UNIVERSITÉ DE LORRAINE | IUT Metz Informatique

Département Informatique
Promotion 2017-2018

SNMP et Java : monitoring

Rapport final présenté en octobre 2017
par

Benoît SCHNEIDER
Mathilde CORSIGLIA

en vue de l'obtention de la LP SIL ASR

Sommaire

Introduction

Ces dernières années, l'informatique a fini par s'imposer dans notre quotidien. On y est dépendant au point de ne plus pouvoir se passer de nombreux services, qui pour la plupart, sont rendus possibles à l'aide réseaux. Cela se vérifie notamment en entreprise, lors de transactions, de visioconférences, de communication par e-mail et plus largement de toute forme d'échange informatique. Ces services étant devenus presque indispensables, il faut bien évidemment veiller à leur bon fonctionnement. De manière générale, on répond à ce besoin en surveillant l'activité sur le réseau dans le but de corriger la moindre anomalie relevée.

De nombreux éléments sont donc à surveiller sur les réseaux physiques (bande passante, charge sur le réseau, câblage, circulation de l'information, etc.). À cet effet, on retrouve des périphériques à observer tels que les serveurs, les routeurs, les commutateur/hubs, les stations de travail, les imprimantes sans compter différents autres périphériques. Ainsi, en cas de panne ou d'anomalie, l'administrateur doit savoir interpréter l'information reçue afin de localiser l'origine du problème.

Simple Network Management Protocol (abrégé SNMP ou « protocole simple de gestion de réseau » en français) est un protocole de communication conçu par CISCO, HP et Sun Microsystems (aujourd'hui Oracle) permettant aux administrateurs réseau de gérer les différents équipements réseau, de contrôler et de diagnostiquer des problèmes réseaux et matériels à distance par le biais d'une architecture distribuée de gestion des systèmes et des agents. SNMP peut donc être utilisé pour configurer des périphériques, contrôler les performances réseau (vitesse de traitement, débit, etc.) ou encore détecter les défaillances et autres accès non autorisés (déclenchements d'alertes).

Le sujet s'articule autour de l'étude de ce protocole et de la création d'un client ainsi que d'un serveur SNMP en Java. Celui-ci doit permettre d'effectuer des remontées de statistiques dans le temps, selon une durée saisie. L'agent (serveur) va remonter des données d'occupation de disque et de mémoire tandis que le manager (client) va pouvoir envoyer des requêtes sur n'importe quel appareil utilisant le protocole SNMP du réseau.

Fonctionnement de SNMP

Vocabulaire et principe

Dans un sens, SNMP reste un protocole que l'on peut qualifier de « simple ». Celui-ci permet la gestion de réseaux complexes dits « hétérogènes » (reliant des périphériques interopérables). Il est par exemple utilisé afin de gérer à distances des applications, des bases de données, des serveurs, etc.

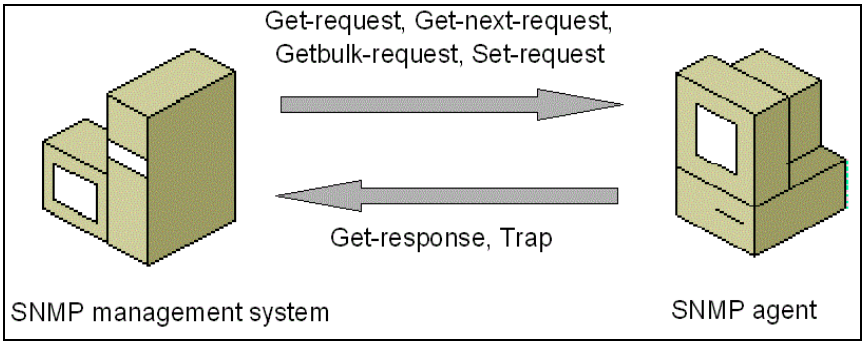
Basé sur un fonctionnement asymétrique, SNMP est constitué d'un ensemble de requêtes, de réponses et d'alertes (ou « **traps** »). Chaque appareil du réseau pouvant être administré est appelé un « **nœud manageable** ». Ces derniers sont par exemple des machines hôtes (station, serveur), des éléments d'interconnexion (hubs, routeurs, commutateurs, etc.). Ci-dessous se trouve un récapitulatif des 3 types d'instruction SNMP :

Requêtes	Réponses	Alertes
GetRequest	GetResponse	ColdStart
GetNextRequest	noSuchObject	WarmStart
GetBulk		LinkDown
SetRequest		LinkUp
		AuthentificationFailure

Liste exhaustive des instructions SNMP (v1)

On installe sur chaque nœud manageable que l'on veut superviser un **agent SNMP**, programme ayant pour mission d'enregistrer certaines informations concernant l'appareil en les stockant dans une base de données locale (la **MIB**), et ce, en permanence. Il est ensuite possible d'interroger chaque nœud manageable depuis une station d'administration (connaitre son état, vérifier le nombre d'octets émis/reçus, configurer l'appareil, ses ports autorisés, etc.

Ce protocole utilise le modèle **client-serveur,** dans lequel le client est représenté par la station d'administration (**manager SNMP**) capable d'interroger des agents SNMP. Le manager envoie donc des requêtes aux agents, lesquels retournent des réponses adéquates. En plus de cela, l'agent envoie une alerte (**trap**) à la station d'administration lorsqu'une anomalie est relevée sur un appareil du réseau. L'image qui suit schématise le fonctionnement de SNMP :



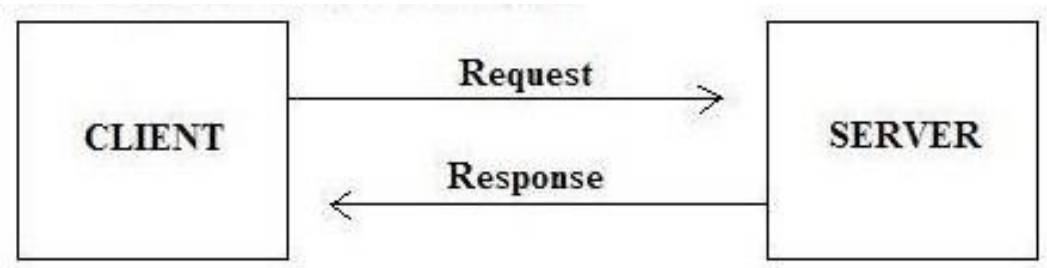
Fonctionnement SNMP entre la station d'administration (gauche) et un agent (droite)

Opérant au **7ème niveau du modèle OSI** (cf. image ci-dessous) tout en reposant essentiellement sur le protocole **UDP** (User Datagram Protocol), SNMP nécessite qu'on lui accorde un numéro de port afin de communiquer. On distingue deux ports habituellement alloués au protocole SNMP. En premier lieu il y a le **port 161**, sur lequel le manager émet des requêtes vers l'agent qui lui répond sur ce même port. Ensuite vient le **port 162**, sur lequel les messages d'alertes envoyés par l'agent sont reçus par la station d'administration.

	Modèle OSI	Modèle DOD
7	Application	SNMP
6	Présentation	
5	Session	
4	Transport	UDP
3	Réseau	IP
2	Liaison	
1	Physique	

Place du protocole SNMP dans les modèles OSI et DOD

Il faut savoir qu'UDP sert à transmettre de faibles quantités de données dans le cas où créer des connexions et les maintenir est bien trop couteux en ressources par rapport aux données à envoyer. C'est pourquoi UDP est généralement utilisé pour les applications suivant le modèle de type « interrogation-réponse » (cf. image ci-dessous). SNMP fonctionne donc sans contrôle des données transmises (mode non connecté).

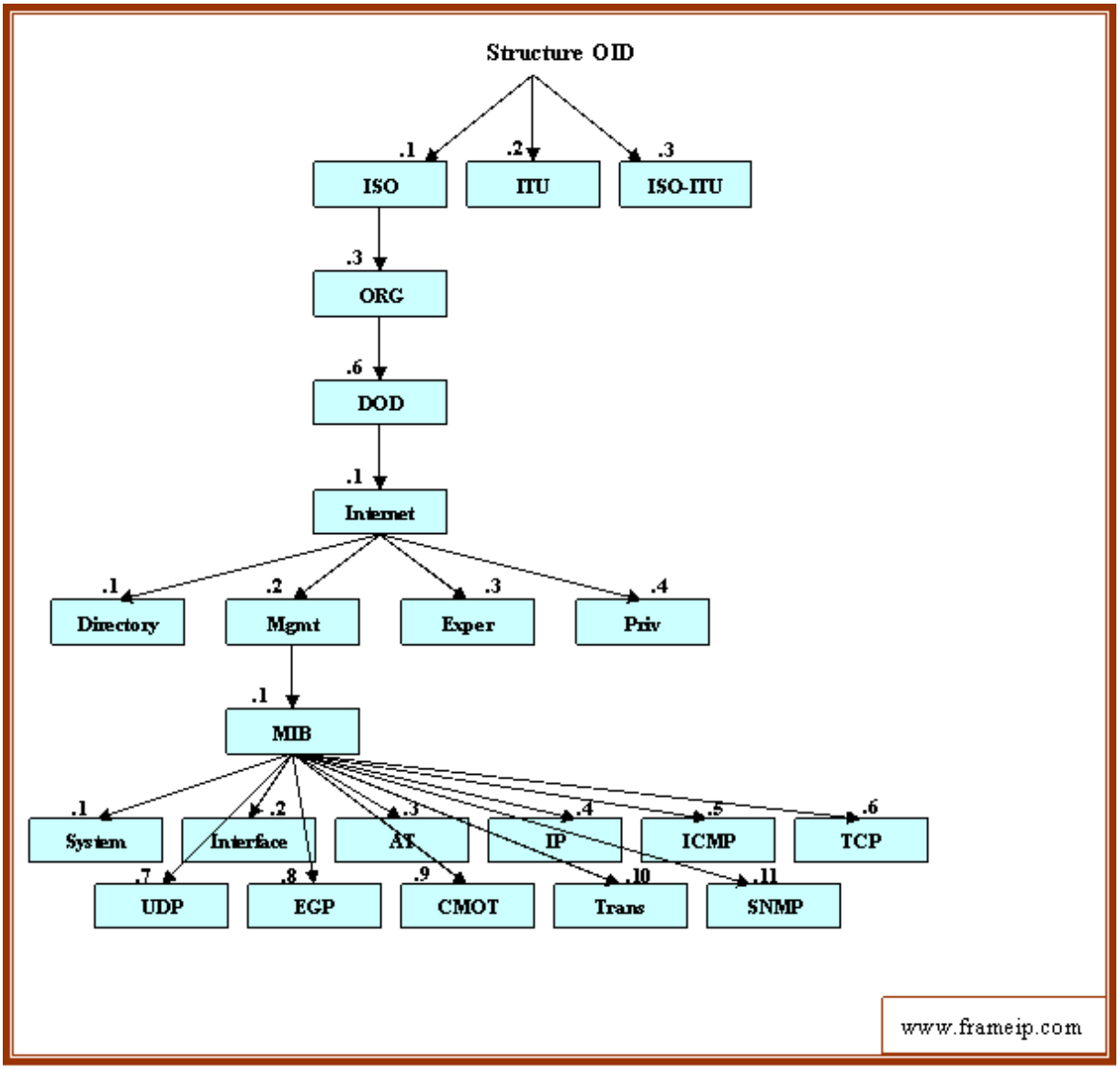


Modèle « interrogation réponse » utilisé par UDP

La MIB (Management Information Base)

La MIB est la base de données des informations de gestion propre à chaque agent, la station d'administration va donc accéder à cette dernière pour récupérer lesdites valeurs. Elle contient les descriptions de variables, tables et alertes sous la forme d'un document texte encodé en langage **ASN.1** (Abstract Syntax Notation 1). On peut en distinguer deux types qui sont « MIB 1 » (standard) et « MIB 2 » (permettant de rajouter des objets dans certaines catégories de la MIB standard).

Étant donné qu'il y a de nombreuses variables à stocker dans les MIB, l'OSI a instauré un système d'arborescence appelé **SMI** (Structure of Management Information). A partir de cette classification, des identifiants objet appelés **OID** (Object Identifier) sont établis. Chacune des variables SNMP dans une MIB peut ainsi être retrouvée de deux manières : à l'aide de son nom ou de son identifiant objet. Rappelons aussi que chaque appareil à surveiller possède sa propre MIB, sans oublier que la structure et les chemins des différents nœuds de l'arborescence le sont aussi. Ci-dessous se trouve un exemple de structure de table MIB :



Exemple de structure de table MIB

En prenant l'exemple précédent comme support, supposons que l'on veuille récupérer la variable « System » d'un nœud manageable : deux options s'offrent à nous. La première est d'**interroger directement la variable**, la seconde est d'**interroger la variable ayant pour OID la valeur correspondante à la variable recherchée**. Ainsi, on peut interroger « System » comme on peut interroger « 1.3.6.1.2.1.1. », qui est en fait un index numérique qu'utilise SNMP pour y accéder. Ce genre d'appellation n'existe que pour gagner en clarté lors de la navigation dans la MIB.

Il peut aussi arriver qu'une entreprise veuille établir son propre ensemble de variable de gestion. Pour ce faire, elle va devoir enregistrer son numéro d'objet sous le nœud privé prévu à cet effet qui est « **iso.org.dod.internet.private.entreprise** » (ou « **1.3.6.1.4.1.** »).

Deux techniques de supervision

Dans un premier temps, l'utilisation des alertes (**traps**) est de mise. Cette technique de vérification est dite « **passive** ». Il s'agit ici de configurer l'agent SNMP pour qu'il puisse contacter un autre agent lorsqu'un problème survient. Paramétrer un appareil réseau pour que ce dernier envoie des alertes SNMP lors d'actions spécifiques est donc également possible avec SNMP. Si l'on prend l'exemple d'un routeur, ce dernier peut envoyer une alerte lorsqu'il détecte que la liaison est interrompue. Lorsqu'un tel évènement se produit, l'agent du périphérique envoie l'alerte vers une destination configurée au préalable appelée « **trap host** ». Cette adresse dispose de son propre agent SNMP qui va s'occuper des alertes lorsqu'elles arrivent. Le traitement des alertes est géré par des « handlers » qui peuvent répondre de manière appropriée, en envoyant par exemple un e-mail à une personne responsable.

Dans un second temps, nous avons le « **polling** » : on envoie tout simplement une requête à intervalles réguliers dans le but de récupérer une valeur particulière. On peut qualifier cette forme de vérification d'« **active** ». On peut également vérifier si les valeurs sont conformes à l'aide de petits programmes ou de scripts. Dans le cas où la requête retourne une erreur, il est fort probable qu'il y ait un problème avec l'appareil sur lequel est installé l'agent. Étant donné que SNMP utilise la technologie UDP, il est avisé de renvoyer la requête pour confirmer la présence d'un problème (notamment dans le cas d'une vérification via Internet).

Versions 1 et 2

Origine

SNMPv1 est la première version du protocole, telle que définie dans le RFC 1157. Une première ébauche du protocole SNMP prend forme à la fin des années 80 : le protocole **SGMP** (utilisé pour gérer des routeurs IP). Suite à cela, le protocole a été amélioré et standardisé courant 1990. Cette version compte tout de même un bon nombre de lacunes parmi lesquelles on retrouve un manque de hiérarchie, de faibles performances, une sécurité laxiste (vérification basée sur une chaine de caractère sous la forme d'un identifiant de groupe de machine « **Community** »). Pour remédier au problème de sécurité une nouvelle version de SNMP voit le jour, SNMPsec ajoutant une sécurité supplémentaire basés sur des groupes.

Fonctionnement standard

Comme expliqué précédemment, le protocole SNMP est constitué d'un ensemble de messages du type « **Requête-Réponse** » entre le manager et l'agent (cf. schéma et tableau ci-après). Toutefois si un évènement anormal apparait sur un élément du réseau alors l'agent envoie une alerte au manager (Trap).

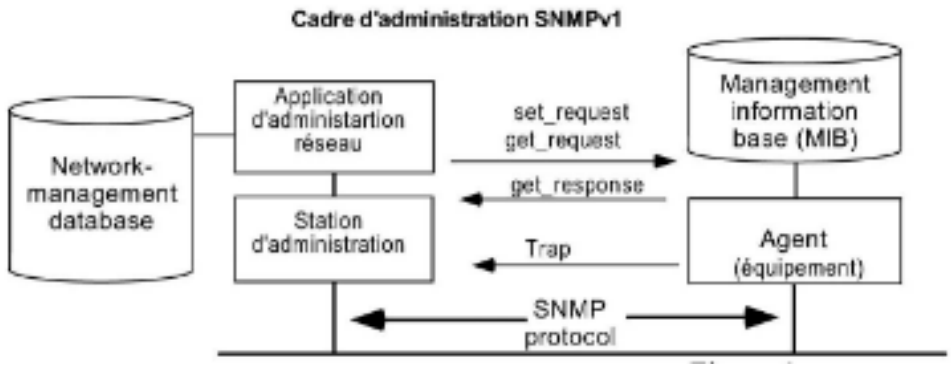


Schéma fonctionnel de SNMPv1

Les requêtes (manager)	GetRequest	Cherche la valeur d'une variable sur un agent
	GetNextRequest	Permet de lire la valeur d'une variable suivante
	SetRequest	Modifie la valeur d'une variable sur un agent
Les réponses (agents)	GetResponse	Permet de renvoyer une réponse à une requête
	NoSuchObject	Envoyée dans le cas où la variable est indisponible
Les alertes (agents)	Trap	Déclenchée un évènement inattendu (ColdStart, WarmStart, Linkdown, LinkUp, Authentification failure)

Tableau récapitulatif des instructions existantes (SNMPv1)

La communication des instructions entre manager et agent passant par une trame UDP, on va détailler ici les informations qu'elle contient dans le cas de SNMPv1. Il faut savoir qu'elle est découpée en 2 parties : l'entête et le paquet. Comme le montre l'image ci-après, l'entête regroupe la déclaration de la version du protocole (ex : « 3 » pour SNMPv3) ainsi que la chaine de caractères « Community » (qui regroupent un ensemble d'agents sur un réseau).

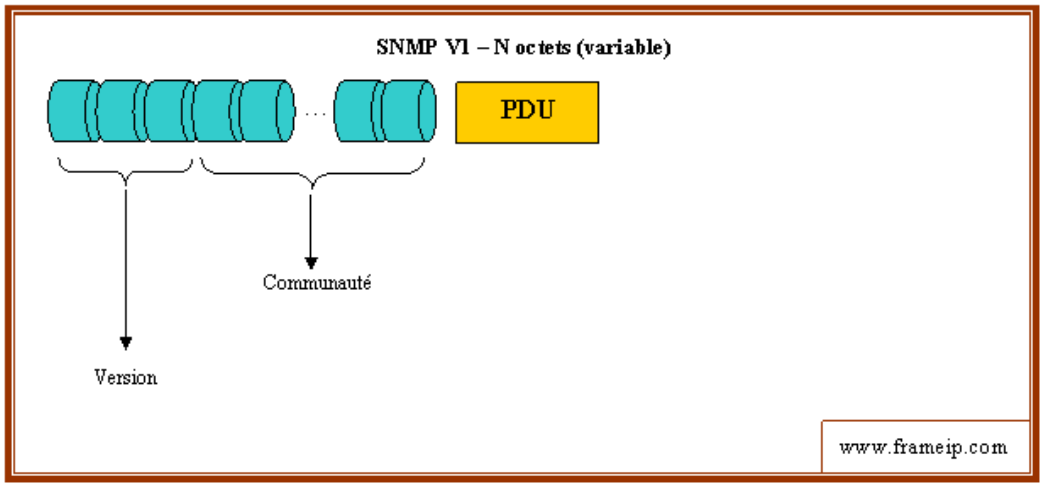


Schéma de l'entête SNMP (SNMPv1)

Le paquet contenant les informations est quant à lui composé de 6 éléments distincts. Le premier est le « **Type PDU** », qui définit le type d'instruction SNMP auquel on a affaire (ex : « 0 : Get-request »). Le second est le « **RequestID »**, qui permet au manager d'associer les réponses à ses requêtes. Le troisième élément quant à lui est « **Error Status** »**,** il indique si l'information est une erreur. Ce dernier vaudra « 0 » en l'absence de problème, sinon « 1 ». Le quatrième est « **Error Index** » qui indique le type d'erreur (ex : « Reponse=NonAccess : Accès interdit »). Le cinquième est l' « **OID »** ou l'identifiant objet permettant d'accéder à une variable dans la MIB de l'agent. Pour finir, le dernier élément de la trame est « **Value »**, soit la valeur de la variable décrite par l'OID. Le schéma qui suit résume les différentes parties composant un PDU suivant l'entête SNMP :

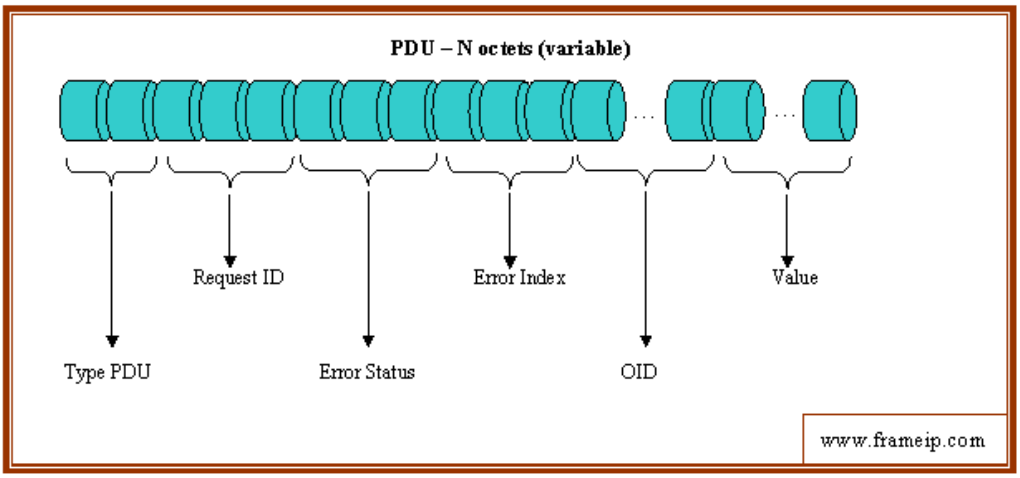


Schéma du paquet SNMP (SNMPv1)

Faiblesses de la version 1

Les défauts du protocole SNMPv1 sont multiples. Cependant, sa plus grande faiblesse est l'absence d'un système viable pour assurer la sécurité et la confidentialité des informations circulant sur le réseau (la chaine « Community » circule en clair sur le réseau, n'importe qui pourrait s'infiltrer et modifier des données).

Autres points faibles : l'authentification et le chiffrement. Les données et le mot de passe sont envoyés sans êtres chiffrées sur le réseau. La méthode « **GetNexRequest** », quant à elle, est mal optimisée. Si un manager désire récupérer un objet complexe, il va devoir passer pour toutes les variables une à une (ce qui va créer énormément de requêtes).

Il y a également aucune limitation ni contrôle d'accès à la MIB d'un agent sur le réseau, ce problème de sécurité rend SNMPv1 de type «**SHOW AND TELNET** ». On utilise SNMP pour acquérir des données de gestion, en parallèle le protocole Telnet permet d'effectuer les contrôles. Par ailleurs, le nommage des variables trop longues (tout le parcours dans la MIB, exemple « 1.3.6.1.2.1 ») a un impact non négligeable sur la bande passante du réseau.

Les améliorations des versions 2

Différentes version de SNMPv2 ont vu le jour pour essayer de résoudre les différents problèmes de la première version de SNMP. On en compte pas moins de 4 : **SNMPv2p**, **SNMPV2c**, **SNMPv2u**, **SNMPv2***. Beaucoup sont restées au stade expérimental et SNMPv2 n'a jamais réussi à devenir un standard à cause de plusieurs désaccords techniques entre chaque version.

Notons que SNMPv2c a présenté une nouveauté majeure : l'opération « **GETBULK** ». Cette requête permet à une plateforme de gestion, de faire un gros transfert de variables dans la MIB de l'agent. Chose qui n'était pas possible dans SNMPv1 qui obligeait la plateforme à faire un « **GETNEXT** » et à passer par toutes les variables et les réponses. En plus de cela, l'opération « **INFORM** » a elle aussi été ajoutée au protocole. Elle permet à une machine d'administration d'envoyer une alerte vers autre machine d'administration.

L'amélioration alertes SNMP et le début de la création de plusieurs mécanismes de sécurité tels que les messages cryptés (**DES**), un support multi-protocole, l'authentification par à l'aide d'un hachage (**MD5**).

Version 3

Objectif

Le protocole SNMPv3 est la dernière version en date du protocole SNMP. Elle corrige le souci de sécurité concernant les transactions de données sur un réseau (public ou non) en donnant la possibilité de rendre privée l'identification et les échanges entre des managers et agents. En résumé, les nouveautés apportées sont l'authentification, le chiffrement, l'autorisation et le contrôle d'accès.

USM (User Security Module)

SNMPv3 a développé une sécurité basée sur 2 concepts : **USM** (User-based-Security Model) et **VACM** (View-based Access Control Model). Le premier concept utilise trois mécanismes de sécurité : l'authentification, le chiffrement et l'estampillage de temps. Chaque mécanisme a été élaboré pour empêcher un type d'attaque sur le paquet SNMPv3.

Authentification

L'authentification permet d'assurer que personne ne puisse modifier le paquet SNMPv3 lors de la transmission et usurpe le mot de passe de la personne envoyant la requête. Pour comprendre le fonctionnement de l'authentification, il faut comprendre fonctions de hachage et l'utilisation d'un mot de passe «partagé » entre deux entités qui communiquent. Cette technique résout en partie le problème de sécurité lors de la transmission d'un paquet sur le réseau mais elle ne chiffre ou cache pas le paquet. En effet, elle empêche simplement la modification de celui-ci par quelqu'un qui ne possède pas le mot de passe. L'image qui suit illustre le fonctionnement de l'authentification :

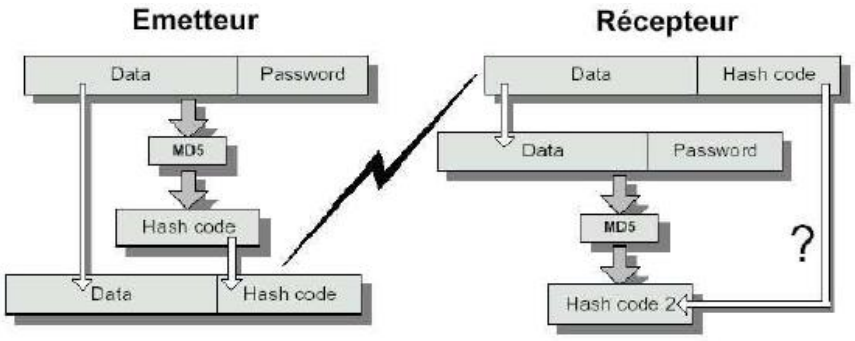


Schéma du fonctionnement de l'authentification

Les différentes étapes de l'authentification sont les suivantes :

- L'émetteur rassemble les données à transmettre avec le mot de passe
- Les données et le mot de passent dans une fonction de hachage ce qui donne un « **hash code** »
- Les données et le hash code sont envoyés sur le réseau
- Le récepteur va récupérer les données et le hash code
- Il va passer dans une fonction de hachage les données qu'il vient de recevoir et le mot passe qu'il possède en mettant de côté le hash code qu'il a reçu
- Le récepteur compare l'hash code qu'il vient de créer à celui qu'il lui a été transmis, l'émetteur sera authentifié si les deux codes sont identiques

Chiffrement des données

Cette technique permet qu'aucune information de gestion contenue dans le paquet SNMPv3 ne puisse être lue par qui que ce soit. Le chiffrement tout comme l'authentification se base sur un couple clé/mot de passe connu du manager et de l'agent communiquant. On utilise une clé de 64 bit et le chiffrement DES pour chiffrer les informations avant d'envoyer l'information sur le réseau. Les données cryptées sont ensuite récupérées puis déchiffrées avec la clé que possède l'autre utilisateur. Si la clé est correcte alors les données pourront être lues. Le schéma ci-dessous vise à résumer le chiffrement DES :

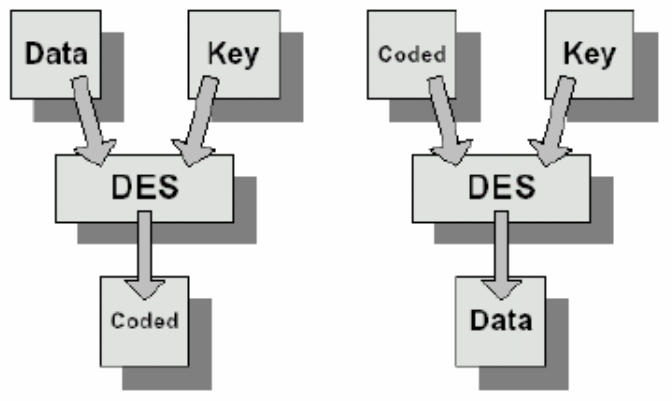


Schéma du fonctionnement du chiffrement DES

Contrôle dans le temps

L'estampillage de temps permet qu'un paquet SNMPv3 valide et qui a déjà été envoyé par une personne ne soit pas réutilisé. L'authentification et le chiffrement résolvent en partie le problème de la sécurité mais ils ne peuvent pas empêcher que quelqu'un récupère un paquet sans même le modifier pour le renvoyer plus tard sur le réseau.

Afin de palier à ce genre de problème, chaque paquet a une horloge (« **TIME** ») et un compteur (« **BOOTS** ») qui permettent de savoir depuis combien de temps l'agent est lancé et combien de fois l'équipement a été mis en route. Si la différence entre le temps actuel et le temps du paquet est supérieur à 150 secondes, alors le paquet est ignoré.

VACM (View Access Control Model)

VACM est une option de SNMP permettant de contrôler l'accès à la MIB en attribuant des droits de lecture et d'écriture à des groupes ou des utilisateurs.

Technologies utilisées

Java

Technologie imposée pour la création de l'application, Java fût initialement développé par Sun Microsystems puis ensuite par Oracle après le rachat de Sun. Elle est utilisée dans de nombreuses plateformes allant des systèmes embarqués et des téléphones mobiles aux les ordinateurs individuels, aux serveurs, aux applications d'entreprise, aux superordinateurs, etc. Lors de son lancement, Java avait pour but de pallier un manque des langages de programmation, à savoir **une conception faite pour des machines et des logiciels hétérogènes**. Cela s'avère très pratique car l'application pourra fonctionner sur n'importe quel système que ce soit Windows, Linux ou MacOS. Ci-dessous se trouve un schéma illustrant le fonctionnement de Java :

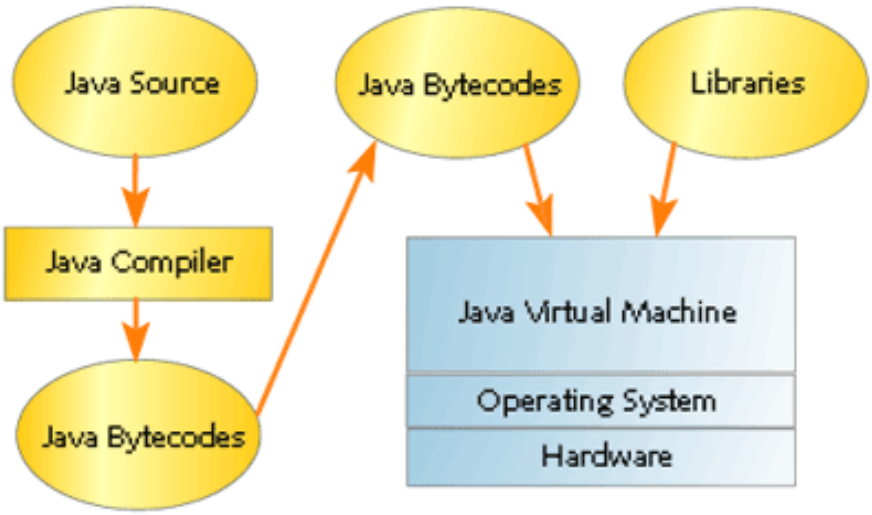
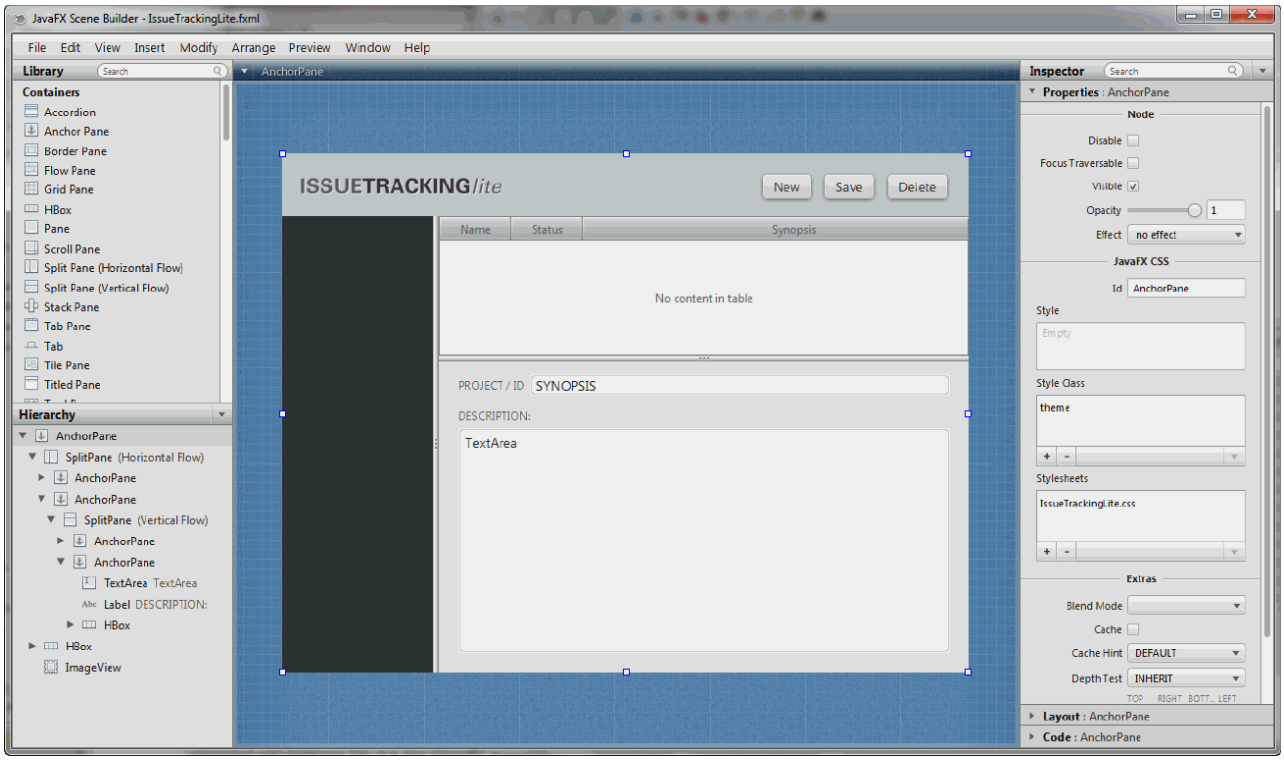


Schéma fonctionnel de Java

La JVM (**Java Virtual Machine**), le compilateur, ainsi que de nombreux outils de développement sont disponibles gratuitement. Java est devenu bien plus qu'une simple solution Internet, c'est encore maintenant un langage utilisé pour tous types de projets : distribué ou non, client lourd ou léger, etc.

JavaFX

La technologie **JavaFX,** intégrée à Java 8 depuis 2014, se présente comme la bibliothèque de création d'interface graphique officielle pour Java. Elle **remplace « Swing »** (aujourd'hui abandonnée) et s'intègre dans tous types d'applications (mobiles, clients lourds, clients légers). Son fonctionnement est relativement simple : il faut voir l'API comme une pièce de théâtre. L'élément central, la fenêtre JavaFX, est appelée « **scène** » et contient différents éléments (listes, boutons, etc.) appelés « **acteurs** » dont on va programmer les instructions lorsqu'ils seront actionnés par l'utilisateur. Une multitude de projets libres viennent compléter JavaFX en fournissant de nouveaux éléments (JFXtras et ControlsFX). L'image ci-dessous est une capture de l'outil « **Scene Builder** », destiné à faciliter la création de vues.



« Scene Builder » ou l'outil de génération de vues (FXML)

SNMP4J

SNMP4J est une « **API »** (interface de programmation) orientée objet dédiée au développement d'agent de et de manager SNMP en Java. Dans notre cas, nous avons utilisé les ressources libres de l'API (SNMP4J et SNMP4J-Agent, cf. image ci-dessous) afin de réaliser un agent collectant des **données d'utilisation disque et mémoire** ainsi qu'un manager capable d'envoyer des requêtes SNMP « **GET** » vers d'autres appareils sur le réseau.

Application réalisée (SNMP2c)

Agent

Il a été décidé en premier lieu de réaliser l'agent. Ce dernier est composé de 4 classes :

- « **Agent** » : Paramètre et créé une instance de l'agent avant de le démarrer
- « **MOCreator** » : Permet de créer et de récupérer des objets de la MIB
- « **DemarrerAgent** » : Permet de démarrer la scène JavaFX de l'agent
- « **ControleurAgent** » : Permet d'instancier l'agent et de manipuler la scène JavaFX

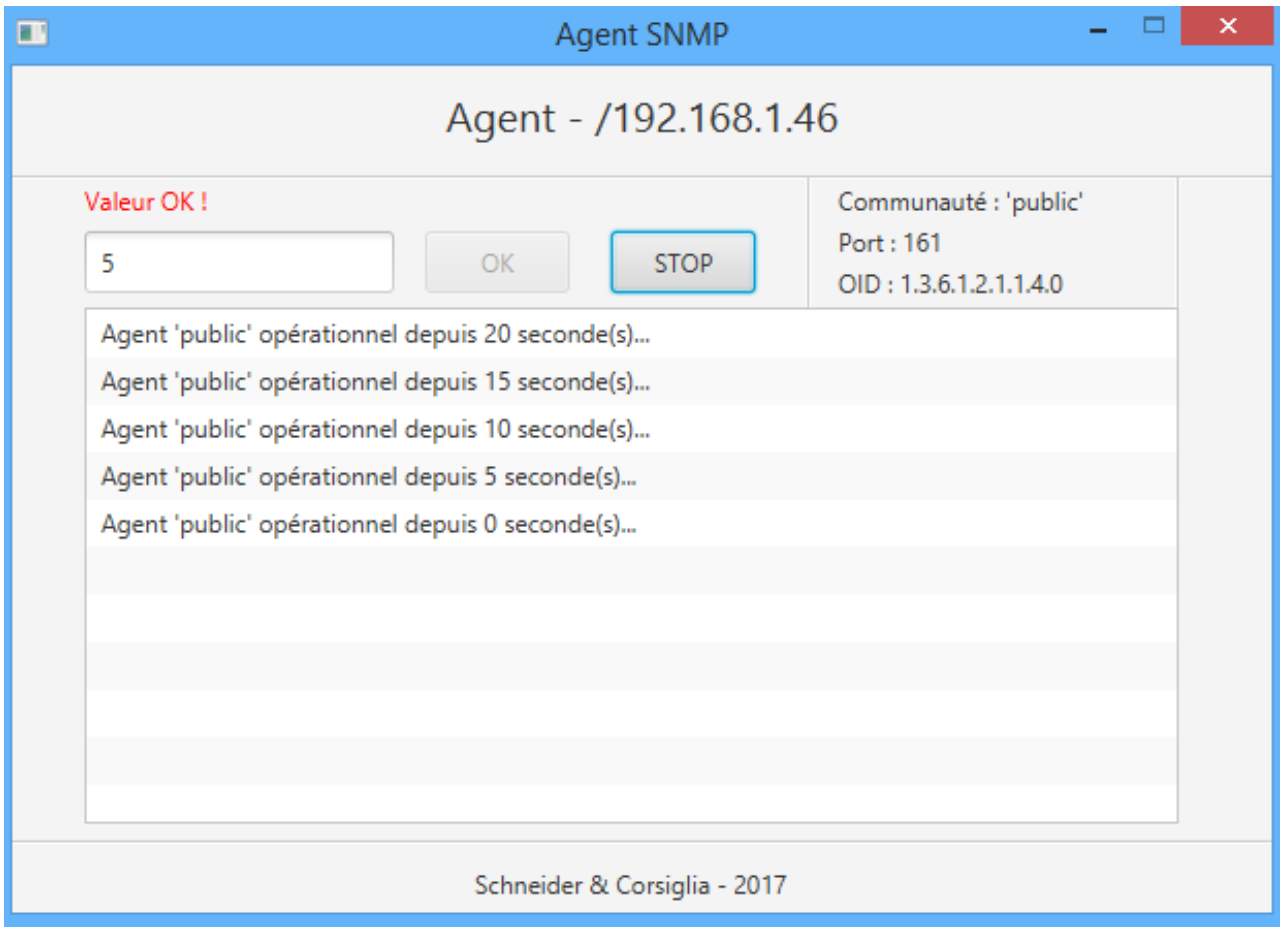
Une fois l'agent démarré, l'utilisateur se trouve face la fenêtre JavaFX. On affiche dans la zone prévue à cet effet l'adresse IP de l'agent sur le réseau. Toutefois, récupérer cette adresse n'a pas été aussi simple qu'il ne paraît. Un poste pouvant disposer de plusieurs cartes réseaux, il est difficile de déterminer laquelle sera utilisée par l'agent sur le réseau. L'astuce est d'effectuer une tentative de connexion au port 80 du DNS de Google via une socket pour déterminer l'adresse IP correspondant à la carte réseau par laquelle la trame remonte au routeur donnant sur internet.

Plusieurs valeurs conventionnelles sont affichées à titre indicatif : La communauté et le port. Celles-ci ont été affectées « en dur » dans le but de simplifier au maximum l'agent SNMP. L'utilisateur doit renseigner une valeur de temps en secondes afin de démarrer et de définir le cycle d'actualisation de l'agent.

Le bouton « OK » permet quant à lui de valider la valeur de temps, celle-ci va être testée pour vérifier qu'elle est bien conforme et une erreur s'affichera pour demander une nouvelle saisie dans le cas échéant. On instancie ensuite la classe « Agent » et on initialise le numéro de port ainsi que la chaine communauté. Cependant, quelques recherches ont montrées qu'il était assez compliqué de créer une MIB à partir de rien, si bien qu'il a été préféré « d'emprunter » arbitrairement un nœud et sa valeur dans la MIB générée par la JVM. Cette variable de support est « **sysContact** » (OID : « **.1.3.6.1.2.1.1.4.0** ») et est habituellement destinée à contenir l'adresse électronique de la personne responsable de l'agent à contacter, ce qui est en soi guère important dans le cadre de ce projet. Suite à cette opération, la MIB initialement créée par JVM est vidée du contenu de ses valeurs afin de d'inscrire uniquement la valeur à remontrer dans la MIB et de laisser les autres valeurs en stand-by pour effectuer d'autres types de remontées à l'avenir. On inscrit donc dans « sysContact » les informations récupérées sur des variables d'environnement (ici l'état des disques et la mémoire disponible dans la JVM) sous la forme d'une chaine de caractères.

En parallèle à cela, on lance un thread pour gérer la boucle d'affichage. En effet, passer par une tache asynchrone pour actualiser la liste s'est avéré nécessaire car l'utilisation d'une boucle « while » dans le thread principal (la fenêtre JavaFX) faisait geler la vue utilisateur.

Finalement, le bouton « STOP » permet de terminer le thread de la tache asynchrone. Car si on ferme l'application à l'aide de la petite croix, seulement le thread principal est terminé (la tache tourne donc dans le vide). La capture ci-après montre l'agent en fonctionnement.



Capture d'un agent s'actualisant toutes les 5 secondes

Manager

Le manager fût pour sa part réalisé dans un second temps. Ce dernier est composé de 2 classes :

- « **DemarrerManager** » : Permet de démarrer la scène JavaFX du manager
- « **ControleurManager** » : Permet d'instancier le manager et de manipuler la scène JavaFX

Lorsque le manager est démarré, l'utilisateur se retrouve face à la scène JavaFX. Il est rappelé dans le titre de l'application que le manager permet d'effectuer des requêtes SNMP « GET » envoyées en boucle à un agent sur le réseau.

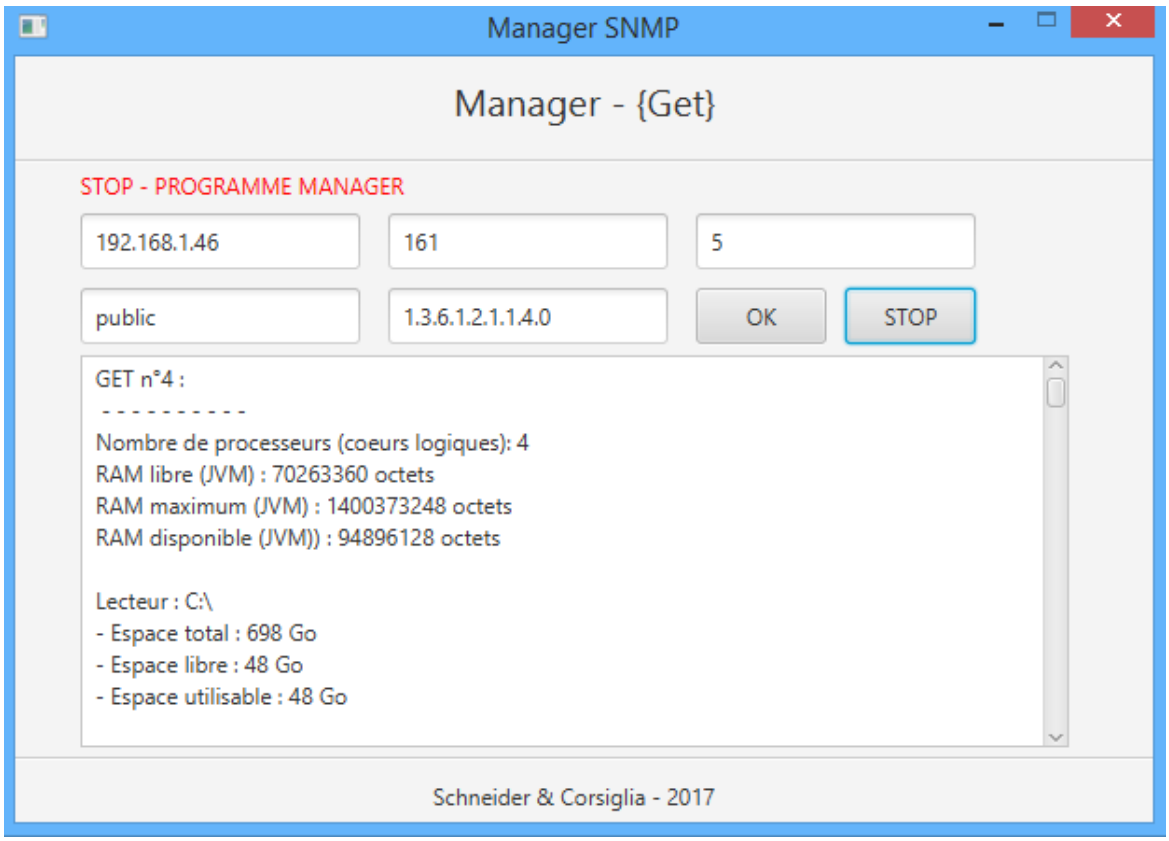
Plusieurs champs sont à renseigner pour établir la connexion à l'agent et récupérer des données : l'adresse IP de l'agent, le port de communication, la communauté de l'agent, l'OID permettant d'accéder à la variable cible et le temps d'attente entre chaque envoi de « GET ».

Suivant le même principe que l'agent, le bouton « OK » va tester les valeurs et, si elles sont recevables, on instancie la classe avant de démarrer la session SNMP en écoute afin de pouvoir traiter les réponses aux requêtes. Un récapitulatif de la requête envoyé à l'agent avec l'adresse IP, le port, la communauté, l'OID et le temps d'actualisation est ensuite affiché dans la fenêtre.

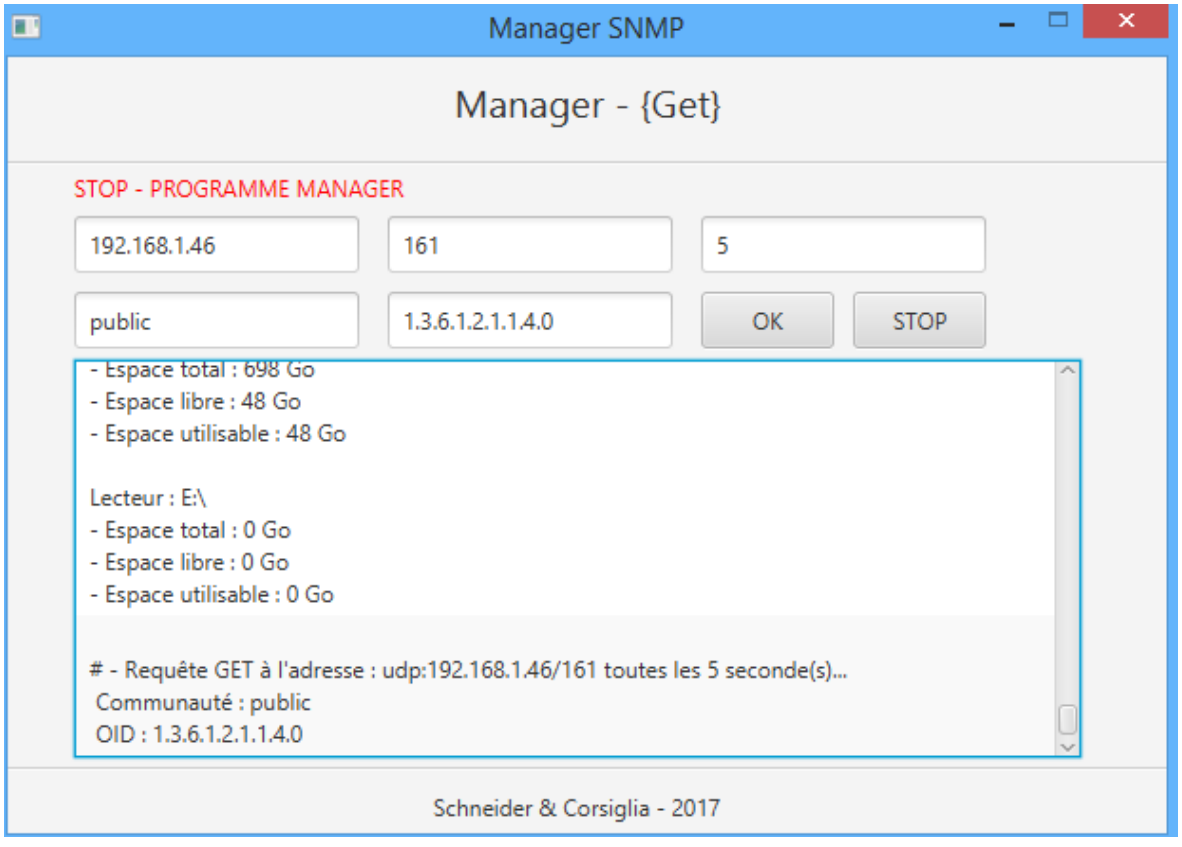
Au même instant une tâche asynchrone est lancée sur le même principe que l'agent (ne pas geler la fenêtre) avec pour mission de d'envoyer la requête en boucle vers l'agent. La génération de cette dernière se fait en deux étapes. **Premièrement, on renseigne l'entête de la trame**, à savoir : l'IP cible, la communauté, le nombre de tentatives, le délai d'attente ainsi que la version du protocole. **On ajoute ensuite à cette trame le paquet SNMP** dans lequel on spécifie l'OID cible et le type de requête avant d'envoyer la trame à l'agent. Le manager récupère ensuite la valeur correspondant à l'index dans la MIB de l'agent sous la forme d'une chaine de caractères. Les informations remontées sont par la suite affichées dans la liste prévue à cet effet à chaque requête « GET » suivant le rythme du temps d'actualisation saisi par l'utilisateur.

Pour finir, Le bouton « STOP » termine le thread de la tâche asynchrone, chose utile pour deux raisons. La première est d'éviter que le thread continue de tourner dans le vide après que le thread principal soit fermé et la seconde est de laisser la possibilité à l'utilisateur de regarder les résultats des requêtes en détail.

Les deux captures ci-après illustrent des requêtes « GET » effectuées par le manager Java sur l'agent Java (actif sur une machine distante) précédemment décrit.



Capture d'un manager remontant les valeurs de l'agent développé (1/2)



Capture d'un manager remontant les valeurs de l'agent développé (2/2)

Périphériques Cisco

Activation du service

La connexion directe aux périphériques se fait par le biais du port série (COM) en passant par le logiciel « **PuTTY** » qui sert à émuler un terminal ainsi qu'un client pour les protocoles SSH, Telnet, rlogin, et TCP. Une fois connecté, il faut passer en mode « **permission** » puis en mode « **configuration** » pour configurer le service à l'aide de la commande « **snmp-server community** » ou l'on indique la communauté et les droits sur la MIB qui en découlent :

```
router2(config)#snmp-server community public RO
router2(config)#snmp-server community private RW
```

Configuration du service SNMP (Routeur Cisco)

On enregistre ensuite la configuration de l'appareil à l'aide de la commande « **write memory** » :

```
router2(config)#exit
router2#
*Oct 19 08:03:12.589: %SYS-5-CONFIG_I: Configured from console by console
router2#write memory
Building configuration...
[OK]
router2#
```

Enregistrement de la configuration (Routeur Cisco)

Par ailleurs, il faut avoir défini son interface virtuelle d'un périphérique pour envoyer des requêtes sur ce dernier afin de le rendre accessible par le manager comme suit :

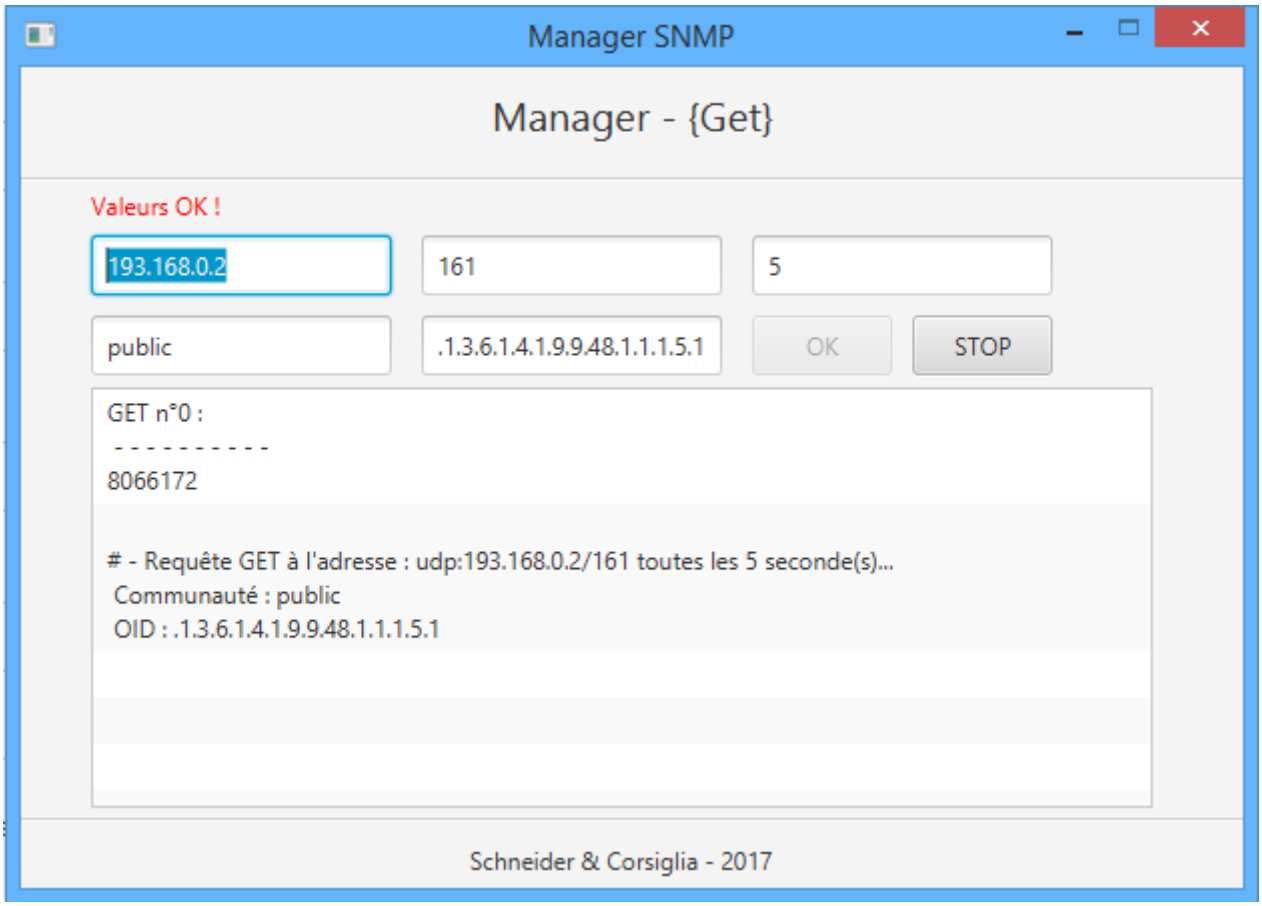
```
interface Vlan1
 ip address 193.168.0.2 255.255.0.0
 no ip route-cache
!
```

Informations relatives à l'interface virtuelle (Routeur Cisco)

Notons que deux commandes sont utiles pour vérifier qu'un périphérique est bien configuré : « **show running-config** » et « **show snmp** ». La première affiche toute la configuration de l'appareil (interfaces, services, état) tandis que la seconde permet de vérifier l'état du service SNMP et de nombre d'instructions/trames qu'il a géré.

Exemple de requête « GET »

La capture qui suit illustre la récupération de l'OID correspondant à l'utilisation de la mémoire en octet sur commutateur Cisco configuré au préalable :



Utilisation mémoire en octets via SNMP (Commutateur Catalyst 2960 series)

Quelques OID de variables Cisco :

- Utilisation CPU sur 5 secondes (%) : **.1.3.6.1.4.1.9.9.109.1.1.1.1.3.1**
- Utilisation CPU sur 1 minute (%) : **.1.3.6.1.4.1.9.9.109.1.1.1.1.4.1**
- Utilisation CPU sur 5 minutes (%) : **.1.3.6.1.4.1.9.9.109.1.1.1.1.5.1**
- Utilisation mémoire (octets) : **.1.3.6.1.4.1.9.9.48.1.1.1.5.1**

Conclusion

On ne le dira jamais assez, le réseau informatique constitue aujourd'hui un atout indispensable au sein d'une entreprise. SNMP peut donc être perçu comme un moyen technique de détecter les éventuelles anomalies en supervisant des équipements connectés à un réseau.

Depuis 1988, trois versions de SNMP ont vu le jour et pourtant la première reste la plus répandue malgré d'évidentes lacunes au niveau de la sécurité (peu d'entreprises font évoluer leur infrastructure). Différentes approches ont ensuite été utilisées pour concevoir la seconde version (qui visait à pallier aux lacunes de la première version) mais aucune d'entre elles n'a pu être choisie comme standard. La troisième et dernière version du protocole a ensuite été conçue le but de concilier les atouts des deux précédentes, pour finalement être décrétée comme standard en 2002.

L'application réalisée est entièrement fonctionnelle et correspond aux attentes du sujet. Le manager est capable d'effectuer des requêtes SNMP « GET » vers un appareil à travers le réseau avec de remonter périodiquement des informations. L'agent quant à lui fonctionnement correctement en vidant la MIB de la JVM avant d'inscrire les informations à remonter sur la variable « sysContact », jugée superflue, en se basant sur un numéro de port et une communauté définis au préalable. On remarquera qu'il aurait été plus ergonomique de pouvoir paramétrer l'agent directement à partir de la fenêtre de ce dernier, de même pour l'arrêt « propre » de l'application à l'aide des boutons « STOP » afin de pouvoir relancer des opérations sans avoir à recharger la fenêtre.

Sources

- **frameip.com**
- **supinfo.com**
- **wiki.monitoring-fr.org**
- **linux-france.org**
- **d.nouchi.free.fr**
- **2001.jres.org**
- **igm.univ-mlv.fr**
- **snmp4j.org**
- **alvestrand.no**
- **jitendrazaa.com**