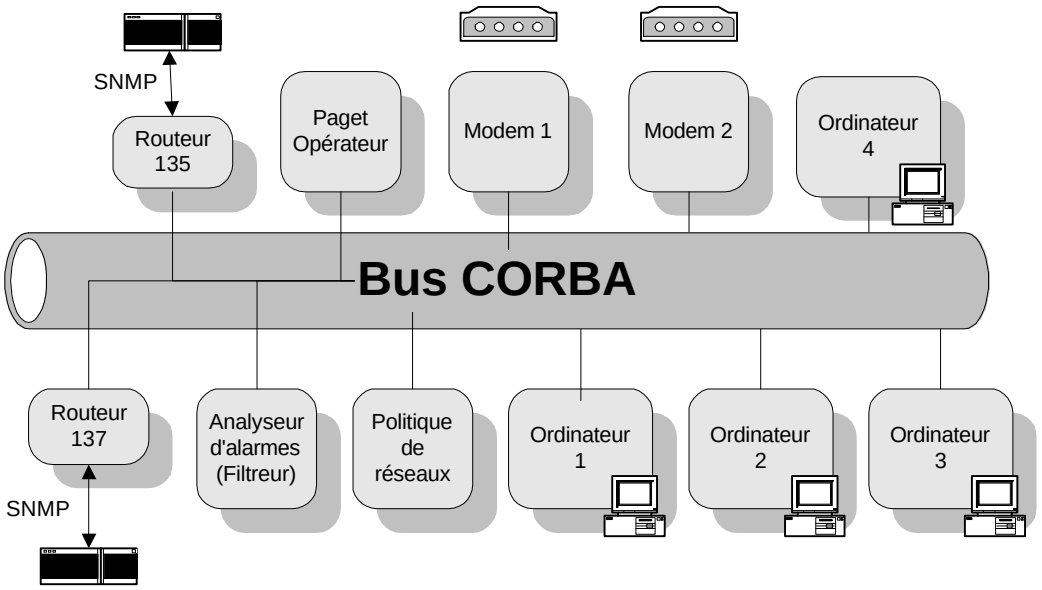


Figure 8 : Exemple de réseau géré avec CORBA

Les avantages du modèle basé sur CORBA

Le langage IDL permet de définir les composantes du modèle sans se préoccuper de la mise en œuvre de la ressource. L'utilisation des « Dynamic Invocation Interface » (DII), permet de connaître les méthodes invocables d'un objet au moment de l'exécution et permet l'ajout ou la suppression des objets gérés à la volée.

Enfin, pour assurer une interopérabilité sans faille avec les protocoles et standards existants, deux approches peuvent être utilisées :

- Encapsuler les systèmes existants en utilisant les interfaces en IDL;
- Utilisation des passerelles

1.6.5 Les solutions de gestion de réseaux et SNMPv3

On constate que les solutions de gestion de réseaux présentées dans cette section offrent bien plus qu'une façon de transporter les données de gestion d'un point à l'autre. Les autres

solutions offrent : une architecture de gestion, une description des composantes et dans certains cas, un modèle d'information [ISO 89] par lequel on peut exprimer les informations de gestion.

Les standards SNMPv1 et SNMPv2c décrivent essentiellement un protocole de transport des données entre un agent et une plate forme de gestion. Rien n'est mentionné sur l'architecture ou l'environnement dans lequel SNMP doit être utilisé. Aussi, il revient aux documents de standardisation des MIBs de décrire les données qui sont transportées. Or, les MIBs n'offrent pas de modèle d'information, ni de méthode par laquelle les informations d'un agent peuvent être étendues pour couvrir les besoins spécifiques de chaque instrumentation. Il y a donc beaucoup de MIBs propriétaires, définis par des compagnies, qui sont utilisés mais dont les spécifications et la description de leurs comportements ne sont pas connues.

Avec l'arrivée de SNMPv3, le standard SNMP décrit maintenant une architecture pour la plateforme de gestion et l'agent. Toutefois, SNMP reste un protocole simple, dont la vocation première est d'être placé dans les instruments de réseau. SNMP est donc un bon protocole pour obtenir des informations de gestion à partir des équipements de gestion, maintenant encore meilleur avec l'ajout de la sécurité. Le chapitre 3 donne les détails sur le standard SNMPv3.

Conclusion au chapitre

Dans ce chapitre, on a brièvement vu le monde de la gestion de réseau et quelques-unes des solutions qui sont en usage. Toujours en visant la construction d'un logiciel SNMPv3, le prochain chapitre aborde des thèmes indispensables à la bonne conception d'un logiciel de gestion de réseau.

Chapitre II

Ce chapitre a deux volets. On commence par une introduction rapide de la conception orientée objet, puis on entre plus en détail dans la conception d'un logiciel de gestion de réseau en abordant des problèmes qui ont déjà été observés par le passé.

Dans le deuxième volet, on décrit les composantes d'une architecture de tests. Ceci est nécessaire pour développer des tests efficaces pour un logiciel.

2.1 Le génie logiciel appliqué aux protocoles de communications

L'orienté objet est une technique qui est à la base de tous les nouveaux projets de développement de logiciels. La mise en œuvre de logiciels réseau n'échappe pas à cette tendance. Comme tous les autres projets logiciels, les logiciels réseaux peuvent profiter des avantages de l'orienté objet [ROSP 92]. La prochaine section décrit les fondements de l'orienté objet. On utilisera ces principes pour développer SNMP-Modulaire, et la mise en œuvre de SNMPv3 au chapitre 4.

La justification pour l'utilisation du modèle objet dans un logiciel SNMP (et comme beaucoup de logiciels) est qu'il permet la construction d'un programme solide, bien structuré, facile à visualiser et qui est facile à modifier et à maintenir.

2.1.1 Les principes de la méthodologie de conception orientée objet

L'orienté objet est une méthodologie qui permet d'exprimer la structure et le comportement de systèmes complexes [SBAP 94]. Cette section décrit sommairement la méthodologie orientée objet. Les notions comme l'abstraction, l'encapsulation, la classification, l'héritage, l'extension, l'évolution, et les éléments de construction seront développés. On verra par la suite comment cette technique a été appliquée à la conception du protocole SNMPv3.

L’orienté objet est un paradigme, un cadre qui est utilisé pour modéliser des problèmes. Les fondations de l’orienté objet sont basées sur l’habilité humaine à cataloguer les choses.

Quand on catalogue des choses, on peut considérer une grande quantité d’unités en même temps. Les classes ou catégories d’objets ont tels comportements ou tels attributs. Le nom donné aux catégories (classes) d’éléments, permet de pouvoir en parler dans l’abstrait (tous les mammifères sont…).

Une bonne technique de modélisation permet une classification basée sur la structure et le comportement. Dans le tableau périodique des éléments, la structure de l’élément est décrite par les rangées et le comportement par les colonnes. Une des raisons de la popularité de la technique orienté objet est l’attrait intellectuel à capturer la structure et le comportement dans une seule abstraction d’un objet.

Certains modèles, comme l’entité-relation, décrivent uniquement le domaine du problème dans des termes structurels. Ou d’autres, comme le diagramme de transition d’états, décrivent uniquement le comportement. Ces modèles sont bien adaptés à certains problèmes. Toutefois, beaucoup d’autres problèmes peuvent aussi être décrits par ces deux modèles.

Le problème de la communication sur les réseaux est un exemple de problème qui comporte en même temps, un aspect structurel et comportemental. La description des réseaux sous forme strictement structurelle montrerait les différentes composantes du réseau et les relations entre elles. Toutefois, ce modèle ne montre pas le comportement entre les entités (comme la séquence des messages transmis).

Pour décrire correctement un réseau , on doit décrire les deux aspects. La modélisation objet permet de décrire correctement ces deux aspects. On pourra donc décrire dans un seul cadre, la structure et le comportement.

La capacité d’identifier des regroupements d’objets similaires est une partie importante de la méthodologie. On doit aussi trouver les caractéristiques statiques et dynamiques de chaque objet. Le regroupement d’objets en classes est, dans sa nature, une généralisation du problème. On factorise des éléments communs et on identifie les exceptions. Les thèmes de l’analyse orientée objet sont : l’abstraction, l’encapsulation et l’héritage. Même si différentes notations

existent, toutes les notations orientées objets expriment toutes les mêmes concepts fondamentaux. La transition d'un formalisme à un autre est donc souvent facile et même « automatique ».

2.1.2 L'abstraction

L'abstraction est un mécanisme qui permet de travailler avec les problèmes complexes. C'est une façon de se concentrer seulement sur les détails du problème spécifique que l'on a à résoudre [SBAP 94][ROSP 92]. On dit aussi que l'abstraction est l'exercice de déterminer quels sont les éléments d'information importants et lesquels ne le sont pas.

Le processus d'abstraction dessine une ligne autour d'un objet. On marque une délimitation conceptuelle. Dans cet objet, on retrouve les caractéristiques essentielles de l'objet qui le rendent différent de tous les autres. Ces caractéristiques sont appelées les « Propriétés » de l'objet. Les caractéristiques qui sont englobées dans l'objet doivent représenter la structure et le comportement de cet objet. On a donc des caractéristiques « Attributs » et des caractéristiques « Fonction ».

Aucune abstraction ne peut décrire complètement un objet. Si un objet doit représenter la réalité, il doit avoir beaucoup de propriétés. Toutefois, les propriétés ne sont pas toutes intéressantes dans le contexte où on utilise l'objet. Il faut donc choisir entre l'exactitude et la performance, quand on décrit un objet. Une abstraction devra être choisie selon le contexte de l'application. L'abstraction devra être telle que les objets ne sont pas décrits avec des détails qui ne sont pas importants au problème.

2.1.3 L'encapsulation

On fait de l'encapsulation quand on cache de l'information. On cache les détails du fonctionnement d'un élément, et on sépare le comportement visible à l'extérieur [SBAP 94][ROSP 92]. On dit donc que les détails internes d'un objet ne sont pas importants tant qu'on les connaît, et qu'on peut prédire le comportement extérieur de l'objet.

L’encapsulation n’est pas unique à l’orienté objet. Mais ensemble avec l’abstraction, elle nous permet de décrire clairement des interfaces internes et externes des objets.

2.1.4 La classification

On dit que plusieurs objets appartiennent à la même classe d’objets si ces objets ont le même comportement, ou s’ils ont une ressemblance structurelle. On peut regrouper les objets semblables en classes, on fait donc de la classification.

Avec l’abstraction, la classification permet de grouper les éléments semblables du système, et de les traiter d’un seul coup pendant la conception.

Il n’y a pas de façon unique de regrouper des objets en classes. On peut généralement se baser sur beaucoup de critères.

2.1.5 Les attributs

Les attributs décrivent les propriétés structurelles d’un objet [ROSP 92][SBAP 94]. Les attributs sont des valeurs qui rendent différent un objet d’un autre, qui est dans la même classe. Toutefois, un attribut est assigné à une classe d’objets.

Chacun des attributs a un type. Le type d’un attribut définit la nature et les valeurs possibles de l'attribut. Quelquefois, des attributs possèdent des valeurs qui sont dépendantes des autres objets du même type. Par exemple, on peut décider que la valeur d’un attribut est identique pour tous les objets d’une classe, on peut aussi choisir que chaque objet d’une classe aura une valeur unique d’attribut.

2.1.6 Les fonctions

Les fonctions d’un objet décrivent son comportement. Les fonctions disent ce que l’objet fait, comment il agit. Une description complète d’une fonction spécifie les paramètres d’entrées et de sorties, ainsi que le traitement effectué par la fonction.

2.1.7 L'héritage

Les classes permettent de classifier des objets, l'héritage permet de classifier les classes. Quand plusieurs classes ont des points communs, on les regroupe en super classes. On factorise donc ce que les sous-classes ont en commun dans la super-classe.

L'héritage apporte donc un deuxième ordre d'abstraction. On n'est pas limité à manipuler des classes d'objets. Avec l'héritage, on effectue des opérations sur des groupes de classes.

2.1.8 L'instanciation

Une fois qu'une classe est définie, on peut créer des instances de cette classe. On crée donc les individus de cette classe. Ces individus peuvent avoir (et ont généralement) des attributs différents l'un de l'autre. L'activité de créer un nouvel objet d'une classe est appelée une instanciation.

On dit d'une classe dont on peut créer des objets qu'elle est une classe concrète, par opposition aux classes abstraites, qui ne sont utilisées que pour regrouper les attributs communs d'autres classes.

2.1.9 L'agrégation

L'agrégation est aussi appelée la composition. On y fait référence quand on crée un objet complexe à partir d'objets plus simples [SBAP 94]. Tous les systèmes qui sont le moindrement complexes sont généralement une composition de systèmes plus simples.

Comme il n'est pas facile, sinon impossible, de comprendre un système complexe avec tous ses détails d'un seul coup, il est important de comprendre comment un système est divisé en sous-systèmes, et on peut par la suite comprendre les sous-systèmes individuellement. Généralement, on divise le système jusqu'à l'obtention des différentes pièces, que l'on peut comprendre facilement. On appelle ceci la décomposition.

La décomposition n'est pas unique à la modélisation objet. Toutefois, quand on découpe un modèle objet, on doit s'assurer que le découpage ne se fera pas n'importe comment. Chaque partie de la décomposition doit être telle qu'elle représente un objet en elle-même. Donc,

chacun des sous-systèmes doit conserver une encapsulation et des interfaces bien définies [ROSP 92].

Les différentes parties d'un système pourront être rassemblées en utilisant l'agrégation. On peut décrire l'assemblage de toutes les pièces du système comme une hiérarchie d'agrégation. Souvent, on désire réutiliser des pièces d'un système pour en construire un autre. Si les morceaux du système sont bien définis, on pourra les réutiliser ailleurs.

2.2 La conception de logiciels de gestion de réseau

Avant de concevoir un logiciel qui est basé sur SNMP, il est important de connaître ce qui a déjà été fait dans le domaine. Il existe beaucoup d'informations sur la construction d'un agent SNMP et la construction d'une MIB dans cet agent [DPAR 97]. Comme SNMPv3 garde le même système de MIB que ses prédécesseurs (SNMPv1, SNMPv2c…), cette connaissance peut-être utilisée. On retrouvera l'architecture d'une MIB et l'architecture d'un agent dans la conception de SNMP-Modulaire au chapitre IV.

2.2.1 L'architecture logicielle d'une MIB

Ceux qui utilisent SNMP peuvent quelquefois avoir la fausse impression qu'une MIB est une base de données ou une liste, qui est contenue dans l'agent et dont on peut lire et écrire les valeurs. En vérité, une telle liste n'existe pas dans l'agent, les informations de gestion se trouvent dans l'instrumentation. Certaines informations de gestion sont une abstraction, qui n'existe qu'au moment où il y a une requête pour cette information.

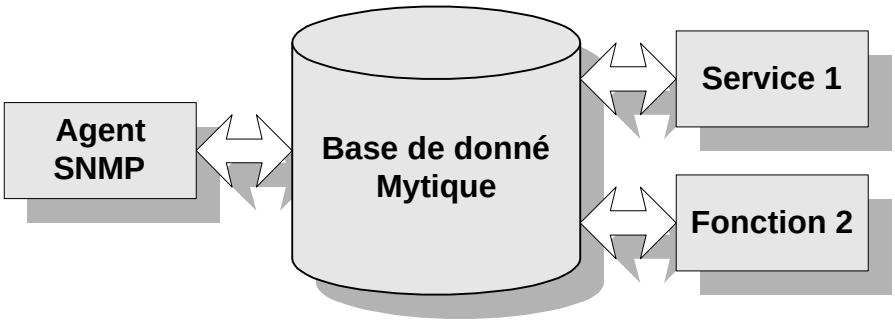


Figure 9: Base de données mytique

Si l'information de gestion était réellement conservée dans une base de données, la construction d'un agent pour ces données serait très facile. Toutefois, la consommation en mémoire et en temps processeur d'un tel agent seraient très élevée. Toutes les informations, comme les compteurs et les indicateurs qui changent souvent, devraient être changées dans la base de données à chaque fois. Aussi, deux copies de l'information existeraient, dans l'instrumentation et dans la base de données de l'agent.

En réalité, les agents SNMP utilisent un « Dispatch table », une table qui indique pour chaque OID, la méthode qu'il faut appeler pour obtenir la valeur de cet OID. On garde aussi les informations sur la méthode à appeler pour changer la valeur d'un OID. Quand il y a une requête pour des informations de gestion, les méthodes correspondantes aux OIDs demandées sont appelées. Ces méthodes font le nécessaire pour chercher ou calculer les informations de gestion.

Le « Dispatch table » [DPAR 97] doit contenir tous les OIDs de gestion en ordre lexicographique. Ceci est nécessaire pour répondre aux requêtes GETNEXT (Donne le prochain) et GETBULK (donne les n prochains). On pourrait attendre la requête et, une fois la requête reçue, demander à l'instrumentation la valeur de ce OID existant. Mais ceci fonctionne uniquement pour les GET et SET. Un « Dispatch table » ordonné est essentiel pour répondre à GETNEXT et GETBULK.

2.2.2 L'Architecture d'un agent

Il existe deux approches prédominantes à la construction de la MIB d'un agent. Pour des petits systèmes on utilise une approche monolithique. Dans cette approche, l'agent, les méthodes et la plupart de l'instrumentation sont placés dans un seul bloc. On utilise l'approche monolithique quand toutes les composantes du système sont connues d'avance.

Autrement, on utilise l'approche dite extensible. Cette approche garde l'agent et l'instrumentation complètement séparé jusqu'au chargement ou l'exécution du système. Cette approche est plus complexe, résulte en un programme plus gros et légèrement moins rapide que

l'approche monolithique. Toutefois, cette approche est préférée quand on a un ordinateur à plusieurs processeurs ou quand on désire l'ajout et le retrait de MIBs dynamiquement, pendant l'exécution.

Si un agent doit effectuer la gestion de plusieurs instrumentations, il y a trois techniques qui peuvent être utilisées :

- Un seul point de contact
- L'agent d'encapsulation
- Les adresses non-standards

Il est important de rappeler que l'on parle de regroupement des ressources gérées dans un seul système. D'autres techniques comme le « Proxie » peuvent être utilisées pour regrouper plusieurs agents qui sont à l'extérieur du système.

2.2.2.1 Un seul point de contact

Cette technique est toujours préférable [DPAR 97]. Un agent offre à tous l'instrumentation, une seule interface par laquelle l'instrumentation peut communiquer avec l'agent et vice versa. Tel que montré à la Figure 10.

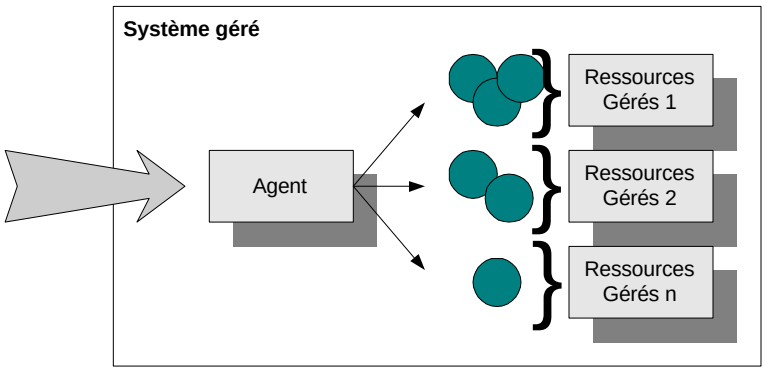


Figure 10, Regroupement de MIBs par un seul point de contact

2.2.2.2 L’agent d’encapsulation

Une seule interface avec toute l’instrumentation est impossible, on peut utiliser un agent d’encapsulation. Une série d’agents monolithiques, conçu pour des instrumentations spécifiques, sont placés derrière un autre agent qui relaie les requêtes de gestion (voir Figure 11).

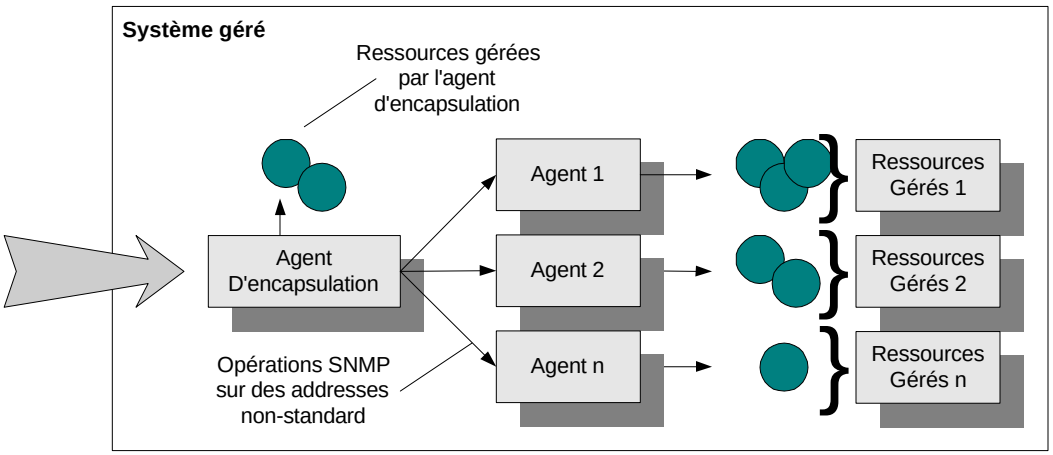


Figure 11 : Regroupement des MIBs par agent d'encapsulation

Si cette technique est bien appliquée, il sera impossible pour la plate forme de gestion de savoir qu’un agent d’encapsulation est utilisé.

2.2.2.3 Les adresses non-standards

Une autre technique est de laisser à la plate forme de gestion le travail de regrouper des différentes instrumentations d’un système. Pour ce faire, le système exécute plusieurs agents SNMP, chacun d’eux fait la gestion d’un ou plusieurs instruments.

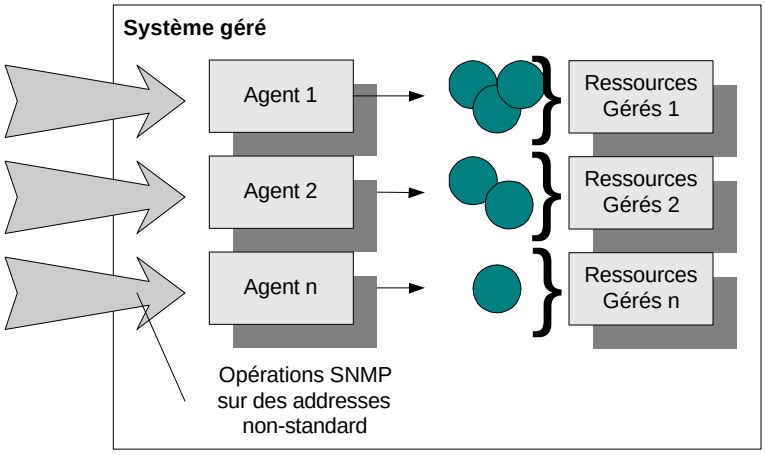


Figure 12 : Utilisation d'adresses non-standards

La plate forme de gestion doit, dans ce cas ci, connaître l'adresse de chacun des agents et communiquer individuellement avec chacun d'eux.

Cette approche a l'avantage que si un problème surgit avec un des agents, il peut être isolé facilement. Toutefois, le plus gros problème est que les plate formes de gestion ne sont pas présentement conçues pour travailler avec cette technique.

2.3 Les méthodologies d'essais (tests)

Pour qu'un logiciel soit fiable, il doit être soumis à des tests. Les tests visent à assurer le bon fonctionnement du produit final. On s'assure aussi que le produit correspond bien à ce qui a été demandé.

Dans le domaine du logiciel, on doit utiliser des méthodologies et développer des architectures de tests pour permettre des tests efficaces et le plus rapide possible. Cette section vise à introduire des méthodologies et des architectures de tests. On appliquera ces techniques sur le programme SNMP-Modulaire au chapitre V.

2.3.1 L'introduction aux tests

Le cycle de développement de logiciel se compose d'un certain nombre d'activités dont une

bonne partie est manuelle [ROSP 92][WPER 95]. Ceci entraîne inévitablement un certain risque d'erreurs. Pour remédier à cette situation, les chercheurs en génie logiciel ont proposé d'inclure à la fin de chaque phase du cycle de développement de logiciel une étape de vérification. Cette dernière permet de valider les résultats de la phase en cours avant de passer à la phase suivante.

Cette même philosophie est utilisée à la fin de la phase de réalisation. L'étape de tests représente la dernière activité de cette phase, elle suit immédiatement celle de la mise en œuvre. Cette activité permet de vérifier l'implantation avant de certifier le produit. L'étape de test est donc l'étape qui garantit la qualité d'une implantation. Elle consiste à valider le comportement par rapport à des données d'entrées spécifiques (cas de test) sélectionnées à l'avance. L'objectif de cette étape est alors la stimulation des fautes, l'observation de leur manifestation (erreur), leur localisation et éventuellement leur correction. Dans la littérature, les deux dernières activités (localisation et correction des erreurs) peuvent être regroupées dans une étape indépendante appelée « debogage ».

2.3.2 Les tests de conformité ou de validation

Ce genre de tests est très important dans le domaine du génie logiciel. Il consiste à vérifier la conformité de l'implantation par rapport à la spécification de référence. Ces tests se concentrent sur le comportement extérieur de l'implantation. Ils consistent à appliquer un ensemble d'entrées spécifiques (suite de tests) au système à tester et à vérifier la conformité aux spécifications.

2.3.3 Les types de tests

Il existe plusieurs sortes de tests de logiciel. Chacun vise à vérifier un aspect particulier du comportement ou de la structure du logiciel. Ces aspects sont plus ou moins indépendants et peuvent être traités séparément.

Parmi ces aspects, on peut considérer les entrées - sorties. Un tel test ne peut pas être exhaustif vu le nombre important de cas à traiter. Les entrées sont alors sélectionnées de façon à obtenir une couverture maximale, c'est à dire de manière à détecter le maximum de fautes et à vérifier

les fonctions importantes du système à tester.

2.3.3.1 Les tests de l'usager

Ces tests sont effectués par l'usager en vue de vérifier si 1e produit final répond bien à ses besoins. Généralement, ils consistent à essayer le système sur des situations réelles et propres à l'environnement de l'usager. Le résultat de ces tests est l'acceptation, l'acceptation conditionnelle ou 1e refus du produit.

2.3.3.2 Les tests d'interopérabilité

Ces tests vérifient si le système développé interagit correctement avec d'autres systèmes extérieurs avec lesquels il est en relation. Ces tests nécessitent l'observation des échanges entre les systèmes. Ceci se fait par l'intermédiaire de modules spéciaux placés entre le système à tester et les autres systèmes avec lesquels il est en relation. Ces modules sont appelés les arbitres.

2.3.3.3 Les tests de performance

Ces tests ont pour but de vérifier tout ce qui se rapporte à la performance du système comme le débit, le temps de réponse du système. La performance est vérifiée en soumettant le système à différentes conditions d'utilisation. Ces conditions sont parfois extrêmes. Ce genre de tests est très important pour les systèmes en temps réel.

2.3.3.4 Les tests de robustesse

Ces tests s'intéressent au degré de résistance de l'implantation à des événements externes ou à des erreurs non prévues par la spécification; c'est-à-dire les capacités du système à fonctionner dans des situations non prévues par la spécification.

2.3.3.5 Les tests de matériel

Comme dans le logiciel, l'étape de test est celle qui garantit la qualité d'un circuit. En effet, avant de commercialiser un circuit, il faut s'assurer de l'exactitude de son fonctionnement. Cette vérification se fait au cours de l'étape de test. Cette étape est constituée de trois principales phases: la génération, l'application et la vérification du test. La génération de tests

consiste à utiliser des techniques algorithmiques ou aléatoires pour générer des entrées de test. Ces dernières sont appelées vecteurs de test. La deuxième phase consiste à appliquer les vecteurs de test au circuit à vérifier. Finalement, au cours de la dernière phase, on compare les réponses à celles produites par une unité de référence.

2.3.4 Les principales philosophies de test

Selon ce que l'on veut tester dans un système (la logique, la structure, les fonctions...), les tests varieront et s'intéresseront au code, à l'architecture ou aux entrées sorties. Les tests de conformité par exemple ne s'intéressent qu'aux entrées/sorties du système. Ces tests sont appelés tests de boîte noire. D'un autre côté, les tests d'intégration s'intéressent plutôt à l'architecture et à la structure modulaire du système, ces tests sont appelés tests de boîte grise. Finalement, les tests unitaires se basent sur le code de chaque module pour mener à bien leur vérification, ce genre de tests est appelé le test de boîte blanche.

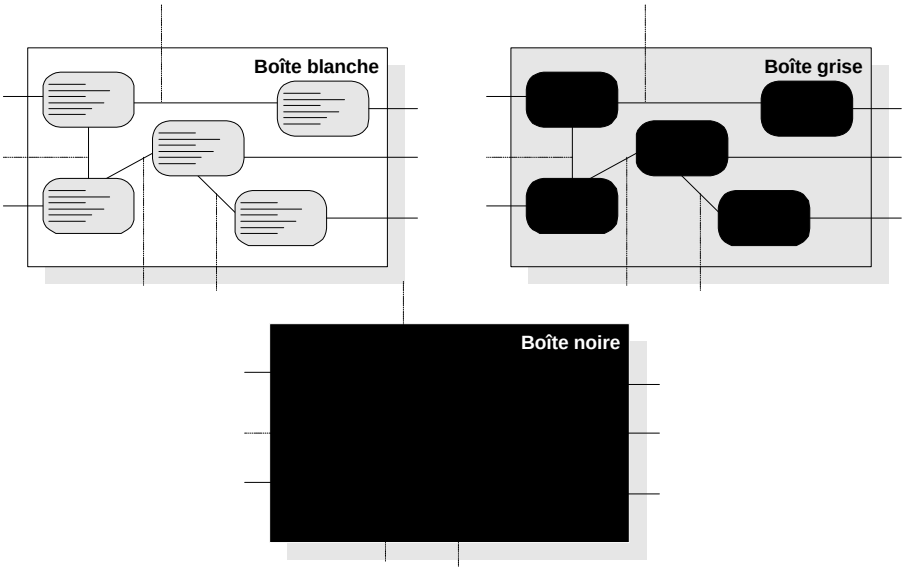


Figure 13, Architectures des tests

2.3.5 Les tests de boîte blanche

Ils sont aussi appelés tests de structure. Ces tests sont effectués sur des produits dont la structure interne est accessible. Ils s'intéressent principalement aux structures de contrôle et

aux détails procéduraux. Ils permettent de vérifier si les aspects intérieurs de l'implantation ne contiennent pas d'erreurs de logique. Ils vérifient si toutes les instructions de l'implantation sont exécutables (contrôlabilité). Ils permettent aussi de vérifier les chemins de base de l'implantation en vue de détecter les erreurs de logique tels que les blocages, les boucles infinies... Plusieurs techniques de tests peuvent être utilisées dans cette catégorie, nous citerons par exemple, les tests de chemins de base, les tests de conditions, les tests de Rot de données, les tests de boucles...

2.3.6 Les tests de boîte noire

Ils sont aussi appelés tests fonctionnels. L'objectif de ces tests est la vérification de la conformité des fonctionnalités de l'implantation par rapport à une spécification de référence. Les détails internes des logiciels ne sont pas inspectés. Ils ne s'intéressent qu'au comportement extérieur du système à tester. Ces tests permettent de détecter une large gamme d'erreurs comme des fonctions incorrectes ou manquantes, des erreurs d'interfaces, des erreurs de structures de données ou d'accès aux bases de données extérieures, des erreurs de performance ainsi que des erreurs d'initialisation ou de terminaison.

Les tests fonctionnels sont orientés données. Pour des systèmes de taille réelle, les données sont nombreuses, complexes et appartiennent à de très larges domaines. Ceci rend impossible l'exhaustivité des tests fonctionnels. Le nombre de cas de tests nécessaires à affirmer l'exactitude de l'implantation est très grand. Les cas de tests sont alors choisis de façon à couvrir un maximum de fautes ou de fonctionnalités.

Il existe plusieurs techniques pour la sélection des cas de test. L'idée de base est d'échantillonner les données en classes plus ou moins indépendantes qui serviront pour le choix des cas de tests les plus significatifs (dans le sens qu'ils couvrent toutes les classes). Générale ment, la sélection est faite à partir d'une spécification du système décrite soit en langage naturel, en langage formel ou sous un formalisme mathématique tel que les automates à états finis.

2.3.7 Les tests de boîte grise

Cette approche est utilisée lorsque la structure modulaire du logiciel est accessible; c'est à dire lorsqu'on peut observer les interactions entre les modules qui composent le logiciel. L'observation des interactions se fait via des points spécifiques du logiciel appelés points d'observation. Cette approche n'a donc pas besoin des détails internes des procédures, mais exige l'observabilité de la structure du logiciel via les points d'observation. La visibilité de la structure interne du logiciel peut aider lors de l'étape de débogage.

2.3.8 L'architecture de test

Pour pouvoir tester les fonctionnalités d'un système (tests de conformité), on a besoin de lui appliquer des séquences d'entrée et d'analyser les séquences de sorties. Généralement, l'accès direct aux points d'entrées/sorties du logiciel est difficile. Pour ces raisons, les entrées sorties du logiciel sont indirectement contrôlables. L'application des entrées et l'analyse des sorties doivent être synchronisées. Toutes ces difficultés peuvent être résolues en utilisant un système qui permet de tester le logiciel appelé testeur. L'architecture du testeur varie et dépend de l'accessibilité des points d'entrées sorties du logiciel.

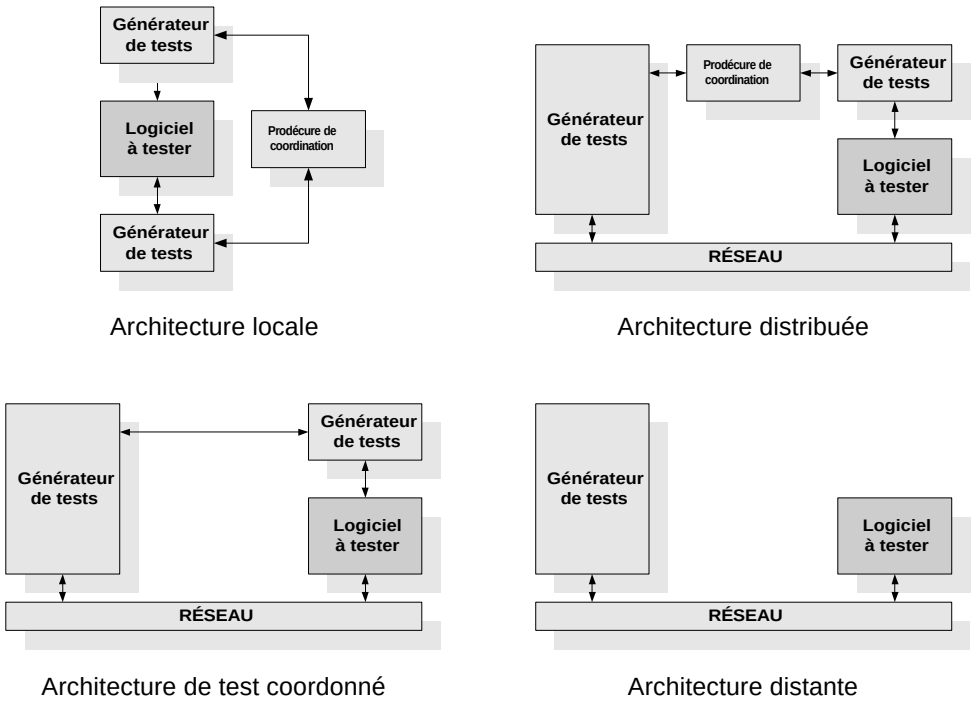


Figure 14 : Architecture de tests distribués

Les principales architectures de tests sont :

- **Architecture locale** : Dans cette architecture, les points d'accès aux entrées sorties de l'IST sont directement accessibles. Le logiciel à tester est alors directement « stimulé » par les générateurs de tests. Les 3 modules sont localisés sur le même site et sont synchronisés grâce à une procédure de coordination qui permet l'échange d'informations et obtenir les données de tests.
- **Architecture distribuée** : Les tests de conformité ne sont pas effectués par le développeur de logiciel lui-même. Ils sont réalisés par des institutions nationales ou internationales. La nécessité d'une architecture non locale est alors apparue. Le logiciel à tester se trouve sur le même site que l'un des générateurs de tests, et un autre générateur est placé sur un autre site et communique via un réseau. Une procédure de coordination synchronise les deux générateurs de tests.
- **Architecture de tests coordonnés** : Cette architecture est presque similaire à

l'architecture distribuée, sauf que les procédures de coordination sont situées dans les générateurs de tests, et on utilise un protocole de gestion des tests.

- **Architecture distante** : Cette architecture offre moins de possibilités de contrôle des tests et de détection d'erreurs. Cette architecture s'applique lorsqu'on ne peut pas accéder aux interfaces du logiciel à tester. Le logiciel à tester est indirectement observable par le bas et non observable par le haut. Cette architecture a des limites de contrôle et d'observation.

2.3.9 Les tests ascendants

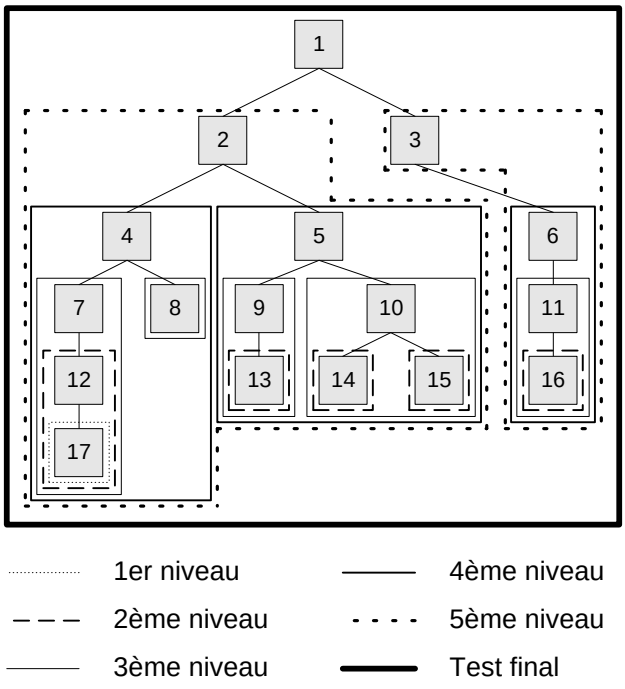


Figure 15 : Tests ascendants

C'est une stratégie classique de test. Elle consiste à tester les modules, ensuite les sous-systèmes et en dernier le système en entier. La première phase consiste à tester les modules dans l'objectif de découvrir des erreurs de logique, de fonctionnalité ou de structure. Les modules sont testés individuellement dans un environnement qui simulera le sous-système qui les englobe. Par la suite on passe au test des sous-systèmes en vérifiant les interactions entre les différents modules du sous-système. Ils s'intéressent principalement aux interfaces des

modules et aux échanges entre les modules. Les tests des sous-systèmes sont accomplis par un processus répétitif qui permet d'intégrer les modules de bases avec leur sous- systèmes, les sous-systèmes d'un certain niveau avec les sous-systèmes d'un niveau plus haut et ainsi de suite jusqu'à intégrer tous les systèmes. Ces tests sont donc hiérarchiques et doivent se faire du bas vers le haut. Les techniques de tests ascendants utilisent plusieurs types de tests comme : les tests unitaires pour tester les modules et les tests d'intégration pour tester les sous-systèmes.

Les tests descendants

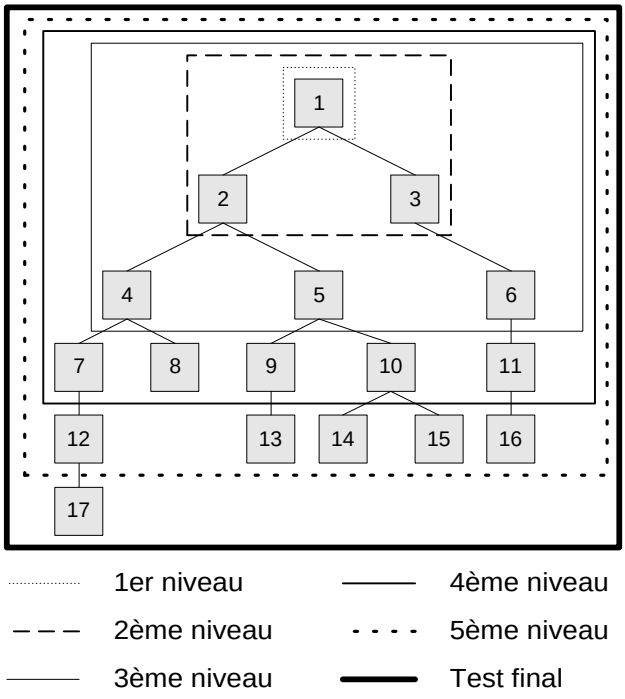


Figure 16 : Tests descendants

Cette approche est moins naturelle que la précédente. Elle commence par tester le programme principal avec ses sous routines immédiates. Après que cette tâche ait été effectuée on passe aux tests de niveaux plus bas. Ces derniers se basent sur ce qui a été déjà validé pour tester les modules d'un niveau plus bas. Ces techniques ont besoin d'utiliser des modules factices qui seront placés à un niveau plus bas que les modules à tester et qui ont pour rôle de récolter les

sorties des modules à tester. Cette stratégie suppose que la structure du système est hiérarchique. Dans plusieurs cas, elle est impossible à effectuer, car les entrées qui servent à tester un module proviennent d'autres modules appartenant à des niveaux supérieurs. Le problème ici, c'est la difficulté de trouver des entrées aux modules de niveaux supérieurs pour qu'ils puissent donner les bonnes entrées de test pour le module en question.

La conclusion au chapitre

En conclusion à ce chapitre, on peut indiquer que la construction d'un logiciel SNMP peut tirer beaucoup des recherches antérieures dans les domaines de l'orienté objet, de la construction d'autres logiciels de gestion de réseau et de conception de tests. Beaucoup de travaux dans le génie logiciel peuvent être utilisés pour assurer le succès du projet. Le succès d'un logiciel SNMP dans un cadre universitaire passe aussi par l'innovation d'autres solutions originales. Le chapitre suivant traite SNMPv3 en détail.

Chapitre III

Ce chapitre décrit en détail le protocole SNMPv3 et comment il a été découpé dans les documents du standard SNMPv3. On montre le fonctionnement des modules et décrit leurs interfaces. On voit aussi comment SNMPv3 assure la sécurité de transactions sur le réseau et l'identification des parties.

3.1 L'architecture du protocole SNMPv3

3.1.1 Les objectifs de SNMPv3

La nouvelle version du protocole SNMP vise essentiellement à inclure la sécurité des transactions. La sécurité inclut l'identification des parties qui communiquent et l'assurance que la conversation soit privée, même si elle passe par un réseau public.

Le protocole SNMPv3 est décrit en sept documents qui doivent être assez spécifiques pour assurer que si quelqu'un construit un logiciel SNMPv3 à partir de ces documents, le programme puisse fonctionner avec d'autres mises en œuvres de SNMPv3. Toutefois, les documents du standard ne visent pas à décrire comment construire un logiciel SNMPv3.

3.1.2 Le standard SNMPv3

La Figure 18 est tiré directement du document SNMPv3 (RFC 2271). Elle montre le fonctionnement et les divisions d'un gestionnaire SNMP traditionnel. On remarque qu'on a des divisions entre le transport, le traitement, la sécurité et les applications.

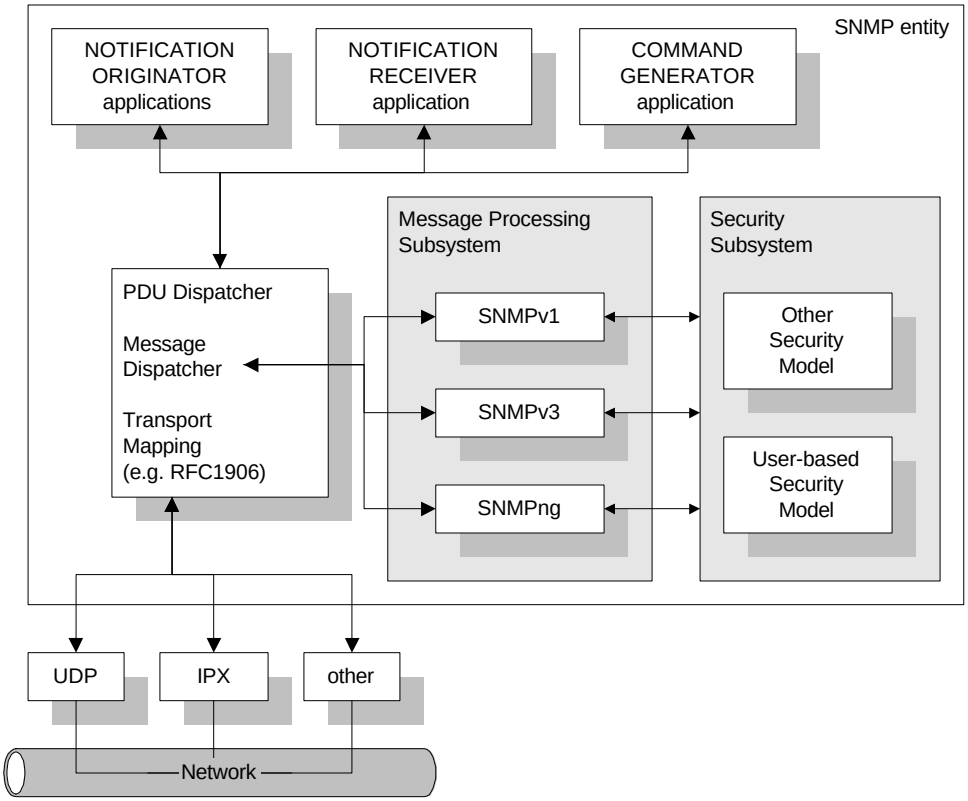


Figure 17 : Architecture SNMPv3 d'une plate-forme de gestion

Le prochain diagramme montre un agent SNMP selon le document SNMPv3 (RFC 2271). Dans un agent SNMPv3, on retrouve le module « access control » et les applications qui communiquent avec l'instrumentation.

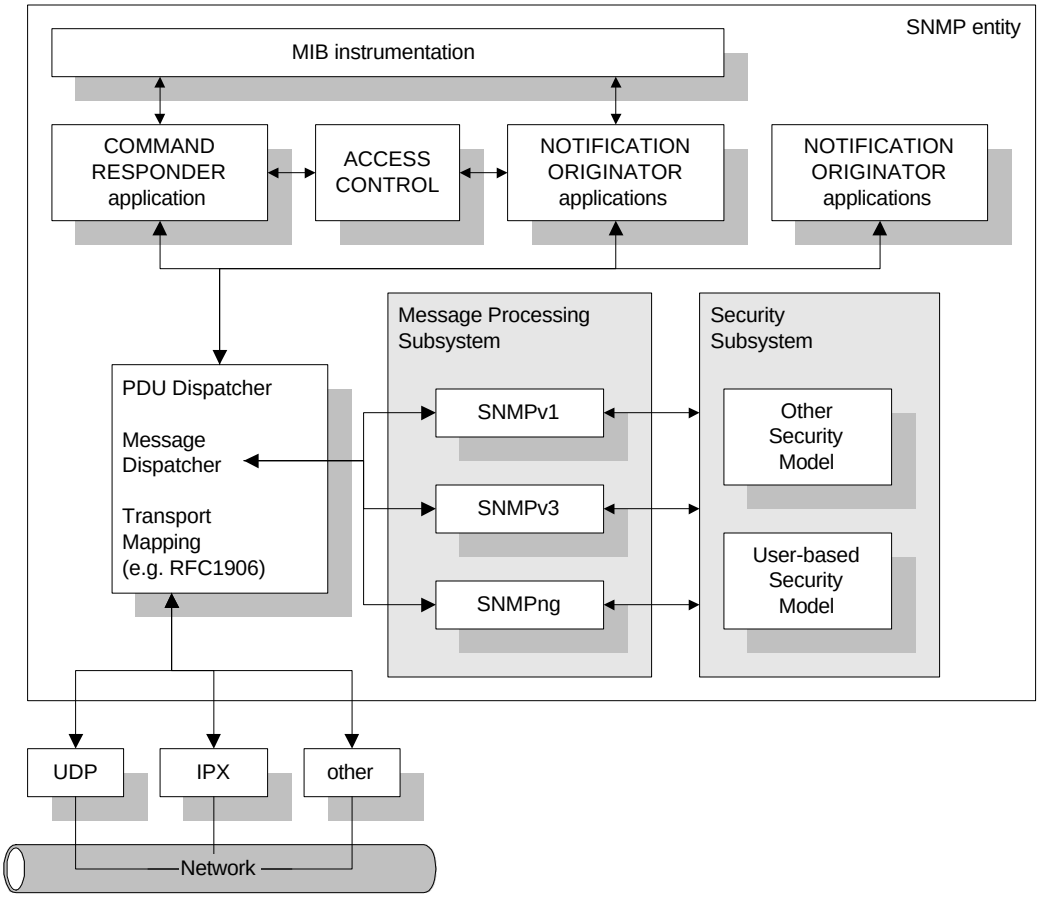


Figure 18 : Architecture SNMPv3 d'un agent

Le modèle SNMPv3 comporte donc plusieurs modules dont les plus importants sont :

- Le transporteur (Transport mapping);
- Module de traitement (Processing Module);
- La sécurité (Security Subsystem);
- Les applications.

Le standard ne fait pas que diviser les modules, il décrit aussi les primitives d'appels entre les modules. Il est important de rappeler que le standard n'impose rien quant à une mise en œuvre d'un logiciel SNMPv3. Un programmeur n'est pas encouragé à suivre le découpage des modules ou les interfaces décrits dans le standard. Le document du standard décrit donc les primitives d'appels entre les modules pour faciliter les changements et les ajouts au standard

SNMPv3. Le découpage en modules a pour seul usage de permettre le découpage du standard SNMPv3 en plusieurs documents.

3.1.3 La description des composantes

Voici une courte description de chaque module décrit dans le standard.

3.1.3.1 Les modules de transport

(UDP, IPX, autres)

Le protocole SNMP n'est pas lié à un protocole réseau et de transport particulier, même si en pratique il est plus que souvent utilisé sur le protocole UDP [RFC 768]. Pour permettre l'indépendance du moyen de transport, le module qui est responsable du transport est externe au moteur SNMP. Il est possible d'en placer plusieurs qui fonctionnent simultanément. On pourrait avoir par exemple, un module de transport UDP, un AppleTalk ou un RS-232.

3.1.3.2 Les modules de traitement

(MPv1, MPv3, MPng)

Un module de traitement est requis pour faire le décodage des paquets SNMP qui arrivent du réseau et l'encodage de ceux qui s'apprêtent à partir dans le réseau. Pour garder SNMP flexible, et permettre les changements futurs, chaque paquet SNMP est marqué d'un numéro de version SNMP (voir la description du paquet SNMP, page 53). Ce numéro de version permet à un moteur SNMP de déterminer à quel module de traitement ce message est destiné. Normalement, chaque module de traitement prend en charge une version de SNMP. Ce module a aussi la tâche d'imposer le respect d'un standard SNMP, pour la version qui le concerne.

Le module de traitement est modulaire, ceci permet l'adaptation aux prochaines versions de SNMP. Un ajout d'un nouveau module de traitement pourrait permettre au moteur SNMP de

comprendre des messages SNMPng (Snmp Next Generation). Ceci permet l'évolution du protocole SNMP sans entraîner les coûts de reconstruction complète du moteur.

Un des grands avantages est qu'il est possible de faire fonctionner plusieurs modules de traitements simultanément. Il est donc possible pour un moteur SNMP de traiter des messages SNMPv1, SNMPv3, SNMPng tous en même temps.

3.1.3.3 Le système de sécurité

Certains environnements requièrent des interactions sécuritaires. La sécurité est normalement appliquée en deux étapes: dans la transmission et la réception du message et dans le traitement du contenu du message. Trois fonctions sont généralement communes à tous les modules de sécurité: l'authentification, l'encryption et la vérification du temps.

Encore une fois, plusieurs modules de sécurité peuvent être actifs simultanément dans un moteur SNMP.

3.1.3.4 Les applications

Les applications sont des processus qui interagissent avec le moteur SNMP en utilisant des messages qui peuvent être définis dans le protocole, ou des messages décrits par une mise en œuvre spécifique du moteur.

Les applications sont développées pour effectuer des opérations de gestion spécifiques. Les objectifs peuvent être fort variés d'une application à l'autre. Toutefois, toutes les applications utilisent en commun le même moteur SNMP pour effectuer les opérations de gestion.

Par exemple, une application de gestion pourrait contrôler des éléments de gestion. Un proxy pourrait effectuer le relais de messages entre les médiums différents ou des versions différentes de SNMP.

3.1.3.5 Le contrôle d'accès

Le module du contrôle d'accès doit décider si une requête est permise et si une réponse peut être envoyée, ou l'ignorer si une personne non autorisée a fait la requête.

Chaque application a le choix d'accepter ou de rejeter une requête. Mais si plusieurs applications fonctionnent sur un même agent, il serait avantageux de centraliser le contrôle d'accès. Le standard SNMPv3 place donc un module de contrôle d'accès qui peut être interrogé par les applications. On peut donc faire une seule configuration des droits des accès.

Ce module peut autoriser une requête sur les critères de qui l'a demandé, quel type de requête et quelle information est touchée par la requête. On doit noter qu'à ce stade, l'authentification de l'entité (usager ou application) qui a effectué la requête a déjà été faite. Ce module n'a donc pas à s'en préoccuper.

3.1.4 La description du paquet SNMPv3

Le format du paquet réseau SNMPv3 est très différent du format de SNMPv1, ils sont toutefois codés tous deux dans le format ASN.1 [ISO 87] (qui assure la compatibilité des types de données entres les ordinateurs d'architectures différentes). Pour rendre plus facile la distinction entre les versions, le numéro de la version SNMP est placé tout au début du paquet. La figure suivante montre un paquet SNMPv3:

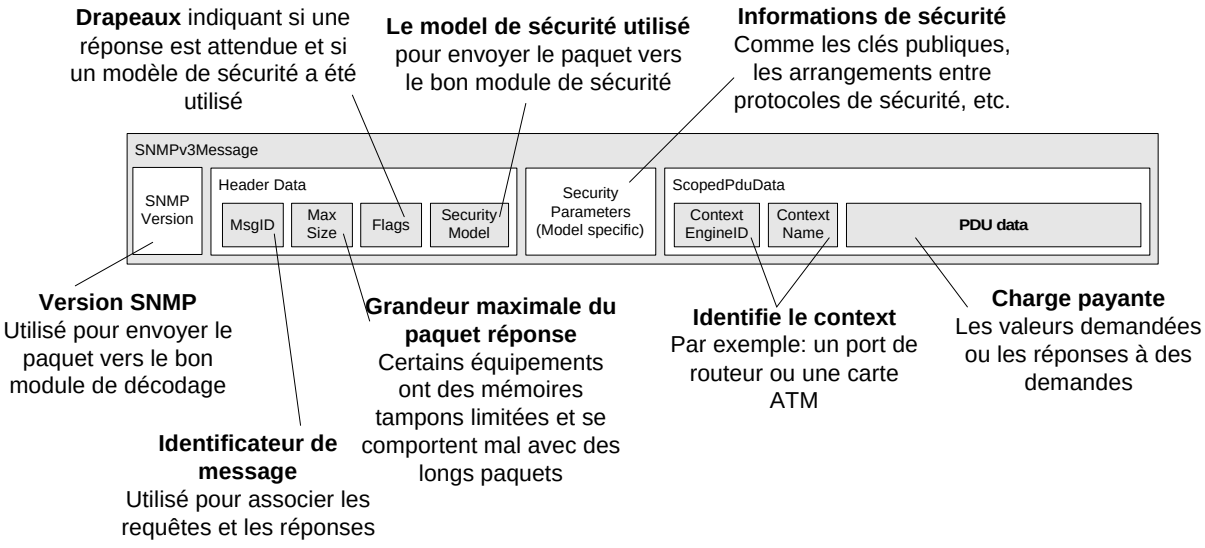


Figure 19 : Description du paquet SNMPv3

La figure montre les différents champs du paquet SNMPv3. Toutefois, le contenu de chaque champ varie selon la situation. Selon que l'on envoie une requête, une réponse, ou un message d'erreur, les informations placées dans le paquet respectent des règles bien définies dans le standard.

Voici comment les champs d'un paquet SNMPv3 sont remplis :

Version SNMP

Pour SNMPv3, on place la valeur 3 dans ce champ. On place 0 pour un paquet SNMPv1.

Identificateur de message

Ce champ est laissé à la discrétion du moteur SNMP. On retrouve souvent des algorithmes, où le premier message de requête est envoyé avec un nombre aléatoire et les suivants avec les incréments de 1. Les paquets qui sont émis en réponse à une requête portent la même identification que le paquet de la requête.

Taille maximale

Le moteur choisit la taille maximale d'une réponse à une requête selon ses capacités en mémoire tampon et ses limites à décoder de longs paquets. Quand on envoie une réponse à une requête, on doit veiller à ne pas dépasser la taille maximale.

Drapeaux

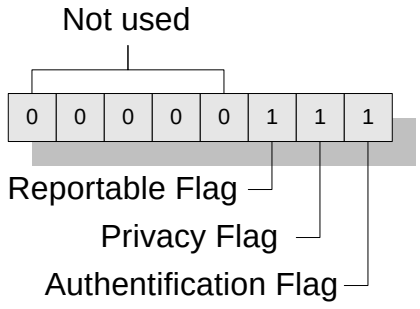


Figure 20, Les drapeaux SNMPv3

Présentement, trois bits sont utilisés (sur les 8) pour indiquer :

- Si une réponse est attendue à la réception de ce paquet. (Reportable Flag)
- Si un modèle d'encryption a été utilisé (Privacy Flag)
- Si un modèle d'authentification a été utilisé (Authentification Flag)

On note qu'il n'est pas possible d'envoyer un paquet qui utilise un modèle d'encryption sans qu'il soit aussi authentifié, donc qu'il utilise un modèle d'authentification.

3.1.4.1 Le modèle de sécurité

Ce module identifie le type de sécurité qui est utilisé pour encrypter le reste du paquet. Cet identificateur doit identifier de façon unique chaque module de sécurité. Le groupe de l'IETF recommande qu'il y ait relativement peu de modules de sécurité pour permettre de maximiser l'intercompatibilité des équipements. Présentement, l'algorithme d'encryption DES (Data Encryption Standard) et l'algorithme d'authentification HMAC-MD5-96 ont été choisis comme algorithmes utilisés dans SNMPv3. HMAC-SHA-96 est optionnel.

3.1.4.2 Les informations de sécurité

Ces informations ne sont pas décrites dans le standard SNMPv3, ce bloc est laissé au soin des modules de sécurité. D'un module de sécurité à un autre, ces informations seront différentes. Le module de sécurité DES a standardisé le contenu de ce bloc.

3.1.4.3 Les identificateurs de contextes

Avec SNMPv1, on ne pouvait avoir une seule base d'informations (MIB) par agent. Ceci n'est pas suffisant pour certains équipements qui peuvent avoir plusieurs fois les mêmes variables. Par exemple, un commutateur ATM contient plusieurs cartes qui peuvent avoir chacune leurs propres bases d'informations. Le contexte permet de distinguer entre plusieurs bases d'informations et même plusieurs agents. On distingue entre des agents, par exemple, quand on

a un moteur qui agit comme passerelle entre SNMPv3 et du SNMPv1. On envoie donc un paquet SNMPv3 en identifiant à quel agent SNMPv1 on désire que le paquet soit retransmis.

3.1.5 Le PDU

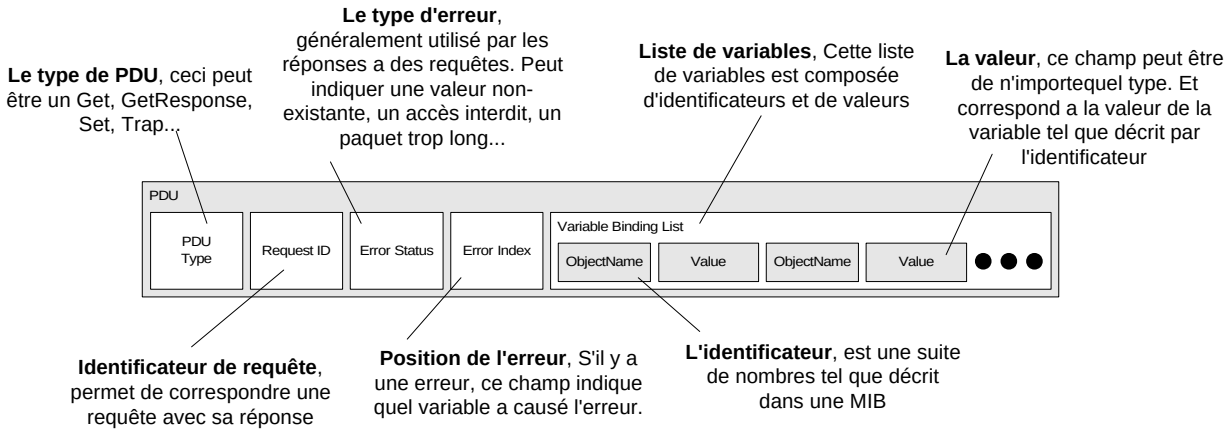


Figure 21 : Description du PDU

Le PDU (Protocol Data Unit) est la charge utile qui contient les variables de la requête ou les valeurs de la réponse. On y précise aussi quelle opération on désire effectuer. Les détails exacts du PDU sont décrits dans le RFC1905.

3.2 Le fonctionnement de la sécurité dans SNMPv3

Cette section introduit les mécanismes de sécurité utilisés dans SNMPv3 par le « User Security Module ». Quatre mécanismes sont utilisés. Chacun de ces mécanismes a pour but d'empêcher un type d'attaque.

- L'authentification

  Empêche quelqu'un de changer le paquet SNMPv3 en cours de route et valider le mot de passe de la personne qui transmet la requête.

- La localisation des mots de passes

Ce mécanisme empêche quelqu'un de compromettre la sécurité d'un domaine d'administration, même si la sécurité d'un des agents du domaine est compromise.

- L'encryption

  Empêche quiconque de lire les informations de gestions contenues dans un paquet SNMPv3.

- L'estampillage du temps

  Empêche la réutilisation d'un paquet SNMPv3 valide que quelqu'un a déjà transmis.

Nous allons explorer le fonctionnement de chacun de ces mécanismes. Pour plus de détails sur le fonctionnement des mécanismes de sécurité, nous référons le lecteur au RFC2274. Les mécanismes de sécurité sont également expliqués dans [BSCH 96].

3.2.1 L'échange de mots de passes

Avant d'aborder les mécanismes de sécurités, on doit aborder l'échange des mots de passes. Les mécanismes qui seront expliqués dans cette section utilisent des mots de passes « Partagé », des mots de passes qui sont connus que par la plate-forme de gestion et l'agent. Il existe aussi un autre système bien connu de mots de passes « Public » et « Privé » qui est utilisé, par exemple, dans les fureteurs web. Les systèmes à clé publique sont expliqués dans [BSCH 96] mais ne font pas partie des mécanismes utilisés dans SNMPv3.

Quand on parle de mots de passes « partagé » (On dit en anglais « shared passwords »), on se demande comment placer les mots de passes dans la plate-forme et l'agent. Idéalement, un administrateur qui a un accès physique à la plate-forme de gestion et à l'agent place le mot de passe sans que ces mots de passes passent par le réseau. Comme les agents ont souvent une interface limité avec l'usagé (pas d'écrans, ni de claviers) la seul façon pratique de configurer un agent est d'utilisé SNMP. Comme SNMPv3 est sécuritaire, il suffit de connaître un mot de passe secret pour configurer tous les autres mots de passes. Dans le cas d'agent SNMPv3, les manufacturiers pourront placer un mot de passe initial qui sera changé par l'administrateur.

3.2.2 L'authentification

L'authentification a pour rôle d'assurer que le paquet reste inchangé pendant la transmission, et que le mot de passe est valide pour l'usager qui fait la requête.

Pour construire ce mécanisme, on doit avoir connaissance des fonctions de hachage à une seule direction. Des exemples de ces fonctions sont : MD5 et SHA-1. Ces fonctions prennent en entrée une chaîne de caractères de longueur indéfinie, et génèrent en sortie une chaîne d'octets de longueur finie (16 octets pour MD5, 20 octets pour SHA-1).

Ces fonctions de hachage à une seule direction ont la propriété suivante:

> *Étant donné une chaîne d'octets qui est le résultat d'une fonction de hachage à une direction. Il doit être très difficile de trouver une quelconque chaîne d'entrée qui, une fois passée dans la fonction, donne cette même chaîne en sortie.*

On peut dire aussi que :

> *Il est très difficile de trouver deux chaînes de caractères qui passées dans la fonction de hachage à une direction, donnent le même résultat. Si on trouve deux chaînes qui ont le même code de hachage, on dit que l'on a trouvé une collision.*

Avant d'utiliser une fonction de hachage à une direction pour faire de la sécurité, il est important de définir ce que veut dire « très difficile ». Dans le cas de SHA-1, les chances qu'une deuxième chaîne de caractères donne un résultat identique à la première est de l'ordre[4] de 1 sur 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000. Il n'est donc pas réaliste de trouver une telle combinaison dans des temps acceptables, même avec des ordinateurs très puissants.

---

[4] Source : SET « Secure Electronic Transaction Specification », Un groupe formé par Visa, MasterCard et d'autres.

Pour authentifier l'information qui va être transmise, on doit aussi avoir un mot de passe qui est « partagé ». Le mot de passe ne doit donc être connu que par les deux entités qui s'envoient les messages, et préférablement par personne d'autre.

La Figure 22 montre le mécanisme d'authentification.

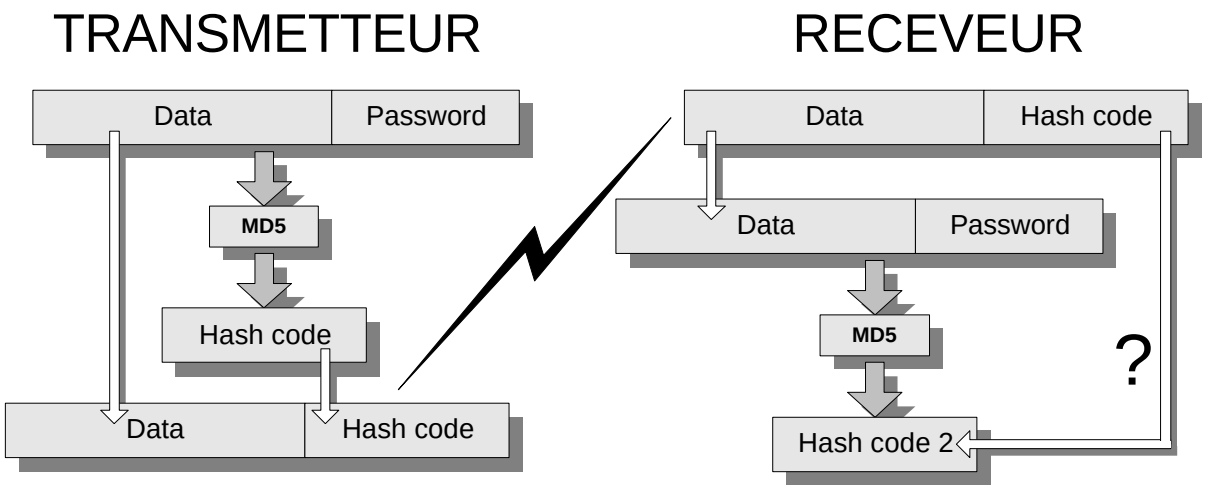


Figure 22 : L'authentification d'un message

Les étapes d'authentification sont les suivantes :

- Le transmetteur groupe des informations à transmettre avec le mot de passe.
- On passe ensuite ce groupe dans la fonction de hachage à une direction.
- Les données et le code de hachage sont ensuite transmis sur le réseau.
- Le receveur prend le bloc des données, et y ajoute le mot de passe.
- On passe ce groupe dans la fonction de hachage à une direction.
- Si le code de hachage est identique à celui transmis, le transmetteur est authentifié.

Avec cette technique, le mot de passe est validé sans qu'il ait été transmis sur le réseau. Quelqu'un qui saisit les paquets SNMPv3 passants sur le réseau ne peut pas facilement trouver le mot de passe.

Pour ce qui est de SNMPv3, l'authentification se fait à l'aide de HMAC-MD5-96 et HMAC-SHA-96, qui est un peu plus compliqué que ce qui a été montré ici. Le résultat de la fonction de hachage est placé dans le bloc des « Paramètres de sécurité » du paquet SNMPv3. L'authentification se fait sur tout le paquet.

Il est important de rappeler que l'étape d'authentification ne vise pas à cacher l'existence du paquet ou à le rendre illisible. Si uniquement l'authentification est appliquée, les personnes qui saisissent les paquets passants sur le réseau peuvent encore voir le contenu du paquet. Toutefois, elles ne peuvent pas en changer le contenu sans connaître le mot de passe.

3.2.3 La localisation

SNMP pose un problème de sécurité particulier. Une plate-forme de gestion a l'habitude de communiquer avec des dizaines ou quelquefois des centaines d'agents. Si le même mot de passe est utilisé par chaque agent, on court le risque qu'un des agents soit volé ou compromis. Le mot de passe serait alors connu et pourrait être utilisé pour administrer tous les autres agents du même domaine d'administration.

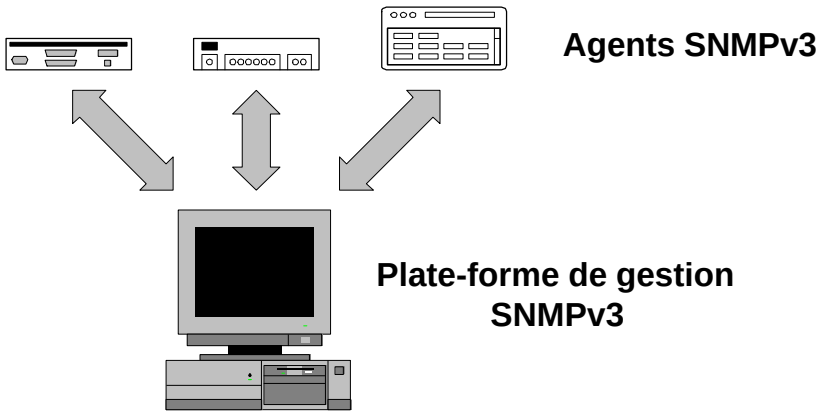


Figure 23 : Une plate-forme de gestion communique avec plusieurs agents SNMPv3

Une solution serait d'utiliser un mot de passe différent pour chaque agent. Ceci est toutefois impraticable car il n'est pas raisonnable pour un administrateur de connaître des dizaines ou des centaines de mots de passes différents.

La solution adoptée par SNMPv3 est d'utiliser un seul mot de passe, mais de passer par une étape de « localisation ». Un mot de passe localisé fonctionne qu'avec un seul agent.

Avant de localiser, il nous faut une chaîne de caractères qui soit unique à chaque agent[5]. Avec SNMPv3, on utilise le « ContextEngineID ». Cette chaîne est générée par un ensemble de données comme : l'adresse MAC de la carte Etherner, l'adresse IP, des nombres aléatoires ou une chaîne spécifiée par l'administrateur.

La figure montre le mécanisme de localisation du mot de passe :

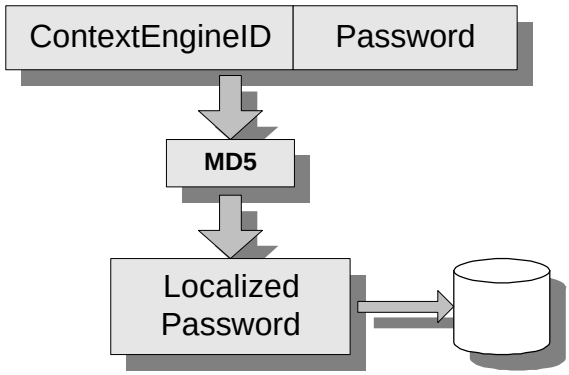


Figure 24 : La localisation d'un mot de passe

On commence par trouver le ContextEngineID de l'agent auquel on veut envoyer une requête. On groupe le ContextEngineID et le mot de passe ensemble et on passe le groupe dans une fonction de hachage à une direction (ces fonctions sont expliquées dans la section sur l'authentification à la page 58).

C'est le mot de passe localisé qui est mémorisé dans l'agent et qui est utilisé par la plate-forme. Le mot de passe localisé est utilisé dans l'authentification et l'encryption des paquets SNMPv3.

Les détails de la localisation sont décrits dans le RFC2274. Cette étape est très coûteuse en temps processeur. En Java, le temps de localisation varie de 3 à 20 secondes selon

[5] Au minimum, cette chaîne doit être unique dans le domaine administratif

l'environnement sur lequel on exécute le code Java. Même en code natif, 1 à 2 secondes sont nécessaires pour localiser un mot de passe. Les plate-formes de gestion sont donc instruites d'utiliser une cache pour éviter de répéter ce calcul plusieurs fois.

3.2.4 L'encryption

L'encryption a pour but d'empêcher que quelqu'un n'obtienne les informations de gestion en écoutant sur le réseau les requêtes et les réponses de quelqu'un d'autre, lequel est autorisé à obtenir ces informations.

Avec SNMPv3, l'encryption de base se fait sur un mot de passe « partagé » entre la plate-forme et l'agent. Ce mot de passe ne doit être connu par personne d'autre. Pour des raisons de sécurité, SNMPv3 utilise deux mots de passe : un pour l'authentification et un pour l'encryption. On recommande à l'usager d'utiliser deux mots de passe distincts. Ceci permet au système d'authentification et au système d'encryption d'être indépendants. Un de ces systèmes ne peut pas compromettre l'autre.

SNMPv3 se base sur DES (Data Encryption Standard) pour effectuer l'encryption. DES ressemble à une fonction XOR compliquée, dans le sens qu'elle préserve presque la même propriété : DES est réversible.

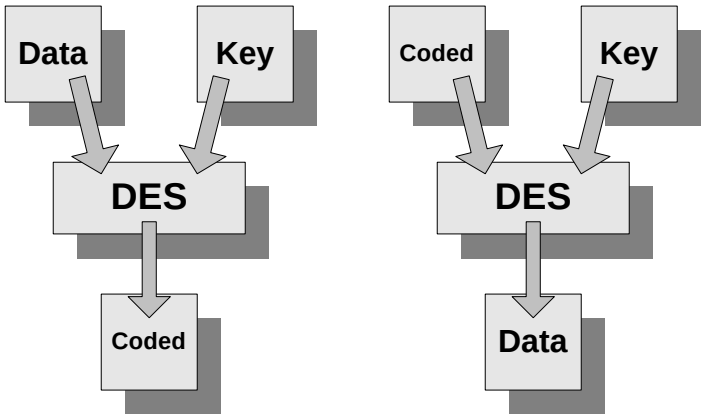


Figure 25 : Illustre le mécanisme d'encryption DES

On utilise une clé de 64 bits[6] et DES encrypte 64 bits à la fois. Comme les informations que l'on doit encrypter sont plus longues que 8 octets, on utilise du chaînage de blocs DES de 64 bits.

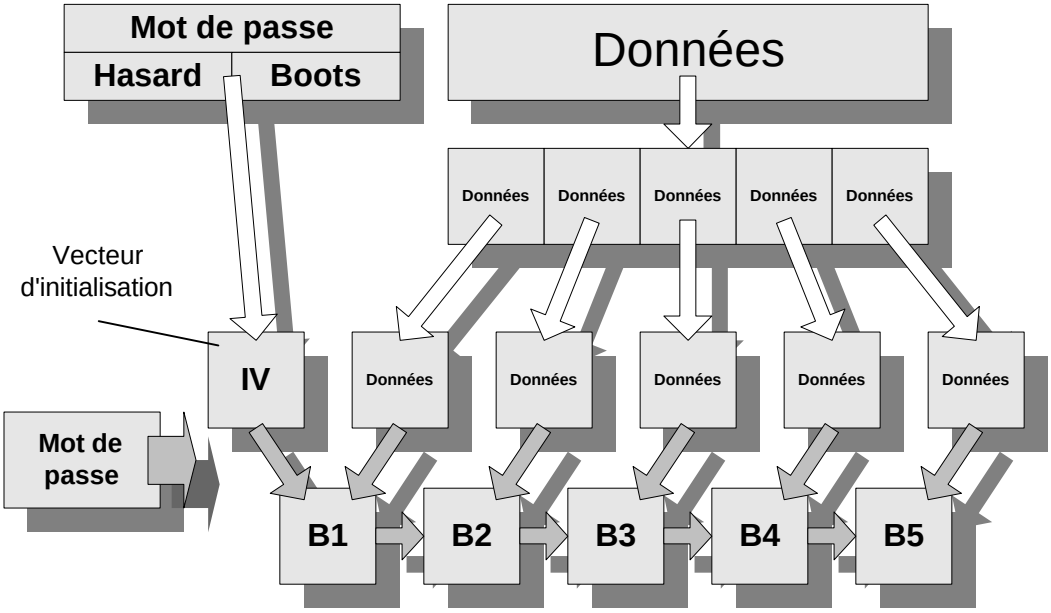


Figure 26 : Montre le fonctionnement de l'encryption pour le chaînage des blocs DES

Une combinaison du mot de passe, d'une chaîne aléatoire et d'autres informations forme le « Vecteur d'initialisation ». Chacun des blocs de 64 bits est passé par DES et est chaîné avec le bloc précédent avec un XOR. Le premier bloc est chaîné par un XOR au vecteur d'initialisation. Le vecteur d'initialisation est transmis avec chaque paquet dans les « Paramètres de sécurité », un champs qui fait partie du paquet SNMPv3.

Contrairement à l'authentification qui est appliquée à tout le paquet, l'encryption est seulement appliquée sur le « Scoped-PDU », voir Figure 19 : Description du paquet SNMPv3.

[6] 8 des 64 bits sont des parités, la clé réelle est donc longue de 56 bits.

3.2.5 L’estampillage du temps

Si une requête est transmise, les mécanismes d’authentification, de localisation et d’encryption n’empêchent pas quelqu’un de saisir un paquet SNMPv3 valide du réseau et de tenter de le réutiliser plus tard, sans modification.

Par exemple, si l’administrateur effectue l’opération de remise à jours d’un équipement; quelqu’un peut saisir ce paquet et tenter de le retransmettre à l’équipement à chaque fois que cette personne désire faire une mise à jours illicite de l’équipement. Même si la personne n’a pas l’autorisation nécessaire, elle envoie un paquet, authentifié et encrypté correctement par l’administration à l’équipement.

On appelle cette attaque le « replay attack ». Pour éviter ceci, le temps est estampillé sur chaque paquet. Quand on reçoit un paquet SNMPv3, on compare le temps actuel avec le temps dans le paquet. Si la différence est plus que supérieur à 150 secondes, le paquet est ignoré.

Toutefois, les agents SNMP sont souvent des petits micro-controlleurs qui n’ont pas à leur disposition une horloge avec l’heure courante et précise. Les horloges doivent être mises à l’heure, et il est pratique de les garder à l’heure même quand l’équipement est éteint. De plus, il est difficile de garder toutes les horloges synchronisées.

SNMPv3 n’utilise pas l’heure normale, on utilise plutôt une horloge différente dans chaque agent qui garde le nombre de secondes depuis que l’agent a été mis en circuit. On garde aussi un compteur qui compte le nombre de fois où l’équipement a été mis en circuit. On appelle ces compteurs BOOTS (Nombre de fois en circuit) et TIME (Nombre de secondes depuis la dernière fois que l’équipement a été mis en circuit).

La combinaison du BOOTS et du TIME donne une valeur qui augmente toujours, et qui peut être utilisée pour l’estampillage. Comme chaque agent a sa propre valeur du BOOTS/TIME, la plate-forme de gestion doit garder une horloge qui doit être synchronisée pour chaque agent qu’elle contacte. Au moment du contact initial, la plate-forme obtient la valeur du BOOTS/TIME de l’agent et synchronise une horloge distincte.

La conclusion du chapitre

Ce chapitre a décrit les détails du protocole SNMPv3 et comment il est découpé. On a aussi vu comment SNMPv3 assure l'authentification et la confidentialité des messages. On constate que le standard SNMPv3 est assez modulaire pour permettre beaucoup de changements, et qu'il est basé sur des principes de sécurité bien éprouvés. On peut maintenant aborder, dans le chapitre suivant, la construction de « SNMPv3 Modulaire ».

Chapitre IV

Ce chapitre décrit en détail la conception et la construction de « SNMPv3 Modulaire ». On décrit les choix de conception, et le fonctionnement du logiciel. Ce chapitre fait appel aux connaissances acquises dans les chapitres 2, 3, 4 pour concevoir un logiciel qui, en plus de fonctionner, offre des solutions innovatrices.

4.1 L'architecture de SNMPv3-Modulaire

4.1.1 Les objectifs de l'architecture SNMPv3-Modulaire

L'architecture que nous avons sélectionné permet :

- Être proche de l'architecture décrite dans le standard de SNMPv3, soit le RFC2271 « An Architecture for Describing SNMP Management Frameworks ».
- Elle permet la jointure (attachement) et le détachement dynamique des différents modules.
- Elle permet de fabriquer un jeu de programmes qui soit facile à tester.
- Elle permet le contrôle, attachement, détachement des modules par un seul point.
- Elle permet, de façon simple, la gestion par SNMP des modules.

4.1.2 Une architecture proche du standard SNMPv3

Le standard SNMPv3 se démarque des autres versions, car il décrit une architecture et un partitionnement du protocole, ainsi que des primitives d'appels entre des modules. Pour faciliter la conception du programme, et la compréhension du produit final, le choix a été fait en faveur d'une architecture logicielle proche de celle décrite dans le standard. SNMPv3-

Modulaire utilisera donc, le plus fidèlement possible, le découpage modulaire et les primitives d'interfaces décrites dans le RFC2271, « La description de l'architecture de SNMPv3 ».

Ceci permet à un utilisateur qui est familier avec les documents du standard SNMPv3, d'être immédiatement à l'aise avec le logiciel SNMPv3-Modulaire. Cet usager pourra développer et utiliser les modules exactement comme décrit dans le standard.

Un usager pourrait fabriquer, par exemple, un nouveau module de sécurité en ne portant attention qu'au document qui traite de ce module (par exemple, RFC2274). Ce document décrit le fonctionnement du module, et utilise les primitives d'interfaces pour raccorder ce module au reste du moteur. La programmation d'un tel module peut se faire sans consulter les autres documents du standard, ou avoir une connaissance du programme SNMPv3-Modulaire dans son ensemble.

Voici, par un exemple, comment l'architecture sélectionnée est proche du standard :

La section 4.2.1 du RFC2271 décrit une interface offerte dans les modules de traitements qui permet au dispatcher de demander qu'un paquet SNMP soit construit. On y passe les paramètres nécessaires et le paquet SNMP est retourné. Ce paquet est prêt à être envoyé sur le réseau.

```
statusInformation =              -- success or errorIndication
     prepareOutgoingMessage(
     IN   transportDomain           -- transport domain to be used
     IN   transportAddress          -- transport address to be used
     IN   messageProcessingModel    -- typically, SNMP version
     IN   securityModel             -- Security Model to use
     IN   securityName              -- on behalf of this principal
     IN   securityLevel             -- Level of Security requested
     IN   contextEngineID           -- data from/at this entity
     IN   contextName               -- data from/in this context
     IN   pduVersion                -- the version of the PDU
     IN   PDU                       -- SNMP Protocol Data Unit
     IN   expectResponse            -- TRUE or FALSE
     IN   sendPduHandle             -- the handle for matching
                                    -- incoming responses
     OUT  destTransportDomain       -- destination transport domain
     OUT  destTransportAddress      -- destination transport address
     OUT  outgoingMessage           -- the message to send
     OUT  outgoingMessageLength     -- its length
          )
```

Dans le code Java de « SNMPv3-Modulaire », on retrouve la même méthode. Dans MPModule.java :

```
public abstract StatusInformation prepareOutgoingMessage(
        int transportDomain,             // IN  as specified by
application
        IpAddress transportAddress,      // IN  as specified by
application
        int messageProcessingModel,      // IN  as specified by
application
        int securityModel,               // IN  as specified by
application
        String securityName,             // IN  as specified by
application
        int securityLevel,               // IN  as specified by
application
        String contextEngineID,          // IN  as specified by
application
        String contextName,              // IN  as specified by
application
        int pduVersion,                  // IN  the version of the PDU
        byte[] PDU,                      // IN  as specified by
application
        boolean expectResponse,          // IN  as specified by
application
        int sendPduHandle,               // IN  as determined in step
3(RFC)
        int destTransportDomain[],       // OUT destination transport
domain
        IpAddress destTransportAddress[],// OUT destination address
        byte[] outgoingMessage[],        // OUT the message to send
        int outgoingMessageLength[]      // OUT the message length
    );
```

Note :

Dans un appel à une méthode en Java, les objets sont passés en référence et les types de base (int, float, byte...) sont passés par copie. Quand un objet est passé, le pointeur à l'objet est copié, et donc ne peut pas être changé pour pointer sur un autre objet. Pour s'assurer qu'on reçoit correctement le type en sortie (OUT), on passe pour tous les types en sortie un tableau de longueur un. Cette technique est utilisée dans toutes les interfaces de « SNMPv3-Modulaire ».

Les interfaces sont identiques. Pour permettre que les interfaces Java soient exactement identiques avec celles du standard, certains nouveaux types ont été créés. C'est le cas de IpAddress[7] et StatusInformation.

4.1.3 Une jointure (attachement) dynamique des modules

Comme un moteur SNMP peut jouer plusieurs rôles (voir **Error! Reference source not found.** en page **Error! Bookmark not defined.**), il est important pour l'usager du SNMP d'être capable d'assembler de toutes pièces un moteur SNMP pour jouer un rôle particulier.

Le moteur, tel que décrit dans les pages suivantes, est « tout assemblé » pour jouer le rôle d'agent et de plate-forme de gestion, et ceci simultanément pour les version SNMPv1 et SNMPv3.

L'assemblage d'un nombre minimum de modules est quelquefois préférable pour éviter le gaspillage de ressources mémoire dans les ordinateurs. Un agent SNMPv1 peut être assemblé avec aussi peu que trois modules. Ceci réduit considérablement la taille de la mémoire requise.

Le standard permet une jointure (attachement) et un détachement dynamique des modules. Ceci veut dire qu'un module peut être joint ou détaché au cours de l'exécution du moteur SNMP. On peut profiter des avantages du Java pour effectuer le branchement de classes supplémentaires pendant l'exécution, puisque le Java offre un service de chargement dynamique de fichiers « .class ». La classe « SnmpEngine.java » offre une méthode pour ajouter des modules. Cette méthode est utilisée pendant l'initialisation du moteur pour charger tous les modules. Pour ajouter un nouveau module, il s'agit de créer une instance d'une classe qui dérive de « SnmpModule.java » et de passer cette objet dans le paramètre de la méthode « AddModule() ».

---

[7] Le type InetAddress qui existe en Java contient seulement l'adresse IP mais pas le port. Le nouveau type IpAddress est construit avec l'adresse IP et le port.

4.1.4 La simplicité des tests sur le produit final

L'architecture doit prendre en considération que le produit final devra passer une batterie de tests pour assurer la qualité du logiciel et la conformité aux standards. Pour ce faire, il est préférable de choisir une architecture qui simplifie les procédures de tests. Dans notre cas, l'objectif de modularité et celui de testabilité sont tous les deux très proches.

Comme tous les modules sont détachables et que leurs interfaces sont clairement identifiées et comprises dans les documents de standard [RFC2271-RFC2275], on peut appliquer facilement des séquences de tests.

4.1.5 Une gestion des modules par SNMP

La gestion d'un moteur SNMP est généralement effectuée par un interface usager, ou par le protocole SNMP lui-même (à l'aide d'une MIB). Tous les documents du standard SNMP, même des versions antérieures, décrivent des MIB standard pour la gestion des différents modules SNMP. L'architecture de SNMPv3-Modulaire doit donc être choisie pour faciliter la gestion par SNMP de tous les modules, qu'ils soient joints (attachés) dynamiquement ou pas.

4.2 L'architecture proposée

L'architecture proposé pour la mise en œuvre de SNMPv3-Modulaire:

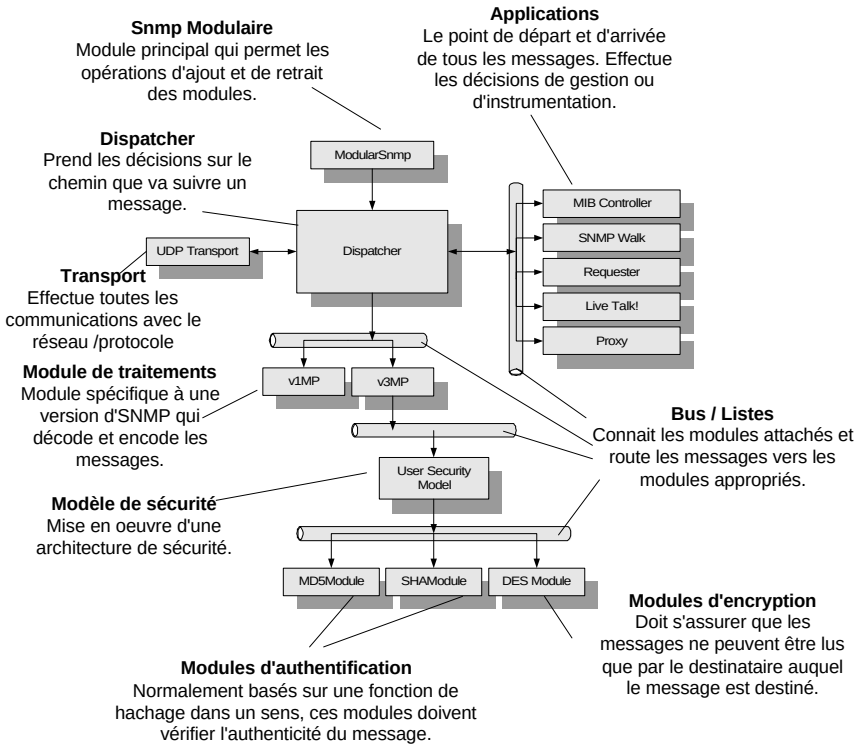


Figure 27, Architecture de SNMPv3-Modulaire

Cette architecture est très proche de celle décrite dans le standard. On y retrouve le corps du programme, le “dispatcher” et les modules: modules de traitements, modèles de sécurités, modules de sécurités et les applications.

Les librairies Java offrent un accès aux services UDP et TCP (qui sont sur IP). Toutefois, Java n’offre pas d’autres services de transport (IPX, Appletalk...). Même si SNMP peut être utilisé sur d’autres protocoles que UDP, Snmp-Modulaire est construit pour n’offrir qu’un seul service de transport.

Le diagramme d’architecture montre les modules qui contiennent les fonctionnalités liées au protocole SNMP, et les « bus » (Appelés « ListHandlers » dans le code Java) qui font le routage des messages vers les modules.

4.2.1 Le parcourt d’un paquet SNMP

Pour faciliter la compréhension de l’architecture, et du rôle de chaque module, on peut observer le traitement effectué pendant le parcours d’un message SNMP dans le logiciel.

Le diagramme suivant montre le chemin effectué par un paquet SNMPv3 qui est reçu du réseau. La destination d’un paquet validé est toujours une des applications.

On doit noter que ce parcours est effectué en sens inverse par un message qui provient d’une application qui envoie une requête, ou qui répond à une requête de gestion. À tout moment, le parcours d’un message SNMP peut être interrompu pour les raisons suivantes:

- Une erreur s’est produite pendant le décodage du message
- Un module nécessaire au décodage d’un message n’est pas disponible (pas attaché)
- L’étape de décryption ou l’authentification du message n’a pas été exécutée avec succès (Mauvais mot de passe)
- Une erreur inattendue s’est produite (Voir « La tolérance aux fautes par les fils d’exécutions », page 82)

Selon le paquet SNMP qui est traité, le parcours peut être raccourci. Par exemple, si un paquet SNMPv1 est reçu, il ne passe pas par la sécurité. Ce paquet est donc décodé et envoyé directement à l’application concernée.

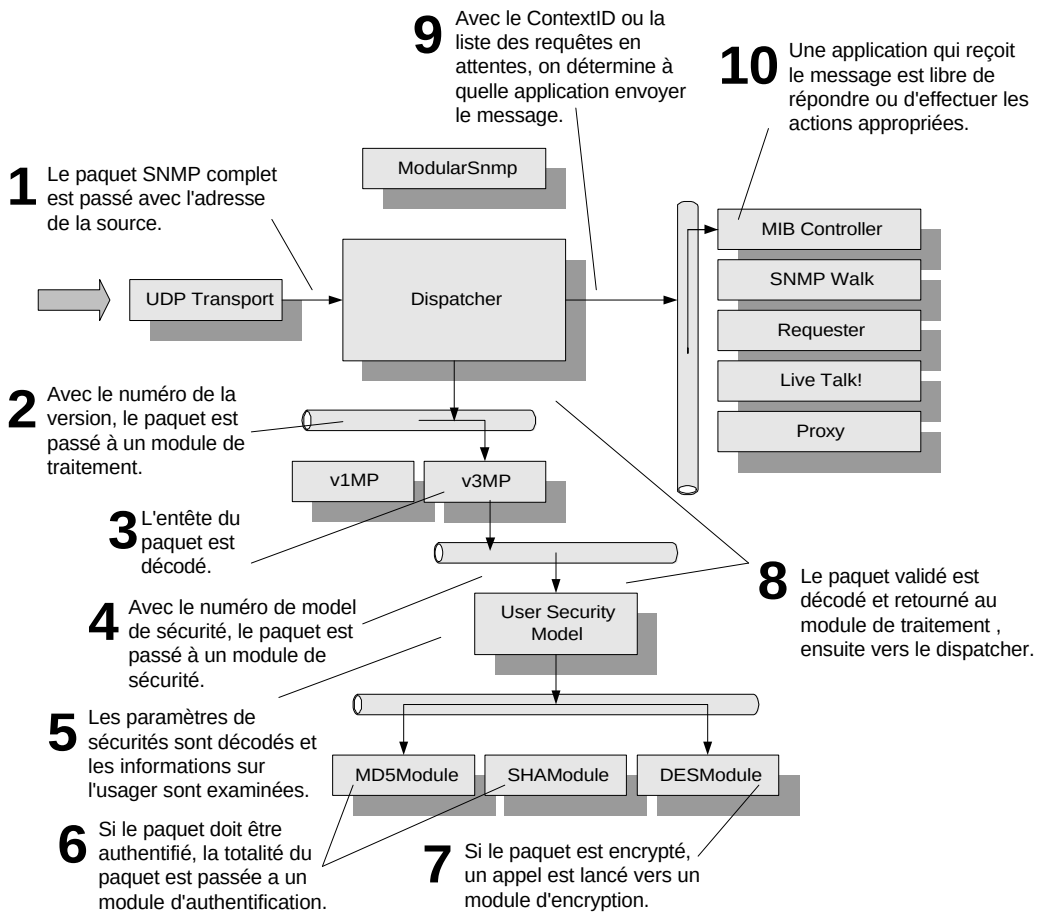


Figure 28, Trajet d'un message SNMP

4.2.2 Le fonctionnement des bus

L'architecture de « SNMP Modulaire » est composée de modules et de bus entre les modules. Ces buses servent à garder une liste des modules présents et assez d'informations sur chaque module pour pouvoir acheminer les messages vers le bon module.

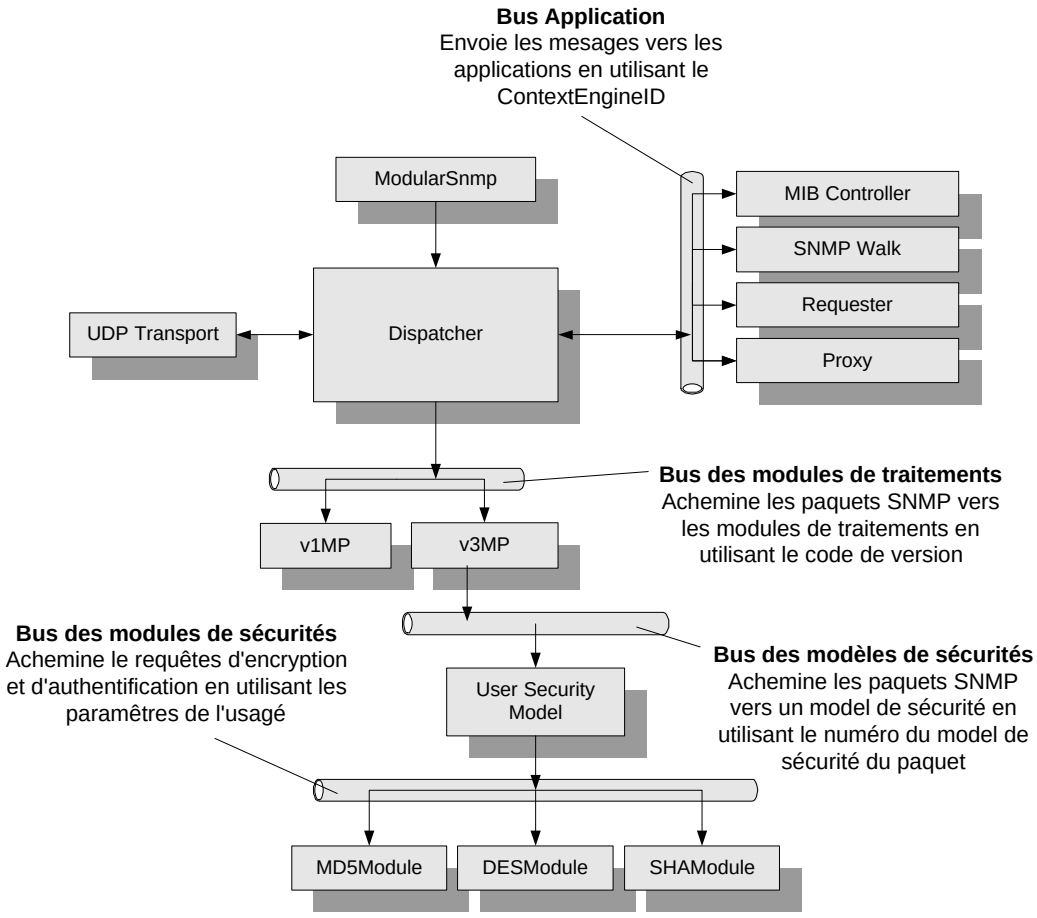


Figure 29 : Fonctionnement des bus

Note :

On retrouve dans le code Java, une classe « Bus » responsable de garder les informations sur le module attaché pour chaque bus dans le diagramme. Le nom de ces classes termine toujours par « Handler ». Par exemple : « AppListHandler.java » ou « SecurityModuleHandler.java ».

4.2.3 Le stockage des données

Une des façons de comprendre la mise en œuvre de SNMPv3-Modulaire est par l'observation des différents endroits où des données sont stockées. Les modules ne gardent l'état ou l'information entre chaque traitement de message SNMP que dans des entrepôts de données

spécialement conçus pour faire ce travail. On distingue deux types d'entrepôts, ceux qui sont temporaires et ceux qui sont permanents.

Entrepôt temporaire

- Situé uniquement en mémoire
- S'efface à chaque redémarrage du moteur SNMP
- Subit des changements fréquents (À chaque message SNMP)
- Certain de ces entrepôts ont des algorithmes de nettoyage automatique

Entrepôt permanent

- Situé en mémoire avec une copie sur disque
- Un changement est effectué au démarrage du moteur, une sauvegarde est faite à chaque changement des données.
- Peu de changements, uniquement l'administration de l'usager

Chaque « bus », ou point de branchements 1 à n dans l'architecture comporte un module de stockage temporaire. Ces informations servent à faire le lien entre les modules attachés au « bus » et la fonctionnalité offerte par chaque module. Par exemple, le « bus » situé entre le dispatcher et les modules de traitements, contient une table (temporaire) qui fait la correspondance entre la version SNMP et le module (pointeur au module) qui traite ces paquets.

Version SNMP	Pointeur au module de traitement
0	Instance de « MPv1.class »
3	Instance de « MPv3.class »
n	Version future de SNMP

Tableau 4, Exemple de table à stockage temporaire

(Noter que SNMPv1 porte le numéro de version zéro)

La figure suivante montre la distribution des entrepôts de données dans l'architecture de SNMPv3-Modulaire :

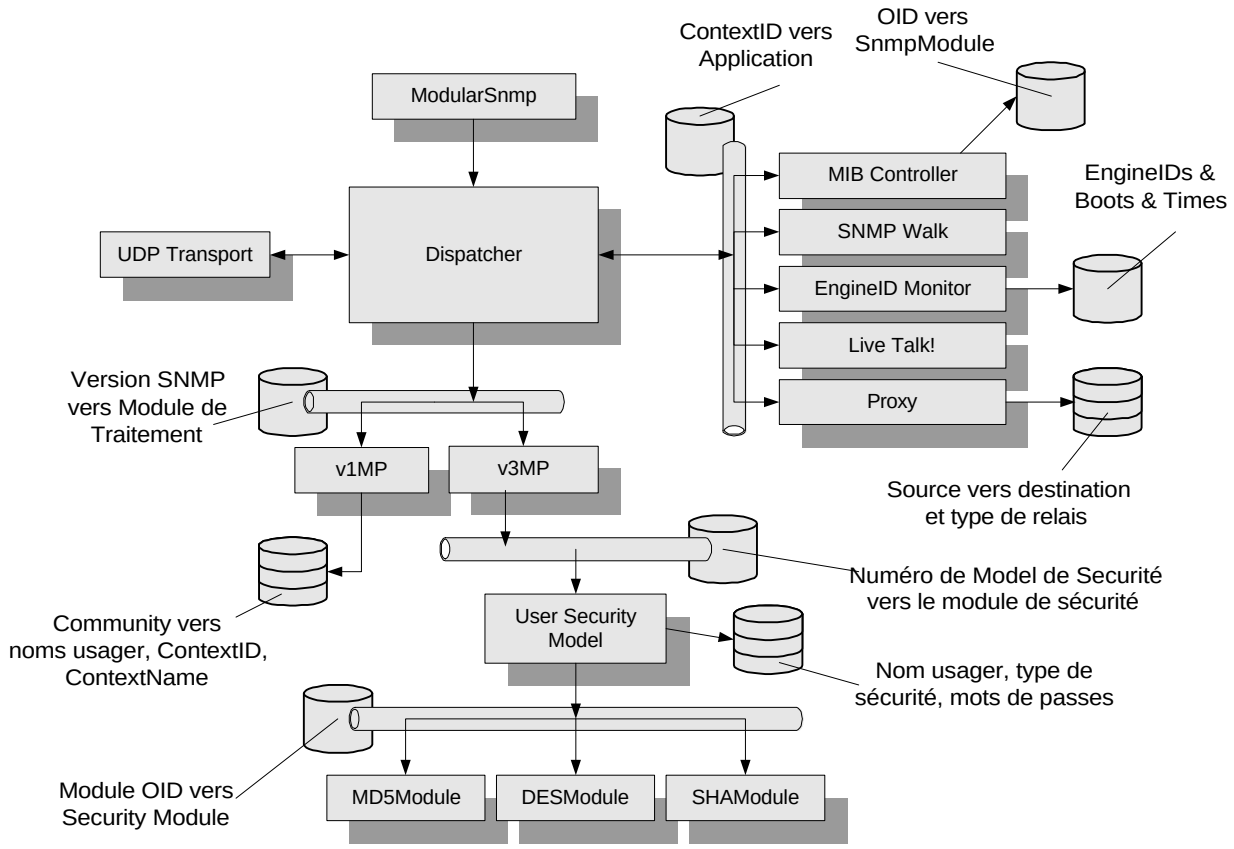


Figure 30, Modules de stockage

Dans cette figure, les banques de données avec deux lignes horizontales sont des entrepôts permanents, les autres sont des entrepôts temporaires.

Cette figure montre la position actuelle des entrepôts. Toutefois, de nouveaux modules peuvent être programmés et attachés au moteur. Ces nouveaux modules sont libres d'avoir d'autres entrepôts de données.

4.2.4 La gestion d'un moteur SNMP

SNMP est un protocole de gestion qui est souvent utilisé pour configurer et obtenir des informations de gestion d'un équipement de réseau. Les mêmes fonctionnalités qui sont utilisées pour gérer une instrumentation quelconque sont aussi utilisées pour faire la gestion du moteur SNMP. Le moteur SNMP est donc lui aussi une entité gérable.

Dans la version 3 de SNMP, on peut effectuer des changements de mots de passe, ou la création de nouveaux usagers sur l'agent à travers SNMP. Pour ce faire, chaque module, en plus d'accomplir sa tâche initiale (application, traitement, sécurité...), doit souvent faire la gestion d'une MIB, répondre aux requêtes et effectuer des traitements suite à des requêtes de gestion.

Voici un diagramme qui montre l'architecture de SNMPv3-Modulaire et pour certains modules, la MIB à laquelle ils sont associés dans les standards.

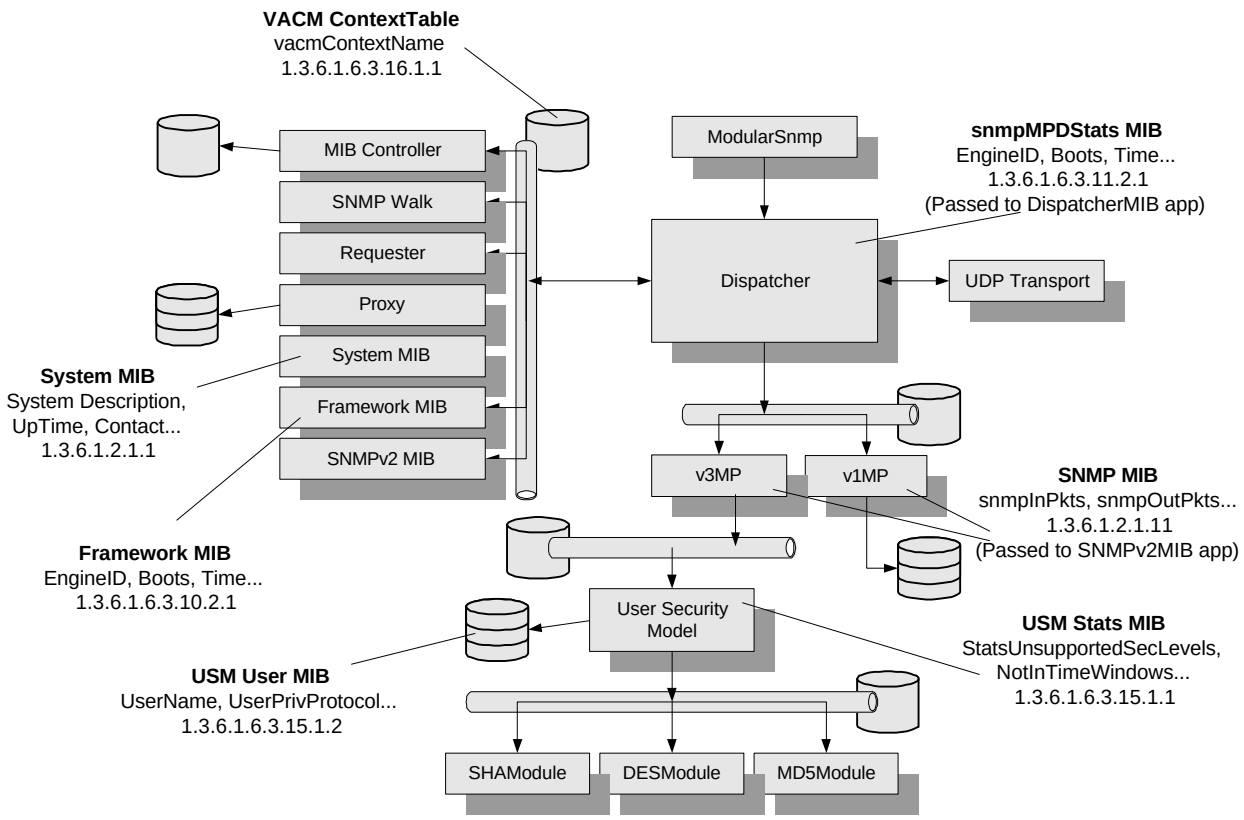


Figure 31, Entreposage des données

Une application spéciale « MIB Controller » prend les requêtes de gestion destinées au moteur SNMP et les envoie vers les modules concernés par la requête. Pour effectuer cette tâche, il faut que le contrôleur de MIB garde des listes complètes triées de toutes les variables (OID) actives. Quand une requête SNMP arrive du moteur, elle est décomposée par cette application en une série de GET et SET qui sont envoyés aux modules qui ont demandé d'être gérés. Ceci est nécessaire car une seule commande de gestion SNMP comme GET ou SET peut faire référence à plusieurs variables situées dans plusieurs modules.

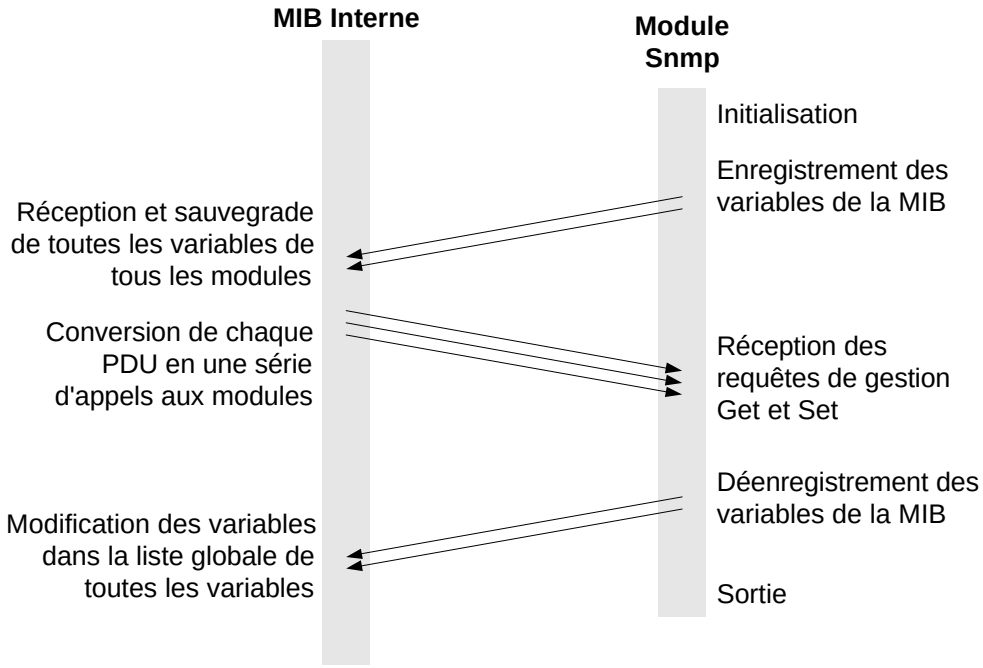


Figure 32, Fonctionnement de la MIB interne

Pour fonctionner, chaque module doit enregistrer toutes ces variables (OID) avec le contrôleur de MIB interne. Cet enregistrement se fait avec les méthodes héritées de « SnmpModule » (Voir « L'héritage des classes », page 81).

À chaque fois qu'une application enregistre un OID, SnmpModulaire stocke le OID dans une liste. Cette liste contient tous les OID enregistrés et un pointeur à l'application qui s'occupe de répondre aux requêtes à cette OID. Pour exécuter la commande SNMP GETNEXT on doit garder la liste triée. On peut donc facilement trouver le prochain objet dans l'ordre lexicographique.

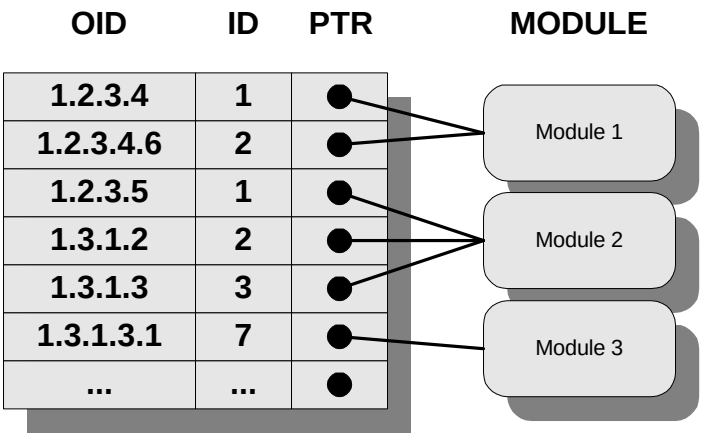


Figure 33, Table des OIDs telle que construite dans SnmpModulaire

À chaque fois qu'une requête pour un OID arrive, on appelle le module associé à l'OID. Quand cet appel est fait, on pourrait passer le OID demandé à l'application, toutefois pour des raisons de performance, on passe un identificateur numérique. Cette technique est aussi utilisée par la mise en œuvre de SNMPv1 du Carnegy Mellon University (CMU). Au moment de l'enregistrement d'un OID, l'application donne un identificateur numérique. Cet identificateur sera utilisé pour demander cette information de gestion. Le gain de performance est largement du à la rapidité de la commande « Switch » en C++ et en Java et à la possibilité d'utiliser l'identificateur comme index dans une table. À la réception d'un identificateur entier, l'application peut :

- Effectuer un « Switch » et brancher rapidement sur le code qui est lié à cet OID. Ce code obtiendra/calculera la valeur demandée.
- L'identificateur est utilisé comme index dans une table. On lira le $n^{ième}$ élément de la table pour obtenir la réponse.

Souvent, on utilise une combinaison de ces deux techniques pour obtenir rapidement des valeurs (isolées ou dans des tables).

4.2.5 L'héritage des classes

La méthodologie objet permet l'héritage des classes. Deux avantages dont notre architecture profite beaucoup sont:

- Les classes peuvent hériter de la fonctionnalité d'autres classes
- Permettre le polymorphisme, c'est-à-dire cacher la fonctionnalité de différentes classes en arrière d'une interface commune.

Voici la structure d'héritage pour SNMPModulaire:

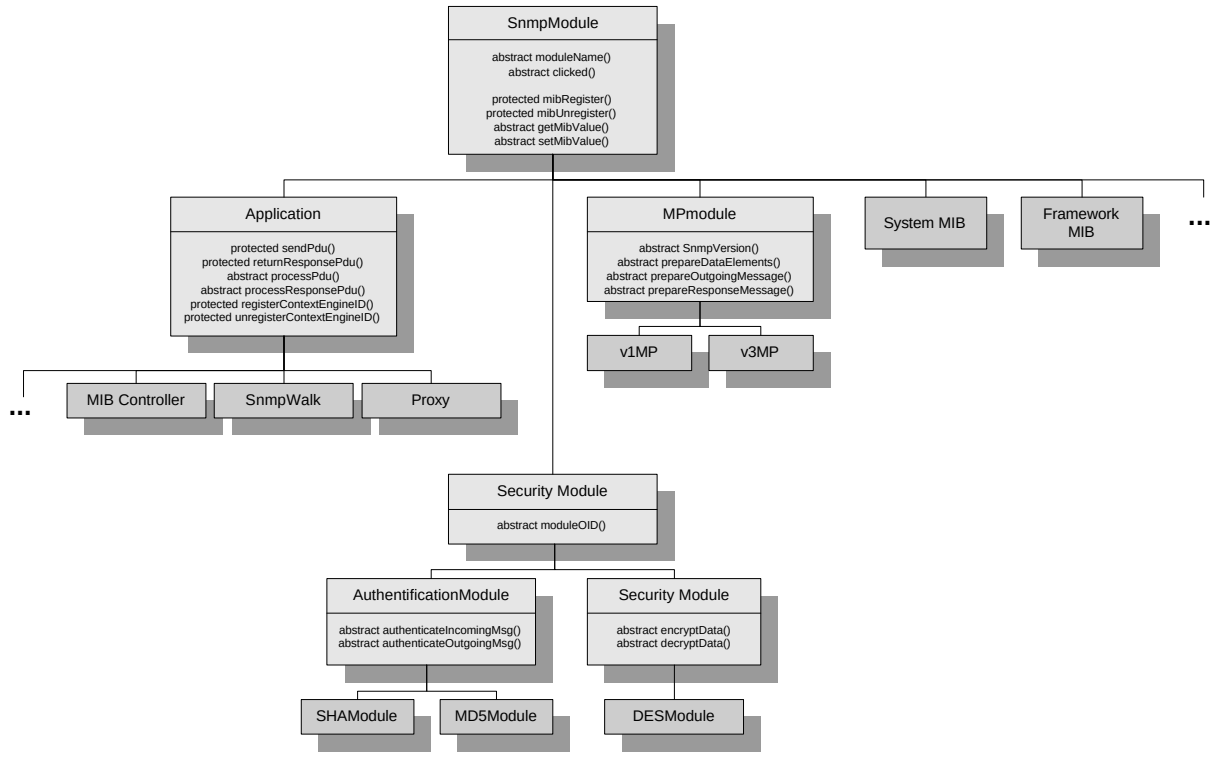


Figure 34, Structure d'héritage des classes

En gris foncé sont les classes qui contiennent la mise en œuvre d'algorithmes, d'applications ou toute autre portion du programme. Chaque module de l'architecture (voir

Figure 27) est aussi placé dans la structure d'héritage comme un module gris foncé. Les classes en gris pâle sont les classes abstraites, c'est-à-dire qu'on ne peut pas les instancier.

Ces classes abstraites offrent aux classes qui les héritent des fonctionnalités. Si une classe hérite de la classe application, elle hérite des méthodes pour la transmission de messages, ou l'enregistrement de contextes. Comme toutes les classes héritent de la classe « SnmpModule » toutes les classes dans la hiérarchie peuvent profiter des services de la gestion SNMP (voir section 4.2.4).

Ces classes abstraites servent aussi à imposer la mise en œuvre de certaines méthodes aux classes qui l'héritent. Par exemple, une classe qui veut jouer le rôle d'une application doit avoir des méthodes pour recevoir les messages SNMP, les réponses aux messages, etc... Pour le programme qui doit faire la gestion des applications, ce système de classe abstraite permet une interface unique et identique pour toutes les applications.

Les méthodes notées « Abstract » doivent être codées dans les classes qui héritent de la classe parent. La méthode notée « Protected » est une des méthodes qui offrent des services aux classes qui héritent de la classe parent.

4.2.6 La tolérance aux fautes par les fils d'exécutions

La tolérance aux fautes est le comportement qu'a un programme quand il est placé dans des situations qui ne sont pas prévues pas les spécifications. Dans le cas de SNMPv3-Modulaire, on peut s'assurer qu'une faute dans l'exécution d'un traitement n'empêche pas le traitement d'autres messages par la suite. Contrairement à un programme conventionnel qui arrête complètement lors d'une faute, les programmes Java génèrent une exception et terminent le fil d'exécution.

Ainsi, on peut profiter des avantages du Java pour développer une stratégie de gestion des fautes.

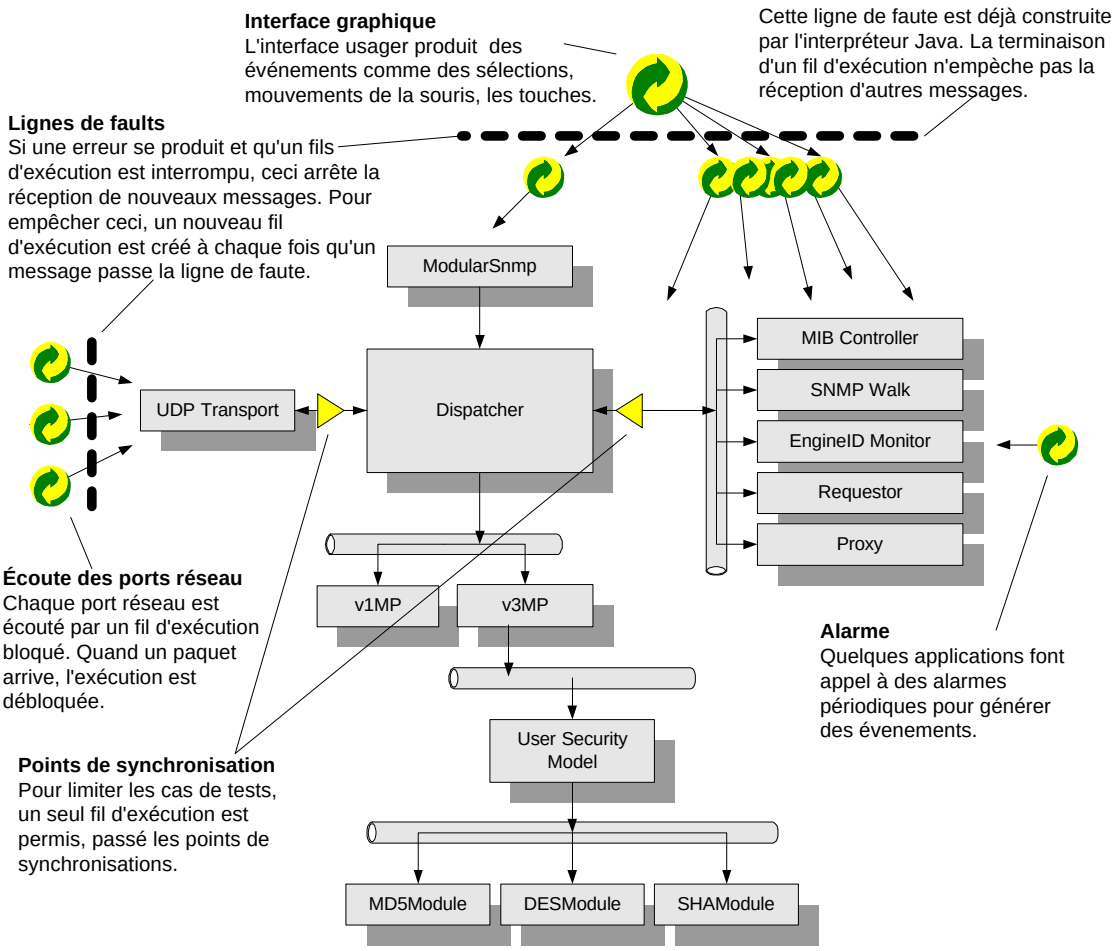


Figure 35, Architecture des fils d'exécutions

Le Java utilise déjà dans la librairie AWT (Abstract Windowing Toolkit) ces mêmes techniques, il s'agit de les adapter à notre architecture logicielle. Si un fil d'exécution bloque en attente de messages sur le réseau, il est impératif que ce fil d'exécution continue son travail après une faute dans le logiciel. Pour ce faire, à la réception d'un paquet du réseau, on créera un nouveau fil d'exécution qui fera le traitement du paquet. On libère donc le fil d'exécution original, qui retourne immédiatement écouter pour d'autres paquets sur le réseau. Si une faute se produit pendant le traitement du paquet, cela ne devrait pas avoir d'incidence sur les nouveaux paquets.

Toutefois, la création de nouveaux fils d'exécutions pour chaque paquet, pour chaque événement de l'interface usager et pour les alarmes, peut produire une situation où plusieurs fils d'exécutions sont en même temps dans la même portion du programme. Cette situation peut mener à des problèmes d'inconsistance des variables.

On dit d'un module qui est conçu pour avoir plusieurs fils d'exécution actifs en même temps qu'il est « réentrant ». Ces modules doivent subir des tests additionnels pour assurer qu'un fil d'exécution aura toujours des variables cohérentes et qu'aucun des fils d'exécution ne sera jamais en attente perpétuelle de l'autre fil d'exécution. Comme le désir de SNMPv3-Modulaire est d'avoir des modules faciles à programmer et facilement attachables au moteur, la meilleure avenue est de ne pas permettre à deux fils d'exécution de traverser le corps du programme en même temps.

Pour ce faire, Java utilise des primitives simples pour synchroniser l'entrée des fils d'exécutions dans le programme. Deux points stratégiques ont été choisis pour placer ces points de synchronisation, soit de chaque côté du « dispatcher ».

4.2.7 Le mécanisme de traces

Pour simplifier la mise en œuvre du SNMPv3-Modulaire et pour permettre de trouver rapidement les erreurs, un mécanisme de traces accompagne l'architecture. Normalement, tous les modules affichent les messages de traces directement à la console. Comme chaque module affiche ces messages, il est possible de ne plus savoir d'où vient le message. Aussi, si l'usager veut diriger les messages de traces vers une autre destination, l'interprétation sera très compliquée.

Le mécanisme de traces proposé envoie tous les messages de traces à un seul point dans le programme.

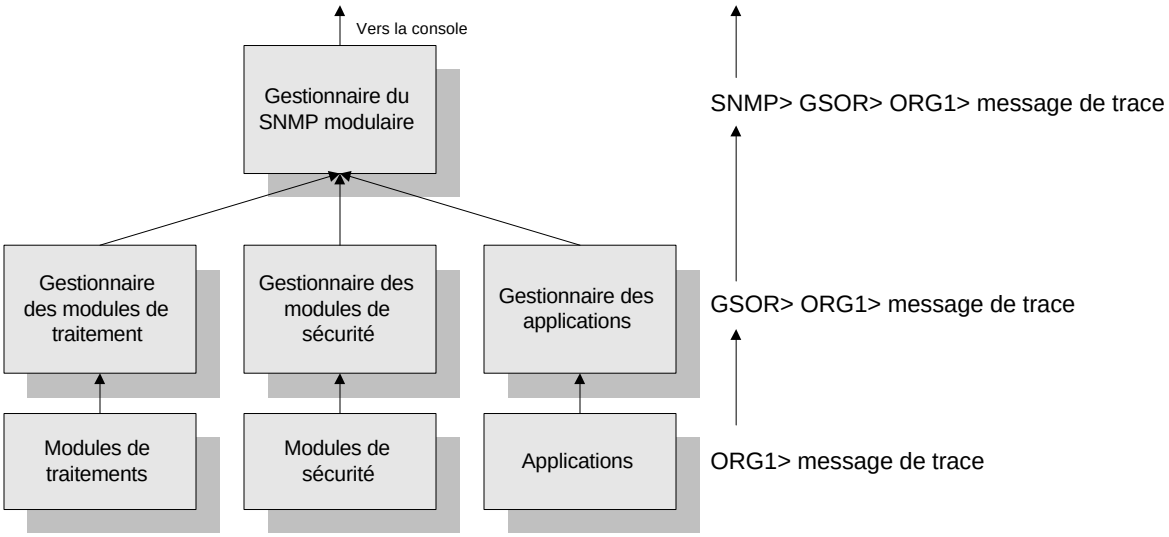


Figure 36, Mécanisme de trace

Chaque module envoie tous ses messages de traces au module supérieur. Pour s'assurer que l'on connaîtra la provenance du message, chaque module ajoute une signature (identificateur). On peut donc, avec la lecture des signatures, retracer la provenance du message.

Chaque classe qui désire émettre des messages de traces doit activer une interface de trace. Cette interface permet d'activer ou d'interrompre le traçage de ce module. On peut ainsi choisir chaque module dont on veut un rapport de traces.

Dans chaque module on a donc un lot d'instructions similaires à celui-ci:

```
private boolean trace = false;
public  void printTrace(String s) {Superior.printTrace("CODE> " + s);}
private void trace(String s) {if (trace) printTrace(s);}
public  void trace(boolean t) {trace = t;}
```

Ce code se retrouve dans tous les modules sauf le module principal, qui doit décider quoi faire avec les messages. On va généralement les imprimer à la console, mais on peut aussi les envoyer dans un fichier ou sur le réseau.

4.2.8 La justification de l'architecture

Cette section présente l'architecture proposée pour « SNMPv3 Modulaire ». Toutefois, il faut justifier cette proposition. D'autres architectures ont été explorées (Architecture à bus unique, architecture basée sur CORBA ou RMI...). Le choix de l'architecture a été influencé par certains facteurs :

- « SNMPv3 Modulaire » se veut un outil pédagogique où d'autres étudiants universitaires pourraient greffer facilement d'autres applications et outils. Pour ce faire, un étudiant qui écrit une application SNMPv3 doit avoir une interface logiciel simple. Il doit pouvoir attacher son nouveau module rapidement. L'architecture proposée accomplit ceci en offrant une seule méthode « AddModule() » et une seule interface « Application.java ». Plusieurs exemples (SnmpWalk, Requestor) d'applications sont fournis pour aider des étudiants à fabriquer leurs variantes.
- « SNMPv3 Modulaire » doit être facile à comprendre et à modifier. Même si en regardant l'ensemble du programme, on le trouve compliqué, la division du programme en modules facilite la compréhension. Aussi, comme le découpage en modules est fait exactement comme le découpage dans le standard SNMPv3 et que les interfaces sont identiques, les personnes qui désirent modifier le noyau de « SNMPv3 Modulaire » peuvent utiliser les documents du standard SNMPv3 [RFC2271-RFC2275] comme documents de références.
- « SNMPv3 Modulaire » offre un service de MIB interne. C'est-à-dire que n'importe quel module peut ajouter des variables à la MIB interne du moteur. Ceci permet à quelqu'un de fabriquer facilement et rapidement une MIB d'instrumentation quelconque. Un exemple (SystsemMIB.java) permet à quelqu'un de faire la mise en œuvre d'une MIB sans problème. Une seule interface (SnmpModule.java) est utilisée pour la construction d'une MIB.
- En construisant les modules SNMP autour des classes Java (Un module = une classe Java) et en utilisant les interfaces du standard SNMPv3 [RFC2271] comme interfaces entre les classes, on peut utiliser le chargement dynamique des classes Java pour charger des modules.

La justification majeure de cette architecture est qu'elle est proche du standard et donc plus facile à comprendre. Plusieurs étudiants universitaires ont déjà développé des applications qui fonctionnent sur « SNMPv3 Modulaire », c'est le cas de [AYCH 98] avec une application de notification et un contrôle d'accès.

4.3 La compatibilité avec SNMPv1

Contrairement à ce que l'on pourrait penser, SNMPv3 n'est pas compatible avec SNMPv1. Ceci n'empêche pas SNMPv1 de fonctionner à coté de SNMPv3 (« Dual stack »), il suffit de placer deux agents ou deux plate-formes de gestion côte à côte.

Les documents du standard SNMPv3 (RFC2271) sont très clairs à l'effet qu'aucune mesure n'a été prise dans le développement de SNMPv3 pour être compatible avec SNMPv1. Toutefois, comme l'architecture de SNMPv3 a été décrite de façon modulaire, on peut espérer construire un module de traitement des paquets SNMPv1 qui fonctionne dans le cadre de SNMPv3. La Figure 18 « Architecture SNMPv3 d'un agent » montre ceci, même si aucune explication n'est fournie.

Un module de traitement des paquets SNMPv1 doit être construit pour qu'il fonctionne parfaitement dans l'environnement SNMPv3. Ce module devra utiliser le dispatcher, « l'access control », les applications et surtout les primitives d'interfaces de SNMPv3, tout en communiquant avec à des agents SNMPv1. Tous les modules construits pour acheminer les paquets SNMPv3 doivent fonctionner sans modification.

Pour ce faire, le module de traitement SNMPv1 doit se comporter comme un module de décodage SNMPv3. Voici les principaux obstacles :

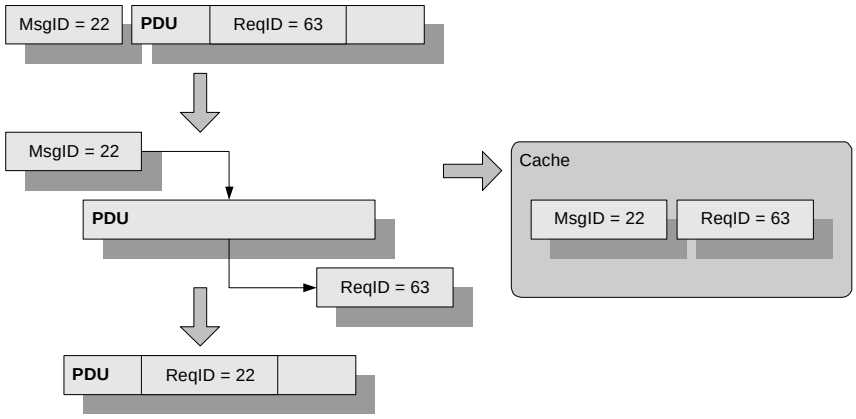
- Les paquets SNMPv1 n'ont pas de « MessageID »
- Les paquets SNMPv1 ne contiennent pas de « ContextEngineID » ni de « ContextName » ni de « Username ».

Toutefois, les PDU SNMPv1 sont identiques aux PDU de SNMPv3. Ceci implique que si le PDU d'un paquet SNMPv1 se rend à une application SNMPv3, l'application pourra le décoder.

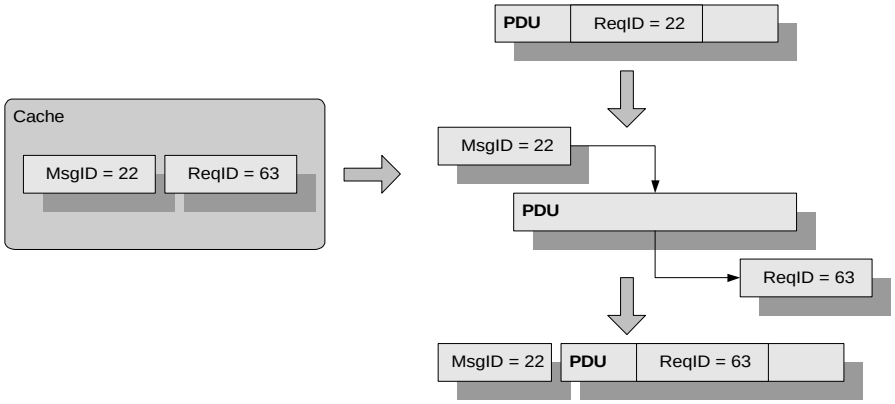
La responsabilité d'être compatible avec SNMPv1 est donc complètement laissée au constructeur du moteur SNMP. Plusieurs ont choisi d'utiliser leur propre cadre de programmation pour résoudre le problème, d'autres ont opté pour deux moteurs SNMP distincts. SnmpModulaire utilise un module de traitement des paquets SNMPv1 dans le cadre et l'architecture SNMPv3. Les problèmes énoncés plus tôt y ont été résolus.

4.3.1.1 Le « MessageID » dans SNMPv1

Le « MessageID » a la fonction d'identifier de façon unique chaque paquet envoyé sur le réseau. Quand une réponse est reçue, le dispatcher utilise ce champ pour identifier l'application qui a envoyé la requête. Le « MessageID » est un entier qui est généré par le dispatcher et donné au module de traitement avec le PDU. Quand un paquet SNMP réponse est décodé, le dispatcher s'attend à recevoir le même « MessageID » que le paquet qui a généré la requête. Pour offrir ce service, SnmpModulaire substitue le « RequestID » du paquet SNMPv1 par le « MessageID ».



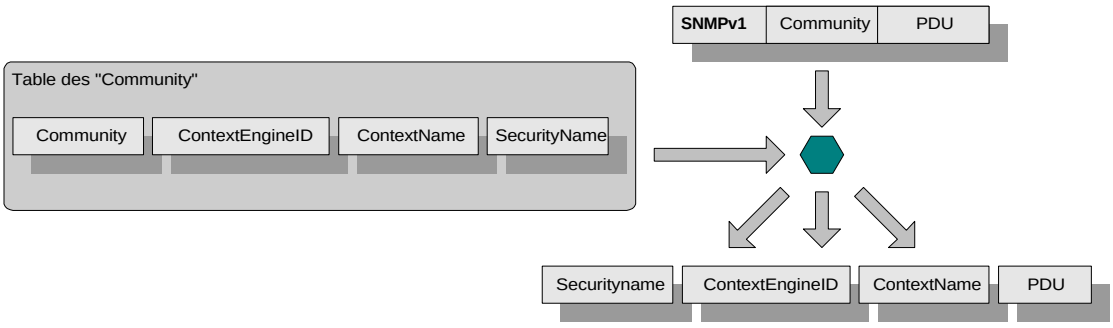
Une cache de toutes les substitutions est gardée en mémoire. Quand le paquet SNMP arrive avec la réponse à la requête, la substitution inverse a lieu.



Il est important de replacer le « RequestID » de la requête car les applications peuvent utiliser le « RequestID » pour faire certaines associations. Les applications s'attendent donc à avoir le même « RequestID » pour la requête et la réponse.

4.3.1.2 Les « Contextes » dans SNMPv1

Quand une requête SNMPv1 arrive, certaines informations sont requises par le dispatcher pour acheminer le PDU vers sa destination. Comme le paquet SNMPv1 n'a pas de « ContextEngineID », de « Contextname » et de « Securityname », SnmpModulaire les obtient d'une table fournie par l'administrateur.



Cette table convertit le « Community String » du paquet SNMPv1 vers un « ContextEngineID, ContextName, SecurityName ».

4.4 Le Logiciel et ses modules

Cette section se veut un guide pour la lecture du code Java « SNMPv3 Modulaire ». Chaque fichier Java y est décrit. Pour faciliter la lecture de cette section, les fichiers Java sont classés en 8 catégories :

- **Les interfaces et services**

  Les classes abstraites, celles qui offrent des services génériques.

- **Les modules centraux**

  Le dispatcher et les classes qui l'entourent.

- **Les modules de traitements**

  SNMPv1 et SNMPv3

- **Les applications**

  Requestor, SNMP Walk,

- **Le transporteur UDP**

  Les classes reliées à la communication avec le réseau

- **« User Security Model »**

  Les classes reliées aux RFC2274

- **Les modules de sécurité**

  Algorithmes DES, MD5, SHA-1, CBC, HMAC.

- **Les types et autres modules**

  Des types définis dans les primitives d'interface du standard et autres classes.

4.4.1 Les interfaces et Services

SnmpModule.java (197 octets, Package : *ModularSnmp*)

Cette classe est le parent de toutes les classes qui peuvent être attachées dynamiquement au moteur SnmpModulaire. Cette classe offre à toutes les classes qui l'héritent un service de gestion à distance.

TraceInterface.java (238 octets, Package : *ModularSnmp*)

Interface qui permet aux gestionnaires des modules de recevoir des messages de trace des modules.

Application.java (4,773 octets, Package : *ModularSnmp*)

Interface dont doivent hériter toutes les applications SNMP (SnmpWalk, Requestor, Proxy). Cette interface permet à un module d'envoyer et de recevoir des PDUs.

MPmodule.java (5,870 octets, Package : *ModularSnmp*)

Interface dont doivent hériter tous les modules de traitements des messages SNMP. (v1MP, v3MP)

SecurityModule.java (838 octets, Package : *ModularSnmp*)

Interface qu'héritent tous les modules de sécurité. Cette interface n'est pas héritée directement, on hérite plutôt AuthentificationModule.java ou EncryptionModule.java.

AuthentificationModule.java (745 octets, Package : *ModularSnmp*)

Interface qui doit hériter tous les modules d'authentification. (HMAC-MD5, HMAC-SHA)

EncryptionModule.java (781 octets, Package : *ModularSnmp*)

Interface qui doit hériter tous les modules d'encryption. (CBC-DES)

SecurityModel.java (4,189 octets, Package : *ModularSnmp*)

Interface qui doit hériter tous les modèles de sécurités. (USM)

4.4.2 Les modules centraux

MasterFrame.java (5,144 octets)

Masterframe est la classe de démarrage du logiciel. Cette classe crée une instance du moteur SnmpModulaire et attache les modules de traitement, de sécurité, les applications et autres.

ModularSnmp.java (5,037 octets, Package : */ModularSnmp*)

Module principale de SnmpModulaire. Une instance de ce module crée une instance du moteur SNMP. Cette classe offre les méthodes nécessaires pour attacher et détacher des modules.

APListHandler.java (1,310 octets, Package : */ModularSnmp*)

Liste de tous les modules qui sont attachés à SnmpModulaire.

SecurityDispatcher.java (5,862 octets, Package : */ModularSnmp*)

Classe qui achemine les messages vers le bon modèle de sécurité. Cette classe consulte « SecurityModelListHandler » pour connaître les modèles qui sont attachés.

ContextListHandler.java (1,825 octets, Package : */ModularSnmp*)

Table d'enregistrement de tous les « ContextEngineID » et quelle application est responsable pour chacun des contextes. On utilise généralement cette table quand une requête est reçue par le moteur et qu'on doit décider à quelle application cette requête doit être acheminée.

MsgDataCache.java (2,899 octets)

Cache des requêtes reçues dont on attend qu'une des applications réponde. Cette classe contient un algorithme de nettoyage des entrées qui sont trop vieilles.

OutstandingMsgCache.java (1,422 octets)

Mémoire « cache » qui emmagasine toutes les requêtes dont on attend une réponse. Quand une réponse est reçue, on utilise les informations contenues dans cette classe pour acheminer la réponse vers l'application qui en a fait la demande.

Dispatcher.java (16,446 octets)

Classe responsable d'acheminer les messages vers les modules appropriés. Cette classe est aussi responsable du comportement du moteur en cas d'erreur.

MPListHandler.java (1,830 octets)

Liste les modules qui traitent des messages SNMP (v1MP, v3MP). Cette classe connait les modules de traitements attachés au moteur et leur numéro respectif (SNMPv1 = 0, SNMPv3 = 3...).

SecurityModelListHandler.java (2,167 octets)

Liste de tous les modèles de sécurités qui sont attachés et le numéro associé à chacun d'eux (User Security Model = 3);

SecurityModuleListHandler.java (1,451 octets)

Liste de tous les modules de sécurités qui sont attachés et le OID associé à chacun d'eux.

EngineIDHandler.java (4,993 octets)

Cette classe fait la correspondance entre les « ContextEngineID » et l'adresse IP des agents extérieurs dont le contact initial a été effectué. À chaque contact initial, une entrée est ajoutée à cette liste. La liste des agents qui ont été contactés et leurs « ContextEngineID » peut être facilement consultée en utilisant l'application « Engine Monitor ».

4.4.3 Les modules de traitement

4.4.3.1 SNMPv1

v1MP.java (10,367 octets)

Module de traitement des paquets SNMPv1. Ce module code et décode les paquets SNMPv1. Comme SNMPv1 ne comporte pas toutes les informations nécessaires pour que le message soit acheminé correctement dans une architecture SNMPv3, ce module complète les informations manquantes avec des informations fournies par l'usager (voir « CommunityListHandler.java »).

v1MP_Frame.java (7,515 octets)

Interface usager du module v1MP.java. La fonction principale de cette interface est de permettre à l'usager de changer des entrées dans la table «CommunityListHandler.java».

Snmpv1ReqidCache.java (1,376 octets)

Pour utiliser du SNMPv1 dans un environnement SNMPv3, on effectue une substitution du « RequestID » de SNMPv1 avec le « MessageID ». Chaque substitution doit être renversée quand la réponse à une requête SNMPv1 arrive. Cette classe contient la liste de toutes les substitutions qui ont été effectuées. Un algorithme de balayage est utilisé pour enlever les entrées qui sont trop vieilles.

CommunityListHandler.java (4,955 octets)

Table sauvegardée sur disque qui contient les informations manquantes à un paquet SNMPv1 pour être acheminé avec une architecture SNMPv3. Cette table contient pour chaque « Community », un « ContextEngineID », un « ContextName » et un « Username ».

4.4.3.2 SNMPv3

v3MP.java (15,625 octets)

Module de traitement des paquets SNMPv3 tel que décrit dans RFC2272.

4.4.4 Les applications

4.4.4.1 SnmpWalk

SnmpWalk.java (3,011 octets)

Application « SnmpWalk ». Cette application envoie une succession de requêtes GETNEXT dont le but est d'énumérer tous les OIDs de l'agent et toutes les valeurs contenues dans ces objets. Quand le button « Start » est pressé, l'application envoie

une requête GETNEXT avec l'OID initiale, telle qu'entrée dans l'interface usager. À chaque réponse obtenue, « SnmpWalk » envoie un nouveau GETNEXT. Un GETNEXT n'est toutefois pas envoyé si l'erreur NOSUCHNAME est reçue, ou si l'usager presse sur « STOP ».

SnmpWalkFrame.java (7,378 octets)

Interface graphique de l'application « Requester »

4.4.4.2 Le « Requester »

Requester.java (2,864 octets)

Application « Requester ». Cette application envoie une requête GET et attend la réponse. Chaque requête est numérotée avec un compteur qui commence à 1 et augmente de 1 à chaque requête. La valeur du compteur est placée dans le RequestID du PDU envoyé. Quand une réponse est reçue, le RequestID de la réponse est affichée ainsi que le résultat de la requête.

RequesterFrame.java (6,827 octets)

Interface graphique de l'application « Requester »

4.4.4.3 Le « Proxy »

Proxy.java (4,651 octets)

Application « Proxy ».Cette application peut être configurée par l'usager pour effectuer le relais de requêtes SNMP. Cette application peut aussi convertir des requêtes SNMP d'une version vers une autre version.

ProxyFrame.java (10,702 octets)

Interface graphique de l'application « Proxy »

ProxyLinkTableHandler.java (5,724 octets)

Fait partie de l'application « Proxy ». Cette classe garde de façon persistante (fichier : *Proxy.lst*) une table qui associe un ContextEngineID et en ContextName vers une adresse de destination et le type de requête dans lequel on relaie la requête. L'usager utilise l'interface usager pour ajouter/modifier la liste des relais.

ProxyPendingMsgCache.java (2,276 octets)

Fait partie de l'application « Proxy ». Quand une requête est relayée, la réponse à cette requête doit être retournée à la source. Pour ce faire, le module garde la liste de toutes les requêtes qui ont été relayées, dont on attend la réponse. Quand une réponse arrive, la source originale de la requête est retrouvée et la réponse lui est envoyée. Comme la réponse à une requête n'est pas garantie, un mécanisme ordonne régulièrement le balayage de cette liste.

4.4.4.4 SnmpTalk

SnmpTalk.java (5,582 octets)

Application « SnmpTalk! ». Cette application n'est décrite dans aucun document ou standard. Cette application permet à deux usagers de correspondre de clavier à clavier à travers SNMP. Comme SNMPv3 est utilisé, les usagers peuvent profiter de l'authentification et de l'encryption qu'offre SNMPv3. Ce programme est un exemple d'une application SNMP qui agit à la fois comme agent et comme plate-forme. Comme agent, cette application s'enregistre sur le ContextEngineID « Talk! ».

SnmpTalkFrame.java (5,872 octets)

Interface graphique de l'application « SnmpTalk! »

4.4.4.5 Le « EngineID Monitor »

EngineMonitor.java (656 octets)

Cette application permet à l'usager de voir les « ContextEngineID » qui sont identifiés par le moteur SNMP.

EngineMonitorFrame.java (4,296 octets)

Interface graphique de « EngineMonitor ». Chaque ligne à l'écran montre l'adresse IP et le « ContextEngineID » d'un agent dont le contact initial a été effectué. Si un contact initial avec authentification a été fait, l'horloge de l'agent est affichée. La liste est mise à jours à l'écran à toutes les 5 secondes.

4.4.5 Le transporteur UDP

UDP.java (2,624 octets, Package : *ModularSnmp\Transport*)

Cette classe offre un transport UDP de transmission et de réception sur plusieurs ports. Cette classe s'occupe de bloquer des fils d'exécutions sur les ports en écoute et d'allouer la mémoire tampon nécessaire.

UDPHelper.java (1,410 octets, Package : *ModularSnmp\Transport*)

Cette classe s'occupe d'un fil d'exécution qui écoute sur un seul port UDP.

UDPInterface.java (270 octets, Package : *ModularSnmp\Transport*)

Interface que doit utiliser la classe qui désire revoir des paquets UDP.

4.4.6 Le « User Security Model »

UserSecurityModel.java (22,190 octets)

« User Security Model » ou USM telle que décrite dans le document RFC2274.

UserSecurityModelFrame.java (10,648 octets)

Interface graphique du USM. Le principal rôle de cette interface est de permettre à l'usager de changer la table des usagers (Nom, Mots de passes...) Cette interface affiche aussi les valeurs courantes de la MIB USM.

UserSecurityModelFrame2.java (6,952 octets)

Une deuxième interface est utilisée par le « User Security Model » pour entrer un nouvel usager dans la liste ou changer des valeurs existantes. Cette interface affiche les informations d'un seul usager.

UserListHandler.java (9,302 octets)

Liste de tous les usagers qui sont configurés dans le « User Security Model ». Cette liste contient une entrée pour chaque usager/agent qui a été configuré. Ces informations sont gardées sur disque et contiennent les mots de passes localisés et les algorithmes utilisés pour chaque usager/agent.

LocalBootsTime.java (4,866 octets, Package : *ModularSnmp\SecurityModel*)

Cette classe fournit à tout le logiciel la valeur courante de l'horloge (Boots et Time) et garde aussi à jour la valeur de toutes les horloges de tous les agents SNMPv3 connus.

SecurityDataCache.java (1,652 octets, Package : *ModularSnmp\SecurityModel*)

Quand une requête pour des informations des gestions est reçue, certaines informations sur cette requête sont gardées en attendant qu'une application réponde à la requête. Ces informations de sécurité sont utilisées pour sécuriser la réponse à la requête.

4.4.7 Les modules de sécurité

4.4.7.1 MD5

MD5.java (15,115 octets, Package : *ModularSnmp\MD5Authentification*)

L'algorithme MD5 « Message Digest » version 5. Un algorithme de hachage à une direction.

MD5Module.java (5,944 octets, Package : *ModularSnmp\MD5Authentification*)

Module de sécurité HMAC-MD5-96. Ce module offre le service d'authentification et de vérification des paquets SNMP.

TextualConvMD5.java (1,459 octets, Package : *ModularSnmp\MD5Authentification*)

Ce module offre le « MD5 Textual conversion » qui peut être utilisé dans certaines MIBs. Par exemple : Changer les mots de passe des usagers.

4.4.7.2 SHA-1

SHA1.java (9,914 octets, Package : *ModularSnmp\SHAAuthentification*)

L'algorithme SHA-1 « Standard Haching Algorithme » version 1. Un algorithme de hachage à une direction.

SHAModule.java (5,730 octets, Package : *ModularSnmp\SHAAuthentification*)

Module de sécurité HMAC-SHA-96. Ce module offre le service d'authentification et de vérification des paquets SNMP.

TextualConvSHA.java (1,454 octets, Package : *ModularSnmp\SHAAuthentification*)

Ce module offre le « SHA Textual conversion » qui peut être utilisé dans certaines MIBs. Par exemple : Changer les mots de passe des usagers.

4.4.7.3 DES

DESModule.java (7,829 octets, Package : *ModularSnmp\DESPrivacy*)

Module de sécurité CBC-DES. Ce module offre l'encryption et le décryption de blocs d'informations en utilisant le « Cypher Block Chaining » et DES.

DESAlgorithm.java (41,696 octets, Package : *ModularSnmp\DESPrivacy*)

Algorithme DES « Data Encryption Standard ». Cet algorithme encrypte 64 bites (8 octets) à la fois.

4.4.8 Les types et autres modules

NoSecurityModel.java (6,140 octets, Package : */ModularSnmp*)

Ce module est un modèle de sécurité complètement hors standard. Cette classe ne doit être utilisée que pour des tests, ou pour aider à la programmation seulement.

StatusInformation.java (1,195 octets, Package : */ModularSnmp*)

Le document RFC2271 utilise le type « StatusInformation » comme type de retour de plusieurs interfaces. Ce type retourne un succès ou un échec et, en cas d'échec, les informations associées à l'échec.

IpAddress.java (2,627 octets, Package : */ModularSnmp*)

Dans les documents du standard SNMPv3 (RFC2271), on décrit des primitives d'interface. Ces interfaces utilisent le type « Address ». Dans le cas de IP, cette adresse inclut un identificateur de 32 bits et un identificateur de 16 bits pour le port. Le type « Inetadress » en Java n'inclut pas le port, le type « IpAddress » est donc un ajout à « Inetadress ». Ce type permet simplement de respecter le plus possible les primitives d'interface proposées dans les documents du standard.

PduReqIdHandler.java (972 octets, Package : */ModularSnmp*)

Certaines fonctions spécifiques aux paquets SNMPv1 ont été placées dans ce module, car elles sont utilisées à plusieurs endroits dans le logiciel.

Asn.java (27,969 octets, Package : *\Snmp*)

Cette classe effectue tout le codage et le décodage des chaînes ASN.1 partout dans le programme. Même si cette classe est utilisée dans tous les modules de SnmpModulaire, l'utilisation d'une autre librairie ASN.1 est parfaitement acceptable. Comme exemple

d’utilisation de cette classe pour le codage et le décodage d’un PDU, il est recommandé de regarder le fichier SnmpWalk.java.

La conclusion au chapitre

Ce chapitre a exposé une mise en œuvre de SNMPv3, où on a retenu certaines propriétés qui sont uniques à ce logiciel et qui entrent dans les objectifs de recherche d’un projet universitaire. Ces propriétés sont : la modularité du logiciel au temps de l’exécution, la possibilité d’extension du logiciel (même au temps d’exécution), l’affichage de messages internes pour aider le développement et l’entretien, la gestion des fils d’exécution, la tolérance aux fautes, et finalement la compatibilité avec SNMPv1. Toutefois, afin d’assurer que tout fonctionne comme prévu, le chapitre suivant aborde les tests.

Chapitre V

Pour s'assurer d'un bon fonctionnement du logiciel, des tests doivent être faits. Le choix d'une architecture de tests est important pour garantir la qualité du logiciel, tout en minimisant le nombre de tests. Ce chapitre décrit les objectifs des tests et une architecture de tests pour SNMP-Modulaire.

5.1 Les tests

5.1.1 L'objectif des tests

Ce chapitre vise à établir une stratégie complète pour tester le logiciel SnmpModulaire. À la conclusion de ces tests, on doit pouvoir conclure que le programme fonctionne correctement dans toutes les situations prévues par le standard SNMPv3 et que le programme répond adéquatement (dans un cadre raisonnable) à des situations qui n'auraient pas été prévues.

Pour arriver à cette conclusion, le logiciel SnmpModulaire doit être soumis à des tests rigoureux. On doit, par exemple : chercher à parcourir tous les chemins possibles d'un paquet SNMP dans le programme, essayer tous les codages valides et invalides des paquets SNMPv3, toutes les combinaisons de paquets avec chaque configuration du moteur SNMP...

Comme le nombre d'essais possible est très grand, on doit se restreindre à un ensemble limité d'essais. Une stratégie de tests doit être développée pour couvrir toutes les circonstances, sans utiliser un nombre infini de tests. Les tests sont coûteux en ressources humaines, il est donc nécessaire de développer une stratégie de tests qui soit le plus efficace possible (voir la section 2.3).

Comme SnmpModulaire est un moteur sur lequel d'autres applications sont construites, l'information sur les fonctionnalités du moteur doit être communiquée aux programmeurs des

applications. Ceci est nécessaire pour éviter que les applications qui sont construites ne se basent sur des données erronées.

On doit noter que l'objectif de ce chapitre est uniquement d'énoncer une stratégie de tests et de mesure de performance. Un autre document pourra donner les résultats des tests quand une version du logiciel sera choisie pour évaluation.

5.1.2 L'architecture de tests proposée

Après les tests unitaires, l'architecture de tests proposée pour SnmpModulaire se base en grande partie sur une combinaison des tests ascendants et descendants de boîtes noires (voir « Les principales philosophies de test » page 41).

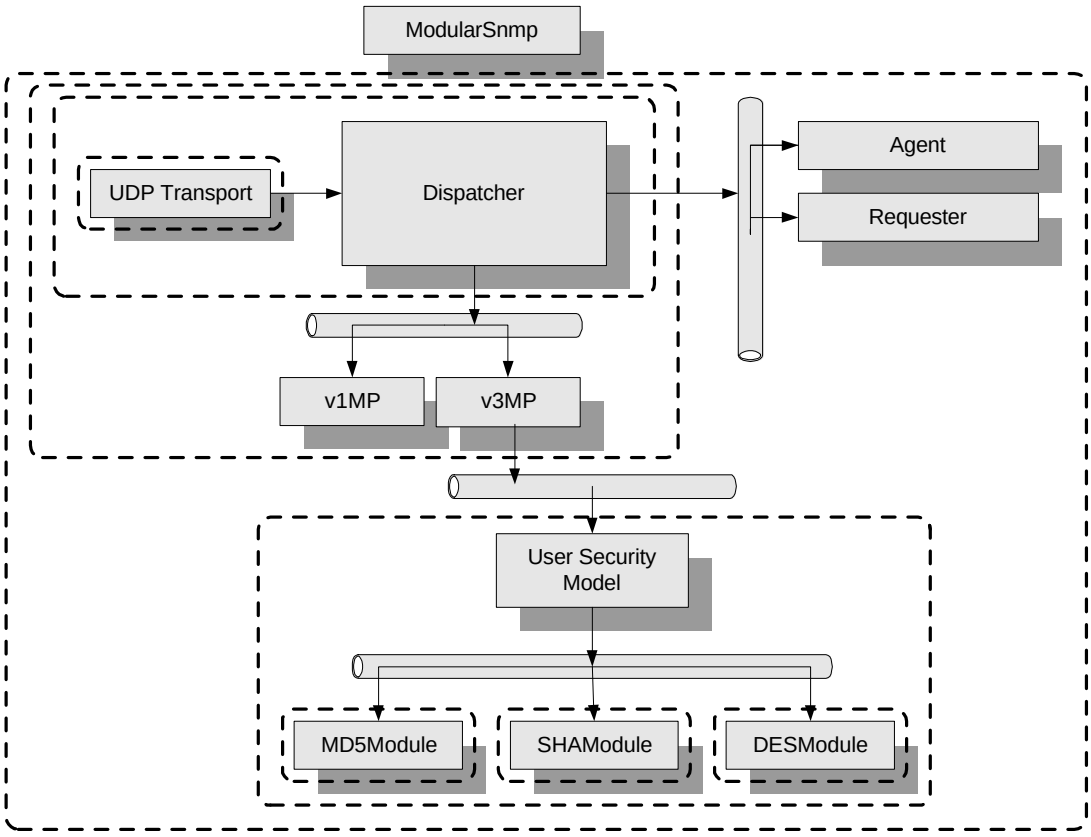


Figure 37, Décomposition des tests

La stratégie proposée est découpée comme suit :

- Les tests unitaires,
- Les tests ascendants et descendants ,
- Les tests d'interopérabilité,
- Les tests de performance.

Les sections suivantes décrivent en détail chacune de ces suites de tests.

5.1.2.1 Les tests unitaires

Certains modules se prêtent bien aux tests unitaires. ModuleMD5, ModuleSHA et le transport UDP sont essayés seuls avec des séquences précises.

5.1.2.1.1 Le module MD5

Le Module MD5 est basé sur l'algorithme MD5 qui est standardisé (RFC1321) et bien connu. Des séquences de tests pour cet algorithme sont décrites dans le document RFC1321.

Cet algorithme est construit de sorte qu'une seule erreur dans le code se propage presque toujours dans le résultat. Il n'y a qu'un seul chemin d'exécution dans cet algorithme.

Une bonne façon de tester cet algorithme est de le comparer à un algorithme déjà prouvé. Par exemple, RSA (www.rsa.com) a une version utilisée partout dans le monde. Des chaînes envoyées à l'algorithme Java et à celui de RSA devraient donner le même résultat.

Le « Module MD5 » offre le service de localisation des mots de passe basé sur HMAC-MD5-96 (RFC2204).

L'algorithme de localisation des mots de passe est propre à SNMPv3 et est décrit dans le User Security Model (RFC2274). Une séquence de test est présentée dans le RFC2274 pour la localisation basée sur HMAC-MD5-96.

5.1.2.1.2 Le module SHA-1

Le Module SHA-1 est testé de la même façon que le Module MD5. On teste l'algorithme SHA-1 (Séquences fournies dans l'annexe) puis on teste l'algorithme HMAC-SHA-96

(RFC2204) (Séquences fournies dans l'annexe), puis on essaie le test proposé dans le RFC2274 (USM) pour la localisation du mot de passe.

RSA (www.rsa.com) a aussi une version de SHA-1 qui peut être utilisée pour des tests de comparaison.

5.1.2.1.3 Le module de transport UDP

Ce module doit être tester pour assurer que :

- Les communications sont correctes (caractères perdus ou ajoutés, messages perdus).
- Le module répond aux commandes d'écoute et d'arrêt sur les ports.
- Le module gère correctement les demandes d'écoute sur des ports déjà occupés.
- Le module fait la gestion du tampon alloué à la réception de message UDP et indique correctement la taille des messages reçus.
- Le module gère correctement les exceptions qui peuvent être lancées.

Pour ce faire, des paquets de différentes tailles sont envoyés vers un port d'écoute. En Java, un tampon est alloué pour la réception de paquets UDP, (Dans SnmpModulaire, cette valeur est arbitrairement placée à 8000 octets ce qui est 5 à 7 fois plus grand que les autres agents SNMP connu). Des paquets UDP doivent être transmis de taille 0, de taille 1 >n>8000 et >8000 avec des données aléatoires.

À chaque test, on vérifie que les données se sont bien rendues, qu'elles sont identiques aux données transmises et que la taille donnée par le module est exacte. Le module de transport UDP doit être testé de la même façon en transmission.

S'il arrive que des caractères de remplissage[8] (Généralement des 0x00) s'ajoutent à la fin de paquets UDP, cette situation est acceptable. Le codage ASN.1 permet d'ignorer ces caractères. La taille de paquet reçu peut donc être plus grande que le paquet transmis.

---

[8] Le « User Security Model » ajoute quelquefois des caractères 0x00 à la fin du paquet avant l'encryption avec DES. Ceci est normal et en raison du ASN1, ne cause pas de problème.

Chaque fonction reliée au réseau doit être passée en revue et chacune des exceptions réseau doivent être attrapées (Try...Catch) et traitées correctement. Quand il est possible de le faire, on essayera de faire un test qui lancera l'exception. On vérifie alors que l'exception ne crée pas de problèmes à l'exécution du programme.

Entre les tests réseau, on demande au module de transport UDP de changer de port d'écoute. Cette opération n'est pas triviale et implique de la gestion de fils d'exécution (threads).

5.1.2.2 Les tests ascendants et descendants

Avant de commencer à intégrer des modules et effectuer des tests, des applications de tests doivent être construites. Ces applications doivent être simples et doivent afficher toutes les informations de transmission et de réception des paquets. Le décodage complet ASN1 de chaque paquet transmis est effectué et affiché à l'écran. Pour effectuer ces essais, chaque paquet transmis et reçu est envoyé sous forme hexadécimale dans un fichier, ainsi que l'heure et la date de réception.

Pour SnmpModulaire, le fichier des entrées sorties ressemble à ceci :

```
Sat May 23 17:12:06 EDT 1998 - Dispatcher -
3050020103301002040cba19a802021f400401040201030419301704 04ac10000202016002013204056e6f
73656304000400301e0404ac1000020400a1141000020201600201320 4056e6f73656304000400301e0404
ac1000020400a114020300aaaa020100020100300730050601290500
Sat May 23 17:12:07 EDT 1998 - MIB Controller - Got GETNEXT PDU type: [USM][N] Block:
0x300730050601290500
Sat May 23 17:12:07 EDT 1998 - MIB Controller - Sending Response:
Sat May 23 17:12:07 EDT 1998 - Dispatcher - Returned response packet
Sat May 23 17:12:07 EDT 1998 - Dispatcher -
3081890201033010020 40cba19a802021f400401000201030425302304 04ac10000202016002013504056e
6f736563040c00000000000000000000000000400304b0404ac1000020412446566617 56c74436f6e746578
744e616d65a22f020300aaaa020100020100302230200 6082b06010201010100041 44d6f64756c6172536e
6d70204a61766120302e39
Sat May 23 17:12:07 EDT 1998 - Dispatcher - Incoming v3 packet from 127.0.0.1:161
...
```

Les paquets en chaînes hexadécimales peuvent être étudiés plus tard avec un décodeur ASN.1

Phase 1 : Le Dispatcher

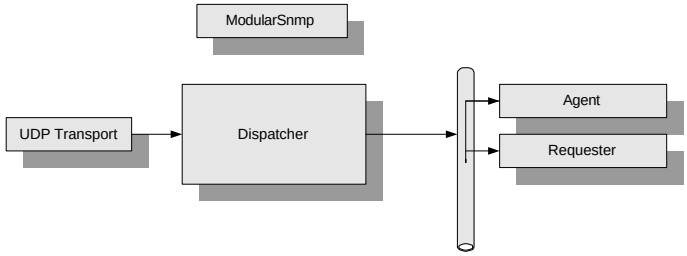


Figure 38 : Phase 1 des tests: Le dispatcher

La première séquence de tests est exécutée avec le dispatcher, le transporteur UDP et des applications de tests. Le but est de voir si le dispatcher achemine bien les paquets. Comme aucun module de décodage n'est présent, aucune erreur SNMP ne sera retournée. Les cheminements possibles sont les suivants :

- Le dispatcher reçoit un paquet SNMP du réseau, le numéro de version SNMP n'a pas pu être décodé.
- Le dispatcher reçoit un paquet SNMP du réseau, le numéro de version SNMP a été décodé avec succès, mais aucun module de décodage qui décode cette version de SNMP n'est présent.
- Une application envoie un PDU, mais le module de décodage voulu n'est pas présent. Une erreur est retournée à l'application.

Phase 2 : Le module de décodage SNMPv1

Le module de décodage SNMPv1 est relativement simple par comparaison avec SNMPv3. Cette deuxième séquence teste le codage et le décodage des paquets SNMPv1 ainsi que le reste du cheminement dans le dispatcher (Cheminement exécuté uniquement après le décodage). Si le module de traitement SNMPv1 était comparable en complexité au module SNMPv3, un module de décodage temporaire aurait dû être construit et utilisé pour tester le reste du dispatcher. Toutefois, comme SNMPv1 est très bien connu, bien documenté dans le domaine publique et que beaucoup de plate-formes et d'agents SNMPv1 existent, l'installation de ce module permet d'exécuter des tests avec de bons outils de références.

Une plate-forme de gestion SNMPv1 et un agent SNMPv1 sont indispensables pour effectuer cette séquence de tests. Avec le dispatcher, UDP et SNMPv1, tous les modules nécessaires au fonctionnement d'application sur SNMPv1 sont en place.

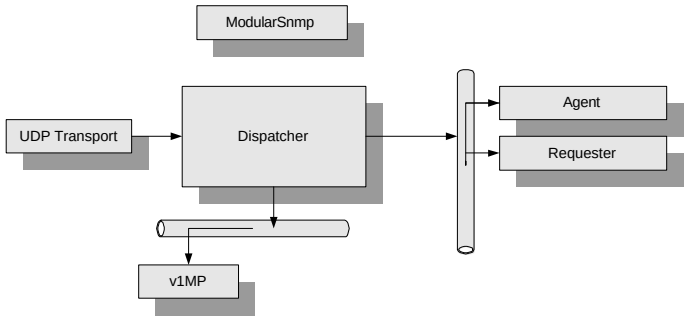


Figure 39 : Phase 2 des tests: Le module SNMPv1

Les chemins intéressants et actions à vérifier sont les suivant :

- Lancement d'une requête vers un agent SNMPv1 extérieur. On détermine si le parcours et le traitement des paquets SNMPv1 sont bien effectués à la transmission et à la réception. On s'assure que le module SNMPv1 utilise le « RequestID » du message SNMPv1 pour retracer l'application qui a envoyé la requête et qui reçoit la réponse. On doit s'assurer que, si une réponse n'arrive jamais, aucun module ne garde de façon permanente des informations sur cette requête.
- On utilise une plate-forme SNMPv1 pour envoyer une requête vers SnmpModulaire. On s'assure que l'application qui s'est enregistrée pour avoir le paquet le reçoit correctement. Une réponse est envoyée, et le cheminement de cette réponse est aussi observé. Quand la réponse est retournée, le moteur ne doit pas garder de trace de cette requête (À l'exception des compteurs dans les MIBs qui sont gardés pour des fins de gestion).
- Une requête SNMPv1 mal formulée devrait être détectée, et ignorée sans réponses et sans conséquences sur les opérations du moteur.
- Les applications agent sont informées que la requête est SNMPv1, que la réponse doit être SNMPv1 et que SNMPv1 n'a aucune Authentification et aucune Encryption.

- Le module de traitement des paquets SNMPv1 substitue le « community » de SNMPv1 par un « Username », « ContextEngineID » et un « ContextName » selon les configurations établies par l'usager.

Phase 3 : Le module de décodage SNMPv3

Pour cette phase, on ajoute le module de traitement des paquets SNMPv3.

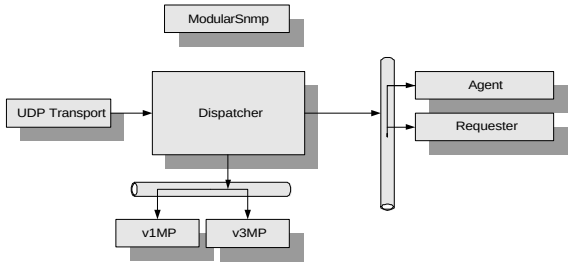


Figure 40 : Phase 3 des tests: Module SNMPv3

- Lancement d'une requête SNMPv3 NoAuth/NoPriv vers l'extérieur. Le paquet doit placer les drapeaux du paquet correctement (ExpectResponse/NoAuth/NoPriv) et doit cheminer correctement dans le moteur. Les informations sur la requête sont gardées pour un temps limité en attendant la réponse à cette requête. La réponse à cette requête doit être traitée correctement, et doit aboutir à une erreur ou un appel à « ProcessResponsePDU ».
- Le lancement d'une requête Auth/NoPriv ou Auth/Priv doit aboutir à un erreur locale « Unsupported Security Level ».
- La réception d'une requête est effectuée. Le moteur distingue entre une réponse à une requête et une requête pour des informations de gestion. La capacité de distinguer entre les deux change beaucoup le parcours du paquet, et la façon dont il est traité.

Phase 4 : Le « User Security Model »

Comme les modules de sécurités MD5, SHA-1 et DES sont testés individuellement, le « User Sécurity Model » et tous les modules de sécurité sont tous attachés pour les prochains essais.

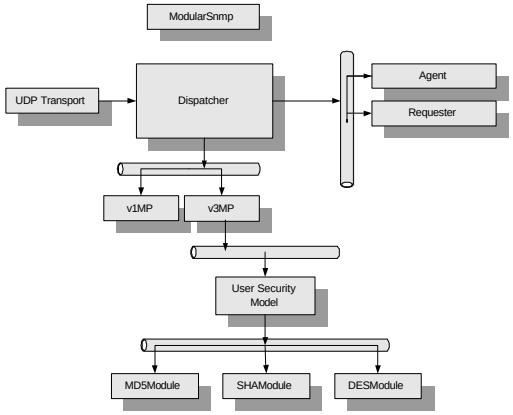


Figure 41 : Phase 4 des tests: User Security Model

On doit aussi placer dans le « User Security Model » le nom des usagers, les mot de passes et les algorithmes utilisés. Les essais entre les membres du groupe de travail de l'IETF se sont faits avec les valeurs suivantes :

```
username   authprotocol   privprotocol   authpassword   privpassword
--------   ------------   ------------   ------------   ------------
initial        md5            des        initialPass    initialPass
initial2       sha            des        initial2Pass   initial2Pass
template1      none           none       -              -
template2      md5            none       template2Pass  -
template3      sha            none       template3Pass  -
template4      md5            des        template4Pass  template4Pass
template5      sha            des        template5Pass  template5Pass
```

- Si le ContextEngineID est vide avec le « Username » à « Initial », ceci indique un contact initial. Une erreur est alors générée « Unknown ContextEngineID » et l'EngineID de l'agent est envoyé.
- Une requête Auth/NoPriv (MD5) est lancée. La première fois qu'une requête authentifiée est envoyée, l'horloge doit être à BOOTS = 0 et TIME = 0. L'erreur « Not In Time Windows » devrait être reçue et l'horloge est synchronisée pour cet agent. Les requêtes subséquentes reçoivent une réponse normale de la MIB.

- Une requête Auth/NoPriv (SHA-1) est lancée. SHA-1 place un code de hachage de 12 octets dans le paquet SNMPv3, la même longueur que MD5.
- Une requête Auth/Priv (MD5, DES) est lancée. Le vecteur d'initialisation change à chaque paquet de façon la plus aléatoire possible.
- Une requête Auth/Priv (SHA-1, DES) est lancée.
- On inscrit un usager configuré sans module d'encryption et sans module authentification. Une requête Auth/NoPriv ou Auth/Priv émise par cette usager retourne « Unsupported Security Level ».
- Un usager qui n'est pas connu par l'agent est configuré. Une requête émise par cet usager retourne l'erreur : « Unknown UserName ».
- On configure un usager avec un mauvais mot de passe. Une requête émise par cette usager retourne « WrondDigest » ou « Decryption Error » selon que le mot de passe de l'authentification ou de l'encryption est mauvais.

5.1.2.3 Les tests d'interopérabilité

Une des parties du cycle de vie d'un standard à l'IETF concerne les tests d'interopérabilité. Pour que les RFC de SNMPv3 passent de « Proposed Standard » à « Internet Standard », on exige que des mises en œuvres du standard fonctionnent et que plusieurs d'entre eux soient testés ensemble.

À cette étape, on vise à trouver toutes les ambiguïtés et les fautes dans le standard. Les tests d'interopérabilité ne peuvent pas être effectués seuls. Il faut communiquer avec un autre groupe qui a une mise en œuvre de SNMPv3, et effectuer les tests avec lui.

Quand on teste avec une autre mise en œuvre de SNMPv3, il faut essayer les séquences suivantes des deux coté (Si chacun est agent et plate-forme) :

Effectuer des requêtes SNMPv1

Le type de requête n'est pas important, un GET est suffisant. Le but est d'essayer le moteur SNMP, pas l'application ou la MIB.

Effectuer un contact initial sans sécurité

La connaissance du ContextEngineID est nécessaire pour toutes les communications en SNMPv3. Pour ce faire, le contact initial est une étape obligatoire. Un contact initial sans authentification est assez pour l'instant.

Effectuer des requêtes SNMPv3 sans sécurité

Comme SNMPv1, un GET est suffisant. Toutefois, un « Walk » dans une MIB est recommandé.

Effectuer un contact initial avec authentification

La synchronisation des horloges est un pré-requis à une communication authentifiée. Le nom de l'usager et un mot de passe doivent être correctement entrés pour que cette étape et les prochaines étapes fonctionnent.

Effectuer des requêtes SNMPv3 avec authentification

On « Walk » dans la MIB. Quand on teste l'authentification, on doit s'assurer que la synchronisation de l'horloge est maintenue. Pour ce faire, on lance une requête, on attend plus de 150 secondes et on fait une autre requête.

A chaque requête lancée avec authentification, on compare l'horloge courante de l'agent avec celle de la plate-forme. Les variations devraient être très petites, de l'ordre de un RTT (Round Trip Time) + le temps de traitement du paquet.

Effectuer des requêtes SNMPv3 avec authentification et encryption

Comme SNMPv1, un GET est suffisant. Toutefois, un « Walk » dans une MIB est recommandé. Il est préférable d'utiliser des mots de passe différents pour l'authentification et l'encryption. Quand les mots de passe sont différents, on vérifie qu'il n'y a pas de confusion dans les mots de passe.

Les paquets SNMP qui sont authentifiés ou encryptés sont capturés avec un « sniffer » pour vérifier qu'ils sont réellement authentifiés et encryptés. Le vecteur d'initialisation (VI) de DES est vérifié, sa valeur doit changer à chaque nouvelle requête qui est envoyée.

Au moment de l'écriture de ce document, des paquets qui proviennent d'autres moteurs SNMPv3 ont été reçus et sont fournis dans l'annexe 2 (page 129) pour des fins de comparaisons.

5.1.2.4 Les tests de performance

On peut calculer la performance du moteur dans les catégories suivantes :

- Transmission et Réception de paquets SNMPv1
- Transmission et Réception de paquets SNMPv3 sans sécurité
- Transmission et Réception de paquets SNMPv3 authentifiés
- Transmission et Réception de paquets SNMPv3 authentifiés et encryptés

Chacun des tests ne doit pas compter le temps passé à créer ou à décoder le PDU. Cette fonction fait partie de l'application.

Pour la transmission, on compte le temps à partir de l'appel de SendPDU[9] ou du SendResponsePDU jusqu'à la transmission sur le réseau. Pour la réception, on compte le temps à partir de la réception du paquet jusqu'à l'appel à ProcessPDU ou ProcessResponsePDU.

Le temps de traitement d'un message SNMP varie beaucoup selon l'environnement d'exécution Java (Java Virtual Machine) et selon la machine (processeur) que l'on utilise. Pour chaque test de performance, il faut noter :

---

[9] SendPDU, SendResponsePDU, ProcessPDU et ProcessResponsePDU font partie le l'interface avec les applications, voir Application.java dans SnmpModulaire.

- Le JVM (Java Virtual Machine) utilisé (Sun, Symantec, Microsoft...) et la version.
- Le processeur que l'on utilise (Intel 80x86, MIPS, SPARC...) et sa performance
- La quantité de mémoire dans la machine
- Toutes autres informations susceptibles de changer la vitesse d'exécution (Processeur mathématique, extension multimédia...)

On effectue toujours les tests en désactivant le mécanisme de trace et le mécanisme de mise sur fichiers des paquets. Ces deux mécanismes sont coûteux en temps processeur et ne sont pas utilisés en temps normal (seulement pour le développement) et les tests réseaux.

Le temps de traitement peut varier légèrement selon la longueur du PDU à traiter. Il est donc important de fournir le PDU qui a été utilisé pour faire les tests de performance. Comme la majorité des requêtes contiennent une seul variable, il est recommandé d'utiliser un PDU relativement court.

La longueur du PDU a un impact plus marqué quand on utilise SNMPv3 avec authentification et encryption. Le chaînage des blocs DES est très vorace, et le nombre de blocs à encrypter augmente avec la longueur du PDU.

5.1.3 Résultats de performance obtenus

Plusieurs tests de performance ont été effectués sur la toute dernière version de SNMPv3 en date du 1 novembre 1998. Le premier compare SNMPv1 avec différentes utilisations de SNMPv3.

Les tests ont été effectués avec les paramètres suivants :

> On utilise le JVM de Symantec version 3.00.029(I) pour JDK 1.1.x sur un Pentium II 400mhz MMX, 128 meg ram avec une carte vidéo 3D Labs Premedia II un disque Quantum 8Gig Ultra-DMA/66 et Windows NT 4.0 SP4.

Lecture des données de gestions en local (sans réseau) avec SNMPv3 Modulaire comme agent et comme plate-forme. On utilise l'application SnmpWalk pour lire 88 valeurs.

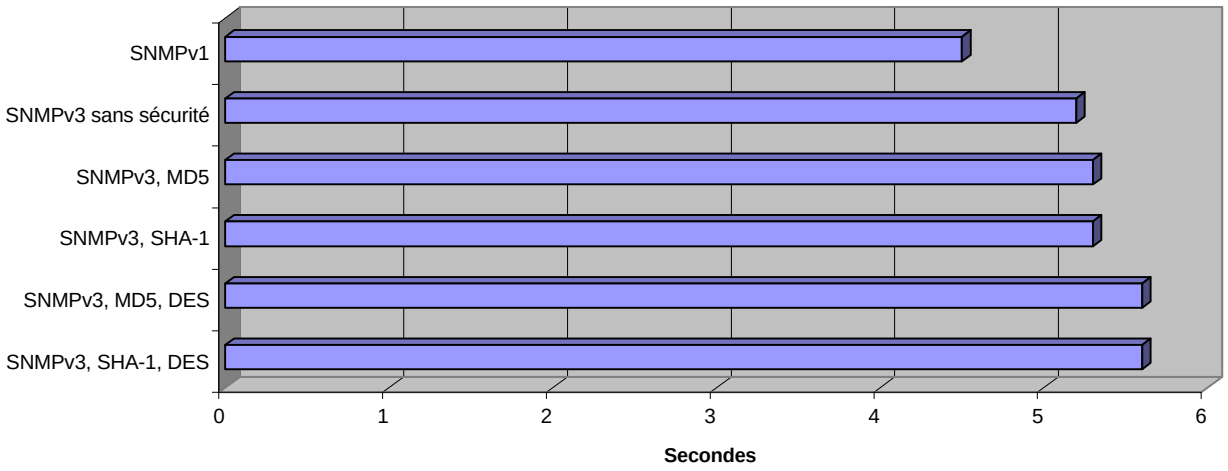


Figure 42 : Comparaison de vitesse entre les différentes versions de SNMP

On constate que toutes les méthodes offrent sensiblement la même vitesse. Le traitement des paquets SNMPv1 est plus rapide que SNMPv3. L'utilisation de DES pénalise légèrement la performance.

Le test suivant compare l'utilisation de l'application SnmpWalk avec SnmpRapidWalk. SnmpRadipWalk utilise la commande GetBulk pour obtenir plusieurs valeurs de gestion en une seule transaction.

Lecture des données de gestions sur un réseau intranet de 100mb, utilisé à moins de 1% avec SNMPv3 Modulaire comme plate-forme. On lie 304 valeurs. L'identification de l'agent est : « FreeBSD xxx.xxx.xxx.com 2.2.2-RELEASE FreeBSD 2.2.2-RELEASE #0: i386 »

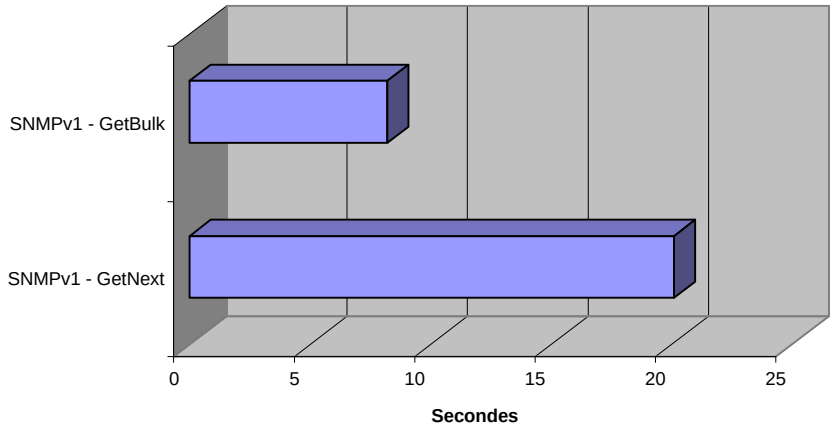


Figure 43 : Comparaison de vitesse entre GetNext et GetBulk

On constate que l'on obtient les valeurs demandées beaucoup plus rapidement. On peut aussi mesurer l'impact des requêtes sur le réseau.

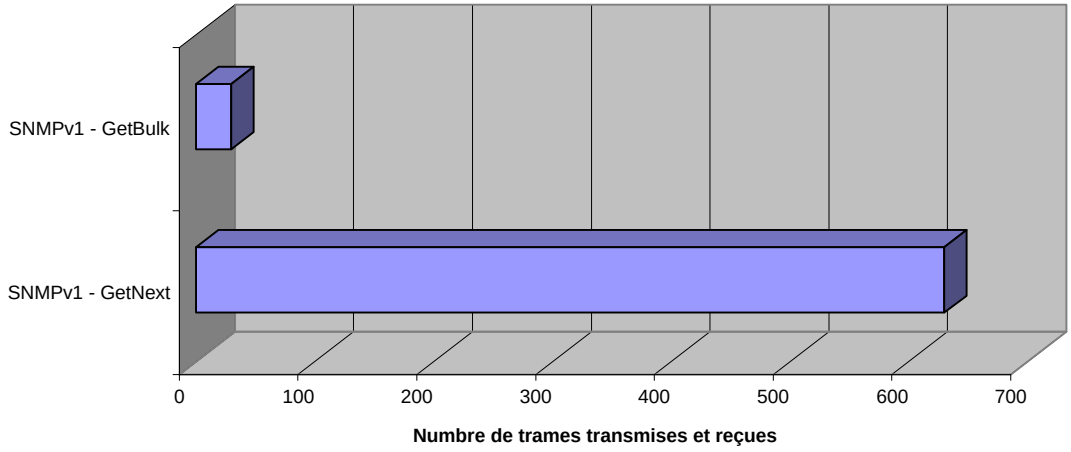


Figure 44 : Comparaison de trafic entre GetNext et GetBulk

L'utilisation du GetBulk génère beaucoup moins de trames. L'utilisation de GetNext nécessite deux trames par transaction. GetBulk a obtenu en moyenne douze valeurs par transaction.

De ces tests, on conclut que le temps n'est pas divisé par douze, car SnmpWalk et SnmpRapidWalk consomme beaucoup de temps à l'affichage de chaque donnée. Ce fait est

accentué quand on utilise SnmpModulaire sur une machine rapide.

La conclusion au chapitre

Ce chapitre a décrit une stratégie de tests, ainsi que les points à vérifier. On a aussi abordé certaines mesures de performances. La stratégie de tests montrée dans ce chapitre prend avantage de la modularité du logiciel. Ainsi, les tests peuvent être effectués plus rapidement, et avec une meilleure couverture. Le logiciel « SNMPv3 Modulaire » change rapidement, des tests exhaustifs seront préférablement conduits sur une version stable dont on a gelé les spécifications. Un ou plusieurs documents de tests seront élaborés seulement alors.

La conclusion

Dans ce document, on a abordé la gestion de réseau et certaines de ses nécessités, pour ensuite traiter de SNMPv3. L'objectif est d'aboutir à une mise en œuvre de SNMPv3 de qualité qui permet la gestion sécuritaire d'un réseau. De plus, comme on cherche à créer un logiciel innovateur, on a profité des travaux faits dans les domaines de la conception de logiciels et de la conception de moteurs SNMP. Ces principales innovations sont :

La modularité du logiciel

> Avec un logiciel modulaire, on utilise la stratégie « diviser pour régner » afin d'aider la conception et la mise en œuvre du logiciel. Cette stratégie offre aussi un meilleur entretien du logiciel. Comme la division en modules suit les divisions du standard SNMPv3, on facilite aussi la compréhension du programme à d'autres programmeurs.

L'extensibilité du logiciel au temps d'exécution

La division en module du logiciel tourne autour de « bus », et avec les avantages du langage Java, on peut attacher et détacher un module au temps d'exécution (sans recompiler le programme). On peut donc changer la configuration et envoyer de nouveaux modules java au moteur SNMP à la volée, par le réseau.

L'affichage de messages internes pour aider le développement et l'entretien

La construction et l'entretien du logiciel, spécialement dans un cadre universitaire, fait partie de la vie du logiciel. Afin de faciliter le traçage du programme, un système de messages de trace rapporte au programmeur ou au programme de test, l'état interne du système. Comme tous des modules utilisent le même système de trace, on obtient des messages consistants dans tout le système, et dont la source peut toujours être retracée grâce à la signature.

La gestion des fils d'exécution

Pour améliorer les interactions avec les acteurs externes (usagers, logiciels, éléments de réseau), il est nécessaire d'utiliser des fils d'exécution, ceci n'augmente pas nécessairement la performance du programme, mais permet plutôt un meilleur service. On ne donne pas l'impression d'être « bloqué », même pendant un travail intensif, on est toujours à l'écoute des événements extérieurs. Les files d'exécution peuvent causer des conflits et SNMPv3 Modulaire a été conçu pour éviter ces conflits, et au moins les rapporter.

La tolérance aux fautes

Avec le langage Java et l'utilisation des fils d'exécution, on peut réagir correctement même dans le cas où une erreur « catastrophique » survient dans le programme suite à un événement extérieur. Dans le pire des cas, une erreur causée par un des fils d'exécution peut forcer l'interruption, mais dans ce cas, les autres fils d'exécution peuvent être conçus pour continuer à assurer le service. La fiabilité est souvent un problème important dans les réseaux, avoir un programme qui réagit correctement même dans le pire cas, est un atout important.

La compatibilité avec SNMPv1

> Une quantité importante d'éléments de réseau est présentement en service avec le protocole SNMPv1, il est donc important pour que le logiciel « SNMPv3 Modulaire » soit viable qu'il puisse interagir avec les éléments de réseau SNMPv1. Même si le standard SNMPv3 n'aborde pas la compatibilité avec SNMPv1, SNMPv3 Modulaire est compatible avec SNMPv1 par un système de conversion. Ce système est aussi utilisé comme pont entre SNMPv1 et SNMPv3.

SNMPv3 Modulaire est encore loin de son plein potentiel, mais déjà beaucoup d'applications peuvent être créées pour profiter du moteur SNMPv3, comme par exemple : Des outils de découverte de réseau, des outils de gestion de la sécurité, des outils de gestion des usagers… et ceci avant d'avoir créé une application qui fait la gestion d'un élément de réseau en particulier.

D'autres essais pourraient être tentés pour changer la conception. Par exemple : Essayer de combiner les 3 ou plus « bus » et les 4 niveaux de communications (messages SNMP, messages de traces, message de gestion, attachement/détachement des modules) en un « bus » et un seul niveau de communication ?

Une fois la mise en œuvre du logiciel complétée, des tests exhaustifs pourraient être effectués. Toutefois, comme SNMPv3 Modulaire est un logiciel de recherche, les tests pourraient être faits dans le cadre d'autres travaux qui chercheraient à trouver de nouvelles techniques dans ce domaine. La modularité du logiciel est un facteur qui contribue beaucoup à la simplicité et à l'efficacité des tests.

Les présentations publiques et essais du logiciel SNMPv3 avec d'autres mises en œuvre de SNMPv3 ont été effectués avec succès. Le logiciel SNMPv3 Modulaire est public sur l'Internet, http://www.teleinfo.uqam.ca/snmp. À ce jour, les réactions au logiciel ont été excellentes.

Bibliographie

[AYCH 98] Ayoub Cherkaoui, Ilyas Guennoun « Gestionnaire d'alarmes sous SNMPv3, Mise en œuvre d'un environnement de développement ».

[BSCH 96] Bruce Schneier « *Applied Cryptography* », Second Edition, WILEY, 1996

[CASE 93] J. Case, K. McCloghrie, M. Rose et S. Waldbusser, « An Introduction to The Simple Management Protocol » Integrated Network Management, April 1993.

[DPAR 97] David Perkins & Evan McGinnis, « *Understanding SNMP MIBs* », Prentice Hall PTR, New Jersey, 1994.

[ISO 87] International Organization for Standardization (ISO), « *Specification of Abstract Syntax Notation One (ASN.1)* », International Standard, ISO-8824, 1987.

[ISO 89] International Organization for Standardization (ISO), « *Basic Reference Model – Part 4 : management framework »*, International Standard, ISO-7498.4, 1989.

[KAMK 94] Kamel Karoui, « *Les Tests, Chapitre 2, Partie orale de l'examen prédoctoral* », mai 94.

[MBAR 95] M. Barbeau and B. Sarikaya, « *An Approach to Conformance Testing of MIB Implementations* », IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, May 1995,.

[OCHE 98] Omar Cherkaoui, « *Télécommunications* », Chenelière/McGraw-Hill, 1998.

[RFC 768] User Datagram Protocol, septembre 1980

[RFC 1067] A Simple Network Management Protocol, août 1988

[RFC 1157] A Simple Network Management Protocol, mai 1990

[RFC 1213] Management Information Base for Network Management of TCP/IP-based internets: MIB-II, mars 1991

[RFC 1321] The MD5 Message-Digest Algorithm, avril 1992

[RFC 1447] Party MIB for version 2 of Simple Network ManagementProtocol (SNMPv2), avril 1993

[RFC 1448] Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2), avril 1993

[RFC 2104] HMAC: Keyed-Hashing for Message Authentication, Informational, février 1997

[RFC 2202] Test Cases for HMAC-MD5 and HMAC-SHA-1, Informational, septembre 1997

[RFC 2271] An Architecture for Describing SNMP Management Frameworks, Standards Track, janvier 1998

[RFC 2272] Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), Standards Track, janvier 1998

[RFC 2273] SNMPv3 Applications, Standards Track, janvier 1998

[RFC 2274] User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), Standards Track, janvier 1998

[RFC 2275] View-based Access Control Model (VACM) for the Simple Network, Management Protocol (SNMP), Standards Track, janvier 1998

[ROSP 92] Roger S. Pressman, « Software engineering, a practitioner’s approche », third edition, McGraw-Hill.

[SBAP 94] Subodh Bapat, « *Object-Oriented Networks, models for architecture, operations and management* », Prentice-Hall, 1994

[WPER 95] William Perry, « *Effective Methods for Software Testing* », John Wiley & Sons, 1995.

[WSTA 93] William Stallings, « *SNMP, SNMPv2, and CMIP – The Practical Guide to Network-Management Standards* » Addison-Wesly, Avril 1993.

Ressources Internet

IETF SNMPv3 Workgroup

The SNMPv3 Working Group is chartered to prepare recommendations for the next generation of SNMP. The goal of the Working Group is to produce the necessary set of documents that will provide a single standard for the next generation of core SNMP functions.

http://www.ietf.org/html.charters/snmpv3-charter.html
http://www.ietf.org

The Simple Times

The Simple Times is an openly-available publication devoted to the promotion of the Simple Network Management Protocol. In each issue, The Simple Times presents technical articles and featured columns, along with a standards summary and a list of Internet resources. In addition, some issues contain summaries of recent publications and upcoming events.

http://www.simple-times.org/

Request For Comments

Pour un index complet des RFC (actifs et historiques)
http://www.cis.ohio-state.edu/htbin/rfc/rfc-index.html

SNMPv3 page

This set of Web pages provides information about the Simple Network Management Protocol Version 3 (SNMPv3). The SNMPv3 specifications were approved by the Internet Engineering Steering Group (IESG) as Proposed Standard on December 5th,

1997. The SNMPv3 specifications are build on the SNMPv2 Draft Standard protocol (RFC 1902-1908) and add security and remote configuration capabilities to SNMPv2.

http://www.ibr.cs.tu-bs.de/projects/snmpv3/

SNMP Research International

A leading vendor of SNMP technology to the Original Equipment Manufacturer (OEM) and developer community.

http://www.snmp.com/

The Free On-Line Dictionary of Computing

FOLDOC is a searchable dictionary of acronyms, jargon, programming languages, tools, architecture, operating systems, networking, theory, conventions, standards, mathematics, telecoms, electronics, institutions, companies, projects, products, history, in fact anything to do with computing.
Editor : Denis Howe

http://wombat.doc.ic.ac.uk/

Web-Based Entreprise Management

Web-Based Enterprise Management (WBEM), a broad industry initiative launched July 1996 by BMC Software Inc., Cisco Systems Inc., Compaq Computer Corp., Intel Corp. and Microsoft® Corp., has successfully provided standards-based technologies to enable the development of tools and products that reduce the complexity and costs of managing an enterprise computing environment.

http://wbem.freerange.com/

Annexe 1,

Séquences des tests

HMAC-MD5-96

(Tel que décrit par le RFC2202)

Clé : « **0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b** » (16)

Chaîne : « Hi There » (8)

Réponse: « **0x9294727a3638bb1c13f48ef8158bfc9d** »

Clé : « **Jefe** » (4)

Chaîne : « what do ya want for nothing? » (28)

Réponse: « **0x750c783e6ab0b503eaa86e310a5db738** »

Clé : « **0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa** » (16)

Chaîne : « **0xdd** » répété 50 fois (50)

Réponse: « **0x56be34521d144c88dbb8c733f0e8b3f6** »

Clé : « **0x0102030405060708090a0b0c0d0e0f101112131415161718 19**» (25)

Chaîne : « **0xcd** » répété 50 fois (50)

Réponse: « **0x697eaf0aca3a3aea3a75164746ffaa79** »

Clé : « **0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c** » (16)

Chaîne : « Test With Truncation" (20)

Réponse: « **0x56461ef2342edc00f9bab995690efd4c** »

Clé : « **0xaa** » répété 80 fois (80)

Chaîne : « **Test Using Larger Than Block-Size Key - Hash Key First** » (54)

Réponse: « **6b1ab7fe4bd7bf8f0b62e6ce61b9d0cd** »

Clé : « **0xaa** » répété 80 fois (80)

Chaîne : « **Test Using Larger Than Block-Size Key and Larger Than One Block-Size Data** » (73)

Réponse: « **0x6f630fad67cda0ee1fb1f562db3aa53e** »

HMAC-SHA-96

Clé : « **0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b 0b0b0b0b** » (20)

Chaîne : « **Hi There** » (8)

Réponse: «**0xb617318655057264e28bc0b6fb378c8ef146be00**»

Clé : « **Jefe** » (4)

Chaîne : « **what do ya want for nothing?** » (28)

Réponse: «**0xeffcdf6ae5eb2fa2d27416d5f184df9c259a7c79**»

Clé : « **0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaa** » (20)

Chaîne : « **0xdd** » repeated 50 times (50)

Réponse: «**0x125d7342b9ac11cd91a39af48aa17b4f63f175d3**»

Clé : « **0x0102030405060708090a0b0c0d0e0f1 0111213141516171819** » (25)

Chaîne : « **0xcd** » repeated 50 times (50)

Réponse: «**0x4c9007f4026250c6bc8414f9bf50c86c2d7235da**»

Clé : « **0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c 0c0c0c0c0c0c** » (20)

Chaîne : « **Test With Truncation** » (20)

Réponse: «**0x4c1a03424b55e07fe7f27be1d58bb9324a9a5a04**»

Clé : « **0xaa** » repeated 80 times (80)

Chaîne : « **Test Using Larger Than Block-Size Key - Hash Key First** » (54)

Réponse: «**0xaa4ae5e15272d00e95705637ce8a3b55ed402112**»

Clé : « **0xaa** » repeated 80 times (80)

Chaîne : « **Test Using Larger Than Block-Size Key and Larger Than One Block-Size Data** » (73)

Réponse: «**0xe8e99d0f45237d786d6bbaa7965c7808bbff1a91**»

Clé : « **0xaa** » repeated 80 times (80)

Chaîne : **«Test Using Larger Than Block-Size Key - Hash Key First** » (54)

Réponse: «**0xaa4ae5e15272d00e95705637ce8a3b55ed402112**»

Clé : « **0xaa** » repeated 80 times (80)

Chaîne : « **Test Using Larger Than Block-Size Key and Larger Than One Block-Size Data** » (73)

Réponse: «**0xe8e99d0f45237d786d6bbaa7965c7808bbff1a91**»

« Password to Key Algorithm » avec MD5

(Tel que décrit par le RFC2274)

Mot de passe : "**maplesyrup**"

Avant localosation :

'9f af 32 83 88 4e 92 83 4e bc 98 47 d8 ed d9 63'

Valeur du « snmpEngineID » :

'00 00 00 00 00 00 00 00 00 00 00 02'

Sortie final de l'algorithme après la localisation :

'52 6f 5e ed 9f cc e2 6f 89 64 c2 93 07 87 d8 2b'

« Password to Key Algorithm » avec SHA-1

(Tel que décrit par le RFC2274)

Mot de passe : "**maplesyrup**"

Avant localisation :

'f1be a9ae 667f 4fb6 341e 51af 0680 7e91 e43b 01ac'

Valeur du « snmpEngineID » :

'00 00 00 00 00 00 00 00 00 00 00 02'

Sortie finale de l’algorithme après la localisation :

'8aa3 d99e 3e30 56f2 bfe3 a9ee f345 d539 5491 12be'

Annexe 2,

Paquets SNMPv3 d'un autre moteur

Ces paquets SNMPv3 ont été envoyés par David Levi (levi@snmp.com) de Snmp Research le 27 avril 1998. Pour toutes les requêtes et réponses suivantes, on demande un GET sur sysName.0 et on reçoit en retour une chaîne de caractères vide.

Transaction SNMPv3 sans sécurité

```
Outgoing Packet, length = 103:
30 65 02 01 03 30 0d 02 01 01 02 02 08 00 04 01 04 02 01 03 04 23 30
21 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 02 01 00 02 01 00 04 07
69 6e 69 74 69 61 6c 04 00 04 00 30 2c 04 0c 00 00 00 63 00 00 00 a1
c0 93 8e 6f 04 00 a0 1a 02 02 01 78 02 01 00 02 01 00 30 0e 30 0c 06
08 2b 06 01 02 01 01 05 00 05 00
Incoming Packet, length = 103:
30 65 02 01 03 30 0d 02 01 01 02 02 08 00 04 01 00 02 01 03 04 23 30
21 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 02 01 00 02 01 00 04 07
69 6e 69 74 69 61 6c 04 00 04 00 30 2c 04 0c 00 00 00 63 00 00 00 a1
c0 93 8e 6f 04 00 a2 1a 02 02 01 78 02 01 00 02 01 00 30 0e 30 0c 06
08 2b 06 01 02 01 01 05 00 04 00
```

Transaction SNMPv3 avec authentification MD5

```
Outgoing Packet, length = 116:
30 72 02 01 03 30 0d 02 01 03 02 02 08 00 04 01 05 02 01 03 04 30 30
2e 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 02 01 02 02 02 00 b0 04
07 69 6e 69 74 69 61 6c 04 0c bd 25 cd 88 98 f8 49 8c 28 25 1a 4c 04
00 30 2c 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 04 00 a0 1a 02 02
2a 93 02 01 00 02 01 00 30 0e 30 0c 06 08 2b 06 01 02 01 01 05 00 05
00
Incoming Packet, length = 116:
30 72 02 01 03 30 0d 02 01 03 02 02 08 00 04 01 01 02 01 03 04 30 30
2e 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 02 01 02 02 02 00 b0 04
07 69 6e 69 74 69 61 6c 04 0c 71 f5 68 b4 df 60 e9 5a eb ae 9d ff 04
00 30 2c 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 04 00 a2 1a 02 02
2a 93 02 01 00 02 01 00 30 0e 30 0c 06 08 2b 06 01 02 01 01 05 00 04
00
```

Transaction SNMPv3 avec authentification MD5 et encryption DES

```
Outgoing Packet, length = 128:

30 7e 02 01 03 30 0d 02 01 04 02 02 08 00 04 01 07 02 01 03 04 38 30
36 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 02 01 02 02 02 00 d0 04
07 69 6e 69 74 69 61 6c 04 0c 39 6b a7 1d b4 09 46 59 ce b7 30 e6 04
08 00 00 00 02 00 65 29 80 04 30 67 c3 79 21 30 07 d8 a6 19 fc 53 0c
c2 a5 ff 4b 7b 6c f3 d0 c6 a8 c7 18 27 04 1a c6 63 ae 91 1d 64 e9 12
09 89 f4 02 94 5c 8e 7f 32 08 fa bb ad

Incoming Packet, length = 128:

30 7e 02 01 03 30 0d 02 01 04 02 02 08 00 04 01 03 02 01 03 04 38 30
36 04 0c 00 00 00 63 00 00 00 a1 c0 93 8e 6f 02 01 02 02 02 00 d0 04
07 69 6e 69 74 69 61 6c 04 0c 3c a7 db e0 f2 6e 19 9b c8 f9 c7 7c 04
08 00 00 00 02 00 78 42 c0 04 30 ef 58 bb 77 7b 68 07 ba 64 0d 50 90
ec aa 3b 22 71 33 4a f6 54 70 c9 bc 95 d1 ec aa 1f 0b ea e4 2d 17 7e
fc 9b dc 73 59 93 43 e3 7b ba a8 76 fe
```