

**Yacine Maghezzi
Rémi Faivre**

Master Informatique 1ère année

Département de Mathématique et Informatique



Réseau
Rapport de projet : Network Discovery Tools

**Délai de remise : 21 mai 2013
Encadrant : Noufissa Mikou**

Table des matières

1	Introduction	2
2	Network Discovery Tool	3
2.1	Pourquoi ce nom ?	3
2.2	Le Sniffer	4
2.2.1	Sniffer : Kesako ?	4
2.2.2	Sniffer : Fonctionnement	4
2.3	Le Scanner	4
2.3.1	Scanner : Kesako ?	4
2.3.2	Sniffer : Comment ça marche ?	4
3	Cahier des charges	5
3.1	Contraintes temporelles et humaines	5
3.2	Contraintes techniques	5
4	Organisation et spécification	6
4.1	Gestion du temps	6
4.2	Matériel et logiciels	7
4.2.1	Netbeans	7
4.2.2	jNetPcap	7
4.2.3	Subversion	7
5	Mise en œuvre	8
5.1	Architecture des fichiers	8
5.2	Fonctionnement des classes	8
5.2.1	La classe SnifferWindow	8
5.2.2	La classe Scanner	12
6	Améliorations possibles	15
7	Conclusion	16

1 Introduction

Un réseau informatique est un ensemble d'équipements reliés entre eux pour échanger des informations. Il permet ainsi de faire circuler des éléments matériels ou immatériels entre chacune de ces entités selon des règles bien définies. C'est la raison pour laquelle, dans le cadre de notre cours de réseau, il nous a été demandé de créer un programme, permettant d'exploiter ces échanges de données, à l'aide d'un sniffer et d'un scanner. Ces techniques développées en java sont des techniques redoutables pour l'exploitation des failles, ou encore l'écoute des canaux de communication. Elle peuvent bien évidemment être utilisées à des fins frauduleuses, mais surtout à des fins pédagogiques.

2 Network Discovery Tool

2.1 Pourquoi ce nom ?



FIGURE 2.1 – Logo de l'application

Le nom de *Network Discovery Tool* provient de l'anglais, qui signifie "Outil de découverte du réseau". Un nom judicieusement choisi, étant donnée les fonctionnalités de notre application. Nous avons choisi le logo appelé "*All seeing eye*", traduit littéralement par *œil qui voit tout*, faisant un clin d'œil à notre application. Le projet comme expliqué précédemment se compose en deux parties distinctes mais qui se rejoignent sur l'utilité de l'application en général.

2.2 Le Sniffer

2.2.1 Sniffer : Kesako ?

Un analyseur de paquets est un logiciel pouvant lire ou enregistrer des données transitant par le biais d'un réseau local non-commuté. Il permet de capturer chaque paquet du flux de données traversant le réseau, voire, décoder les paquets de données brutes, afficher les valeurs des divers champs du paquet, et analyser leur contenu. L'analyseur de paquets permet ainsi la résolution de problèmes réseaux en visualisant ce qui passe à travers l'interface réseau, mais peut également servir à effectuer de la rétro-ingénierie réseau à des buts d'interopérabilité, de sécurité ou de résolution de problème.

2.2.2 Sniffer : Fonctionnement

Lorsqu'une machine veut communiquer avec une autre sur un réseau non-commuté, elle envoie ses messages sur le réseau à l'ensemble des machines et normalement seule la machine destinataire intercepte le message pour le lire, alors que les autres l'ignorent. Ainsi en utilisant la méthode du sniffing, il est possible d'écouter le trafic passant par un adaptateur réseau (carte réseau, carte réseau sans fil, etc.). Pour pouvoir écouter tout le trafic sur une interface réseau, celle-ci doit être configurée dans un mode spécifique, le « mode promiscuous ». Ce mode permet d'écouter tous les paquets passant par l'interface, alors que dans le mode normal, le matériel servant d'interface réseau élimine les paquets n'étant pas à destination de l'hôte. Le packet sniffer décompose ces messages et les rassemble, ainsi les informations peuvent être analysées à des fins frauduleuses (détecter des logins, des mots de passe, des emails), analyser un problème réseau, superviser un trafic ou encore faire de la rétro-ingénierie.

2.3 Le Scanner

2.3.1 Scanner : Kesako ?

Le scanner ou le balayage de port est une technique servant à rechercher les ports ouverts sur un serveur de réseau. Cette technique est utilisée par les administrateurs des systèmes informatiques pour contrôler la sécurité des serveurs de leurs réseaux. La même technique est aussi utilisée par les pirates informatiques pour tenter de trouver des failles dans des systèmes informatiques. Un balayage de port effectué sur un système tiers est généralement considéré comme une tentative d'intrusion, car un balayage de port sert souvent à préparer une intrusion.

2.3.2 Sniffer : Comment ça marche ?

Un balayage de ports vise typiquement le protocole TCP, car c'est celui qui est utilisé par la majorité des applications. L'objectif du balayage est de savoir si un logiciel est en écoute sur un numéro de port donné. Si un logiciel écoute, on dit que le port est ouvert, sinon on dit qu'il est fermé. Le balayage d'un port se passe en deux étapes :

- l'envoi d'un paquet sur le port testé
- l'analyse de la réponse.

3 Cahier des charges

3.1 Contraintes temporelles et humaines

Comme demandé, le rapport doit être rendu pour le 20 mai 2013, et le rendu de l'archive du projet doit être fait lors de la présentation du programme, dans la semaine du 20 au 27 mai. Le projet devait être réalisé en binôme.

3.2 Contraintes techniques

Le projet devait être écrit en JAVA ou en C, et utiliser Jpcap. Cependant, cette librairie n'était pas fonctionnelle nous nous sommes donc penché sur une alternative : JNetPcap.

4 Organisation et spécification

4.1 Gestion du temps

Afin de pouvoir montrer de manière simple et explicite la gestion du temps de ce projet, nous avons effectué un diagramme dit de "Gantt", regroupant les dates et étapes approximatives qui ont conduit à la réalisation de celui ci.

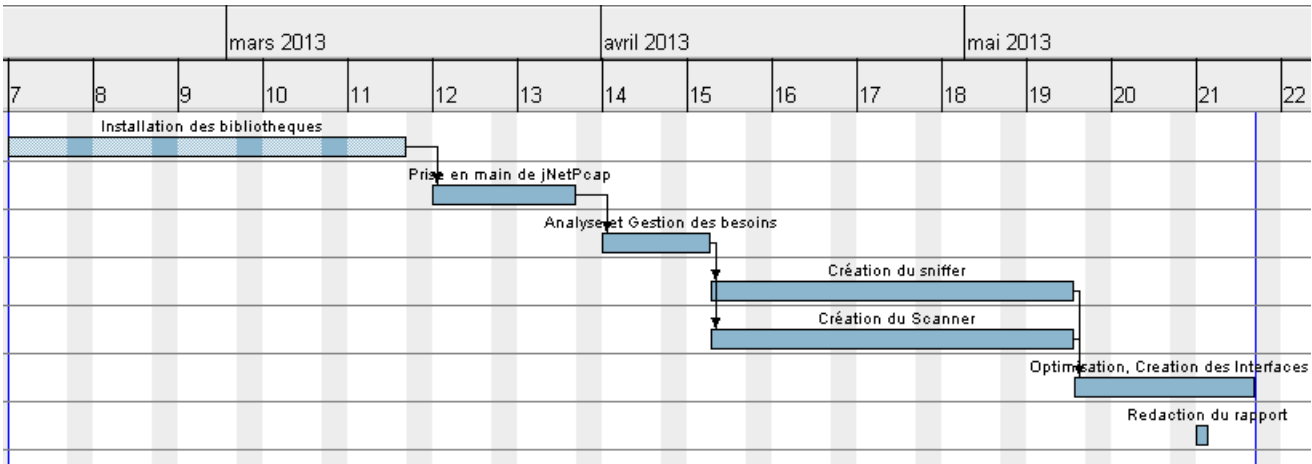


FIGURE 4.1 – Diagramme de gestion du temps

4.2 Matériel et logiciels

4.2.1 Netbeans



NetBeans est un environnement de développement intégré (IDE¹), placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2. En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web). Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris.

Grâce à son aide au développement d'interface graphique, nous avons pu créer de manière rapide et précise une interface "User-Friendly". Il est à noter que nous avons dû utiliser JDK7 afin de pouvoir bénéficier de toute la puissance de jNetPcap.

4.2.2 jNetPcap

jNetPcap est une bibliothèque Java open-source permettant le développement de solutions réseau. Les fonctionnalités sont nombreuses, on note cependant que cette bibliothèque permet la capture de paquet en temps réel, permet l'exploitation de beaucoup de protocoles réseau. Les utilisateurs peuvent facilement ajouter leurs propres définitions de protocole à l'aide du SDK Java jNetPcap utilise un mélange d'implémentation native et Java pour une performance optimale de décodage de paquets.



4.2.3 Subversion



Subversion (en abrégé svn) est un logiciel de gestion de versions, distribué sous licence Apache et BSD. Il nous a permis de synchroniser nos travaux tout au long de sa création.

1. Interface De Développement

5 Mise en œuvre

5.1 Architecture des fichiers

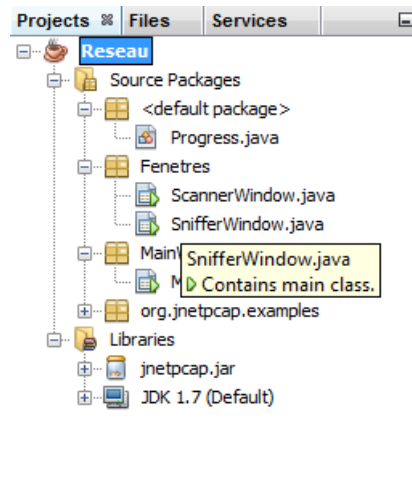


FIGURE 5.1 – Architecture des fichiers

Comme le montre l'image précédente, les fichiers sont classés par fenêtre. Nous avons décidé de créer une classe Java par fenêtre, ce qui nous facilite l'organisation des données et la résolution de problèmes. Notre projet se schématise en 3 classes principales, correspondant aux 3 fenêtres. La classe **MainWindow** représente donc la fenêtre qui va appeler les autres. Les classes **SnifferWindow** et **ScannerWindow** gèrent respectivement le sniffer et le scanner.

5.2 Fonctionnement des classes

5.2.1 La classe SnifferWindow

Cette classe est donc la classe référente au sniffer. En effet c'est grâce aux fonctions qui la compose que nous pouvons sniffer une interface réseau préalablement choisie.

Il suffit donc de sélectionner l'interface réseau choisie et d'entrer le nombre de paquets à sniffer. Il est aussi possible d'avoir plus d'informations en cliquant sur le bouton "?" à côté du menu déroulant. Pour

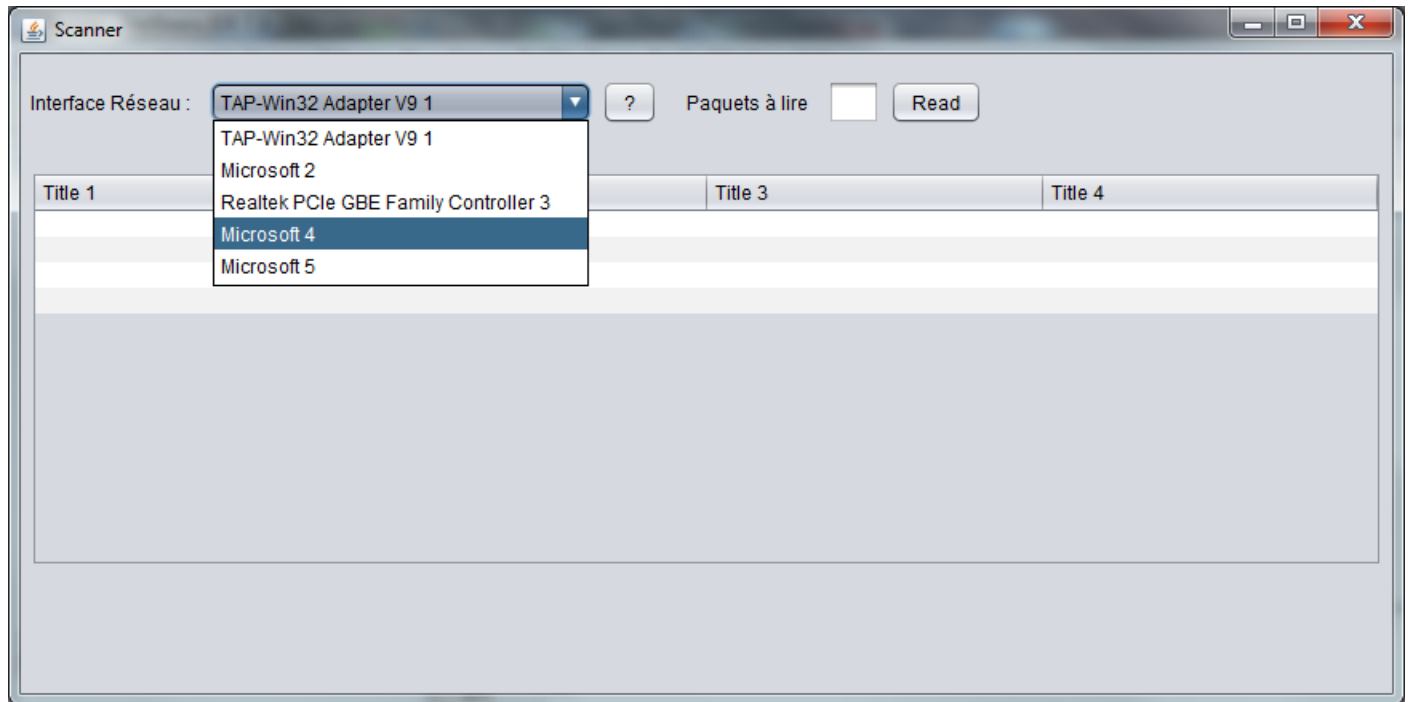


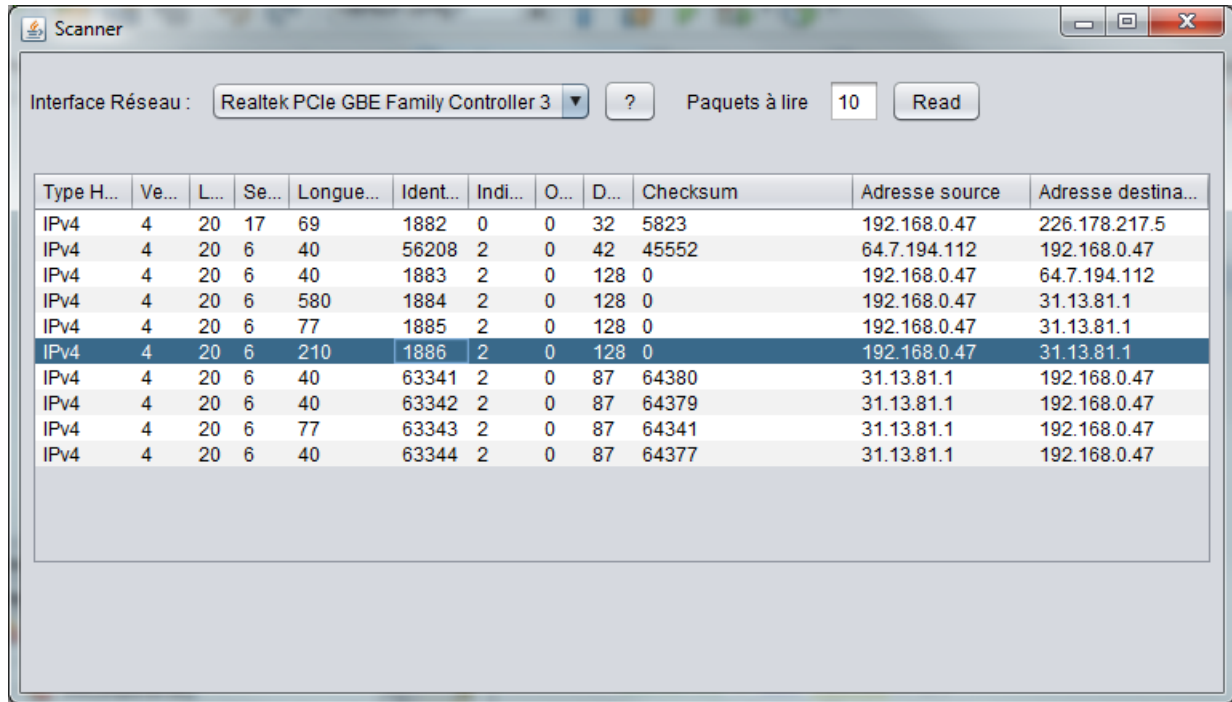
FIGURE 5.2 – Interface Sniffer initiale

ce faire, la fonction **recupererInterfaces()** est appelée dans le constructeur afin qu'elle soit chargée en même temps que le chargement de la fenêtre.

Une fois l'interface souhaitée sélectionnée et le nombre de paquets à sniffer choisi, il est possible de lancer l'écoute sur le réseau. Cette écoute se fait via la fonction **LirePaquets()** qui grâce à la fonction **loop** de jNetPcap, et à la surcharge de la méthode **nextPacket()**. Pour ce faire il est nécessaire d'utiliser un objet comme ci dessous :

```
1 Pcap pcap = Pcap.openLive(device.getName(), snaplen, flags, timeout, errbuf);  
...  
3 PcapPacketHandler<String> jpacketHandler = new PcapPacketHandler<String>();  
...  
5 pcap.loop(numberPacket, jpacketHandler, "");
```

Le programme est indisponible lorsque qu'il est en cours d'écoute. Une fois l'écoute terminée le tableau ci dessous sera rempli de la manière suivante.



The screenshot shows a window titled 'Scanner' with a menu bar and a toolbar. Below the toolbar, there is a section for network interface selection and packet reading. The 'Interface Réseau' dropdown is set to 'Realtek PCIe GBE Family Controller 3'. To its right is a '?' button. Further right, 'Paquets à lire' is set to '10' with a 'Read' button. Below this is a table with 12 columns: 'Type H...', 'Ve...', 'L...', 'Se...', 'Longue...', 'Ident...', 'Indi...', 'O...', 'D...', 'Checksum', 'Adresse source', and 'Adresse destina...'. The table contains 10 rows of data, all of which are IPv4 packets. The fifth row is highlighted in blue.

Type H...	Ve...	L...	Se...	Longue...	Ident...	Indi...	O...	D...	Checksum	Adresse source	Adresse destina...
IPv4	4	20	17	69	1882	0	0	32	5823	192.168.0.47	226.178.217.5
IPv4	4	20	6	40	56208	2	0	42	45552	64.7.194.112	192.168.0.47
IPv4	4	20	6	40	1883	2	0	128	0	192.168.0.47	64.7.194.112
IPv4	4	20	6	580	1884	2	0	128	0	192.168.0.47	31.13.81.1
IPv4	4	20	6	77	1885	2	0	128	0	192.168.0.47	31.13.81.1
IPv4	4	20	6	210	1886	2	0	128	0	192.168.0.47	31.13.81.1
IPv4	4	20	6	40	63341	2	0	87	64380	31.13.81.1	192.168.0.47
IPv4	4	20	6	40	63342	2	0	87	64379	31.13.81.1	192.168.0.47
IPv4	4	20	6	77	63343	2	0	87	64341	31.13.81.1	192.168.0.47
IPv4	4	20	6	40	63344	2	0	87	64377	31.13.81.1	192.168.0.47

FIGURE 5.3 – Interface Sniffer après analyse

Ici nous avons donc 10 paquets que notre programme a réussi à sniffer.

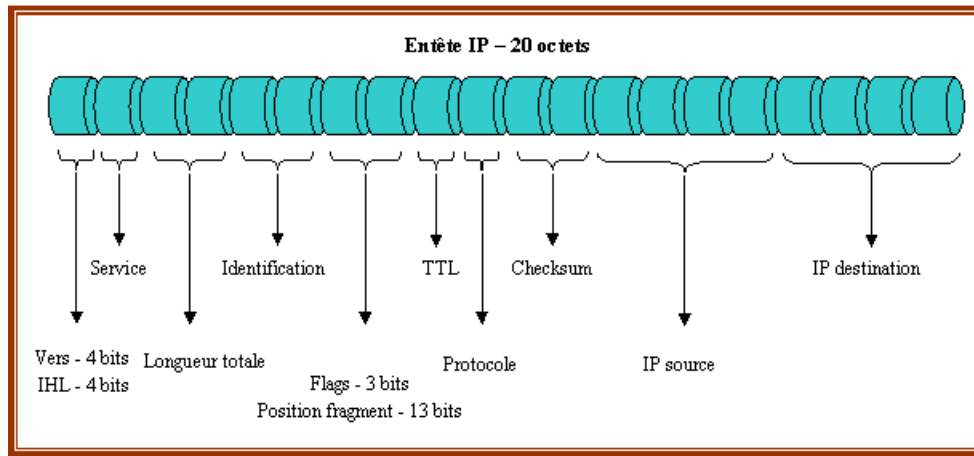


FIGURE 5.4 – Diagramme d'un entête IP

On observe ici que le diagramme correspond bien aux informations sniffée par le programme.

Chaque ligne définit un paquet, et nous affichons les informations sur ces paquets. L'affichage est fait à l'aide d'une *jTable*, qui est utilisable seulement en lui affectant un *DefaultTableModel*. Ce modèle est rempli au préalable dans la fonction **nextPacket()**. Chaque colonne représente un objet *Vector*, qui auquel on affecte la valeur d'une information d'un paquet. Le *Vector* est ensuite affecté au modèle, puis lui-même affecté au *jTable*.

5.2.2 La classe Scanner

La classe Scanner est donc celle qui va gérer notre scanner. Cette partie a été plus difficile à mettre en place, car la manipulation des différents types, ainsi que son affichage sont plus complexes.

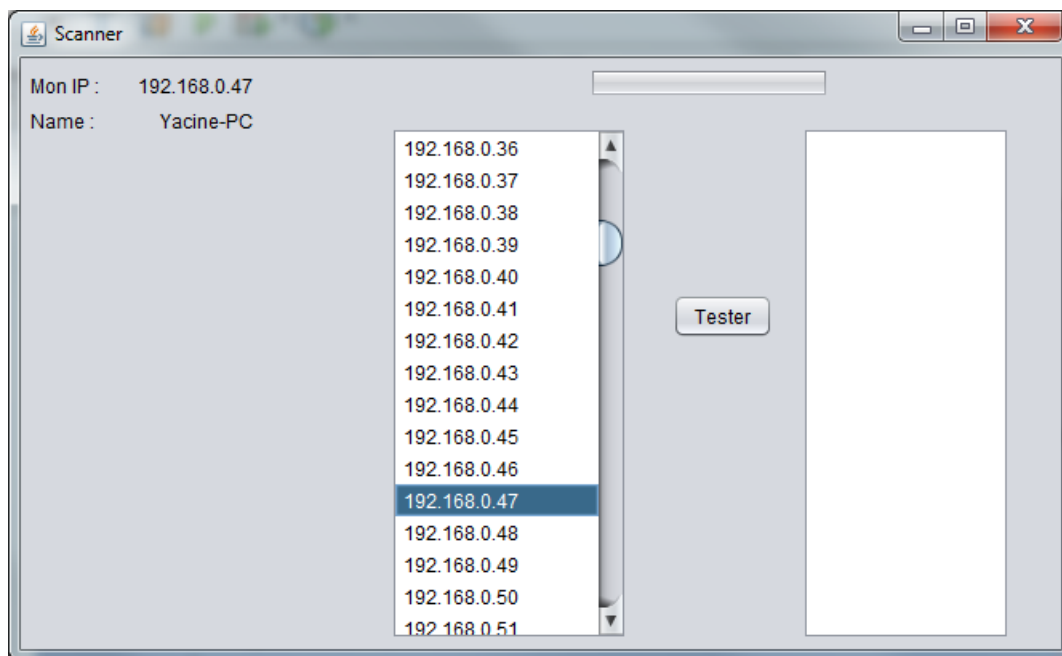


FIGURE 5.5 – Interface Sniffer après analyse

La fenêtre du scanner comporte en haut les informations concernant le nom de notre machine ainsi que l'IP locale de celle-ci. À partir de cette IP, nous allons construire toutes les IP du sous-réseau, qui sont comprises donc entre 0 et 255. Elles sont listées dans la première *JList* qui attend que l'on sélectionne une d'entre elles.

Comment construire les IPs ?

En java, il est possible d'obtenir son adresse IPs de la manière suivante :

```
1 InetAddress thisIp = InetAddress.getLocalHost();
  System.out.println("IP: " + thisIp.getHostAddress());
```

Cependant, la méthode **thisIp.getHostAddress()** nous retourne un tableau *debyte*, très difficilement manipulable. C'est la raison pour laquelle il a été nécessaire de la transformer en *String* afin de pouvoir l'exploiter pleinement.

```
monIP = new Integer[4];
2 StringTokenizer st = new StringTokenizer(thisIp.getHostAddress(), ".");
  int i = 0;
4  while (st.hasMoreTokens()) {
    monIP[i] = Integer.parseInt(st.nextToken());
6    i++;
  }
```

Grâce à cette méthode, il est donc possible de récupérer l'IP sous forme de tableau *d'integer*.

Les fonction **ipTabToIpString**, et **ipTabToIpStringWithCustomizedAddress**, prenant en paramètre un tableau d'integer, nous permette donc de compléter le processus, en transformant ce tableau *d'integer* sous forme d'un *String*.

Analyse des ports

Une fois la liste des adresses IPs construites, il est à présent possible d'en sélectionner une, afin de la mettre en surbrillance. Une fois sélectionnée, il suffit de lancer le scan des ports de l'adresse IP choisie. Le scan se déroule de la manière suivante :

```
1  final ExecutorService es = Executors.newFixedThreadPool(20);
   DefaultListModel dlmPorts = new DefaultListModel();
3  final List<Future<Boolean>> futures = new ArrayList<>();

   for (int port = 1; port <= 200; port++) {
       futures.add(portIsOpen(es, ip, port, timeout));
7       dlmPorts.addElement("Port n " + port + " : " + portIsOpen(es, ip, port, timeout)
           .get());
   }
9  System.out.println("Finboucle");
   es.shutdown();
11  int openPorts = 0;
   for (final Future<Boolean> f : futures) {
13       if (f.get()) {
           openPorts++;
15       }
   }
```

Comme pour le sniffer on définit un modèle, que l'on va remplir. Ce modèle correspondra donc à la disponibilité des ports de l'adresse choisie. Comme écrit ci-dessus, on instancie une liste de booléen qui va contenir *true* ou *false* en fonction de l'état du port. Par défaut, nous scanons 200 ports. En scanner plus serait trop long. Nous scanons donc les ports de 0 à 200. À la fin du scan, un message s'affiche et nous résume le traitement.

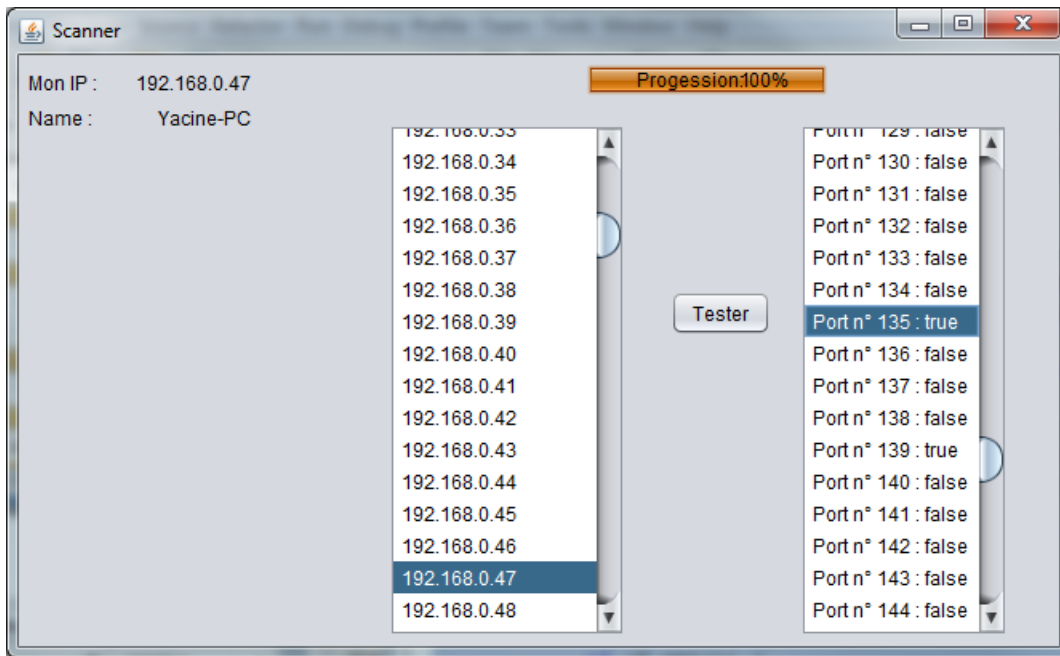


FIGURE 5.6 – Interface Scanner après Scan

6 Améliorations possibles

Afin de mener à bien ce projet, et d'en faire une application réussie et aboutie, l'ajout d'une fonction permettant de "ping" les IPs du même sous réseau, afin que l'utilisateur n'ait plus qu'à choisir quelle IP il veut scanner. L'optimisation de l'interface afin de donner plus de précisions concernant les ports ouverts, comme par exemple l'application qui utilise ce port, depuis quand ... Ce projet continuera à vivre, et nous allons tenter d'apporter des modifications régulièrement afin de pouvoir le rendre optimal.

7 Conclusion

Ce projet nous a permis de consolider nos connaissances en réseau, et d'y découvrir un monde que nous utilisons mais ne connaissons pas. En effet, ce sont les premier pas dans la découverte de failles que nous pourrions exploiter. Les différentes difficultés et les mise en œuvre des interfaces nous ont permis d'être encore plus performant et de consolider nos bases et même les approfondir en Java. De plus, une synergie positive créée par l'union de nos forces de travail nous ont montrés que le travail en binôme est plus que bénéfique, et nous permettra par la suite un meilleur rendement dans nos projets, qu'il soit universitaires ou professionnels.