République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.

Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.
Option : Système d'information

Sujet :

Analyseur du trafic d'un réseau ethernet

Présenté par : MEKFOULDJI Abderezak

Promoteur : BELAZOUG Zahir
Encadreur : BELAZOUG Zahir

Organisme d'accueil : Sonatrach.

Soutenue le : 02/10/2005, devant le jury composé de :

Mr BALA Mahfoud, Professeur ingénieur, USDB	**Président**
Mme MELLAK Assia, Maître assistante, USDB	**Examinateur**
Mr HENTABLI, Ingénieur, USDB	**Examinateur**

- promotion N°35, 2004/2005-

REMERCIEMENTS

Je tiens à remercier mon promoteur monsieur BELAZOUG Zahir pour m'avoir proposé ce sujet, ainsi que pour tous ses précieux conseils, conseils qui m'ont beaucoup servi dans la réalisation de ce travail.

Je remercie vivement monsieur BALA pour avoir présidé le jury, ainsi que les membres du jury, Mme MELLAK et Mr HENTABLI de nous avoir honorés par leur présence.

Mes remerciements s'adressent à toute l'équipe du service réseau de la direction générale de Sonatrach, particulièrement Mr LARBI Sofiane, Mlles MOKRANI et BELABASSI pour leur serviabilité.

J'exprime ma profonde reconnaissance à tous ceux qui m'ont aidé, de près ou de loin, matériellement ou moralement, et qui ont partagé mes peines pour voir naître ce modeste ouvrage. Je citerais Amou Abdelkrim et Abdelaziz, Khalou Mohamed, BENCHIBANE Mohamed, Anis, Toufik, Hamza, Halim et Belkacem.

DEDICACES

Je dédie ce mémoire :

A mes très chers parents qui m'ont beaucoup soutenu et encouragé durant toutes mes années d'études

A ma grand-mère et sa sœur tante Djamila

A mes sœurs : Nawel, Asma et Amira

A mes proches notamment mes oncles, leurs femmes, mes tantes (Farida, Meriem, Dalila, Fatma Zohra et Baya) et leurs maris

A mes cousines et mes cousins

A mes amis : Nacim, Toufik, Walid, Mehdi et Monir, Khalil , Antar, Rafik, Hamza, Hakim, Reda, Rachid, Arilas, Chakib, Amine, ...
(la liste est longue)

A ouled elhouma

A tous mes amis de la promo 2004/2005, de l'université de Blida , de Bab Ezzouar et stagiaires de Sonatrach.

Abderezak M.

RESUME

Dans ce mémoire, j'ai réalisé un aperçu sur le modèle de référence OSI pour faciliter la compréhension de TCP/IP. Ce modèle (établi par l'ISO) définit sept niveaux (couches) différents pour le transfert de données. On a vu en bref le rôle de chaque couche.

TCP/IP est un protocole qui possède des règles de communication entre couches ; il se base sur l'encapsulation de données. Il regroupe certaines couches du modèle OSI dans des couches plus générales. C'est le protocole utilisé dans le réseau local (LAN) que j'ai analysé.

Ce mémoire a abouti à un outil capable d'analyser, de diagnostiquer et de tester les fonctionnalités d'un réseau Ethernet. Cet outil, pour exécuter son travail, a besoin d'obtenir les données qui transitent sur le réseau et les capturer pendant qu'il travaille. Le processus de capture consiste en obtention, en écoute sur le réseau de chaque trame transitant, indépendamment de sa source ou de sa destination.

J'ai présenté une architecture générale du Sniffer (Analyseur) sous les deux plates-formes (Windows et Unix) avant de prévoir les risques liés aux analyseurs. Ensuite, j'ai décrit le pilote de capture des paquets, ses fonctions et sa structure interne. J'ai proposé une architecture de capture à trois niveaux pour la famille des systèmes d'exploitation Win32 de Microsoft et les modules qui la composent. Cette architecture inclut une structure de bas niveau *Winpcap* et des programmes (de niveau utilisateur) qui l'utilisent en interagissant avec les composants du niveau noyau.

La création d'une architecture de capture complète pour Windows avec la performance et les fonctionnalités de BPF pour UNIX a permis le développement de *MekAnal (Analyseur de Mekfouldji)*, une version Windows de *Tcpdump*. Mon analyseur *MekAnal* est capable de :

- Visualiser les informations sur l'utilisation de mon réseau
- Visualiser le contenu des trames
- Localiser et analyser un problème réseau
- Identifier les problèmes de routage et d'adressage
- Editer des statistiques sur l'utilisation du réseau
- Savoir qu'il est nécessaire d'augmenter la bande passante de l'accès Internet

SUMMARY

In this memorandum, I realized a survey about the reference model OSI to facilitate the comprehension of TCP/IP. This model (made by ISO) defines seven different levels (layers) for data transfer. We had a general idea of every layer role's.

TCP/IP is a protocol which has rules of communication between layers, it is based on data encapsulation. It regroups layers from OSI model in more general layers. It's the protocol used in the LAN which we analysed.

This memorandum led to a tool able to analyse, diagnose and test the functionalities of an Ethernet network. This tool, for doing his job, needs to obtain data which transit on the network and capture them while the network is working. The process of capture consists in obtaining and listening, on the network, every transiting frame, independently from his source or destination.

I presented a Sniffer general architecture under the two platforms (Windows and Unix) before forseeing risks due to analysers. Then, I described packet capture driver, its functions and its internal structure. I suggested capture architecture of three levels for Win32 operating systems family of Microsoft and the modules which compose it. This architecture includes a low level structure *Winpcap* and programs (of user level) which use it by interaction with kernel level components.

The creation of complete capture architecture for Windows with the performance and the functionalities of BPF for UNIX allowed the development of *MekAnal (Mekfouldji Analyser's)*, a Windows version of *Tcpdump*. Our analyser *MekAnal* is able to:

- Visualize informations about network use ;
- Visualize frames content ;
- Locate and analyse a network problem ;
- Identify routing and addressing problems ;
- Edit statistics about network's use ;
- Know if it's necessary to increase frequented band of Internet access ;
- Facilitate the work of administrators by helping them to resolve even to anticipate problems.

الملخص

في هذه المذكرة، تطرقت إلى النموذج المرجعي OSI لتسهيل فهم TCP/IP. يعرف هذا النموذج (الذي وضعه ISO) سبع مستويات (طبقات) مختلفة لتحويل المعطيات. ولقد أوردت باختصار دور كل طبقة.

TCP/IP هو بروتوكول ذو قواعد اتصال فيما بين الطبقات، يقوم على كبسلة المعطيات. ويعيد جمع بعض طبقات النموذج OSI في طبقات أعم. إنه البروتوكول المستعمل في الشبكة المحلية (LAN) التي حللتها.

وتوصلت هذه المذكرة إلى أداة قادرة على تحليل وتشخيص واختبار وظائف (إمكانيات) شبكة إيذرنت (Ethernet). حتى تنجز هذه الأداة عملها، يجب أن تحصل على المعطيات التي تمر على الشبكة وتقتنصها أثناء عمل الشبكة. ويقوم سير الاقتناص على الحصول على أو الاستماع إلى كل إطار (هيكل، trame) مار، بصرف النظر عن مصدره أو وجهته.

وقدمت بنية (تقطيعا) للمحلل أو المستنشق (analyseur, Sniffer) تحت الأرضيتين قبل الاحتياط للأخطار المرتبطة بالمحللات. ثم وصفت ربان (سائق) اقتناص الطرود ووظائفه وبنيته الداخلية. وقد اقترحت تقطيع اقتناص ذو ثلاثة مستويات لعائلة أنظمة الاستغلال Win32 من Microsoft والوحدات المشكلة له. ويتضمن هذا التقطيع بنية ذات مستوى منخفض *Winpcap* وبرامج (على مستوى المستعمل) التي تستعملها بالتفاعل مع مكونات مستوى النواة.

مكن إنشاء تقطيع كلي للاقتناص لنظام الاستغلال Windows مع كفاءة وإمكانيات BPF لنظام الاستغلال UNIX من تطوير *MekAnal* (*محلل مقفولجي للشبكات*)، نقل (نسخة) للمحلل *Tcpdump* إلى Windows.
ومحللنا *MekAnal* قادر على :

- إظهار المعلومات المتعلقة باستعمال شبكتنا؛
- إظهار محتوى الأطر (trames)؛
- تحديد و تحليل مشكل ما في الشبكة؛
- تعيين مشاكل الفرز والإرسال؛
- نشر إحصائيات عن استعمال الشبكة؛
- معرفة ضرورة زيادة الشريط الترددي في الدخول للإنترنت؛
- تسهيل عمل الإداريين بمساعدتهم في حل وحتى استباق المشاكل.

SOMMAIRE

Chapitre 3 : TCP/IP

Chapitre 4 : SNIFFERS

CHAPITRE 5 : CONCEPTION ET REALISATION

BIBLIOGRAPHIE

ANNEXES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

Les réseaux informatiques et les technologies de la télécommunication sont utilisés de nos jours dans une grande gamme d'applications. Le succès de l'Internet a amené le réseau à chaque maison et compagnie, et de jour en jour de nouvelles applications et technologies sont créées.

Ainsi, le pouvoir et la complexité des réseaux informatiques grandissent tous les jours. Cela rehausse les possibilités du dernier utilisateur, mais rend plus dur le travail de celui qui doit concevoir, maintenir et établir un réseau sécurisé.

Les réseaux d'aujourd'hui supportent un nombre sans cesse croissant d'applications critiques : ERP, stockage (SAN...) e-commerce, voix sur IP et des connexions point à point (emul, kaza,...) qui réduisent la bande passante, toutes transmises par le protocole TCP/ IP. Parallèlement, les budgets consacrés aux technologies de l'information sont de plus en plus restreints et les ressources disponibles dans le cadre de la maintenance et de la gestion des réseaux sont de moins en moins importantes. Aujourd'hui plus que jamais, les ingénieurs ont besoin d'une solution en matière d'analyse réseau qui leur propose une vision claire et une expertise innovante, dynamique et facile à interpréter.

La solution doit être dans des outils capables d'analyser, de diagnostiquer et de tester les fonctionnalités et la sécurité de réseaux. Ces outils pour exécuter leur travail, ont besoin d'obtenir les données qui transitent habituellement sur un réseau, les capturer pendant que le réseau travaille. Le processus de capture consiste en obtention, en écoute sur le réseau de chaque trame transitant, indépendamment de sa source ou de sa destination.

Le grand nombre de techniques de transmission et protocoles de communication complique cette tâche. De plus, la performance est très importante pour capturer à partir des réseaux rapides à toute vitesse sans perdre des données.

Il y a deux méthodes principales de capture : la première est basée sur l'usage de matériel dédié tandis que la seconde fait usage du matériel d'un PC normal ou d'un poste de travail connecté au canal de communication. Dans cette seconde méthode, la carte réseau de l'ordinateur est utilisée pour obtenir les trames à partir du réseau et le logiciel continue un grand montant du processus de capture.

Habituellement, la solution du logiciel a de plus basses performances, en particulier sur les machines lentes mais c'est le meilleur marché, plus facile à modifier et à améliorer. Pour cette raison, il est largement adopté dans les architectures du réseau les plus utilisées où on n'a pas eu besoin des performances du matériel dédié.

Donc, quelle est l'architecture logicielle la plus adéquate pour analyser le trafic d'un réseau ethernet (cas du Direction Informatique ex CTI de Sonatrach)?

Je propose dans ce mémoire une architecture de capture à trois niveaux pour la famille des systèmes d'exploitation Win32 de Microsoft. Cette architecture inclut une structure de bas niveau Winpcap, cela ajoute aux systèmes d'exploitation Win32 la capacité de capturer efficacement les données de la plupart des familles des réseaux utilisées et des programmes (de notre analyseur MekAnal) ; ceux-ci utilisent la winpcap pour décharger et analyser la circulation du réseau.

Le travail prend l'inspiration du Berkeley Paquet Filtre (BPF), développé par S. McCanne et V. Jacobson à l'Université de Californie. BPF est un composant connu du noyau utilisé comme pilote de capture dans beaucoup de systèmes UNIX pour son efficacité, sa facilité d'implémentation et son interface puissante de niveau utilisateur, la bibliothèque libpcap. L'outil le plus connu qui exploite cette bibliothèque est le célèbre Tcpdump, largement utilisé comme renifleur «Sniffer» indépendant de la plate-forme. Bien que presque chaque plate-forme UNIX a sa propre implémentation de Tcpdump, une version pour Windows n'a jamais été créée à cause de l'absence d'une structure de capture adéquate. Une telle structure est tout à fait difficile de l'implémenter sous Windows, parce qu'elle a besoin d'une interaction profonde avec la portion réseau du noyau ; ceci est plus difficile dans le monde Microsoft où le code source du noyau n'est pas disponible.

Ce travail provient du besoin croissant de haute performance et les outils de capture du réseau croisés plate-forme. Il veut remédier au manque d'une infrastructure de capture libre, complète, facile à utiliser pour les systèmes d'exploitation de la famille Win32.

Les buts principaux de ce travail sont :

- D'abord la création d'une architecture de capture complète pour Windows avec la performance et les fonctionnalités de BPF pour UNIX.
- Ensuite le développement d'une version Windows performante et complète de l'outil d'analyse réseau Tcpdump.

Le premier but vise à obtenir une API de capture système indépendante, capable de travailler sous tous les systèmes Windows et sous UNIX. Cette API devrait être puissante et facile à utiliser, autorisant à développer des outils réseau portatifs sans se soucier des détails de niveau bas. De plus, cela devrait aider le port vers les plates-formes Windows de la pléthore d'applications réseau utiles d'UNIX basée sur libpcap.

Le deuxième but veut tester les fonctionnalités et la performance de cette API de capture avec un outil vrai et exigeant d'analyse du réseau. Mon port permet à un administrateur du réseau d'utiliser le même outil sur toutes les plates-formes les plus répandues.

Ce mémoire est organisé en cinq chapitres précédés d'une introduction générale présentant les sujets discutés par la suite et finis par une conclusion générale. Le paragraphe qui suit donne une description concise de chacun de ces chapitres :

• Le chapitre 1 est une introduction au travail comprenant des notions de base en réseaux ainsi que leur classification.

- Le chapitre 2 décrit le modèle de référence OSI et ses avantages.

- Le chapitre 3 définit les couches de la pile de protocoles TCP/IP et ses éléments de base.

- Le chapitre 4 montre l'architecture générale du Sniffer sous les deux plates-formes avec des exemples et une analyse de risques.

- Le chapitre 5 décrit le pilote de capture du paquet, ses fonctions et sa structure interne comme il contient les renseignements sur l'architecture de capture et les modules qui la composent. Ce chapitre présente également la méthode de conception ,le langage de programmation choisi et la manière de réalisation de MekAnal (Mekfouldji Analyser's).

Chapitre 1

Notions de base en réseau

Chapitre 1 : NOTIONS DE BASE EN RESEAU

I. INTRODUCTION

Un réseau est un ensemble d'entités communicant entre elles. Je vais m'intéresser dans le cadre de ce travail à ce que l'on nomme des réseaux de données ou réseaux informatiques. Ces réseaux sont apparus suite à une demande des entreprises qui recherchaient une méthode pour éviter la duplication des imprimantes et une simplification des communications de données entre des équipements informatiques. Un réseau de données est donc un ensemble d'entités informatiques communicant ensemble. La première classification de réseau que je vais faire s'établit sur la base des distances entre les entités.

II. LAN (Local Area Network), ou réseau local

C'est la forme la plus simple de réseau informatique. Ce n'est rien de plus qu'un ensemble d'équipements situés sur un même site et connectés à un réseau. Les LAN présentent les caractéristiques suivantes :

- Le support des communications est un média physique à large bande passante (par exemple la paire torsadée, le câble coaxial ou la fibre optique) ;
- Couvrent une région géographique limitée (quelques centaines de mètres au maximum) ;
- Permettent un accès multiple aux média de communication

Comme exemples de réseaux locaux on peut citer : le réseau Ethernet (10 Mbps), Fast Ethernet (100 Mbps ou plus), le réseau Token Ring (16 Mbps), FDDI, etc....

III. MAN (Metropolitan Area Network), ou réseau métropolitain

Un réseau métropolitain interconnecte plusieurs lieux situés dans une même ville, par exemple les différents sites d'une université ou d'une administration, chacun possédant son propre réseau local. Il est destiné à couvrir de très grands périmètres qui sont fédérateurs de réseaux locaux.

Un MAN est un système de communication offrant des services de transfert très divers comme les données à haut débit, la voix, ou la vidéo, entre des équipements situés dans un environnement géographique étendu dont la dimension est celle d'une métropole. Ces réseaux peuvent également relier un certain nombre de bâtiments se trouvant sur un même campus; c'est dans ce cas un réseau métropolitain privé.

Les réseaux métropolitains permettent l'interconnexion de réseaux locaux situés à moins d'une centaine de kilomètres les uns des autres. La vitesse de transport doit atteindre un minimum de 100 Mbit/s pour permettre ce type d'interconnexion. Ces réseaux métropolitains sont plus ou moins adaptés au multimédia ; certains ne prennent en charge que les données, d'autres essayent d'intégrer voix et données. [PUJ 98]

IV. WAN (Wide Area Network), ou réseau étendu

Un réseau étendu permet de communiquer à l'échelle d'un pays, ou de la planète entière, les infrastructures physiques pouvant être terrestres ou spatiales à l'aide de satellites de télécommunications. Exemple de WAN : *Internet*.

Les réseaux WAN :

- Couvrent une vaste zone géographique ;
- Permettent l'accès par des interfaces séries plus lentes ;
- Assurent une connectivité pouvant être continue ou intermittente ;
- Relient des unités dispersées à une échelle planétaire.

Lorsque des LAN ou des MAN sont trop éparpillés géographiquement pour qu'on puisse les relier à la vitesse maximale d'un LAN, il est temps de bâtir un WAN. Les réseaux sont constitués d'un ensemble de LAN disséminés géographiquement, et reliés à l'aide de liaisons et de routeurs.

La bande passante disponible pour la transmission à travers un WAN est souvent limitée par celle de la liaison d'accès au réseau (par exemple dans le cas d'une liaison téléphonique entre 56 Kbps et quelques Mbps en utilisant la technologie ADSL). La bande passante des liaisons de transfert dans le WAN dépend de la technologie utilisée (liaisons spécialisées, ATM, etc...) mais elle est souvent bien supérieure à celle des liaisons d'accès.
[HAY 02]

V. CONCLUSION

Contrairement aux LAN, les WAN impliquent toujours des routeurs. Puisque la majeure partie du trafic d'un WAN se situe dans les LAN qui le constituent, les routeurs sont investis d'une mission importante : le contrôle de trafic. Les routeurs doivent être paramétrés avec des informations appelées routes (nous en saurons plus au chapitre consacré à TCP/IP) qui indiquent à ceux-ci comment acheminer des données entre réseaux. Pour faciliter la compréhension de TCP/IP, on va passer par le modèle de référence OSI.

Chapitre 2

Le modèle OSI

CHAPITRE 2 : LE MODELE OSI

I. INTRODUCTION

La première évolution des réseaux informatiques a été des plus anarchiques, chaque constructeur développant presque sa propre technologie, d'où, comme résultat, une quasi-impossibilité de connecter différents réseaux entre eux. Pour pallier à cela, l'Institut de normalisation -ISO- (International Standardization Organization) décida de mettre en place un modèle de référence théorique décrivant le fonctionnement des communications réseaux, le modèle OSI (Open System Interconnexion), à partir des structures réseaux prédominantes de l'époque que sont DecNet et SNA. Le but de ce modèle est d'analyser la communication en découpant les différentes étapes en sept couches, chacune de ces couches remplissant une tâche bien spécifique.

II. DESCRIPTION DES COUCHES DU MODELE

À chaque couche du modèle OSI correspond un ensemble préétabli de fonctions qui doivent être effectuées pour qu'il y ait communication. Une description de chacune de ces couches est donnée ci-dessous :

Tableau 1 : Les 7 couches du modèle OSI

N°	Nom	**Description**
1	Physique	Envoi sur le média physique
2	Liaison de données	Préparation de l'envoi sur le média
3	Réseau	Sélection du chemin
4	Transport	Qualité de transmission
5	Session	Contrôle du dialogue
6	Présentation	Gestion de la syntaxe
7	Application	Communication avec les logiciels

Couche 1 : *Couche physique*

La couche 1, ou couche *physique* (Physical layer), est celle qui est située au plus bas niveau du modèle OSI. Cette couche transmet le flux de bits bruts non structuré, par l'intermédiaire d'un support physique (par exemple le câble réseau). La couche *physique* définit les interfaces électriques, optiques, mécaniques et fonctionnelles avec le câble. En outre, elle gère les signaux qui transmettent les données produites par toutes les couches supérieures.

Cette couche établit la façon dont le câble est connecté à la carte réseau. Par exemple, elle définit le nombre de broches du connecteur ainsi que la fonction de chacune. Elle détermine également la méthode de transmission utilisée pour envoyer des données sur le câble réseau.

La couche *physique* est responsable de la transmission des bits entres deux ordinateurs. Les bits n'ont pas de signification en eux-mêmes à ce niveau. Cette couche définit l'encodage des données et la synchronisation des bits : quand un hôte émet un bit à '1', la couche *physique* assure qu'il sera reçu comme un '1' et non pas comme un '0'. Elle définit également la durée de chaque bit, ainsi que la façon dont il est traduit en impulsion électrique ou optique pour le câble réseau. Pour nous souvenir facilement des fonctions de la couche 1, pensons aux signaux et aux médias.

Couche 2 : ***Couche liaison de données***

La couche 2, ou couche *liaison* de données (Data Link layer), envoie des trames de données depuis la couche *réseau* vers la couche *physique*. Sur l'ordinateur récepteur, la couche *liaison* regroupe dans des trames les bits bruts provenant de la couche *physique*. Une trame est une structure logique et organisée dans laquelle on peut placer des données.

Cette couche est responsable du transfert sans erreur des trames entre deux ordinateurs par l'intermédiaire de la couche *physique*, permettant ainsi à la couche *réseau* d'assumer une transmission théoriquement sans erreur sur le réseau.

En général, lorsque la couche *liaison* envoie une trame, elle attend un accusé de réception de la part du destinataire. La couche *liaison* du récepteur détecte tous les problèmes qui ont pu survenir avec la trame lors de la transmission. Les trames qui n'ont pas fait l'objet d'un accusé de réception ou qui ont été altérées lors de la transmission sont réexpédiées. Pour nous souvenir facilement des fonctions de la couche 2, pensons à l'attribution des noms, aux trames et aux adresses MAC.

Couche 3 : ***Couche réseau***

La couche 3, ou couche *réseau* (Network layer), se charge de l'adressage des messages et de la traduction des adresses et des noms logiques en adresses physiques. Cette couche définit également le routage des messages entre l'ordinateur source et l'ordinateur cible : elle détermine le chemin en fonction de l'état du réseau, de la priorité du service et d'un certain nombre d'autres facteurs. En outre, cette couche gère les problèmes de trafic sur le réseau, tels que la commutation de paquets, le routage, le contrôle de l'encombrement des données, etc....

Si la carte réseau du routeur ne peut pas transmettre un paquet de données de la même taille que celui envoyé par l'ordinateur source, la couche *réseau* du routeur découpe les données en unités plus petites. Sur l'ordinateur de destination, la couche *réseau* ré-assemble les données. Pour nous souvenir facilement des fonctions de la couche 3, pensons à la sélection de trajet, à la commutation, à l'adressage et au routage.

Couche 4 : ***Couche de transport***

La couche 4, ou couche *transport* (Transport layer), fournit un niveau de connexion supplémentaire au-dessous de la couche *session*. Elle s'assure que les paquets ont été reçus sans erreur, dans l'ordre, et sans perte ni duplication de données. Cette couche réorganise les messages : elle découpe les messages longs en plusieurs paquets et regroupe les petits paquets en un seul, de manière à les transmettre plus efficacement sur le réseau. Sur l'ordinateur récepteur, la couche *transport* extrait les messages des paquets, ré-assemble les messages d'origine et envoie en principe un accusé de réception.

En fournissant un service fiable, la couche de *transport* procure des mécanismes permettant l'établissement, la maintenance et la terminaison ordonnée des circuits virtuels, la détection et la reprise sur incident ainsi que le contrôle du flux d'information, afin d'empêcher qu'un système n'en surcharge de données un autre système. Pour nous souvenir facilement des fonctions de la couche 4, pensons à la qualité de service et à la fiabilité.

Couche 5 : ***Couche session***

La couche 5, ou couche *session* (Session layer) ouvre, gère et ferme les sessions entre les applications. Une session est un dialogue entre deux entités de présentation ou plus. La couche *session* fournit ses services à la couche de présentation.

En plus d'assurer la régulation de base des conversations (sessions), la couche *session* synchronise des tâches utilisateur en plaçant des points de contrôle dans le flux de données. De cette façon, si le réseau tombe en panne, seules les données venant après le dernier point de contrôle seront retransmises. Cette couche met également en œuvre le contrôle du dialogue entre les processus communicants : elle décide du côté qui transmet, du moment et de la durée de la transmission, etc... Pour nous souvenir facilement des fonctions de la couche 5, pensons aux dialogues et aux conversations.

Couche 6 : ***Couche de présentation***

La couche 6, couche *présentation* (Presentation layer), détermine le format utilisé pour échanger des données entre les ordinateurs du réseau. Elle peut être considérée comme le traducteur du réseau. Sur l'ordinateur émetteur, cette couche traduit les données de façon à ce que le format issu de la couche application soit remplacé par un format intermédiaire et communément reconnu. Sur l'ordinateur récepteur, cette couche convertit le format intermédiaire en un format utilisable par la couche *application* de cet ordinateur.

La couche *présentation* se charge de la conversion des protocoles, de la traduction et de l'encodage des données, de la conversion du jeu de caractère ainsi que de l'exécution de commandes graphiques. La couche *présentation* s'occupe également de la compression des données, qui permet de réduire le nombre de bits à transmettre. Pour nous souvenir facilement des fonctions de la couche 6, pensons au chiffrage, à la représentation des données.

Couche 7 : *La couche application*

La couche *application* est la couche OSI la plus proche de l'utilisateur ; elle fournit des services réseau aux applications de l'utilisateur. Elle se distingue des autres couches du fait qu'elle ne fournit des services qu'aux processus d'application à l'extérieur du modèle OSI et point aux autres couches OSI. Exemples de processus d'application : tableurs, traitement de texte et logiciels de terminaux bancaires.

La couche *application* identifie et détermine la disponibilité des partenaires de communication voulus, synchronise les applications coopératives et établit une entente sur les procédures de reprise sur incident et de contrôle de l'intégrité des données. Elle détermine également si les ressources sont suffisantes pour la communication prévue. Pour nous souvenir facilement des fonctions de la couche 7, pensons aux navigateurs.

III. LES AVANTAGES DE CE MODELE

Les avantages de ce modèle sont :

- Une division de la communication réseau en éléments plus petits et plus simple pour une meilleure compréhension.
- L'uniformisation des éléments afin de permettre le développement multi-constructeur.
- La possibilité de modifier un aspect de la communication réseau sans modifier le reste (Exemple : un nouveau média). [WLA 01]

IV. L'ENCAPSULATION

Pour communiquer entre les couches et entre les hôtes d'un réseau, OSI a recourt au principe d'encapsulation.

Lorsque deux hôtes communiquent, on parle de communication d'égal à égal ; c'est-à-dire que la couche *n* de la source communique avec la couche *n* du destinataire. Lorsqu'une couche de la source reçoit des données, elle encapsule ces dernières avec ses informations puis les passe à la couche inférieure.

Le mécanisme inverse a lieu au niveau du destinataire où une couche réceptionne les données de la couche inférieure, enlève les informations la concernant puis transmet les informations restantes à la couche supérieure. Les données transitant à la couche *n* de la source sont donc les mêmes que les données transitant à la couche *n* du destinataire. Pour identifier les données lors de leur passage au travers d'une couche, l'appellation «*Unité de données de protocole (PDU)*» est utilisée. [WLA 01]

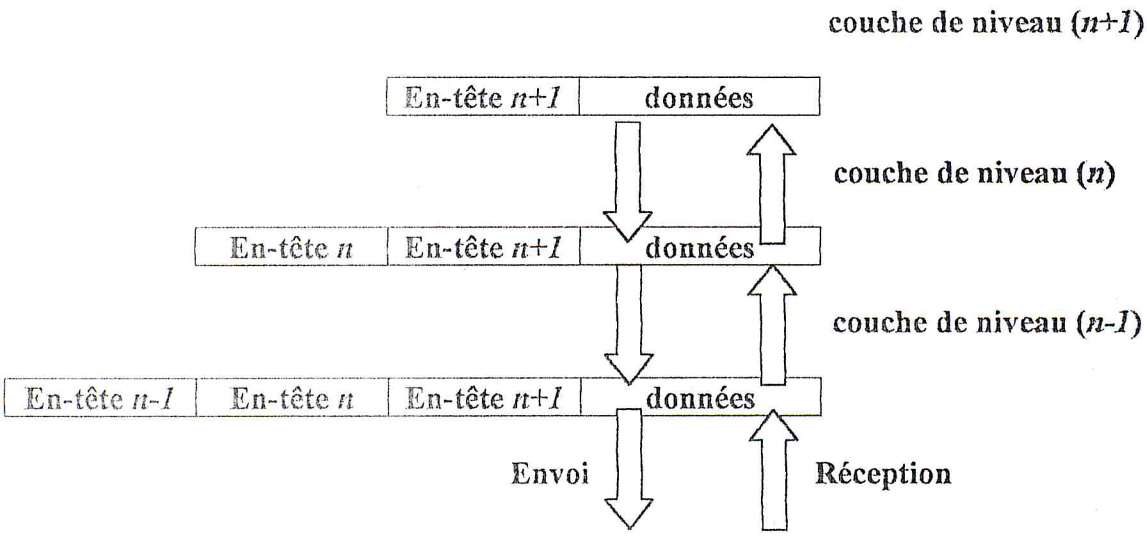


Figure 2.1 : Encapsulation de données.

V. LES PDU DES DIFFERENTES COUCHES

Le modèle OSI comprend deux couches dites « matérielles » en opposition aux couches logicielles. La couche 1 -ou couche *physique*- englobe les médias, les signaux ainsi que les bits se déplaçant sur diverses topologies.

La couche 2 -ou couche *liaison*- a pour fonction de combler tous les manques de la couche *physique* afin de permettre la communication *réseau*. On va voir les PDU des différentes couches (Tableau 2) :

Tableau 2 : Les équivalences de PDU

Couche	**Désignation**
1	Bits
2	Trame
3	Paquets
4	Segment
5	Données
6	Données
7	Données

V.1. Bits (Couche 1) :

La transmission des bits se fait sur un support physique (Ethernet, Token Ring, X.21, X.25, FDDI). La composante de base de l'information dans les réseaux est le bit. Dans le cas d'un signal électrique, un bit correspond à une impulsion signifiant 0 ou 1.Il existe différents facteurs pouvant affecter le signal et de ce fait les bits transportés sur le média :

- La propagation de signaux réseau :

Le terme de propagation fait référence au temps que met un bit, c'est-à-dire une impulsion, à se déplacer dans le média. Il est impératif que la propagation soit homogène dans le réseau.

- L'atténuation du signal réseau :

Perte de la force du signal. Ce problème est limitable par un bon choix des médias réseaux utilisés.

- La réflexion réseau :

Retour d'énergie causé par le passage des impulsions dans le média. Si ce retour est trop fort, il peut perturber le signal des impulsions suivantes. Le système binaire -et donc à 2 états- peut être perturbé par ces énergies supplémentaires se déplaçant dans le média.

- Le bruit :

Ajout indésirable à un signal. Des sources d'énergie situées à proximité du média fournissent un supplément d'énergie venant perturber le signal.

- *Diaphonie* : bruit ajouté au signal d'origine d'un conducteur par l'action du champ magnétique provenant d'un autre conducteur
- *Paradiaphonie* : diaphonie causée par un conducteur interne au câble

Le bruit peut être causé par des sources d'alimentations externes, des variations thermiques, des interférences électromagnétiques ou encore des interférences de radio fréquences.

- La dispersion :

Etalement des impulsions dans le temps. Si la dispersion est trop forte, le signal d'un bit peut recouper le signal du précédent ou du suivant. La durée d'une impulsion est fixe, la dispersion correspond à une modification de cette durée au fur et à mesure que le signal se propage dans le média.

- La gigue :

Les systèmes numériques sont synchronisés, tout est réglé par des impulsions d'horloge. Si les horloges de la source et du destinataire ne sont pas synchronisées, on obtient alors une *gigue de synchronisation*.

- La latence :

Retard de transmission. Principalement dû au déplacement du signal dans le média et à la présence de composants électroniques entre la source et la destination.

- La collision :

Se produit lorsque deux ordinateurs utilisant le même segment de réseau émettent en même temps. Les impulsions se mélangent, détruisant alors les données.

Dès qu'un bit accède au média, il est sujet à tous ces paramètres pouvant perturber la transmission. Dans la mesure où le but n'est pas de transmettre un bit mais des quantités gigantesques (parfois 1 milliard de bits à la seconde), ces paramètres ne sont pas à négliger car le moindre défaut peut avoir des conséquences importantes sur la qualité de la transmission. Je vais donc m'intéresser ici aux méthodes de transmission de bits de façon brute entre l'émetteur et le récepteur.

La transmission de plusieurs bits peut s'effectuer :

1. En série : les bits sont envoyés les uns derrière les autres de manière synchrone ou asynchrone :

   - Dans le mode synchrone l'émetteur et le récepteur se mettent d'accord sur une base de temps (un top d'horloge) qui se répète régulièrement durant tout l'échange. À chaque top d'horloge (ou k tops d'horloge k entiers fixés définitivement) un bit est envoyé et le récepteur saura ainsi quand arrivent les bits.

   - Dans le mode asynchrone, il n'y a pas de négociation préalable mais chaque caractère envoyé est précédé d'un bit de start et immédiatement suivi d'un bit de stop. Ces deux bits spéciaux servent à caler l'horloge du récepteur pour qu'il échantillonne le signal qu'il reçoit afin d'y décoder les bits qu'il transmet.

2. En parallèle : Les bits d'un même caractère sont envoyés en même temps chacun sur un fil distinct, mais cela pose des problèmes de synchronisation et chaque fil n'est utilisé que sur de courtes distances (bus par exemple).

V.2. Trames (Couche 2) :

Les normes IEEE sont actuellement les normes prédominantes. Selon l'IEEE, on divise la partie matérielle du modèle OSI en 2 parties :

- La norme LLC 802.2, ne dépendant pas de la technologie du média utilisé.
- Les éléments spécifiques, tributaires de la technologie qui intègrent la couche physique du modèle OSI.

De plus cette division sépare la couche *liaison* de données en 2 parties :

- Media Access Control (MAC) : transmission vers le bas jusqu'au média.
- Logical Link Control (LLC) : transmission vers le haut jusqu'à la couche *réseau.*

La norme IEEE définit ses propres PDU, ses interfaces, et ses protocoles qui chevauchent les couches 1 et 2 du modèle OSI. La principale raison de cette différence est le fait qu'OSI est un modèle convenu et que l'IEEE a écrit par la suite ses normes afin de faire face à différents problèmes réseaux.

L'autre différence à noter est au niveau de la carte réseau. En effet, une carte réseau contenant l'adresse matérielle (MAC) de couche 2 devrait être classée dans les équipements de couche 2. Cependant, elle comprend également un *émetteur récepteur* de couche 1. Dès lors, il serait exact de dire qu'elle appartient aux couches 1 et 2 du modèle OSI.

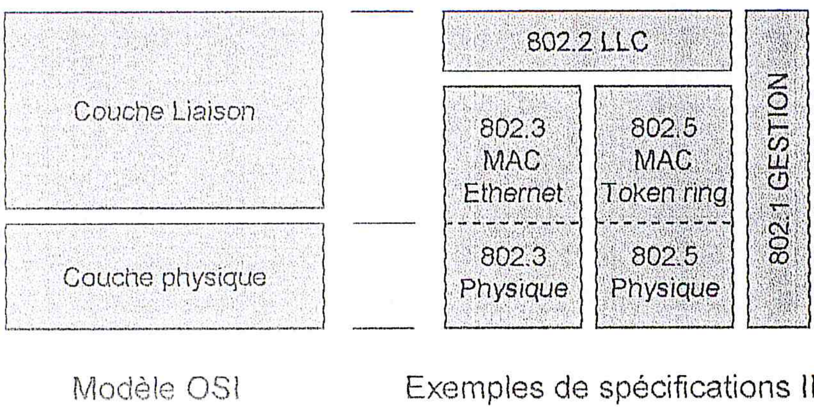


Figure 2.2 : Différences entre le modèle OSI et les spécifications de l'IEEE.

V.2.1. Les adresses MAC :

Une adresse MAC est une adresse matérielle, c'est-à-dire une adresse unique non modifiable par l'administrateur et stockée sur une mémoire morte (ROM) de la carte réseau. Les adresses MAC comportent 48 bits et sont exprimées sous la forme de 12 chiffres hexadécimaux :

- 6 chiffres sont administrés par l'IEEE et identifient le fabricant de la carte.
- 6 chiffres forment le numéro de série de la carte.

On peut les représenter de deux manières différentes : par groupe de 4 chiffres séparés par des points ou par groupe de 2 chiffres séparés par des tirets.

Exemple : *0000.0c12.3456* ou *00-00-0c-12-34-56*

Les LAN de type Ethernet et 802.3 sont des réseaux dits de *broadcast*, ce qui signifie que tous les hôtes voient toutes les trames. L'adressage MAC est donc un élément important afin de pouvoir déterminer les émetteurs et les destinataires en lisant les trames. Le principal défaut de l'adressage MAC est qu'il est non hiérarchique : on ne peut pas faire de classement d'adresses.

V.2.2. Le verrouillage de trames :

Une Trame est le PDU de couche 2. Le verrouillage de trame est un concept permettant de récupérer les informations essentielles normalement impossibles à obtenir avec les trains binaires comme par exemple :

- Quels sont les ordinateurs en communication ?
- Début et fin de la communication ?
- Qui est autorisé à parler ?
- Quelles sont les erreurs survenues ?

Une trame est donc comme un tableau encadrant les bits et ajoutant les informations nécessaires à la compréhension de ces bits par les hôtes.

V.2.3. Structure de trame générique :

A	B	C	D	E	F
Champ de début de trame	Champ d'adresse	Champ de type/ longueur	Champ de données	Champ FCS	Champ de fin de trame

Figure 2.3 : Les champs d'une trame générique.

- Champ de début de trames : annonce l'arrivée d'une trame
- Champ d'adresse : contient les informations d'identification (source et destination)
- Champ de longueur/type : dépend de la technologie, il peut indiquer la longueur de la trame, le protocole de couche 3 ou encore rien du tout

- Champ de données : contient les informations à transmettre, parfois accompagnées d'octets de remplissage pour que les trames aient une longueur minimale à des fins de synchronisation

- Champ de FCS : permet de détecter les erreurs, c'est une séquence de contrôle permettant au destinataire de vérifier le bon état de la trame. Exemple : le *CRC* ou Code de Redondance Cyclique : calculs polynomiaux sur les données.

- Champ de fin de trame : permet d'annoncer la fin de la trame

V.2.4. Le contrôle de lien logique (LLC) :

La sous couche LLC a été créée afin de permettre à une partie de la couche *liaison* de données de fonctionner indépendamment des technologies existantes. Cela assure la polyvalence des services fournis aux protocoles de couche réseau situés en amont de cette couche tout en communiquant avec les différentes technologies utilisées pour véhiculer les informations entre la source et la destination.

Le rôle de cette sous-couche est de réceptionner le paquet IP et d'y ajouter les informations de contrôle pour en faciliter l'acheminement jusqu'à la destination. Elle ajoute deux éléments d'adressage décrits dans la spécification LLC 802.2 :

- Le point d'accès DSAP : point d'accès SAP du nœud réseau désigné dans le champ de destination du paquet.

- Le point d'accès SSAP : point d'accès au service du nœud réseau désigné dans le champ source du paquet.

La sous couche LLC gère les communications entre les dispositifs sur une seule liaison réseau. La norme IEEE 802.2 définit un certain nombre de champs dans les trames, qui permettent à plusieurs protocoles de couche supérieure de partager une liaison physique de données. Ce paquet IP encapsulé se rend ensuite à la sous-couche MAC où la technologie utilisée effectue une encapsulation supplémentaire.

V.2.5. La sous-couche MAC :

La sous-couche MAC concerne les protocoles que doit suivre un hôte pour accéder au média. Dans un environnement de média partagé, il permet de déterminer quel ordinateur peut parler. On distingue deux types de protocoles MAC :

- Déterministe : chacun à son tour, Exemple *: Token Ring*

- Non déterministe : premier arrivé premier servi, Exemple *: Ethernet*

V.2.6. Structure de trame Ethernet et IEEE 802.3 :

Ethernet et IEEE 802.3 définissent des technologies semblables :

- Utilisation de CSMA/CD pour l'accès au média.

- Concept de réseaux de *broadcast*.

Il existe cependant quelques différences subtiles. En effet, Ethernet offre des services correspondants aux couches 1 et 2 du modèle OSI alors que IEEE 802.3 définit la couche 1 ainsi que la partie MAC de la couche 2.

?	1	6	6	2	46-1500	4
Préambule	Délimiteur de début de trame	Adresse de destination	Adresse d'origine	Type	Données	FCS

Trame Ethernet

?	1	6	6	2	64-1500	4
Préambule	Délimiteur de début de trame	Adresse de destination	Adresse d'origine	Longueur	Données	FCS

Trame IEEE 802.3

Figure 2.4 : Structure de trames Ethernet et IEEE 802.3.

- Préambule : composé de 1 et de 0 en alternance, annonce si la trame est de type Ethernet ou 802.3 ;
- Début de trame IEEE 802.3 : l'octet séparateur se termine par deux bits à 1 consécutifs servant à synchroniser les portions de réception des trames de toutes les stations ;
- Champ d'adresse d'origine : toujours de type *unicast* ;
- Champ d'adresse de destination : peut être de type *unicast*, *multicast* ou *broadcast* ;
- Type (Ethernet) : précise le type de protocole de couche supérieure qui reçoit les données ;
- Longueur (802.3) : indique le nombre d'octets de données qui suit le champ ;
- Données :
  - Ethernet : une fois le traitement de couche 1 et 2 terminé, les données sont transmises au protocole de la couche supérieure indiqué dans le champ type. On peut avoir recours à des octets de remplissage s'il n'y a pas assez de données pour remplir les 64 octets minimaux de la trame.
  - IEEE 802.3 : une fois le traitement de couche 1 et 2 terminé, les données sont transmises au protocole de la couche supérieure indiqué dans le champ données de la trame. On peut aussi avoir ici recours à du remplissage ;
- FCS : Séquence de contrôle de trame. Cette séquence contient un code de redondance cyclique de 4 octets permettant à l'unité réceptrice de vérifier l'intégrité des données.

V.2.7. MAC Ethernet :

Ethernet et 802.3 utilisent un principe d'accès au média non déterministe : CSMA/CD (Carrier Sense Multiple Access / Collision Detect). Les hôtes se partagent le média ; si l'un d'eux désire émettre, il vérifie au préalable que personne n'est en train de le faire puis commence a émettre (CSMA).

Si cependant deux hôtes émettent en même temps, il se produit alors une collision. La première station qui détecte une collision envoie alors un signal de bourrage, se traduisant par un arrêt d'émission de tous les hôtes : les paquets concernés sont alors détruits. Chaque hôte calcule alors une valeur aléatoire définissant la durée avant de recommencer à émettre, puis le mécanisme de CSMA se remet en fonction.

[WLA 01]

V.3. Paquets (Couche 3) :

Les informations provenant de la couche 4 sont encapsulés dans le PDU de couche 3 : **le paquet**

Les paquets sont des tronçons de données. Ils doivent donc comporter des informations sur leur provenance, leur destination et sur la façon dont ils devront être examinés pour savoir s'ils comportent ou non des erreurs et sur la façon dont ils seront regroupés pour reconstituer le fichier ou message originel. Pour englober toutes ces informations, un paquet comporte trois parties : un en-tête, les données mêmes et une queue.

Figure 2.5 : **Structure d'un paquet.**

Comme son nom l'indique, l'en-tête précède le tronçon des données du paquet et inclut les adresses réseau de la source et de la destination ainsi que des informations de contrôle et de synchronisation pour garantir que l'ensemble soit transmis de façon appropriée. La section des données contient le tronçon des informations à transmettre. Selon le réseau, elle aura une taille de 512 octets à 4 ko. La queue renferme, entre autres choses que nous verrons plus loin, des informations de vérification d'erreurs, qui permettent à l'ordinateur destinataire de vérifier que les données lui sont bien parvenues intactes. [WOO 00]

VI. CONCLUSION

Pour s'y retrouver plus facilement dans l'ensemble des protocoles, l'International Standard Organization (ISO) a défini un modèle de base, le modèle OSI. Ce modèle définit sept **niveaux** différents pour le transfert de données. Ces niveaux sont également appelés **couches**.

Le premier niveau, la couche *physique*, gère les connections matérielles. Le deuxième niveau, la couche *liaison* de données, s'occupe du codage, de l'adressage, et de la transmission des informations. Vient ensuite le troisième niveau, la couche *réseau*, qui détermine les routes de transport et qui s'occupe du traitement et du transfert de messages. Le quatrième niveau, la couche *transport*, gère la remise correcte des informations. Le cinquième niveau, la couche *session*, s'occupe de l'établissement, de la gestion et de la coordination des communications. Le sixième niveau, la couche *présentation*, s'occupe de la mise en forme des textes et des conventions d'affichage. Le septième niveau, la couche *application*, gère le transfert des informations entre programmes.

A chacun de ces niveaux, on encapsule un *en-tête* et une fin de trame qui comporte les informations nécessaires en suivant les règles définies par le protocole utilisé. La dernière trame, celle qu'on obtient après avoir encapsulé les éléments de la couche *physique*, est celle qui sera envoyée sur le réseau. Les protocoles TCP/IP sont parmi les protocoles les plus utilisés et que nous allons voir par la suite.

Chapitre 3

TCP/IP

Chapitre 3 : TCP/IP

I. INTRODUCTION

La forme actuelle de TCP/IP résulte du rôle historique que ce système de protocoles a joué dans le parachèvement de ce qui allait devenir Internet. A l'instar des nombreux développements de ces dernières années, Internet est issu des recherches lancées aux Etats-Unis par le DOD, département de la défense.

A la fin des années 60, les officiels du DOD se rendent compte que les militaires du département de la défense possèdent une grande quantité de matériel informatique très divers, mais ces machines travaillent pour la plupart de manière isolée ou encore en réseaux de taille très modeste avec des protocoles incompatibles entre eux, ceci rendant une interconnexion impossible.

Les autorités militaires se sont alors demandé s'il était possible, pour ces machines aux profils très différents, de traiter des informations mises en commun. Habitués comme ils le sont aux problèmes de sécurité, les responsables de la défense ont immédiatement réalisé qu'un réseau de grande ampleur deviendrait une cible idéale en cas de conflit. La caractéristique principale de ce réseau, s'il devait exister, était d'être non centralisé.

Ses fonctions essentielles ne devaient en aucun cas se trouver en un seul point, ce qui le rendrait trop vulnérable. C'est alors que fut mis en place le projet Arpanet (Advanced Research Projects Agency du DOD), qui allait devenir par la suite le système d'interconnexion de réseau qui régit ce que l'on appelle aujourd'hui l'Internet : TCP/IP.

II. LES QUATRE COUCHES DU TCP/IP

A l'origine développé pour ARPAnet, TCP/IP, est désormais, avec l'émergence d'Internet et de l'inter-réseau dans le royaume du PC et du modèle client-serveur, le protocole à la mode. Il s'agit, en réalité, d'un standard réseau, en partie pour les motifs suivants :

- Il est robuste.
- Il permet la communication entre différents systèmes.
- Il est disponible sur une grande variété de plates-formes informatiques.

Bien sûr, il fournit aussi l'accès à Internet, ce qui intéresse de plus en plus les connexions réseau des entreprises.

II.1. Présentation des couches :

TCP/IP est un modèle comprenant 4 couches :

Tableau 3 : Les 4 couches de TCP/IP

N°	Nom	Description
1	Liens (pilotes et cartes d'interface)	Reprend les couches 1 et 2 du modèle OSI
2	Inter-réseau (IP, ICMP et IGMP)	Sélection du chemin
3	Transport (TCP, UDP)	Qualité de transmission
4	Application (Telnet, FTP, ...)	Couches 5 à 7 du modèle OSI

Chaque couche possède un rôle différent :

- La couche de **liens de données** (appelée parfois la couche **interface réseau**), prend en charge la communication directe avec le réseau. Elle doit comprendre l'architecture de réseau utilisée, telle que Token-Ring ou Ethernet et offrir une interface permettant à la couche internet de communiquer directement avec elle.

- La couche **inter-réseau** (appelée parfois couche **internet**), gère la circulation des paquets à travers le réseau. Le routage des paquets, par exemple, se situe à ce niveau. IP (Internet Protocol), ICMP (Internet Control Message Protocol) et IGMP (Internet Group Management Protocol), forme la couche inter-réseau de la série de protocoles TCP/IP.

- La couche **transport** gère le flux de données entre 2 machines. Elle prend en charge des tâches comme l'envoi vers la couche réseau des données émises par une application en tronçons de tailles appropriées, et l'acquittement des paquets reçus. Elle établit des temporisations pour s'assurer que l'autre extrémité acquitte les paquets qui sont émis, et ainsi de suite. Puisque c'est la couche transport qui prend en charge cet important flux de données, la couche applicative ignore tout de ces détails.

- La couche **applicative** prend en charge les détails de communication d'une application particulière. De très nombreuses implémentations de TCP/IP regroupent les applications : Telnet pour la prise de contrôle à distance – FTP, (File Transfer Protocol), le protocole de transfert de fichiers – SMTP, (Simple Mail Transfer Protocol), pour le courrier électronique – SNMP, (Simple Network Management Protocol), pour l'administration et bien d'autres.

[STE 98]

TCP/IP repose sur deux protocoles, dont il tire son nom :

TCP (Transmission Control Protocol, ou protocole de contrôle de transmission), qui opère sur la couche transport et est chargé de livrer correctement les informations. TCP établit une connexion entre deux machines, il est conçu pour vérifier que tous les paquets envoyés par une machine sont reçus à l'autre bout.

IP ou Protocole Internet, s'occupe avant tout du routage et de la livraison des paquets au travers du protocole IP. Tous les protocoles de la couche transport doivent utiliser IP pour transmettre les données. Le protocole IP comprend des règles sur la manière d'adresser et de diriger les paquets, de fragmenter et d'assembler les paquets, de fournir des informations de sécurité, et d'identifier le type de service utilisé.

La suite TCP/IP comprend en outre le protocole de niveau transport appelé UDP (User Datagram Protocol, ou protocole des datagrammes utilisateur), qui s'appuie aussi sur IP pour le routage des paquets. A la différence de TCP, qui est un transport fiable basé sur la connexion, UDP est un transport non fiable, sans connexion. En d'autres termes, TCP établit une connexion entre l'émetteur et le destinataire avant de transférer les données (en fragments appelés des segments), retransmet les segments lorsqu'il ne reçoit pas d'acquittement de réception de la part du destinataire et peut contrôler le flux d'informations pour garantir que l'ordinateur émetteur ne surcharge pas les tampons, ou zones de réception du destinataire.

À l'opposé, UDP ne comporte aucun moyen d'assurer la livraison. Il fait ce qu'il peut mais demeure «non fiable» puisqu'il n'établit pas de connexion entre émetteur et récepteur avant de transmettre ses informations en éléments appelés des datagrammes. Il n'attend ni n'exige aucun accusé de réception du destinataire. De plus, les datagrammes transmis par UDP étant indépendants les uns des autres, même quand ils font partie de la même transmission, ils peuvent arriver abîmés. Les datagrammes, comme les moyens de transmission sans accusé de réception, sont sans connexion.

Ces protocoles sont à la base de La suite TCP/IP, en ce sens qu'ils fournissent des services à un certain nombre d'autres protocoles, dont la plupart sont impliqués dans les questions relatives aux applications et qu'ils partent à l'hypothèse de TCP/IP et/ou UDP prennent soin des problèmes relevant du transport de bas niveau et du réseau.

[WOO 00]

II.2. Le protocole IP :

Grâce à l'essor formidable d'Internet, TCP/IP est devenu une norme de fait pour la connexion des réseaux hétérogènes.
Les services rendus par le protocole IP correspondent à peu près à la couche réseau du modèle OSI. Ce protocole définit en particulier le format des paquets de données échangés : les datagrammes IP.

II.2.1. Adressage IP :

Une distinction doit être faite entre noms, adresses, et chemins. Un nom indique ce que nous cherchons. Une adresse indique où cela se trouve. Un chemin indique comment y aboutir. Le protocole Internet s'occupe essentiellement des adresses. C'est à des protocoles de niveau plus élevé (ex., hôte vers hôte ou application) que revient la tâche de lier des noms à des adresses. Le module Internet déduit de l'adresse Internet IP une adresse réseau local. La tâche qui consiste à transcrire l'adresse de réseau local en termes de chemin (ex, sur un réseau local ou dans un routeur) revient au protocole de bas niveau.

Actuellement l'organisme chargé d'attribuer les adresses IP est l'Internic (Internet Network Information center). Les adresses ont une longueur fixe de 4 octets (32 bits). Une adresse commence toujours par un numéro de réseau ou l'**id de réseau** qui identifie les machines qui se trouvent sur un même réseau physique, suivi d'un **id d'hôte** qui identifie une station, un serveur, un routeur ou tout autre périphérique. Les adresses IP sont représentées sous forme de quatre entiers décimaux séparés par un point décimal.
Exemple : 10.125.25.30
Ci-dessous un schéma représentant chaque classe d'adresses :

	Bits 0 – 8 – 16 – 24 – 31	
Classe A	0 id réseau	id hôte
Classe B	10 id réseau	id hôte
Classe C	110 id réseau	id hôte
Classe D	1110	Multicast
Classe E	11110	Réservé

Figure 3.1 : Différentes classes d'adresses IP.

Il existe trois formats ou classes d'adresses Internet principales : pour la classe A, le bit de poids fort vaut zéro, les 7 bits suivants désignent le réseau, les derniers 24 bits désignent l'adresse locale de la machine. Pour la classe B, les deux bits de poids fort valent 1 et 0, les 14 bits suivants désignent le réseau et les 16 derniers bits l'adresse locale de machine. Pour la classe C, les trois bits de poids fort forment le schème 110, les 21 bits suivants forment l'adresse réseau et les 8 derniers bits l'adresse locale. Les adresses de classe D désignent un groupe de hôtes qui ne sont pas forcément du même réseau. Pour la classe E, ses adresses sont réservées.

Le rôle de la couche réseau est d'acheminer les données entre l'émetteur et le destinataire au travers de différents réseaux en mettant en place un système d'adressage hiérarchique pour combiner aux manques de l'adressage MAC.

Les protocoles de la couche réseau utilisent un système d'adressage garantissant l'unicité des adresses sur le réseau et définissant une méthode d'acheminement des informations entre les réseaux.

Donc, sur Internet, les ordinateurs communiquent entre eux grâce au protocole TCP/IP qui utilise des numéros de 32 bits, que l'on écrit sous forme de 4 numéros allant de 0 à 255 (4 fois 8 bits), on les note donc sous la forme xxx.xxx.xxx.xxx où chaque xxx représente un entier de 0 à 255.

Ces numéros servent aux ordinateurs du réseau pour se reconnaître, ainsi il ne doit pas exister deux ordinateurs sur le réseau ayant la même adresse IP.

II.2.1.1. Les règles d'adressage :

Comme nous l'avons vu une adresse IP est une adresse 32 bits notée sous forme de 4 nombres entiers séparés par des points. On distingue en fait deux parties dans l'adresse IP:

- Une partie des nombres à gauche désigne le réseau (on l'appelle id réseau)
- Les nombres de droite désignent les ordinateurs de ce réseau (on l'appelle id hôte)

Les adresses IP ne peuvent communiquer qu'avec des adresses ayant le même numéro de réseau, y compris si des stations se trouvent sur le même segment. C'est ce même numéro qui permet au routeur d'acheminer le paquet au destinataire.

II.2.1.2. Les classes d'adresses IP :

Les adresses IP sont réparties en plusieurs classes, en fonction des bits qui les composent. Ce qui suit donnera plus de détails sur les adresses IP et leurs significations :

Classe A : Dans une adresse IP de classe A, le premier octet représente le réseau. Le bit de poids fort (le premier bit, celui de gauche) est à zéro, ce qui signifie qu'il y a 2 puissance(7), (de 00000000 à 01111111) possibilités de réseaux, c'.à.d 128.

Toutefois le réseau 0 (00000000) n'existe pas et le nombre 127 est réservé pour désigner votre machine, les réseaux disponibles en classe A sont donc les réseaux allant de **1.0.0.0** à **126.0.0.0**.

Les trois octets de droite représentent les ordinateurs du réseau, le réseau peut donc contenir :

2 puis(24)-2 = 16777214 ordinateurs.

Une adresse IP de classe A, en binaire, ressemble à ceci : 0 xxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Classe B : Dans une adresse IP de classe B, les deux premiers octets représentent le réseau. Les deux premiers bits sont 1 et 0, ce qui signifie qu'il y a 2 puis(14), (de 10 000000 00000000 à 10 111111 11111111) possibilités de réseaux, c'.à.d 16384.

Les réseaux disponibles en classe B sont donc les réseaux allant de **128.0.0.0** à **191.255.0.0** .

Les deux octets de droite représentent les ordinateurs du réseau, le réseau peut donc contenir:

2 puis(16)-2 = 65534 ordinateurs.

Une adresse IP de classe B, en binaire, ressemble à ceci : 10 xxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Classe C : Dans une adresse IP de classe C, les trois premiers octets représentent le réseau. Les trois premiers bits sont 1,1 et 0, ce qui signifie qu'il y a 2 puis(21) possibilités de réseaux, c'.à.d 2097152. Les réseaux disponibles en classe C sont donc les réseaux allant de **192.0.0.0** à **223.255.255.0** .

L'octet de droite représente les ordinateurs du réseau, le réseau peut donc contenir :

2 puis(8)-2 = 254 ordinateurs.

Une adresse IP de classe C, en binaire, ressemble à ceci : 110 xxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Il arrive fréquemment dans une entreprise qu'un seul ordinateur soit relié à Internet, c'est par son intermédiaire que les autres ordinateurs du réseau accèdent à Internet (on parle généralement de proxy).

Dans ce cas, seul l'ordinateur relié à Internet a besoin de réserver une adresse IP auprès de l'Internic. Toutefois, les autres ordinateurs ont tout de même besoin d'une adresse IP pour pouvoir communiquer ensemble de façon interne.

Ainsi, l'Internic a réservé une poignée d'adresses dans chaque classe pour permettre d'affecter une adresse IP aux ordinateurs d'un réseau local relié à Internet sans risquer de créer de conflits d'adresses IP sur le réseau. Il s'agit des **adresses privées** suivantes:

- **10.0.0.1** à **10.255.255.254**
- **172.16.0.1** à **172.31.255.254**
- **192.168.0.1** à **192.168.255.254**

II.2.1.3. Adresse de réseau et adresse de broadcast :

Une adresse réseau est une adresse IP dont tous les bits hôtes sont occupés par des 0 binaires. Cette adresse désigne le réseau lui-même et non pas un hôte précis. Exemple, dans un réseau de classe A, 113.0.0.0 désigne le réseau comprenant l'hôte 113.1.2.3.

L'adresse de broadcast est une adresse utilisée pour joindre en même temps tous les hôtes d'un réseau. Tous les bits hôtes de celle-ci sont à 1. Exemple : pour le réseau 192.168.10.0, l'adresse de broadcast est 192.168.10.255

NB : Ces adresses ne peuvent donc pas être utilisée pour identifier un hôte sur le réseau.

II.2.1.4. Les sous réseaux :

Afin d'augmenter les capacités de gestion de trafic dans un réseau, il est possible de subdiviser ce dernier en plusieurs sous réseaux afin de permettre une segmentation des domaines de broadcast.

Pour cela, on emprunte à la partie hôte des bits que l'on désigne comme champ de sous réseaux. Le nombre minimal de bits à emprunter est de 2 et le nombre maximal est égal à tout nombre laissant 2 bits à la partie hôte.

II.2.1.5. Le masque de sous réseau :

Un masque de sous réseau est une adresse de 32 bits contenant des 1 aux emplacements des bits que l'on désire conserver, et des 0 pour ceux que l'on veut rendre égaux à zéro. Une fois ce masque créé, il suffit de faire un ET logique entre la valeur que l'on désire masquer et le masque afin de garder intacte la partie que l'on désire et annuler le reste.

Ainsi, un masque réseau se présente sous la forme de 4 octets séparés par des points (comme une adresse IP), il comprend (dans sa notation binaire) des zéros au niveau des bits de l'adresse IP que l'on veut annuler (et des 1 au niveau de ceux que l'on désire conserver).

Il y a plusieurs avantages à utiliser ce procédé. Un d'entre eux est de pouvoir connaître le réseau associé à une adresse IP. En effet, comme nous l'avons vu précédemment, le réseau est déterminé par un certain nombre d'octets de l'adresse IP (1 octet pour les adresses de classe A, 2 pour les adresses de classe B, et 3 octets pour la classe C). De plus, nous avons vu que l'on note un réseau en prenant le nombre d'octets qui le caractérise, puis en complétant avec des 0.

Ainsi, le réseau associé à l'adresse 34.56.123.12 est 34.0.0.0 (puisqu'il s'agit d'une adresse de classe A).

Il suffit donc pour connaître l'adresse du réseau associé à l'adresse IP 34.56.123.12 d'appliquer un masque dont le premier octet ne comporte que des 1 (ce qui donne 255), puis des 0 sur les octets suivants (ce qui donne 0..).

Le masque est : 11111111.00000000.00000000.00000000

Le masque associé à l'adresse IP 34.208.123.12 est donc 255.0.0.0.

La valeur binaire de 34.208.123.12 est : 00100010.11010000.01111011.00001100

Un ET logique entre :

00100010.11010000.01111011.00001100
ET
11111111.00000000.00000000.00000000, donne :

00100010.00000000.00000000.00000000

C'.à.d 34.0.0.0, c'est bien le réseau associé à l'adresse 34.56.123.12

En généralisant, on obtient les masques suivants pour chaque classe :

- Pour une adresse de Classe A, seul le premier octet nous intéresse, on a donc un masque de la forme 11111111.00000000.00000000.00000000, c'.à.d en notation décimale : 255.0.0.0.
- Pour une adresse de Classe B, les deux premiers octets nous intéresse, on a donc un masque de la forme 11111111.11111111.00000000.00000000, c'.à.d en notation décimale : 255.255.0.0.
- Pour une adresse de Classe C on s'intéresse aux trois premiers octets, on a donc un masque de la forme 11111111.11111111.11111111.00000000, c'.à.d en notation décimale : 255.255.255.0.

II.2.1.6.Création de sous réseau :

Reprenons l'exemple du réseau 34.0.0.0 et supposons que l'on désire que les deux premiers bits du deuxième octet permettent de désigner le réseau.

Le masque à appliquer sera alors : 11111111.11000000.00000000.00000000 c'.à.d 255.192.0.0.Si on applique ce masque, à l'adresse 34.208.123.12 on obtient : 34.192.0.0.
En réalité il y a 4 cas de figures possibles pour le résultat du masquage d'une adresse IP d'un ordinateur du réseau 34.0.0.0 :

- Soit les deux premiers bits du deuxième octet sont 00, auquel cas le résultat du masquage est 255.0.0.0
- Soit les deux premiers bits du deuxième octet sont 01, auquel cas le résultat du masquage est 255.64.0.0
- Soit les deux premiers bits du deuxième octet sont 10, auquel cas le résultat du masquage est 255.128.0.0
- Soit les deux premiers bits du deuxième octet sont 11, auquel cas le résultat du masquage est 255.192.0.0

Ce masquage divise donc un réseau de classe A (pouvant admettre 16777214 ordinateurs) en 4 sous réseaux pouvant admettre 2 puis(22) ordinateurs, c'.à.d 4194304 ordinateurs. Au passage on remarque que le nombre d'ordinateurs possibles dans les deux cas est au total de 16777214 ordinateurs (4 x 4194304 - 2 = 16777214). Le nombre de sous réseaux dépend du nombre de bits que l'on attribue en plus au réseau (ici 2).

II.2.2. La sélection du chemin :

Les méthodes de sélection du chemin permettent aux équipements de couche 3; les routeurs; de déterminer la route (chemin) à suivre pour acheminer les informations au travers de différents réseaux.

Les services de routage utilisent les informations de topologie du réseau pour évaluer les chemins. Ce processus est aussi appelé routage des paquets et prend en compte divers paramètres; comme :

- Densité du trafic ;
- Nombre de routeurs à franchir pour joindre la destination ;
- Vitesse des liaisons ;
- ...etc.

II.2.2.1. Le routage :

Pour sélectionner le chemin, nous avons vu que le routeur utilisait sa table de routage. Une table de routage fait la correspondance entre les réseaux et les interfaces du routeurs qui leur sont connectés. Il existe deux manières de mettre à jour ces tables. La première est le routage statique, ou l'administrateur configure manuellement les routes que le routeur doit utiliser. La seconde consiste en l'utilisation de protocoles dits de « routage », permettant l'échange d'informations sur la topologie du réseau entre les différents routeurs.

Ces protocoles permettent donc aux routeurs de cartographier le meilleur chemin vers n'importe quel autre routeur ou segment réseau dans le même réseau ou encore sur Internet. Ces échanges de messages sont consommateurs de bande passante ; ce qui présente un désavantage par rapport au routage statique ; cependant, le routage dynamique permet une meilleure réactivité du réseau aux pannes car celui-ci peut s'adapter de lui-même aux changements de topologie.

Il existe de nombreux protocoles de routage, chacun utilisant certaines caractéristiques du réseau pour fonder ses décisions. Ces caractéristiques, pouvant être la bande passante, la fiabilité ou encore l'encombrement d'un segment, sont appelées des métriques.

II.2.2.2. Exemple de protocole de routage : le protocole RIP

RIP est le protocole le plus utilisé à ce jour dans les réseaux actuels. Il calcule la distance jusqu'à un hôte en mesurant le nombre de sauts (routeurs) et privilégie le chemin le plus court.

On appelle ce type de protocole basé sur le nombre de sauts des protocoles de routage à vecteur de distance. Le protocole RIP met à jours les tables de routage toutes les 30 secondes et autorise un nombre de saut maximal de 15.

II.2.2.3. Les différents protocoles de routage :

Il faut savoir qu'un système autonome est un ensemble d'équipements gérés par la même administration. Une première classification se fait entre les protocoles de routage selon qu'ils soient :

- IGP : Interior Gateway Protocol (dans un système autonome)
- EGP : Exterior Gateway Protocol (entre les systèmes autonomes)

Parmi les protocoles IGP les plus courant, on retrouve :

- IGRP : développé pour résoudre les problèmes associés au routage dans de grands réseaux multi fournisseurs, c'est un protocole à vecteur de distance, cependant il prend également en compte d'autres métriques qu'il est possible de pondérer afin de privilégier certains aspects du chemin : bande passante, charge, délai, fiabilité.

- EIGRP : version améliorée d'IGRP.

- OSPF (Open Shortest Path First) : Protocole dit de «routage à état de lien», il incluse des métriques de coûts tenant compte de : la vitesse d'acheminement, du trafic, de la fiabilité et de la sécurité.

II.2.2.4. Le routage indirect :

Le protocole IP permet également l'utilisation d'une « passerelle par défaut » c'est-à-dire l'utilisation d'une route à utiliser si le routeur ne connaît pas le réseau de destination.

Si un routeur reçoit un paquet dont il ne connaît pas le réseau de destination, il le transmet donc à un autre routeur susceptible de le connaître.

Un protocole routable est un protocole pouvant être acheminé au travers de différents réseaux. Exemple : IP, IPX, Appletalk. Par opposition, un protocole non routable ne peut être routé. Exemple : NetBEUI

[WLA 01]

II.2.3. Spécification :

Un résumé du contenu de l'en-tête Internet suit :

<table>
<tr><td>0</td><td>4</td><td>8</td><td>16</td><td>19</td><td>24</td><td>31</td></tr>
<tr><td>VERS</td><td>HLEN
(LET)</td><td>Type de service</td><td colspan="4">Longueur totale</td></tr>
<tr><td colspan="3">Identification</td><td>Indicateurs
(flags)</td><td colspan="3">Décalage de fragment
(fragment offset)</td></tr>
<tr><td colspan="2">Durée de vie</td><td>Protocole</td><td colspan="4">Somme de contrôle d'en tête
(checksum)</td></tr>
<tr><td colspan="7">Adresse IP source</td></tr>
<tr><td colspan="7">Adresse IP de destination</td></tr>
<tr><td colspan="5">Options IP (si nécessaire)</td><td colspan="2">Remplissage
(bourrage)</td></tr>
<tr><td colspan="7">Données (data)</td></tr>
<tr><td colspan="7">...</td></tr>
</table>

Figure 3.2 : Structure de l'En-tête IP.

- Version : (4 bits) Le champ Version renseigne sur le format de l'en-tête Internet. Indique la version de protocole IP utilisée.

- Longueur d'En-tête : (4 bits) Le champ Longueur d'En-tête (LET) ou **HLEN** (IP header length) code la longueur de l'en-tête Internet, l'unité étant le mots de 32 bits, et de ce fait, marque le début des données. Notez que ce champ ne peut prendre une valeur en dessous de 5 pour être valide.

- Type de Service : (8 bits) Le Type de Service donne une indication sur la qualité de service souhaitée, qui reste cependant un paramètre "abstrait". Ce paramètre est utilisé pour "guider" le choix des paramètres des services actuels lorsqu'un datagramme transite dans un réseau particulier. Certains réseaux offrent un mécanisme de priorité, traitant préférentiellement un tel trafic par rapport à un trafic moins prioritaire (en général en acceptant seulement de véhiculer des paquets d'un niveau de priorité au dessus d'un certain seuil lors d'une surcharge momentanée). Principalement, le choix offert est une négociation entre les trois contraintes suivantes : faible retard, faible taux d'erreur, et haut débit.
  Bits 0-2 : Priorité.
  Priorité :
  111 - Network Control
  110 - Internetwork Control
  101 - CRITIC/ECP
  100 - Flash Override
  011 - Flash
  010 - Immediate
  001 - Priority

000 - Routine
Bit 3 :
0 = Retard standard,
1 = Retard faible.

Priorité	D	T	R	0	0

0 8 bits

Figure 3.3 : Structure du champ Type de Service.

Bits 4 :
0 = Débit standard,
1 = Haut débit.
Bits 5 :
0 = Taux d'erreur standard
1 = Taux d'erreur faible.
Bit 6-7 : Réservé.

L'utilisation des indications en termes de retard, débit, et qualité de transmission peut augmenter le "coût" (d'un certain point de vue) du service. Dans la plupart des réseaux, de meilleures performances pour l'un de ces paramètres s'obtiennent au prix d'une dégradation des performances pour un autre. A moins d'une situation exceptionnelle, il sera préférable de ne pas activer plus de deux optimisations sur les trois.
Le "Type de Service" sert à préciser le traitement effectué sur le datagramme pendant sa transmission à travers Internet.
La priorité dite "Network Control" est stipulée comme étant une priorité à l'intérieur d'un seul réseau. Le fait d'utiliser cette option instaure une priorité pour chaque section traversée. La priorité "Internetwork Control" n'est gérée que par les routeurs. Si l'utilisation de ces priorités a une signification particulière ou supplémentaire pour l'un des réseaux, il est de la responsabilité de ce dernier de lire et d'interpréter les présentes informations.

- Longueur Totale : (16 bits) Le champ "Longueur Totale" est la longueur du datagramme entier y compris en-tête et données, mesurée en octets. Ce champ ne permet de coder qu'une longueur de datagramme d'au plus 65 535 octets. Une telle longueur rendrait de toutes façon les datagrammes impossible à gérer pour la plus grande partie des réseaux. Les hôtes devront au moins pouvoir accepter des datagrammes d'une longueur jusqu'à 576 octets (qu'il s'agisse d'un datagramme unique ou d'un fragment). Il est de même recommandé que des hôtes ne décident d'envoyer des datagrammes de plus de 576 octets que dans la mesure où ils sont sûrs que la destination est capable de les accepter. Le nombre 576 a été choisi pour permettre à un bloc de données de taille raisonnable d'être transmis dans un datagramme, tenant compte des données à ajouter pour constituer les en-têtes de protocole.
Par exemple, cette taille permet la transmission d'un bloc de 512 octets, plus 64 octets d'en-tête dans un datagramme unique. (NB : je rappelle ici que la taille de 512 octets correspond à un secteur sur la plupart des supports de stockage) La taille maximale d'un en-tête Internet étant de 60 octets, et sa taille typique étant de 20 octets, ce nombre permet de

conserver une bonne marge pour les données protocolaires de plus haut niveau.

- Identification : (16 bits) Une valeur d'identification assignée par l'émetteur pour identifier les fragments d'un même datagramme.

- Indicateurs ou Flags : (3 bits), Divers commutateurs de contrôle.
  Bit 0 : réservé, doit être laissé à zéro
  Bit 1: (AF) 0 = Fragmentation possible,
  1 = Non fractionnable.
  Bit 2: (DF) 0 = Dernier fragment,
  1 = Fragment intermédiaire.

0	AF	DF

Figure 3.4 : **Structure du champ Flags.**

- Décalage de fragment ou Fragment Offset : (13 bits) Ce champ indique le décalage du premier octet du fragment par rapport au datagramme complet. Cette position relative est mesurée en blocs de 8 octets (64 bits). Le décalage du premier fragment vaut zéro.

- Durée de vie : (8 bits) Ce champ permet de limiter le temps pendant lequel un datagramme reste dans le réseau. Si ce champ prend la valeur zéro, le datagramme doit être détruit. Ce champ est modifié pendant le traitement de l'entête Internet. La durée de vie est mesurée en secondes. Chaque module Internet doit retirer au moins une unité de temps à ce champ, même si le traitement complet du datagramme par le module est effectué en moins d'une seconde. De ce fait, cette durée de vie doit être interprétée comme la limite absolue maximale de temps pendant lequel un datagramme peut exister. Ce mécanisme est motivé par la nécessité de détruire les datagrammes qui n'ont pu être acheminés, en limitant la durée de vie même du datagramme.

- Protocole : (8 bits) Ce champ précise quel protocole de niveau supérieur est utilisé dans la section données du datagramme Internet (paquets entrants après la fin du traitement IP).

- Checksum ou somme de contrôle d'en-tête : (16 bits) Un Checksum calculé sur l'en-tête uniquement. Comme certains champs de l'en-tête sont modifiés (durée de vie par exemple) pendant leur transit à travers le réseau, ce Checksum doit être recalculé et vérifié en chaque point du réseau où l'en-tête est réinterprétée.
  L'algorithme utilisé pour le Checksum est le suivant :
  On calcule le complément à un sur 16 bits de la somme des compléments à un de tous les octets de l'en-tête pris par paires (mots de 16 bits). Lorsque l'on calcule le Checksum, on considère un en-tête dont le champ réservé pour ce même Checksum vaut zéro.

L'algorithme de Checksum peut paraître élémentaire mais l'expérimentation a montré que cette technique était suffisante. Il se peut que cet algorithme soit plus tard remplacé par un calcul de type CRC, suivant la nécessité future.

- Adresse source ou d'origine : (32 bits) L'adresse Internet de la source c.à.d nœud émetteur.

- Adresse destination : (32 bits) L'adresse Internet du destinataire.

- Options : (longueur variable) Les datagrammes peuvent contenir des options. Celles-ci doivent être implémentées par tous les modules IP (hôtes et routeurs). Le caractère "optionnel" concerne leur transmission, et non leur implémentation. Dans certains environnements, l'option de sécurité peut être obligatoire dans tous les datagrammes. Un datagramme peut comporter zéro ou plus options. Voici les deux formats possibles d'une option :
  Cas 1: Une option codée sur un seul octet.
  Cas 2: Un octet codant le type d'option, un octet donnant la taille de l'option, les octets de données d'option.
  La taille de l'option compte tous les octets de l'option y compris le type, son propre octet et tous les octets de donnée d'option.
  L'octet de type d'option est composé de trois champs de bits :
    1 bit indicateur de recopie
    2 bits classe d'option
    5 bits numéro d'option.

- Données : (longueur variable, maximum 64 Ko) Cet élément contient des informations de couche supérieure.

- Remplissage : Des zéros sont ajoutés à ce champ pour s'assurer que l'en-tête IP est toujours un multiple de 32 bits.

[DES 99]

Les stations émettrices et destinatrices peuvent se trouver dans deux réseaux différents. Un système de routage permet de déterminer un chemin entre les deux stations en passant par des routeurs. Certains auteurs utilisent le terme de passerelle (application gateway en anglais) au lieu de routeur, mais il est d'usage de le réserver à des machines qui font la conversion entre protocoles différents au niveau application du modèle OSI (par exemple, pour convertir des messages entre deux formats différents de courrier électronique). IP envoie des datagrammes d'un endroit à l'autre du réseau mais il n'assure pas le contrôle de la bonne arrivée des données (possibilité de perte, duplication, arrivée dans un mauvais ordre des paquets).
[GRI 97]

II.2.4. Fragmentation :

La fragmentation du datagramme Internet devient nécessaire dès lors qu'un datagramme de grande taille arrive sur une portion de réseau qui n'accepte la transmission que de paquets plus courts.

Un datagramme Internet peut être spécifié "non fractionnable", un tel datagramme Internet ne doit jamais être fragmenté quelques soient les circonstances. Si un datagramme Internet non fractionnable ne peut être acheminé jusqu'à sa destination sans être fragmenté, alors il devra être rejeté.

La fragmentation, la transmission et le réassemblage à travers un réseau local hors de vue d'un module de protocole Internet est appelée fragmentation Intranet.

Les procédures de fragmentation et réassemblage Internet doivent pouvoir découper un datagramme Internet en un nombre de "fragments" arbitraire et quelconque pourvu que le réassemblage soit possible. Le récepteur des fragments utilise le champ d'identification pour s'assurer que des fragments de plusieurs datagrammes ne puissent être mélangés. Le champ "Fragment Offset" indique au récepteur la position du fragment reçu dans le datagramme original. Les champs "Fragment Offset" et "Longueur Totale" déterminent la portion du datagramme original que représente le fragment. L'indicateur bit "Dernier Fragment" indique (lors de sa remise à zéro) au récepteur qu'il s'agit du dernier fragment. Ces champs véhiculent suffisamment d'information pour réassembler les datagrammes.

Le champ d'identification sert à distinguer les fragments d'un datagramme de ceux d'un autre datagramme. Le module Internet émetteur d'un datagramme Internet initialise le champ d'identification à une valeur qui doit être unique pour cette paire source-destination et pour ce protocole pendant toute la durée de transmission de ce datagramme. Le module Internet terminant l'émission d'un datagramme met le bit "Dernier Fragment" et le champ "Fragment Offset" à zéro.

Pour fragmenter un long datagramme, un module Internet (par exemple, dans un routeur), crée deux nouveaux datagrammes et copie le contenu des champs d'en-tête Internet originaux dans les deux nouveaux en-têtes. Les données du datagramme original sont divisées en deux portions, la première d'une taille multiple de 8 octets (64 bits) (la taille de la seconde portion n'est donc pas nécessairement un multiple de 8 octets). Nous appellerons le nombre de blocs de 8 octets dans la première portion NBF (ou Nombre de Blocs du Fragment). La première portion de données est placée dans le premier des deux nouveaux datagrammes, et le champ "Longueur Totale" est renseigné avec la taille de ce datagramme.

Le bit "Dernier Fragment" est basculé à 1. La seconde portion de données est placée dans le second des deux nouveaux datagrammes, et le champ "longueur totale" est renseigné avec la taille du second datagramme. Le bit "Dernier Fragment" est placé à la même valeur que celui du datagramme original. Le champ "Fragment Offset" du second datagramme constitué est renseigné avec la valeur du même champ du datagramme original plus NFB. Cette procédure peut être généralisée à une fragmentation en n fragments, plutôt que les deux décrits ci-dessus. Pour réassembler les fragments d'un datagramme Internet, un module Internet (par exemple dans un hôte destinataire) recombine les datagrammes dont les valeurs des quatre champs suivants sont identiques : identification, source, destination, et protocole. La recombinaison est réalisée en replaçant la portion de donnée contenue dans chaque fragment dans un tampon à la position relative indiquée par le champ "Fragment Offset" lu dans l'en-tête correspondant. Le premier fragment sera donc placé en début de tampon, et le dernier fragment récupéré aura le bit "Dernier Fragment" à zéro.

Voici un exemple illustrant l'achèvement d'un datagramme à un réseau (3) à partir d'un réseau (1) par fragmentation [WCE] :

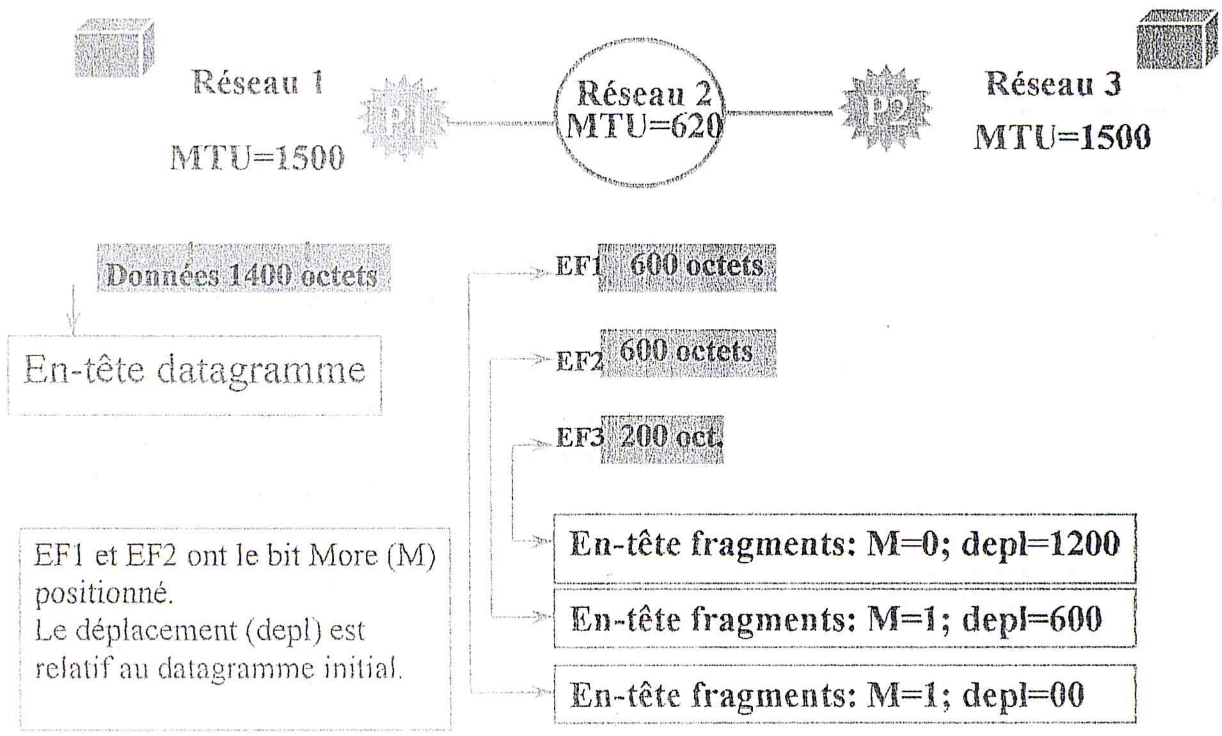


Figure 3.5 : Fragmentation d'un datagramme.

II.3. Le protocole ARP :

Le protocole ARP a un rôle phare parmi les protocoles de la couche Internet de la suite TCP/IP, car il permet de connaître l'adresse physique d'une carte réseau correspondant à une adresse IP, c'est pour cela qu'il s'appelle Protocole de résolution d'adresse (en anglais ARP signifie Address Resolution Protocol).

Chaque machine connectée au réseau possède un numéro d'identification de 48 bits. Ce numéro est un numéro unique qui est fixé dès la fabrication de la carte en usine. Toutefois la communication sur Internet ne se fait pas directement à partir de ce numéro (car il faudrait modifier l'adressage des ordinateurs à chaque fois que l'on change une carte réseau) mais à partir d'une adresse dite logique attribuée par un organisme: l'adresse IP.

Ainsi, pour faire correspondre les adresses physiques aux adresses logiques, le protocole ARP interroge les machines du réseau pour connaître leur adresse physique, puis crée une table de correspondance entre les adresses logiques et les adresses physiques dans une mémoire cache.

Lorsqu'une machine doit communiquer avec une autre, elle consulte la table de correspondance. Si jamais l'adresse demandée ne se trouve pas dans la table, le protocole ARP émet une requête sur le réseau. L'ensemble des machines du réseau va comparer cette adresse logique à la leur. Si l'une d'entre-elles s'identifie à cette adresse, la machine va répondre à ARP qui va stocker le couple d'adresses dans la table de correspondance et la communication va alors pouvoir avoir lieu...

II.4. Le protocole ICMP :

Le protocole ICMP (Internet Control Message Protocol) est un protocole qui permet de gérer les informations relatives aux erreurs aux machines connectées. Etant donné le peu de contrôles que le protocole IP réalise il permet non pas de corriger ces erreurs mais de faire part de ces erreurs aux protocoles des couches voisines. Ainsi, le protocole ICMP est utilisé par tous les routeurs, qui l'utilisent pour reporter une erreur (appelé Delivery Problem).

Les messages d'erreur ICMP sont transportés sur le réseau sous forme de datagramme, comme n'importe quelle donnée. Ainsi, les messages d'erreur peuvent eux-mêmes être à sujet d'erreurs.

Toutefois en cas d'erreur sur un datagramme transportant un message ICMP, aucun message d'erreur n'est délivré pour éviter un effet "boule de neige" en cas d'incident sur le réseau.

[WLA 01]

0	4	8	16	19	24	31
Code		Message	Message			

Figure 3.6 : Structure d'un paquet ICMP.

Le protocole ICMP permet l'échange des diagnostics d'erreurs entre éléments actifs du réseau. Bien que tournant au dessus de IP (couche 3), il est requis par toute machine fonctionnant sous IP. La plupart des paquets contiennent des informations sur les causes de destructions des paquets par exemple TTL expiré, destination inaccessible. Le format des données est extrêmement élaboré par contre il n'y a pas de dialogue entre deux entités ICMP. Tout paquet ICMP commence par un en-tête :
- type : 8 bits,
- code : 8 bits,
- checksum : 16 bits.

Les types de message sont les suivants :
0 Réponse d'écho
3 Destination inaccessible
4 Demande de ralentissement

5 Redirection
8 Echo
9 Annonce de routeur
10 Sollicitation de routeur
11 TTL expiré
12 Problème de paramètre
13 Horodatage
14 Réponse d'horodatage
15 Demande d'information
16 Réponse d'information

Les messages les plus fréquents sont ceux liés aux types Destination inaccessible, TTL expiré, et Demande de ralentissement. L'en-tête est alors suivie de 32 bits de remplissage ainsi que les premiers octets du paquet qui a déclenché le problème. Ces octets permettront au récepteur d'identifier l'application qui a émis le paquet. Par exemple dans le cas de Destination inaccessible le champ code précisera si c'est une erreur de type réseau, protocole, hôte, ...

Les types « écho » permettent de réaliser des fonctions de test du réseau par un utilisateur final. Ces tests sont implémentés par les commandes traceroute et ping :

* Commande ping :

Elle utilise les types 8 en émission et 0 en réception pour vérifier la qualité de la liaison avec un hôte. La réponse est fabriquée en permutant les adresses sources et destination dans le paquet IP. Le paquet ICMP de réponse dans son champ de données recopie le champ de données du paquet reçu. Ainsi on peut par exemple vérifier le séquencement des messages. De plus l'application ping mesure le temps d'aller-retour du message dans le réseau.

Exemple :

```
[maylis] ~ > /usr/etc/ping -s www.cwi.nl
PING info4u.cwi.nl: 56 data bytes
64 bytes from info4u.cwi.nl (192.16.196.148): icmp_seq=0. time=116. ms
64 bytes from info4u.cwi.nl (192.16.196.148): icmp_seq=1. time=1237. ms
64 bytes from info4u.cwi.nl (192.16.196.148): icmp_seq=3. time=83. ms
64 bytes from info4u.cwi.nl (192.16.196.148): icmp_seq=4. time=100. ms
64 bytes from info4u.cwi.nl (192.16.196.148): icmp_seq=5. time=144. ms
64 bytes from info4u.cwi.nl (192.16.196.148): icmp_seq=7. time=82. ms
```

* Commande traceroute :

Elle utilise le type 11. Cette commande permet de vérifier les relais sur la route vers un hôte. Elle procède à l'envoi de paquets successifs en incrémentant à chaque fois le TTL du paquet. Le premier paquet est envoyé avec un TTL égal à 1. Le premier équipement le recevra le décrémentera de 1 et comme il sera devenu nul, détruira le paquet puis émettra un paquet ICMP de type 11 en inversant les champs expéditeur et émetteur dans le paquet IP. Le second paquet initialisé à 2 franchira le premier équipement mais lors de la traversée du second équipement le TTL deviendra nul et provoquera de même l'émission d'un paquet ICMP de type 11,... ainsi de suite.

Exemple :

```
[maylis] ~ > traceroute www.cwi.nl
```

traceroute to info4u.cwi.nl (192.16.196.148), 30 hops max, 40 byte packets
1 iceman (147.210.8.154) 2 ms 2 ms 2 ms
2 b3a1 (147.210.8.254) 2 ms 2 ms 3 ms
3 b9a1.u-bordeaux.fr (147.210.254.253) 2 ms 2 ms 2 ms
. . . .
18 cwi-gw.cwi.nl (192.16.183.32) 94 ms 92 ms 85 ms
19 laxeer.cwi.nl (192.16.191.31) 85 ms 89 ms 87 ms
20 info4u.cwi.nl (192.16.196.148) 91 ms 88 ms 106 ms

[DES 99]

II.5. Le protocole TCP :

TCP est placé au-dessus de IP ; il correspond à peu près à la couche transport et à une partie de la couche session du modèle OSI. TCP assure une communication sûre en ouvrant une connexion full-duplex (possibilité pour les deux machines d'émettre en même temps) entre deux machines et en vérifiant qu'il n'y a pas d'erreur dans la transmission des données. TCP assure un contrôle de bout en bout : si la liaison entre l'émetteur et le destinataire passe par plusieurs routeurs, le contrôle se fait à chaque bout de la connexion et pas sur chaque liaison entre routeurs.

TCP divise les données en segments qui sont insérés dans des datagrammes IP. A chaque réception d'un segment, un accusé de réception est renvoyé à l'expéditeur. Si l'expéditeur ne reçoit pas cet accusé de réception avant un certain temps, il réexpédie le datagramme.

Chaque application TCP (rlogin, telnet, rsh,. . .) correspond à un numéro (de 16 bits) de port normalisé qui permet de savoir à quelle application le segment est destiné. On trouve ces numéros dans le fichier /etc/services. L'adresse de l'ordinateur destinataire est indiquée dans le datagramme IP. Le port qui indique à quelle application le message est destiné est enregistré dans la partie du message spécifique TCP. Ces numéros de ports ne changent jamais (ou très rarement) et ils sont gérés par une administration centrale.

TCP prétend fournir un service de communication de processus à processus, dans un environnement réseau complexe. TCP est défini comme un protocole de communication "host to host", c'est à dire de maître à maître (par opposition à "central à terminal").

II.5.1. Format de l'en-tête :

Les paquets TCP sont envoyés sous forme de datagrammes Internet. L'en-tête IP transmet un certain nombre de paramètres, tels que les adresses Internet source et destinataires. L'en-tête TCP est placée à la suite, contenant les informations spécifiques au protocole TCP. Cette division permet l'utilisation de protocoles autres que TCP, au dessus de la couche IP.

0								16	32bits
Port Source								Port Destination	
Numéro de séquence									
Accusé de réception									
Data Offset	Réservé	U	A	P	R	S	F	Fenêtre	
Checksum								Pointeur données urgentes	
Option								Bourrage	
Data									

Figure 3.7 : En-tête TCP.

- **Port source** : (16 bits) Le numéro de port de la source.

- **Port Destinataire** : (16 bits) Le numéro de port du destinataire.

- Numéro de séquence : (32 bits) Le numéro du premier octet de données par rapport au début de la transmission (sauf si SYN est marqué). Si SYN est marqué, le numéro de séquence est le numéro de séquence initial (ISN) et le premier octet à pour numéro ISN+1.

- Accusé de réception : (32 bits) Si ACK est marqué ce champ contient le numéro de séquence du prochain octet que le récepteur s'attend à recevoir. Une fois la connexion établie, ce champ est toujours renseigné.

- Data Offset : (4 bits) La taille de l'en-tête TCP en nombre de mots de 32 bits. Il indique là ou commence les données. L'entête TCP, dans tous les cas à une taille correspondant à un nombre entier de mots de 32 bits.

- Réservé : (6 bits) Réservés pour usage futur. Doivent nécessairement être à 0.

- Bits de contrôle: 6 bits (de gauche à droite) :
  URG : Pointeur de données urgentes significatif
  ACK : Accusé de réception significatif
  PSH : Fonction Push
  RST : Réinitialisation de la connexion
  SYN : Synchronisation des numéros de séquence
  FIN : Fin de transmission
  - Fenêtre : (16 bits) Le nombre d'octets à partir de la position marquée dans l'accusé de réception que le récepteur est capable de recevoir.

  - Checksum : (16 bits) Le Checksum est constitué en calculant le complément à 1 sur 16 bits de la somme des compléments à 1 des octets de l'en-tête et des données pris deux par deux (mots de 16 bits). Si le message entier contient un nombre impair d'octets, un 0 est ajouté à la fin du message pour terminer le calcul du Checksum. Cet octet supplémentaire n'est pas transmis. Lors du calcul du Checksum, les positions des bits attribués à celui-ci sont marquées à 0.

  Le Checksum couvre de plus une pseudo en-tête de 96 bits préfixée à l'en-tête TCP. Cette pseudo entête comporte les adresses Internet source et destinataires, le type de protocole et la longueur du message TCP. Ceci protège TCP contre les erreurs de routage. Cette information sera véhiculée par IP, et est donnée comme argument par l'interface TCP/Réseau lors des appels d'IP par TCP.

Adresse Source		
Adresse Destinataire		
Zéro	PTCL	Longueur TCP

0 32bits

Figure 3.8 : Structure de la pseudo En-tête TCP.

La longueur TCP compte le nombre d'octets de l'en-tête TCP et des données du message, en excluant les 12 octets de la pseudo en-tête.

- Pointeur de données urgentes : (16 bits) Communique la position d'une donnée urgente en donnant son décalage par rapport au numéro de séquence. Le pointeur doit pointer sur l'octet suivant la donnée urgente. Ce champ n'est interprété que lorsque URG est marqué.

- Options : (longueur variable) Les champs d'option peuvent occuper un espace de taille variable à la fin de l'en-tête TCP. Ils formeront toujours un multiple de 8 bits. Toutes les options sont prises en compte par le Checksum. Un paramètre d'option commence toujours sur un nouvel octet. Il est défini deux formats types pour les options:
  Cas 1 : Option mono-octet.
  Cas 2 : Octet de type d'option, octet de longueur d'option, octets de valeurs d'option.
  La longueur d'option prend en compte l'octet de type, l'octet de longueur lui-même et tous les octets de valeur et est exprimée en octets.
  Notez que la liste d'option peut être plus courte que ce que l'offset de données pourrait le faire supposer.
  Un octet de remplissage (padding) devra être dans ce cas rajouté après le code de fin d'options. Cet octet est nécessairement à 0.
  TCP doit implémenter toutes les options.
  Actuellement, les options définies sont (type indiqué en octal) :

Type	Longueur	Description
0	-	Fin de liste d'option
1	-	Nop
2	4	Taille de segment maximal

Définition des options spécifiques :
*Fin de liste d'options

00000000

Type=0

Ce code indique la fin du champ d'options. Sa position peut ne pas coïncider avec l'indication du début du champ de données marqué dans l'Offset de

données. Il doit être placé après toutes les options, et non après chaque option. Il ne doit être utilisé que dans le cas ou la fin des options ne coïncide pas avec le début du champ de données.
*No-Operation

00000001

Type=1
Cette option peut être utilisée entre deux options, par exemple pour aligner le début d'une option sur un début de mot de 16 bits. L'utilisation de ce séparateur n'est pas une obligation. L'implémentation doit donc prévoir de pouvoir prendre en compte une option même au milieu d'un mot.

*Taille maximale de segment

00000010	00000100	Taille max segment

Type = 2, Longueur = 4
Donnée d'option : Taille maximale de segment est 16 bits
Si cette option est présente, elle communique à l'émetteur la taille maximale des segments qu'il pourra envoyer. Ce champ doit être envoyé dans la requête de connexion initiale (avec SYN marqué). Si cette option est absente, le segment pourra être pris de n'importe quelle taille.

- Bourrage (padding): (longeur variable) Les octets de bourrage terminent l'en-tête TCP: de sorte que le nombre d'octet de celle-ci soit toujours multiple de 4 (32 bits), de sorte que l'offset de données marqué dans l'en-tête corresponde bien au début des données applicatives.

II.5.2. Communication de données :

Les données circulant dans la connexion ouverte doivent être vues comme un flux d'octets. L'application indique dans la commande SEND si les données soumises lors de cet appel (et toutes celles en attente) doivent être immédiatement émises par l'activation du flag PUSH.

Par défaut, TCP reste libre de stocker les données soumises par l'application pour les émettre à sa convenance, jusqu'à ce que le signal PUSH soit activé. Dans ce dernier cas, toutes les données non émises doivent être envoyées. Symétriquement, lorsque le TCP récepteur voit le flag PUSH marqué, il devra passer immédiatement toutes les données collectées à l'application destinataire.

Il n'y a à priori aucune corrélation entre la fonction PUSH et les limites des segments. Les données d'un segment peuvent être le résultat d'une seule commande SEND, en tout ou partie, ou celui de plusieurs appels SEND.

La fonction de la fonction push et du flag PUSH est de forcer la transmission immédiate de toutes les données latentes entre les deux TCP. Il ne s'agit aucunement d'une fonction d'enregistrement (Cf. langage Perl).

Il y a par contre une relation entre la fonction push et l'usage des tampons dans l'interface TCP/application. Chaque fois qu'un flag PUSH est associé à des données stockées dans le tampon de réception, celui-ci est intégralement transmis à l'application même s'il n'est pas plein. Si le tampon est rempli avant qu'un flag PUSH soit vu, les données sont transmises à l'application par éléments de la taille du tampon.

TCP dispose d'un moyen d'avertir l'application que, dans le flux de données qu'il est en train de lire, au delà de la position de lecture courante, des données de caractère urgent sont apparues. TCP ne définit pas ce que l'application est sensée faire lorsqu'elle est avisée de la présence de ces données. En général, c'est l'implémentation de l'application qui traitera ces données urgentes selon ses besoins propres.

II.5.3. Priorité et sécurité :

TCP utilise le champ "type de service" et les options de sécurité du protocole Internet pour fournir les fonctions relatives à la priorité et la sécurité des communications TCP, sur un principe de "détection".

Tous les modules TCP ne fonctionneront pas nécessairement dans un environnement sécurisé à plusieurs niveaux; certains pourront être limités à un fonctionnement sans sécurité, d'autres ne pourront prendre en compte qu'un seul niveau à la fois. Par conséquent, les implémentations TCP ne pourront répondre en termes de sécurité qu'à un sous ensembles de cas du modèle sécurisé multi-niveaux.

Les modules TCP opérant dans un environnement sécurisé à plusieurs niveaux devront correctement renseigner les segments sortants en termes de sécurité, niveau de sécurité, et priorité. De tels modules TCP doivent fournir aux applications supérieures telles que Telnet ou THP une interface leur permettant de spécifier ces paramètres.

[DES 99]

II.6. Le protocole UDP :

UDP (User Data Protocol) correspond à TCP mais il ne gère pas la bonne arrivée de données (ce n'est pas un protocole complet pour la couche transport).

C'est un protocole sans connexion qui se contente d'envoyer des datagrammes en ajoutant seulement au dessus de IP la possibilité de :

- détecter des erreurs à l'arrivée par l'ajout d'un checksum aux données transmises.
- distinguer entre les diverses applications travaillant dans un même ordinateur par l'indication d'un port comme pour TCP. Comme pour TCP, on trouve ces numéros dans le fichier /etc/services.

UDP est souvent utilisé pour construire un protocole plus complet. NFS est construit actuellement au dessus de UDP.
[GRI 97]

III. PORTS ET SOCKETS

Les "ports réservés" correspondent à des numéros de port standardisés permettant aux ordinateurs à distance de déterminer à quel port ils doivent se connecter pour bénéficier d'un service de réseau particulier (par exemple TELNET est proposé sur le port 23). Les sockets sont des "ports alloués dynamiquement" qui ne sont attribués aux opérations que si c'est nécessaire (ne sont pas compris dans la plage des numéros de ports standard).

Une "socket" est la concaténation de l'adresse Internet avec l'identificateur du port sélectionné. Elle est unique dans l'ensemble des communications TCP du réseau. Il y a un certain nombre de sockets réservés que TCP ne doit associer qu'avec certains processus bien identifiés.

IV. LA COUCHE APPLICATION

Cette partie présente cinq protocoles de la couche application :

IV.1. Le protocole Telnet :

Le protocole Telnet est un protocole standard d'Internet permettant l'interfaçage de terminaux et d'applications à travers Internet. Ce protocole fournit les règles de base pour permettre de relier un client (système composé d'un affichage et d'un clavier) à un interpréteur de commande (côté serveur).

Le protocole Telnet s'appuie sur une connexion TCP pour envoyer des données au format ASCII codées sur 8 bits entre lesquelles s'intercalent des séquences de contrôle Telnet. Il fournit ainsi un système orienté communication, bi-directionnel (half-duplex), codé sur 8 bits facile à mettre en oeuvre.

Le protocole Telnet repose sur trois concepts fondamentaux :

- Le paradigme du terminal réseau virtuel (NVT, *Network Virtual Terminal*) ;
- Le principe d'options négociées ;
- Les règles de négociation.

Ce protocole est un protocole de base, sur lequel s'appuient certains autres protocoles de la suite TCP/IP (FTP, SMTP, POP3, ...). Les spécifications de Telnet ne mentionnent pas d'authentification car Telnet est totalement séparé des applications qui l'utilisent (le protocole FTP définit une séquence d'authentification au-dessus de Telnet). En outre le protocole Telnet est un protocole de transfert de données non sûr, c'est-à-dire que les données qu'il véhicule circulent en clair sur le réseau (de manière non chiffrée). Lorsque le protocole Telnet est utilisé pour connecter un hôte distant à la machine sur lequel il est implémenté en tant que serveur, ce protocole est assigné au port 23. [RFC xx]

IV.2. Protocole FTP :

Le protocole FTP est un exemple de protocole alliant le dialogue à un codage très précis des données. Nous n'aborderons ici que les grandes lignes. Ce protocole utilise deux ports de connexions

- pour les commandes de contrôle port 21, interpréteur de protocole (PI),
- pour les données port 20, protocole de transfert de données (DTP).

Le PI client reçoit les demandes en provenance de l'interface utilisateur, les transforme en commande et les transmet au PI serveur. Lors d'une commande de transfert de fichier le PI concerné transmet au DTP (dont il a la charge) pour ouvrir la connexion avec le DTP de l'autre partie. Les commandes transmises sont des chaînes de caractères de la forme
<mnémonique> <liste_argument> <CR><LF>.

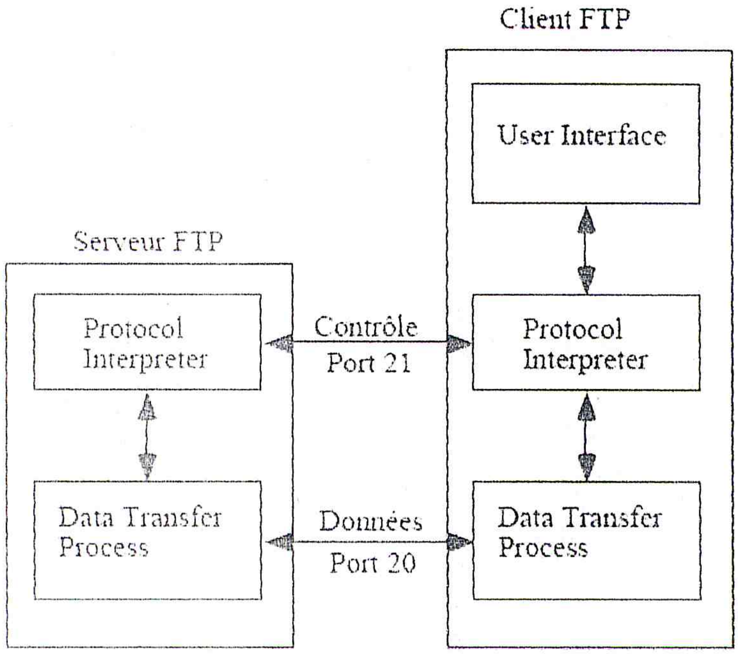


Figure 3.9 : Relation serveur-client FTP.

La réponse est une chaîne de trois chiffres indiquant le statut après traitement (exemple 200 pour commande réussie). Il y a trois groupes de commandes :

- les commandes d'accès

USER : nom de login
PASS : mot de passe
ACCT : compte
CWD : répertoire de positionnement
CDUP : équivalent à cd ..
QUIT : fin de session

- les commandes de paramétrage

TYPE : ASCII (A), EBCDIC (E), Image (I), Local (L), ...
STRU : pour décrire la structure du fichier
MODE : flot continu de données, blocs ou compression

PORT : indique le port ou le DTP serveur doit se connecter (numéro IP suivi de deux octets pour le port)

- les commandes de services

RETR : le DTP serveur envoie le fichier, il est copié sur la machine locale
STOR : le DTP serveur reçoit un fichier, il est copié sur la machine distante
SYST : retourne le nom du système d'exploitation
LIST : équivalent à ls distant
HELP : liste des commandes implémentées dans l'UI
SITE : exécution de commande sur le site distant
DEL : suppression de fichier distant
MKD : création de répertoire distant
RMD : suppression de répertoire distant
NOOP : demande d'accusé de réception

Chaque groupe de commande est décrit par un automate d'état fini. On trouvera figure 3.10, l'automate décrivant le protocole associé à la séquence login et tableau 4, la correspondance avec les codes de signalisation.

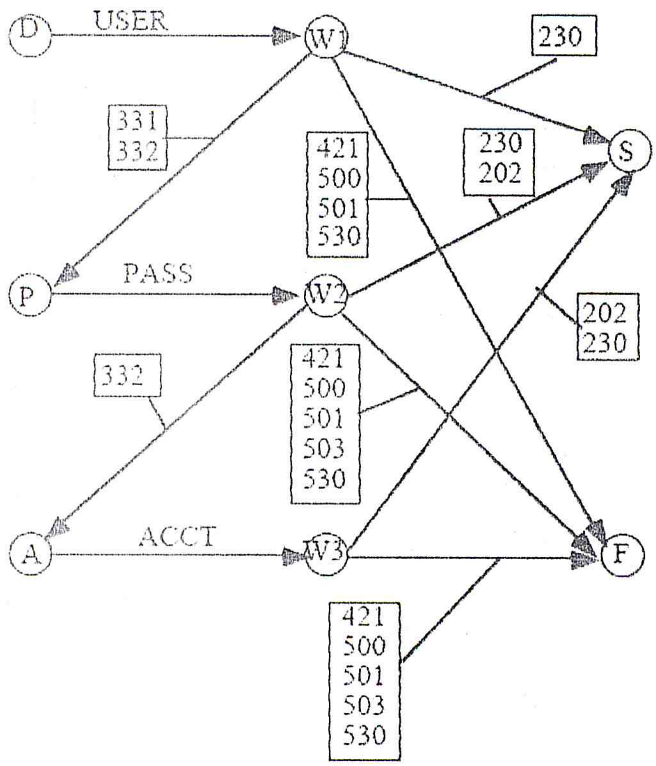


Figure 3.10 : Protocole associé au login.

Tableau 4 : Codes de signalisation.

202	Commande n'est pas implémentée, superflue à ce site
230	Utilisateur enregistré, procéder
331	Nom utilisateur ok, besoin de mot de passe
332	Besoin d'un compte pour début de session

	(login)
421	Service n'est pas disponible, fermer la connexion de contrôle
500	Erreur de syntaxe, commande n'est pas reconnue
501	Erreur de syntaxe dans les paramètres ou les arguments
503	Mauvaise séquence de commandes
530	N'est pas enregistré

Exemple :

[maylis] ~ > ftp -dv firmin
Connected to firmin.
220 firmin FTP server (SunOS 4.1) ready.
Name (firmin:maylis):
---> USER maylis
331 Password required for maylis.
Password:
---> PASS <mon_mot_de_passe>
230 User maylis logged in.
ftp> get toto
---> PORT 147,210,8,138,7,0
200 PORT command successful.
---> RETR toto
150 ASCII data connection for toto (147.210.8.138,1792) (392 bytes).
226 ASCII Transfer complete.
local: toto remote: toto
405 bytes received in 0.036 seconds (11 Kbytes/s)
ftp> put toto
---> PORT 147,210,8,138,7,1
200 PORT command successful.
---> STOR toto
150 ASCII data connection for toto (147.210.8.138,1793).
226 ASCII Transfer complete.
local: toto remote: toto
405 bytes sent in 0.0018 seconds (2.2e+02 Kbytes/s)
ftp> quit
---> QUIT
221 Goodbye.

[FIN]

IV.3. Protocole SMTP :

Le courrier électronique au sein d'Internet est géré par le protocole **SMTP** (Simple Mail Transfer Protocol) bâti sur TCP (port 25). Il permet d'échanger des messages entre un expéditeur et un (ou plusieurs) destinataire pourvu que leurs adresses soient connues. Une adresse de courrier électronique se présente sous la forme nom@domaine est doit être composée de lettres (minuscules ou majuscules sont indifférenciées), de chiffres, de _ (souligné) et de . (point). Il est à noter qu'un mécanisme d'alias permet de définir des équivalences entre adresses, notamment de préciser quelle machine parmi toutes celles d'un même domaine gère réellement le courrier de chaque utilisateur.

Une des caractéristiques principales du protocole SMTP est d'effectuer une remise différée du courrier qui assure que le service sera correctement rendu même si le réseau ou l'ordinateur destinataire sont momentanément en panne ou surchargés.

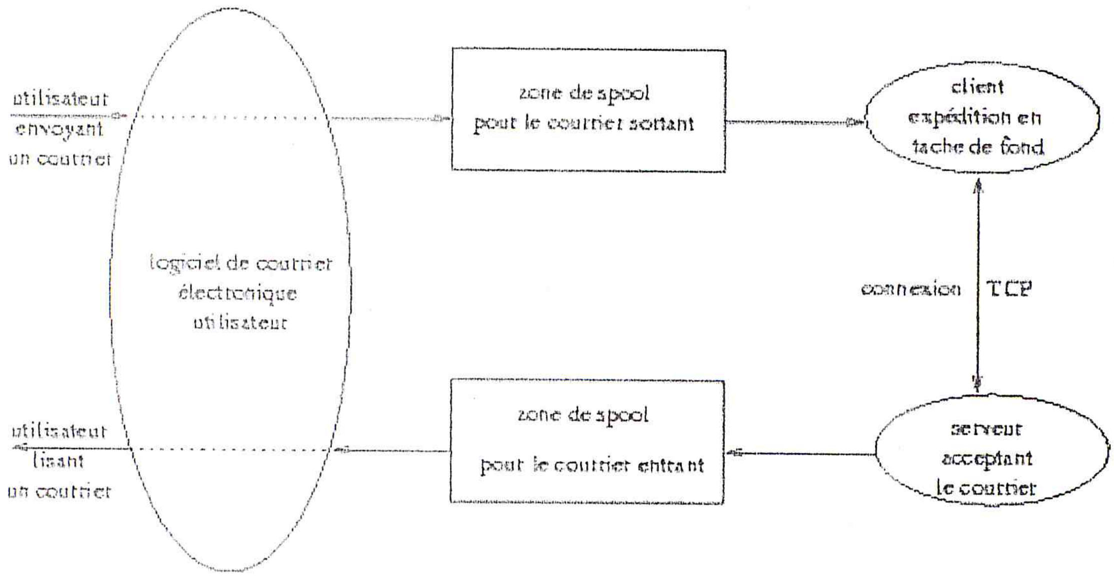


Figure 3.11 : Schéma d'une messagerie SMTP.

Pour cela le système de messagerie fonctionne de la manière décrite en figure 3.11. Un courrier expédié par un utilisateur est d'abord copié dans une mémoire de *spool* accompagné des noms de l'expéditeur, du récepteur, de l'ordinateur destinataire et de l'heure de dépôt. Puis le système de messagerie active en tâche de fond le processus de transfert de courrier qui devient un client. Il associe le nom de l'ordinateur destinataire à une adresse IP et tente d'établir une connexion TCP avec le serveur SMTP de celui-ci. Si cela réussit, le processus de transfert envoie une copie du message au destinataire qui l'enregistre dans une zone de spool spécifique. Lorsque le client et le serveur se sont confirmés l'envoi et l'enregistrement complet du message le client supprime sa copie locale.

Si le client n'arrive pas à établir une connexion TCP, ou si elle est rompue lors du transfert d'un message, il enregistre l'heure de cette tentative et réessaye quelque temps plus tard d'expédier le message. D'une manière générale un système de messagerie examine régulièrement sa zone de spool en envoi et tente d'expédier les messages (nouveau ou en attente à cause d'échec) qui s'y trouvent. Il finira par retourner à son expéditeur un message impossible à expédier après un délai important. Ce mode de fonctionnement (établir une connexion de bout en bout) assure qu'aucun message ne peut se perdre, soit il est délivré, soit son expéditeur est prévenu de l'échec.

Le tableau ci-dessous donne le détail d'une connexion TCP réussie qui envoie un message de l'utilisateur toto@expediteur.dz dont le courrier est géré par l'ordinateur exp.expediteur.dz vers l'utilisateur titi@destinataire.dz dont le courrier est géré par l'ordinateur dest.destinataire.dz. La première colonne décrit les étapes, la deuxième (respectivement troisième) colonne indique les commandes envoyées par l'expéditeur (respectivement destinataire) du courrier.

Tableau 5 : Etapes d'une messagerie SMTP

	client SMTP expéditeur sur exp.expediteur.dz	serveur SMTP destinataire sur exp.destinataire.dz
exp.expediteur.dz demande une connexion TCP sur le port 25 à exp.destinataire.dz		
dest accepte la demande de connexion		220 dest.destinataire.dz...
exp s'identifie	HELO exp.expediteur.dz	
dest accepte l'identification		250 dest.destinataire.dz Hello exp.expediteur.dz pleased to meet you
exp indique l'expéditeur	MAIL From:<toto@expediteur.dz>	
dest accepte l'expéditeur		250 <toto@expediteur.dz> Sender Ok
exp donne le destinataire	RCPT To:<titi@destinataire.dz	
dest a vérifié et accepté le destinataire		250 <titi@destinataire.dz> Recipient Ok
exp va envoyer les données	DATA	
dest est prêt à accepter le message		354 Enter mail, end with...
exp envoie le message terminé par une ligne ne contenant qu'un point.	bla, blabla	
dest accepte le message		250 OK
exp demande à terminer la connexion	QUIT	
dest accepte de terminer la connexion		221 dest.destinataire.dz closing connection

[PNI 99]

IV.4. Serveur de nom de domaine :

Les humains préférant les lettres aux chiffres, le service de nommage est un service qui permet d'associer les numéros INTERNET à des adresses logiques.

Ainsi, 147.210.x.x identifie de manière unique le réseau u-bordeaux.fr. Le nom et le numéro doivent pouvoir être utilisés indifféremment. Le serveur de nom de domaine est attaché au port 53.

L'ensemble de ces informations est géré par un organisme basé à l'Université de Standford, le NIC. Il est géré par trois organismes :

- Network Solutions Inc pour l'enregistrement,
- AT&T pour les répertoires,
- General Atomics/CERFnet pour les informations.

IV.4.1. Principe du service de nommage DNS :

Ce service est basé sur une base de donnée répartie. L'intérêt est de s'assurer que toute l'information sera "disséminable" mais donnée seulement aux sites qui en ont besoin. Le DNS a un domaine racine qui est constitué d'un groupe de serveurs comme le montre l'arborescence figure 3.12.

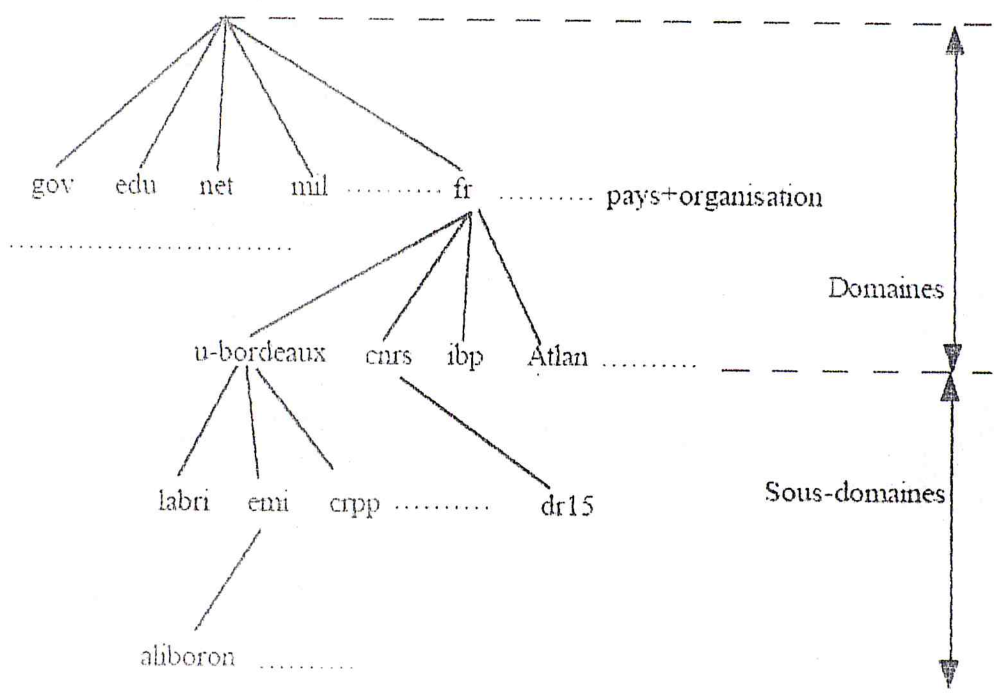


Figure 3.12 : **Les domaines Internet.**

Chaque domaine racine a en charge des domaines qui ont eux mêmes en charge des sous-domaines. On dit qu'un domaine reçoit la délégation d'un autre lorsqu'il reçoit l'autorisation de le gérer.

Par exemple, l'INRIA a délégation pour gérer le domaine .fr, REAUMUR a délégation pour gérer u-bordeaux.fr. Cela signifie par exemple pour REAUMUR que le domaine u-bordeaux.fr a été enregistré auprès du NIC et que tous les sous-domaines sont attribués par REAUMUR.

Une FQDN (Fully Qualified Domain Name) est une adresse dont tous les champs sont précisés. Le fonctionnement du DNS est de type client-serveur. La partie client s'appelle le *resolver*, c'est une bibliothèque. La partie serveur s'appelle le *name server*, c'est un démon.

Il existe trois types de *name server* :

- primaire, possède les tables à jour d'un domaine,
- secondaire, possède les tables à jours provenant d'un autre serveur,
- cache, possède des tables construites à partir des informations traitées.

IV.4.2. Démarrage d'un DNS :

Pour démarrer un domaine, il faut disposer au moins d'une classe C puis obtenir l'enregistrement et la délégation pour ce nom de domaine. On doit également demander la délégation pour le domaine associé in-addr.arpa qui est constitué par le numéro de la classe dans l'ordre inverse suivi de la mention in-addr.arpa. Ceci sert à constituer une arborescence de nom de numéro INTERNET conforme à l'organisation du nom de domaine afin de faciliter le travail d'archivage du DNS.

Exemple : Pour le labri, le domaine est géré par firmin.labri.u-bordeaux.fr d'adresse 147.210.8. La hiérarchie des numéros est l'inverse de celle des noms. On crée donc un domaine dit reverse 8.210.147.in-addr.arpa qui reflète la hiérarchie des noms.

Il faut toujours deux serveurs pour un domaine si possible non situés sur le même câble physique. Le site Aquarel situé au Conseil Régional fournit par exemple un DNS secondaire pour tous les sites Aquarel qui le souhaitent.

On peut demander l'attribution d'un numéro AS (Autonomous System). Ceci peut permettre d'interconnecter des gros réseaux.

Sur le site lui-même, il faut penser à un plan d'adressage pour le domaine concerné. En effet la plupart des opérations sur les routeurs peuvent s'obtenir à partir de masques. Un plan d'adressage bien fait permettra de :

- simplifier l'administration,
- reconnaître une organisation,
- séparer le trafic,
- isoler facilement les problèmes de routages.

[DES 99]

IV.5. Le protocole HTTP :

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990. La version 0.9 était uniquement destinée à transférer des données sur Internet (en particulier des pages Web écrites en HTML. La version 1.0 du protocole (la plus utilisée) permet désormais de transférer des messages avec des en-têtes décrivant le contenu du message en utilisant un codage de type MIME.

Le but du protocole HTTP est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web (appelé d'ailleurs httpd sur les machines UNIX).

Communication entre navigateur et serveur :

La communication entre le navigateur et le serveur se fait en deux temps :

- Le navigateur effectue une requête HTTP
- Le serveur traite la requête puis envoie une réponse HTTP

En réalité la communication s'effectue en plus de temps si on considère le traitement de la requête par le serveur. Etant donné que l'on s'intéresse uniquement au protocole HTTP, le traitement du côté serveur ne sera pas explicité dans le cadre de ce chapitre.

[BFI 97]

V. CONCLUSION

Les deux protocoles OSI et TCP/IP sont très similaires, dans la mesure où les deux sont des modèles de communication à couche et utilisent l'encapsulation de données. On remarque cependant deux différences majeures :

- TCP/IP regroupe certaines couches du modèle OSI dans des couches plus générales.
- TCP/IP est plus qu'un modèle de conception théorique, c'est sur lui que repose le réseau Internet actuel.

Chapitre 4

Sniffers

Chapitre 4 : SNIFFERS

I. INTRODUCTION

Les réseaux informatiques sont des canaux de communication partagés. Cela veut dire que l'interface réseau d'une station sur un réseau Ethernet voit tous les paquets transmis sur ce segment. Chaque paquet a un en-tête indiquant, entre autre, la station destinataire du paquet. Chaque hôte sur le segment Ethernet doit lire, au moins, l'adresse Ethernet destination pour déterminer si le paquet est destiné à lui ou à un autre hôte sur le réseau. En pratique, la manière la plus simple d'accomplir est de lire le paquet en entier dans un buffer et examiner après l'adresse destination. Si la carte réseau reconnaît son adresse, elle déclenche une interruption vers le pilote Ethernet, permettant au pilote de passer le champ données vers la couche réseau pour être traitée. Si la carte Ethernet ne reconnaît pas son adresse, elle attend le prochain paquet, ce qui écrasera le contenu du buffer et par conséquent le paquet qu'il contenait sera rejeté. La plupart des cartes Ethernet ont une option promiscuous qui est activée en mettant un bit à 1 dans le registre hardware. Une carte qui est en mode promiscuous passera tous les paquets vers le pilote Ethernet sans se soucier de l'adresse destination [BEL 97]. En d'autres termes, elle écoutera tout le trafic et pas seulement celui qui lui est destiné.

Un programme spécialement conçu pour placer une interface réseau en mode promiscuous est appelé Sniffer. Le Sniffer est un dispositif logiciel qui, en plaçant l'interface réseau en mode promiscuous, capte les informations qui transitent sur un réseau. Il peut ainsi intercepter tout le trafic réseau. Ce dernier peut employer n'importe quel protocole : TCP/IP, IPX ou autres. [STA 98]

Historiquement, l'origine du terme « sniffer » revient à la société « Network General », qui avait développé un analyseur de trafic réseau appelé « sniffer ». Très vite, ce produit devient leader du marché des analyseurs du réseau (Network Analyzer) et les gens avaient tendance à appeler tout analyseur réseau Sniffer. [DRU 00]

Les administrateurs réseau utilisent les sniffers pour analyser le trafic réseau et ainsi pouvoir déterminer d'éventuels problèmes sur le réseau. Un responsable de la sécurité peut utiliser plusieurs sniffers, stratégiquement placés à travers le réseau, comme un système de détection d'intrusion. Les sniffers sont très importants dans la mesure où, quand ils sont installés, ils peuvent obtenir : noms d'utilisateur (usernames), mots de passes, numéros de carte de crédits, informations personnelles et d'autres informations -la liste est longue- qui peuvent porter une atteinte grave à la confidentialité du système s'ils sont utilisés par une personne malveillante.

Nous allons voir dans ce chapitre l'architecture d'un sniffer, son fonctionnement, des exemples et la prévention contre d'éventuelles attaques par sniffers.

II. ARCHITECTURE D'UN SNIFFER

Dans ce qui suit nous donnerons une description générale du mécanisme de capture de paquets sous les plates-formes Windows et Unix.

II.1. Architecture sous Windows :

Une grande partie des sniffers sous Windows utilisent WinPcap comme API (Application Programming Interface) pour la capture de paquets du réseau.

II.1.1. Définition :

WinPcap est une architecture qui ajoute à la famille des systèmes d'exploitations Win32 la capacité de capture des paquets sur un réseau local Ethernet en utilisant la carte réseau de la machine. De plus, elle offre aux applications une API de haut niveau qui facilite leur interaction avec les niveaux bas. [DEG 00]

II.1.2. Structure de la pile de capture «capture stack» :

Pour qu'une application capture les paquets transférés sur un réseau, elle doit interagir directement avec la partie hardware du réseau. Pour cette raison le système d'exploitation doit offrir un ensemble de primitives de capture pour recevoir les paquet directement de la carte réseau, et les transfère au programme appelant (en dissimulant l'interaction avec la carte réseau).

La partie du kernel responsable de la capture de paquets doit être rapide et efficace car elle doit être capable de capturer les paquets sur des LANs à haut débit en limitant la perte des paquets lorsque le réseau est chargé, et elle doit aussi consommer le moins possible de ressources système.

L'application de capture au niveau utilisateur reçoit les paquets du système, les interprètes et traite les paquets reçus. Le résultat est donné à l'utilisateur dans une forme compréhensible.

Par exemple, Windump.exe est la partie la plus haute de pile de capture de paquets (packets capture stack) qui est composée d'un module qui s'exécute au niveau utilisateur et d'un autre s'exécutant au niveau du kernel.

La figure 4.1 [DEG 00] montre la structure de la pile de capture à partir de la carte réseau jusqu'au niveau application (WinDump.exe, sniffer).

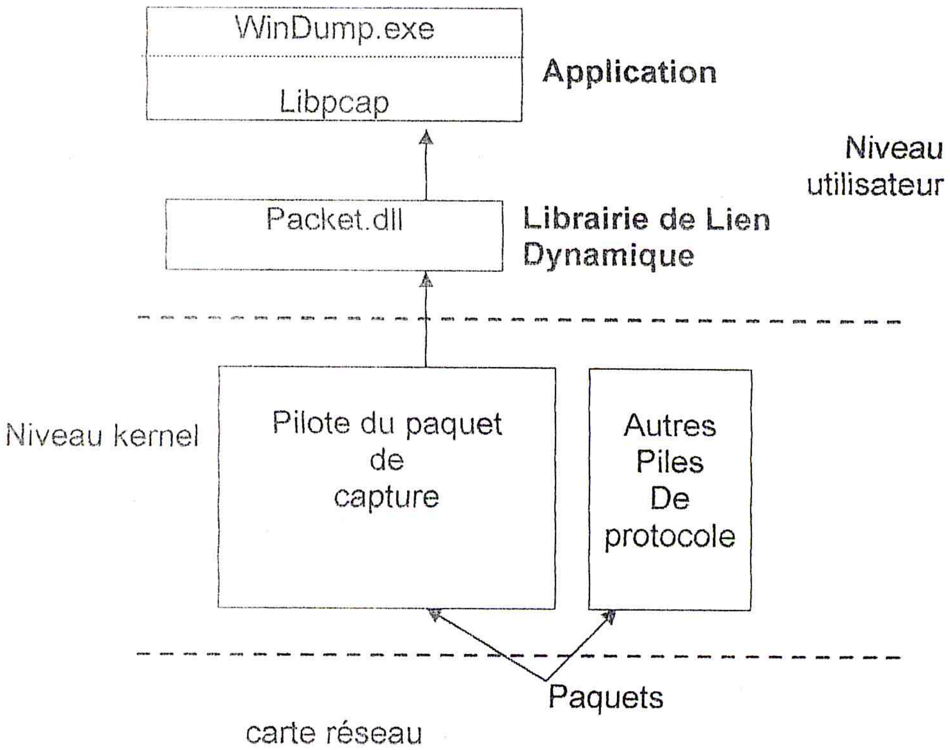


Figure 4.1 : Structure de la pile de capture.

Au niveau le plus bas, nous trouvons la carte réseau. Elle est utilisée pour capturer les paquets qui circulent dans le réseau. Durant une capture, la carte réseau travaille d'habitude dans un mode particulier ('promiscuous mode') qui la force à accepter tout les paquets au lieu de ceux qui sont dirigés seulement vers elle.

Le pilote de capture des paquets est le niveau le plus bas du module logiciel de la pile de capture. C'est la partie qui travaille au niveau kernel et interagie avec la carte réseau pour obtenir les paquets. Il fournit les applications contenant un ensemble de fonctions faisant la lecture et l'écriture de données à partir du réseau au niveau liaison de données.

Packet.dll fonctionne au niveau utilisateur mais elle est séparée du programme de capture. C'est la librairie de lien dynamique qui, en isolant le programme de capture du pilote, va fournir un système indépendant d'interface de capture. Elle permet à WinDump d'être exécuté dans des bouquets différents de Windows sans être recompilé.

La librairie pcap ou libpcap est une librairie statique qui est utilisée par la partie capture du paquet des applications. Elle utilise les services exportés par Packet.dll et fournit aux applications une puissante interface de capture de haut niveau. Prévenir qu'elle est statiquement liée, i.e. elle fait partie de l'application qui l'utilise.

L'interface utilisateur est la partie la plus haute du programme de capture. Elle gère l'interaction avec l'utilisateur et affiche les résultats de la capture.

II.2. Architecture sous Unix :

La plupart des distributions Unix fournissent un mécanisme de capture appelé BPF (BSD Packet Filter). BPF, décrit dans [MCC 92], est une architecture de capture de paquets implémentée dans le noyau d'Unix. BPF est dans l'essentiel un pilote de périphérique qui peut être utilisé par des applications Unix pour lire les paquets du réseau à travers une carte réseau de manière extrêmement optimisée. BPF n'a pas un contrôle direct sur la carte réseau : le pilote de la carte réseau appelle BPF en lui passant les paquets. BPF a deux composantes principales : «Network tap» et «Packet filter».

A) Network tap :

Network tap est une fonction (callback function) qui fait partie du code de BPF mais qui n'est pas exécutée par BPF directement. Elle est appelée par le pilote de la carte réseau lors de l'arrivée d'un nouveau paquet. Le pilote de la carte doit appeler le network tap pour chaque paquet sinon BPF ne fonctionnera pas sur la carte.

Le network tap rassemble les copies des paquets du pilote de la carte réseau et les délivres aux applications en écoute. Les paquets qui arrivent, s'ils sont acceptés par le filtre, sont mis dans un buffer et sont passés à l'application quand le buffer sera plein.

B) Packet filtre :

Le filtre décide si le paquet doit être accepté et copié à l'application en écoute. Le packet filter est une fonction booléenne qui est appliquée à chaque paquet reçu. Si la valeur de la fonction est vraie le kernel copie le paquet à l'application; si c'est faux le paquet est ignoré. Le packet filter ne détermine pas seulement si le paquet doit être gardé, mais aussi le nombre d'octets à garder.

Cette fonctionnalité est très utile si l'application de capture n'a pas besoin du paquet en entier. Par exemple, une application de capture est intéressée par les entêtes et non pas par les données des paquets. Une telle application peut fixer le filtre pour accepter seulement les premiers octets de chaque paquet.

Le processus de capture :

La figure 4.2 illustre l'interaction de BPF avec le reste du système :

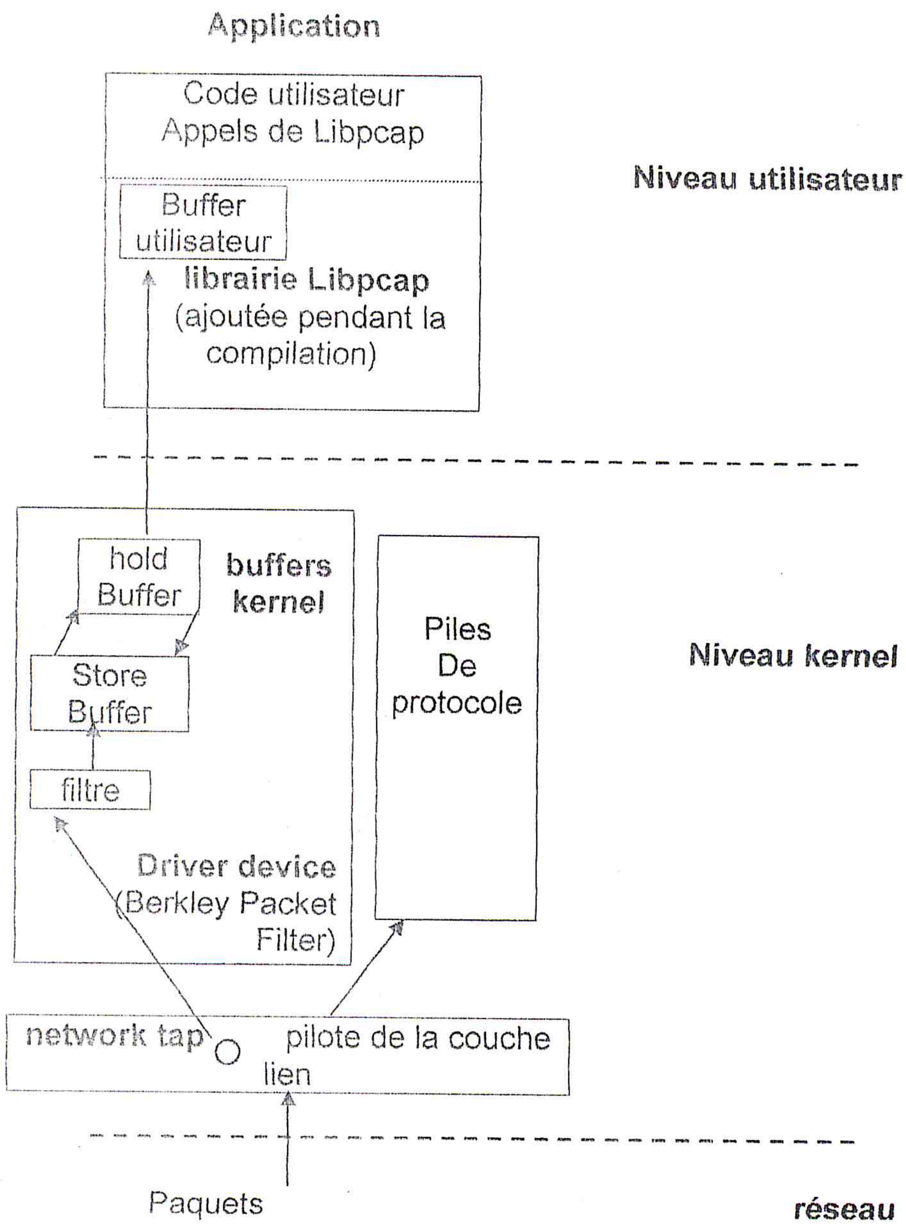


Figure 4.2 : Structure de BPF

BPF associe un filtre et deux buffers à chaque processus de capture demandant ses services. Le filtre est crée par l'application et ensuite passe à BPF à travers un appel IOCTL. Les buffers sont statiquement alloués par BPF et leur dimension est d'habitude 4 KB. Le premier des deux buffers (appelé store buffer) est utilisé pour recevoir les données de la carte, et le second (appelé hold buffer) pour copier les paquets pour l'application. Quand le store buffer est plein et le hold buffer est vide, BPF les troque. Comme ça le processus ne dérange pas le device driver de la carte. Quand un paquet arrive à une interface réseau, le pilote de la couche lien le fait monter au système de pile de protocoles. Mais comme BPF est en train d'écouter cette interface, le pilote commence par appeler la fonction network tap.

Le network tap donne le paquet à chaque application de filtre participante. Ce filtre défini par l'utilisateur décide si un paquet va être accepté et le nombre de bits à enregistrer de chaque paquet. A noter que le filtre est appliqué au paquet pendant qu'il est encore dans la mémoire du pilote de la couche lien, sans le copier. Ceci optimise beaucoup les performances et l'usage de la mémoire, car au cas où le paquet n'est pas accepté, il est abandonné avant n'importe quelle copie. Si le filtre accepte le paquet, le tap copie le nombre d'octets spécifiés par le filtre de la mémoire du pilote de la couche lien au store buffer associé à ce filtre. A ce stade l'interface du device driver re-obtient le contrôle et le processus de protocole normal continue. Un paquet est abandonné si *le store buffer* est plein, et le *hold buffer* n'est pas disponible (i.e. le processus de lecture de ses données du *hold buffer* n'est pas encore terminé).

Le processus fait un appel système de lecture pour recevoir les paquets de BPF. Quand le hold buffer est plein (ou bien quand un timeout s'écoule), BPF les copie vers la mémoire et réveille le processus. Une application peut recevoir plus d'un paquet à la fois. Pour délimiter chaque paquet, BPF encapsule les données de chaque paquet capturé avec une en-tête qui inclue le «time stamp», la longueur et l'offset. Ces données seront utilisées par les applications supérieures.

III. EXEMPLES DE SNIFFERS

De nos jours il existe une multitude de sniffers fonctionnant sur la majorité des plates-formes existantes et disponibles sur Internet. Dans ce qui suit je tenterais d'étudier trois exemples de sniffers.

III.1. Tcpdump :

L'un des plus vieux et des plus communs analyseurs de protocoles TCP/IP. Dans son mode simple, il affiche en ligne de commande le contenu du paquet "déchiffré". Il est considéré comme *le standard des analyseurs de protocoles* utilisés dans l'environnement Unix. Il est disponible à [WTC 01].

Il va permettre de sniffer le réseau, c'est à dire récupérer les paquets circulant sur un segment de votre réseau. Nous traiterons dans cette partie du fonctionnement général du logiciel mais aussi des filtres qui peuvent être utilisés pour rechercher des paquets spécifiques.

L'adage dit que la curiosité est un vilain défaut. Si cela est parfois vrai dans la vie courante, il n'en va pas de même pour l'informatique et en particulier dans le domaine de l'administration réseau. Les manipulations qui vont être décrites ici permettent non seulement la détection d'un éventuel problème physique ou logique, mais également de repérer facilement des problèmes de sécurité trop évidents.

Les données circulant sur un réseau parcourent physiquement, sous la forme d'impulsions électriques, le media utilisé pour relier les machines. Chaque interface fait son travail en n'écoutant que les messages qui lui sont adressés de manière directe ou implicite. De la même manière, lorsqu'une interface envoie sur le réseau une information sous la forme d'un ou plusieurs paquet(s), cette expédition est faite à l'attention d'une machine bien précise. Ceci étant, il n'empêche que les impulsions électriques parcourent l'ensemble du réseau. Il est donc tout à fait concevable qu'une interface puisse écouter passivement tous les paquets qui passent.

Il existe un utilitaire sous GNU/Linux (et pour tous les Unix) qui permet de faire cette manipulation : tcpdump. Cet utilitaire, originellement écrit par Van Jacobson, Craig Leres et Steven McCanne, repose sur la bibliothèque libpcap. Cette dernière est une interface permettant la capture de paquets réseau. Il s'agit là de deux éléments logiciels Libres sous licence BSD. On basera l'ensemble de ce titre sur libpcap version 0.6.1 et tcpdump 3.6.1. Il est conseillé de régulièrement mettre à jour la bibliothèque et l'utilitaire pour plusieurs raisons : chaque version apporte son lot d'extensions et nous avons remarqué un problème de changement de mode de l'interface réseau.

Puisque on parle de mode de l'interface, précisons qu'en temps normal, l'interface ne s'occupe que des paquets qui la concerne. Il faut passer en mode "promiscuous" pour que la carte puisse "écouter" tout ce qui passe. Ne vous inquiétez pas, la libpcap fait cela pour vous.

Commençons tout d'abord avec quelque chose de simple (notez que seul le root peut utiliser tcpdump pour des raisons que vous paraîtront évidentes dans la suite) :

```
# tcpdump
tcpdump: listening on eth0
[...]
```

Je vous informe de la sortie générée sur un réseau d'une trentaine de machines accédant à Internet via une machine NAT (translation d'adresse) et que nous n'abordons pas dans ce travail.

La commande utilisée ainsi, sans aucun paramètre, nous permet tout simplement d'afficher toutes les informations relatives aux paquets qui circulent sur la première interface réseau. Ceci inclut TCP, UDP, IP, ICMP, ... et ce, pour tous les ports. Bref, nous pouvons avoir, selon le réseau, beaucoup de choses à lire. Heureusement, il est possible de filtrer pour n'obtenir que les informations qui nous intéressent.

III.1.1. Filtrage :

Il est possible de sélectionner les paquets à "écouter" en fonction d'expressions. Ainsi, ne seront affichées et/ou traitées que les informations pour lesquelles le résultat de l'expression est vérifié. Une expression est composée d'un mot-clé suivi d'une valeur numérique ou autre. Ainsi, il est possible de classer les mots-clés en catégories :

- Les mots-clés permettant de définir une direction : entrent dans cette catégorie les mots src (source) et dst (destination). Il est possible de combiner ces deux mots-clés avec les opérations logiques *and* (et) et *or* (ou). On peut ainsi définir avec précision la source et/ou la destination des paquets qui nous intéressent.
- Les mots-clés permettant de définir le type de valeur que nous préciserons : il peut s'agir d'un hôte (host), d'un réseau (net) ou encore d'un port (port).
- Enfin, nous avons les mots-clés concernant les protocoles : ici, les mots sont suffisamment clairs pour se passer d'explications (par exemple : IP, TCP, UDP, ...)

Si ce qui précède ne paraît pas limpide, nous allons tâcher de le clarifier avec quelques exemples :

```
# tcpdump src 192.168.0.1
```

Ici, les seuls paquets affichés sont ceux en provenance de 192.168.0.1. Nous pouvons également préciser nos préférences en ajoutant un critère :

```
# tcpdump src 192.168.0.1 and port 80
```

Là, le seul port qui nous intéresse est 80 (http). Si nous poussons le vice, nous pouvons spécifier l'ensemble des paramètres nécessaires :

```
# tcpdump src host 192.168.0.1 and dst host 212.208.225.1 and port 53 and udp
```

Voici une ligne complète qui ne laisse vraiment passer que les paquets qui nous intéressent, c'est à dire en provenance de 192.168.0.1 vers 212.208.225.1, sur le port 53 en UDP. Et ce qui nous intéresse ici, ce sont les requêtes sur le serveur DNS du provider Internet.

Pour faire nos tests, utilisons cette commande (adaptée à nos besoins) sur une console virtuelle et sur une seconde console ; lançons une requête sur le serveur DNS (nslookup www.linux.org par exemple). Nous verrons alors apparaître des lignes ressemblant à ce qui suit sur la première console :

```
tcpdump: listening on eth0

22:33:29.070616 raven.32778 > ns1.rmcnet.dz.domain: 47038+[|domain] (DF)
22:33:29.072863 raven.32779 > ns1.rmcnet.dz.domain: 45315+[|domain] (DF)
22:33:29.128966 raven.32780 > ns1.rmcnet.dz.domain: 47039+[|domain] (DF)
```

Pour poursuivre avec cet exemple, nous allons maintenant utiliser deux des options de la commande tcpdump. Nous devons faire attention, il s'agit d'options qui ne concernent en rien la composition des expressions de filtrage :

```
# tcpdump -x -X -s 0 src host 192.168.0.1 and dst host 212.208.225.1 and port 53
and udp
```

Nous avons demandé l'affichage du contenu des paquets au format hexadécimal et ascii (-x -X) et ce, quelle que soit leur taille (-s 0). Nous obtenons les informations désirées :

```
tcpdump: listening on eth0

20:30:39.189321 raven.32929 > ns1.rmcnet.dz.domain:   5314+ A? www.linux.org.
(31) (DF)
0x0000    4500 003b 0000 4000 4011 ca00 c1fd d9b4    E..;..@.@.......
0x0010    c1fc 1303 80a1 0035 0027 213d 14c2 0100    .......5.'!=....
0x0020    0001 0000 0000 0000 0377 7777 056c 696e    .........www.lin
0x0030    7578 036f 7267 0000 0100 01                ux.org.....
```

Il s'agit de notre requête sur le serveur DNS. Nous pouvons bien sûr inverser source et destination pour obtenir la réponse du serveur :

```
20:40:51.479467   ns1.rmcnet.dz.domain   >   raven.32929:   38062   1/2/2   A
198.182.196.56 (136) (DF)
0x0000  4500 00a4 81b5 4000 f311 94e1 c1fc 1303    E.....@.........
0x0010  c1fd d9b4 0035 80a1 0090 7f6d 94ae 8180    .....5.....m....
0x0020  0001 0001 0002 0002 0377 7777 056c 696e    .........www.lin
0x0030  7578 036f 7267 0000 0100 01c0 0c00 0100    ux.org..........
0x0040  0100 0014 7300 04c6 b6c4 38c0 1000 0200    ....s.....8.....
0x0050  0100 0211 e600 1102 4e53 0849 4e56 4c4f    ........NS.INVLO
0x0060  4749 4303 434f 4d00 c010 0002 0001 0002    GIC.COM.........
0x0070  11e6 0010 034e 5330 0641 4954 434f 4d03    .....NS0.AITCOM.
0x0080  4e45 5400 c03b 0001 0001 0002 11e6 0004    NET..;..........
0x0090  cff5 227a c058 0001 0001 0002 10bb 0004    .."z.X..........

0x00a0       d0ea 0122                             ..."
```

Le résultat obtenu est un peu long mais cela permet de constater combien tcpdump peut être utile.

III.1.2. Récupérer les informations intéressantes :

La récupération des paquets concernant les requêtes DNS est une activité peu informative. Nous pourrions également récupérer les paquets en provenance d'un serveur web et à destination d'un utilisateur sur le réseau local.

Les accès à un serveur POP3 sont du domaine démonstratif. Comme nous l'avons fait pour l'exemple précédent, nous pouvons capturer le contenu de paquets à destination d'un serveur POP3.

```
# tcpdump -x -X -s 0 dst host 192.168.0.10 and src host 192.168.0.1 and port 110
and tcp
```

Le résultat obtenu est le suivant :

```
20:51:50.690745 193.253.217.180.42034 > 193.252.19.209.pop3: P 0:11(11) ack 79
win 5808 (DF)
0x0000   4500 003f 0000 4000 4006 c939 c1fd d9b4        E..?..@.@..9....
0x0010   c1fc 13d1 a432 006e b862 482d 520a 6b11        .....2.n.bH-R.k.
0x0020   8018 16b0 c3cf 0000 0101 080a 00a9 c278        ..............x
0x0030   0bfa 1f75 7573 6572 206f 6b6b 690d 0a          ...uuser.toto..
20:51:50.745300 193.253.217.180.42034 > 193.252.19.209.pop3: P 11:25(14) ack
84 win 5808 (DF)
0x0000   4500 0042 0000 4000 4006 c936 c1fd d9b4        E..B..@.@..6....
0x0010   c1fc 13d1 a432 006e b862 4838 520a 6b16        .....2.n.bH8R.k.
0x0020   8018 16b0 33d0 0000 0101 080a 00a9 c27e        ....3..........~
0x0030   0bfa 1f7a 7061 7373 2079 746b 6b76 7875        ...zpass.coucou
0x0040        0d0a
```

La partie intéressante demeure les deux derniers paquets. On lit clairement dans leur contenu ASCII les mentions "user.toto" et surtout, "pass.coucou". En effet, dans la RFC 1725 décrivant le Protocole Post Office en Version 3 (POP3), on découvre que les informations d'identification et d'authentification circulent de manière non chiffrée entre le client et le serveur. Il serait possible de les récupérer, grâce à tcpdump et un script Perl ou Awk.

[WCO]

III.2. Nmap :

Nmap est le scanner de ports le plus utilisé car il est très puissant. En effet, il dispose d'un nombre impressionnant de fonctionnalités. De plus il est disponible sur un grand nombre de systèmes d'exploitation. Nmap nous sera très utile pour auditer la sécurité de notre réseau mais aussi pour déceler certains dysfonctionnements.

Dans cette partie je vous présenterais les principales fonctionnalités de Nmap. Dans un premier temps je vous ferais découvrir les différentes méthodes de scans. Puis dans un second temps, je détaillerais quelques options intéressantes.

III.2.1. Méthode TCP connect :

Cette méthode utilise l'option -sT.

Fonctionnement :

Pour tester si un port est ouvert ou non, Nmap emploi la fonction connect() du langage C. Elle permet d'initialiser une connexion à une socket. Si le port à tester est ouvert alors l'appel de la fonction connect() fonctionnera sinon cela signifie que le port est fermé. Avec ce type de fonctionnement, on a alors une connexion TCP dite complète (méthode three way handshake). L'inconvénient majeur de cette méthode sera donc une détection assez facile par les pare-feux.

Note :

Il n'est pas nécessaire d'avoir les droits de l'utilisateur root pour utiliser cette méthode.

Exemple :

```
# nmap -sT 192.168.164.80
Starting nmap 3.27 (www.insecure.org/nmap/) at 2003-08-16 13:45 CEST
Interesting ports on nserver (192.168.164.80) :
(The 1617 ports scanned but not shown below are in state : closed)
Port      State    Service
21/tcp    open     ftp
22/tcp    open     ssh
80/tcp    open     http
81/tcp    open     hosts2-ns
111/tcp   open     sunrpc
139/tcp   open     netbios-ssn

Nmap run completed -- 1 IP address (1 host up) scanned in 1.138 seconds
```

Pour chaque port ouvert, Nmap affiche le service qui devrait être lancé sur le port correspondant d'après le fichier nmap-services. Ici Nmap ne vérifie pas si c'est vraiment le service affiché qui tourne sur ce port. En effet, on peut très bien lancer un serveur FTP sur le port 80 qui est habituellement réservé à un serveur HTTP.

III.2.2. Méthode SYN scan :

L'option utilisée cette fois est -sS.

Fonctionnement :

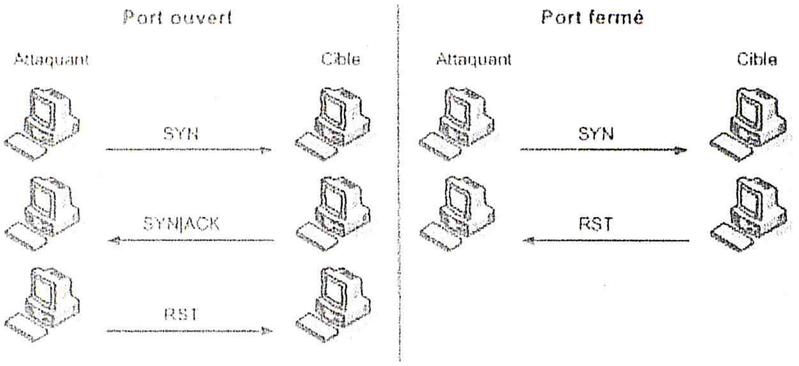


Figure 4.3 : Méthode SYN scan.

Cette méthode consiste à ne pas ouvrir une connexion TCP/IP complète comme précédemment. Cette technique est généralement appelée "half-open" ou scan furtif. Le principe de fonctionnement est le suivant : un flag SYN est envoyé à la station cible et on attend la réponse. Si on reçoit un SYN/ACK cela veut dire que le port est ouvert, sinon le port est fermé si la réponse est un RST. Lorsque Nmap détecte un port ouvert il coupe brutalement la connexion par un RST.

Note :

Il faut obligatoirement être en root pour utiliser cette méthode et celles qui suivront.

Exemple :

```
# nmap -sS 192.168.164.80

Starting nmap 3.27 (www.insecure.org/nmap/) at 2003-08-16 13:52 CEST
Interesting ports on nserver (192.168.164.80):
(The 1617 ports scanned but not shown below are in state : closed)
Port      State     Service
21/tcp    open      ftp
22/tcp    open      ssh
80/tcp    open      http
81/tcp    open      hosts2-ns
111/tcp   open      sunrpc
139/tcp   open      netbios-ssn

Nmap run completed -- 1 IP address (1 host up) scanned in 0.990 seconds
```

Ce type de scan à pour avantage d'être plus rapide et il est moins détectable que la méthode précédente par quelques IDS simplistes. En effet, de nos jours, la majorité des IDS détecte ce type de scan à cause de sa grande popularité.

III.2.3. Méthodes FIN, XMAS et NULL scan :

Ces trois méthodes fonctionnent de la même manière. Elles utilisent respectivement les options :

-sF (FIN scan), -sX (XMAS scan) et -sN (NULL scan).

Fonctionnement :

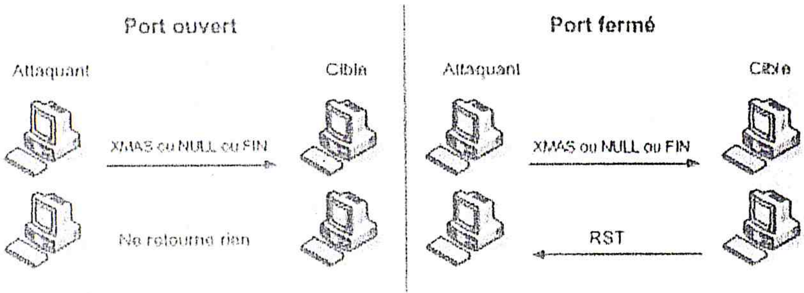


Figure 4.4 : Méthodes FIN, XMAS et NULL scan.

Comme son nom l'indique, la première technique envoie les données avec un drapeau FIN au destinataire. Le scan de type XMAS envoie un datagramme avec les flags FIN, URG et PSH. Et pour finir, le NULL scan transmet à la cible un datagramme sans aucun drapeau. Un port fermé est détecté par la présence d'un RST dans la réponse de la cible.

Exemple :

```
# nmap -sX 192.168.164.80

Starting nmap 3.27 (www.insecure.org/nmap/) at 2003-08-16 14:12 CEST
Interesting ports on nserver (192.168.164.80):
(The 1617 ports scanned but not shown below are in state: closed)
Port       State      Service
21/tcp     open       ftp
22/tcp     open       ssh
80/tcp     open       http
81/tcp     open       hosts2-ns
111/tcp    open       sunrpc
139/tcp    open       netbios-ssn

Nmap run completed -- 1 IP address (1 host up) scanned in 4.325 seconds
```

Le principal inconvénient de cette méthode est qu'elle ne fonctionne pas sous certains systèmes d'exploitation tels que Windows, Cisco, BSDI, HU/UX, MSV et IRIX. Tout simplement parce que ces derniers ne respectent pas les standards établis dans les RFC. En effet, lorsqu'un port est ouvert, ceux-ci renvoient un drapeau RST au lieu de rien du tout. Ces techniques se révèlent aussi plus lentes que les deux précédentes. Il existe aussi des pare-feux bien configurés qui rejettent ces paquets, particulièrement ceux avec le drapeau RST. Cette technique a aussi l'avantage d'être plus ou moins bien détectée par les IDS.

III.2.4. ACK et Window scan :

Ces deux méthodes utilisent respectivement les options -sA et -sW. La principale utilité de cette fonctionnalité est de déterminer le type de pare-feux de la cible, c'est à dire s'il gère le suivi de connexion (stateful) ou non.

Fonctionnement :

Le ACK scan envoie un numéro de séquence aléatoire au port à tester tandis que le Window scan fournit une taille de fenêtre invalide. En retour, si nous obtenons un RST cela indique que le port n'est pas filtré. Il est par contre filtré si on ne reçoit rien ou un ICMP unreachable.

Exemple :

```
# nmap -sW localhost

Starting nmap 3.50 (http://www.insecure.org/nmap/) at 2004-11-23 19:58 CET
Interesting ports on localhost (127.0.0.1):
(The 1658 ports scanned but not shown below are in state: closed)

PORT   STATE    SERVICE
25/tcp filtered  smtp

Nmap run completed -- 1 IP address (1 host up) scanned in 4.173 seconds
```

III.2.5. Scan UDP :

-sU est l'option utilisée pour effectuer un scan de ce type.

Fonctionnement :

Le principe de fonctionnement est simple : il suffit d'envoyer un paquet vide sur le port désiré. Si l'on reçoit un paquet ICMP port unreacheable cela signifie que le port est fermé.

Exemple :

```
# nmap -sU 192.168.164.80

Starting nmap 3.27 (www.insecure.org/nmap/) at 2003-08-16 14:02 CEST
Interesting ports on nserver (192.168.164.80):
```

(The 1465 ports scanned but not shown below are in state: closed)

```
Port      State     Service
111/udp   open      sunrpc
137/udp   open      netbios-ns
138/udp   open      netbios-dgm
2049/udp  open      nfs
32770/udp open      sometimes-rpc4
32771/udp open      sometimes-rpc6
```

Nmap run completed -- 1 IP address (1 host up) scanned in 1471.748 seconds

En observant l'exemple, nous remarquons que cette méthode est extrêmement lente comparée à un scan TCP. En effet, les systèmes d'exploitation imposent une limite sur le nombre d'erreurs ICMP envoyées par seconde. Par exemple, Linux limite les messages ICMP destination unreachable à 80 toutes les 4 secondes. Solaris est encore plus strict avec 2 messages par seconde seulement. Nmap adapte donc automatiquement sa vitesse de scan pour éviter de flooder le réseau avec des paquets qui seraient rejetés par la cible. Un point intéressant à noter : avec les systèmes d'exploitation qui ne respectent pas les normes, comme Windows, nous pouvons les scanner très rapidement.

III.2.6. Idle Scanning :

Le but de cette technique est de scanner la cible en restant anonyme vue de la cible. Pour cela nous utilisons un hôte tierce qui servira d'intermédiaire. La méthode consiste alors à utiliser l'IP spoofing et l'observation de l'IPID de la machine tierce. Cette technique est donc très intéressante pour une personne mal intentionnée puisqu'elle est difficile à tracer et permet d'obtenir des informations sur la cible sans jamais être en contact direct avec celle-ci.

Fonctionnement :

Cette technique se déroule en trois phases : il faut d'abord trouver un zombie (machine intermédiaire) convenable, puis envoyer les paquets à la cible mais avec l'adresse source l'IP du zombie et enfin récupérer l'IPID du zombie et en déduire l'état du port.

Etape 1 : Trouver une machine témoin et récupérer son ID

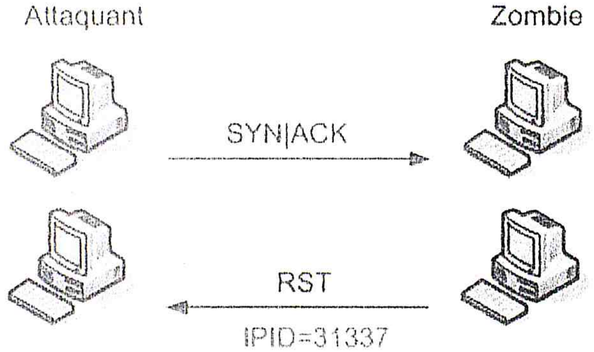


Figure 4.5 : Etape 1 du Idle Scanning.

La machine tierce doit posséder un port ouvert et ne doit pas générer trop de trafic pour ne pas altérer l'incrémentation du champ d'identification du paquet IP. Il faut aussi que le système d'exploitation implémente de façon incrémentale l'IPID. C'est donc en observant le champ ID que nous en déduirons si un port est ouvert ou non sur la cible. Il faut alors dans un premier temps récupérer la valeur de l'ID actuel de la machine témoin. Pour cela, Nmap envoie un SYN|ACK à la machine tierce, celle-ci répond normalement par un RST, ce qui nous permet de récupérer l'ID.

Etape 2 : Envoyer un paquet spoofé avec l'IP du zombie

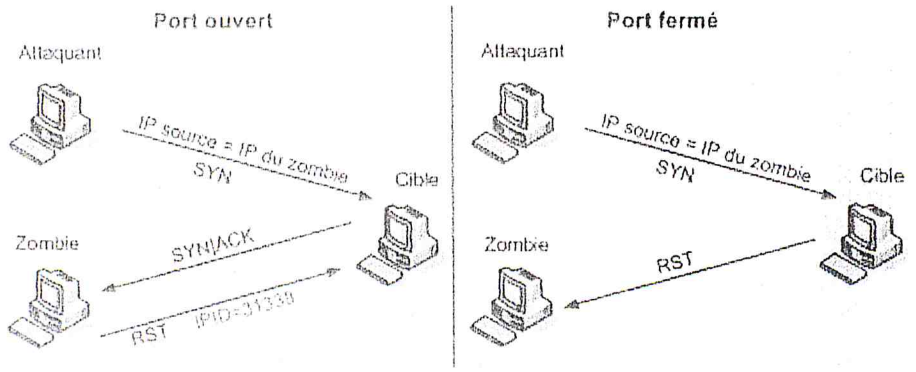


Figure 4.6 : Etape 2 du Idle Scanning.

L'attaquant envoie un SYN sur le port à tester avec l'IP du zombie en adresse source. Nous avons alors deux cas :

- dans le cas d'un port ouvert, la cible répond par un SYN|ACK au zombie. Celui-ci n'ayant rien demandé à la cible va lui retourner un RST et incrémenter son champ ID. C'est grâce à cela que l'on va savoir que le port est ouvert.

- dans le cas d'un port fermé, la cible répond par un RST et si notre zombie respecte les RFC il ne devrait pas répondre. Ainsi son champ ID ne sera pas incrémenté et il aura la même valeur que la dernière fois.

Etape 3 : Interroger le zombie et en déduire l'état du port

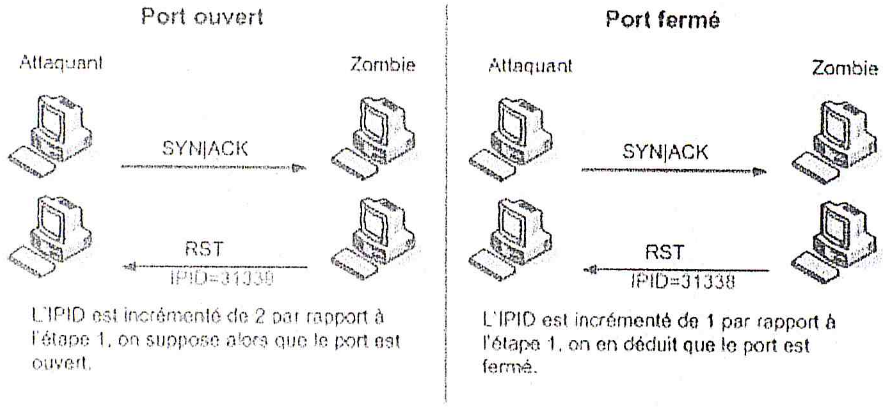


Figure 4.7 : Etape 3 du Idle Scanning.

Pour finir, Nmap interroge de la même manière que la première fois le zombie pour récupérer l'IPID. Ainsi, selon la valeur de l'incrémentation, Nmap supposera que le port est ouvert ou non.

Exemple

```
# nmap -P0 -sI 192.168.164.20:139 10.0.0.1

Starting nmap 3.75 (http://www.insecure.org/nmap/) at 2004-11-02 03:16 CET
Idlescan using zombie 192.168.164.20 (192.168.164.20:139); Class: Incremental
Interesting ports on 10.0.0.1:
(The 1650 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh

Nmap run completed -- 1 IP address (1 host up) scanned in 46.082 seconds
```

Dans cet exemple j'ai pris le port 139 de la machine zombie 192.168.164.20. La cible étant 10.0.0.1, l'option -P0 permet de ne pas pinguer la cible avant le scan. Pour conclure sur cette méthode, nous pouvons dire qu'elle a l'avantage d'être anonyme mais se base sur trop de suppositions pour nous informer sur l'état d'un port.

III.2.7. RPC scan :

Fonctionnement :

Le principe est assez simple, Nmap envoie aux ports ouverts des commandes SunRPC afin de récupérer le nom et la version des programmes qui tournent sur ces ports. Cette méthode s'utilise en la couplant à d'autres techniques de scans (UDP ou TCP). L'option à utiliser ici est -sR.

Exemple :

```
# nmap -sTUR 192.168.164.80

Starting nmap 3.27 (www.insecure.org/nmap/) at 2003-08-16 14:35 CEST
Interesting ports on nserver (192.168.164.80):
(The 3082 ports scanned but not shown below are in state: closed)
Port       State     Service (RPC)
21/tcp     open      ftp
22/tcp     open      ssh
80/tcp     open      http
81/tcp     open      hosts2-ns
111/tcp    open      sunrpc (rpcbind V2)
111/udp    open      sunrpc (rpcbind V2)
137/udp    open      netbios-ns
138/udp    open      netbios-dgm
139/tcp    open      netbios-ssn
2049/udp   open      nfs (nfs V2-3)
32770/udp  open      sometimes-rpc4 (nlockmgr V1-4)
32771/udp  open      sometimes-rpc6 (mountd V1-3)

Nmap run completed -- 1 IP address (1 host up) scanned in 1479.600 seconds
```

Nous avons couplé cette méthode avec le TCP scan et l'UDP scan. Le principal inconvénient pour l'attaquant est que cette technique est voyante : les tentatives s'affichent clairement dans les logs.

III.2.8. Détection du système d'exploitation :

Fonctionnement :

Nmap permet de détecter sur une machine cible la version de son système d'exploitation. Pour cela, Nmap exploite les divergences d'implémentations de la pile IP des différents OS. En combinant toute une batterie de tests et en comparant leurs résultats à sa base de donnée interne, il est capable de déterminer dans la majorité des cas la version de l'OS. Il est aussi possible que Nmap ne puisse pas trouver la version car l'OS-cible dispose d'une pile IP qui a été sécurisée (cf. Grsecurity ou OpenBSD).

III.2.9. Les balayages :

Dans cette section, je vais détailler les différentes techniques utilisées par Nmap pour scanner plusieurs machines en même temps.

1) ICMP ECHO ping sweep

Fonctionnement :

Le fonctionnement est assez simple : un simple ping est envoyé, c'est à dire une requête ICMP avec un "echo request". Cependant, cette méthode n'est pas toujours utilisable. Et pourtant, il est très simple de bloquer ce type de requête ICMP en utilisant n'importe quel type de pare-feu. En revanche, la méthode reste assez rapide pour tester la présence d'un ensemble de machines dans un réseau.

Exemple :

```
# nmap -sP -PI 192.168.164.[1-20]

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-03-06 19:28 CET
Host 192.168.164.1 appears to be up.
MAC Address: 00:60:B0:EC:E9:69 (Hewlett-packard CO.)
Host 192.168.164.2 appears to be up.
MAC Address: 00:0C:41:9D:42:4F (The Linksys Group)
Host 192.168.164.13 appears to be up.
MAC Address: 00:80:5F:F7:CD:02 (Compaq Computer)
Host 192.168.164.17 appears to be up.
Host 192.168.164.19 appears to be up.
MAC Address: 00:0C:41:61:D4:BE (The Linksys Group)
Nmap run completed -- 20 IP addresses (5 hosts up) scanned in 0.811 seconds
```

2) TCP ACK sweep

Exemple :

```
# nmap -sP -PT 192.168.164.[1-20]

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-03-06 19:28 CET
Host 192.168.164.1 appears to be up.
MAC Address: 00:60:B0:EC:E9:69 (Hewlett-packard CO.)
Host 192.168.164.2 appears to be up.
MAC Address: 00:0C:41:9D:42:4F (The Linksys Group)
Host 192.168.164.13 appears to be up.
MAC Address: 00:80:5F:F7:CD:02 (Compaq Computer)
Host 192.168.164.17 appears to be up.
Host 192.168.164.19 appears to be up.
```

MAC Address: 00:0C:41:61:D4:BE (The Linksys Group)
Nmap run completed -- 20 IP addresses (5 hosts up) scanned in 0.818 seconds

3) TCP SYN sweep

Exemple
nmap -sP -PS 192.168.164.[1-20]

Starting nmap 3.75 (http://www.insecure.org/nmap/) at 2005-03-06 19:29 CET
Host 192.168.164.1 appears to be up.
MAC Address: 00:60:B0:EC:E9:69 (Hewlett-packard CO.)
Host 192.168.164.2 appears to be up.
MAC Address: 00:0C:41:9D:42:4F (The Linksys Group)
Host 192.168.164.13 appears to be up.
MAC Address: 00:80:5F:F7:CD:02 (Compaq Computer)
Host 192.168.164.17 appears to be up.
Host 192.168.164.19 appears to be up.
MAC Address: 00:0C:41:61:D4:BE (The Linksys Group)
Nmap run completed -- 20 IP addresses (5 hosts up) scanned in 0.811 seconds

III.2.10. Techniques diverses :

On peut citer la fragmentation, le Decoy scan, le réglage de la vitesse du scan et les scans parallèles.

[WNE 05]

III.3. Etherpeek :

EtherPeek est un puissant analyseur de réseau. Il est disponible en environnement MacOS et Windows. Il permet la surveillance du réseau et le diagnostic des incidents. Le logiciel EtherPeek est un analyseur pour réseau Ethernet puissant et intuitif. Il permet : l'expertise en temps réel sur plusieurs cartes réseau simultanément, l'analyse du temps de réponse des applications, le décodage complet des protocoles, la création de paquets, les alarmes, le déclenchement, l'arrêt automatique de captures, la surveillance et la création de rapports. Il offre :

- une interface utilisateur de qualité qui permet de localiser et de résoudre les problèmes rapidement,
- un système précis qui détaille les causes des problèmes et propose des solutions pour l'ensemble du réseau,
- l'analyse et le contrôle du respect de la politique de sécurité de l'entreprise,
- l'analyse de réseaux distants,
- des statistiques instantanées et historiques qui permettent de dégager des tendances d'évolution,

- une capture simultanée sur plusieurs cartes réseaux et différents segments du réseau,
- un affichage des paquets en temps réel et une analyse experte lors de la capture,
- la possibilité de 'rejouer' les incidents pour mieux les comprendre et les analyser.

III.3.1. Les principaux avantages :

a. Système expert :

- ➢ Identification du problème
- ➢ Evaluation et analyse des 7 couches du modèle OSI
- ➢ Etude par conversation
- ➢ Personnalisable

b. Diagramme des communications :

- ➢ Représentation visuelle des communications
- ➢ Isolement rapide du trafic et des conversations d'un utilisateur ou d'un serveur

c. Surveillance :

- ➢ Décompte en temps réel
- ➢ Création de graphes

d. Création de rapports au format HTML et XML.

e. Alarmes sur plus de 100 conditions détectées en temps réel.

f. Déclencheurs :

Permettent de lancer automatiquement une capture en fonction du contenu d'un paquet.

g. Multiples fenêtres de captures ayant chacune des filtres/ graphes/ experts indépendants permettant de suivre plusieurs conversations en temps réel.

h. Filtrage et sélections automatiques des paquets liés ou identiques.

i. Décodage intégral des paquets du réseau.

j. Création et envoi de paquets

k. Kit pour le développement de nos propres plug-in

III.3.2. Dépannage temps réel :

Etherpeek est le premier outil d'analyse des réseaux qui propose à la fois des diagnostics d'expert et un décodage des trames en temps réel au cours de la capture. L'affichage expert offre une analyse détaillée du délai de transit, du débit et de plus de 85 problèmes relatifs au réseau (présentation par conversation). Nous pouvons adapter le système expert à notre environnement réseau : de nombreux réglages sont disponibles.

Il compte plus d'experts TCP que n'importe quel autre outil d'analyse. Il permet d'obtenir une analyse aussi concise que possible du trafic quel que soit son volume. Il est doté d'un système expert. Cette technologie intelligente, incluant la localisation et l'analyse des problèmes, est basée sur la capture des trames et le décodage des 7 couches OSI mais aussi sur l'étude des conversations entre les ordinateurs.

L'interface utilisateur est bien plus qu'un simple habillage. Elle permet de tirer profit de toute la puissance d'Etherpeek, par exemple :

- Depuis l'onglet 'diagnostics expert' nous pouvons directement sélectionner les paquets s'y rapportant ; cela permet de réduire le temps nécessaire à la localisation et à la résolution d'un problème.

- Les diagnostics sont regroupés par conversations pour optimiser la résolution des problèmes.

- Le système expert fournit des explications détaillées au sujet des problèmes, des causes probables et des solutions envisageables.

- Quelques clics de souris suffisent à mettre en place des filtres en temps réel lors de la capture, ainsi nous explorons plus facilement les problèmes spécifiques de dépannage.

III.3.3. Analyse du temps de réponse des applications :

Le système expert d'Etherpeek analyse les problèmes client/serveur au niveau de la couche applicative (réseaux et serveurs occupés, clients inefficaces, faible débit et délai de transit).

Afin d'évaluer rapidement les performances générales du réseau et d'identifier facilement les problèmes au niveau des applications ou des communications client/ serveur, Etherpeek calcule les statistiques d'envoi et de réception de l'ensemble des stations (moyennes et pourcentages). Le temps de réponse (délai de transit) et le débit de chaque conversation sont analysés de manière indépendante.

III.3.4. Contrôle de la politique de sécurité :

Les réseaux assurent le transport de données de valeur relatives à la propriété intellectuelle, des rapports financiers confidentiels et des données clientèle qui font le succès d'une entreprise. Etherpeek dispose de puissantes fonctions d'analyse de la sécurité qui aident les ingénieurs à maintenir leur réseau au niveau requis par les normes de sécurité propres à l'entreprise. Il permet de détecter des failles dans une installation, de contrôler l'intégrité des données, de répondre automatiquement aux menaces portant atteinte à la sécurité, de détecter, d'analyser et de classer les accès non autorisés.

III.3.5. Dépannage à distance :

PacketGrabber (livré en standard avec Etherpeek) optimise la capacité à dépanner les réseaux de l'entreprise en rassemblant facilement les fichiers de captures collectés sur des stations de travail situées sur des segments distants. Les fichiers de captures peuvent ensuite être envoyés à un ingénieur réseau, via un courrier électronique, un serveur FTP ou un volume partagé, afin d'obtenir une expertise avec Etherpeek qui est capable de rejouer une capture.

RMONGrabber permet aux utilisateurs d'Etherpeek de dépanner de manière interactive des réseaux distants de la même manière et avec les mêmes fonctions puissantes que lors de l'utilisation locale du logiciel.

III.3.6. Analyse de différents segments :

Etherpeek offre la possibilité d'ouvrir simultanément plusieurs fenêtres de capture temps réel, utilisant chacune une carte réseau différente. Il est possible de créer une nouvelle fenêtre de capture permettant de se focaliser sur des éléments spécifiques du trafic détectés dans une capture globale. Chaque fenêtre dispose de ses propres filtres, de son propre expert et de son propre diagramme de communications.

SURVEILLANCE ET CREATION DE RAPPORTS ->

Le diagramme des communications d'Etherpeek permet de visualiser l'ensemble des activités du réseau d'un seul coup d'œil. Le diagramme des communications, présenté sous la forme d'une ellipse orientée verticalement, permet d'afficher l'ensemble des nœuds du réseau. Cette présentation permet à l'utilisateur de visualiser immédiatement l'importance des échanges entre chacun des postes :

- plus la ligne entre les nœuds est épaisse et plus le trafic est important,
- plus le point est gros et plus le trafic émis et reçu par ce nœud est important,
- sélection automatique des 10, 50, 100 principaux nœuds,
- identification des nœuds en fonction de l'adresse physique, IP ou IPX,
- sélection du type de trafic (tout, unicast, multicast, broadcast),
- un glisser- déposer permet de masquer des nœuds.

C'est le seul outil d'analyse du réseau à couvrir un nombre aussi grand de statistiques relatives aux applications, aux protocoles sur le réseau Ethernet. Les statistiques sont converties en graphiques en temps réel ou en alarmes d'un clic droit de la souris. Il est également possible d'associer les statistiques à des graphiques personnalisés complexes. L'utilisateur dispose ainsi d'une étonnante flexibilité lors de la surveillance des activités du réseau.

Grâce aux fonctions permettant d'établir des graphiques et des tendances complexes, il est possible de collecter, d'afficher et de créer un rapport de statistiques relatives aux nœuds, protocoles ou résumés disponibles sur une période de temps définie par l'utilisateur. Ces données peuvent être exportées au format CSV pour une importation dans d'autres applications (gestion, traçabilité, qualité).

III.3.7. Technologie innovante :

Sa technologie innovante fait d'Etherpeek une solution flexible, adaptée aux exigences des professionnels et dotée de fonctionnalités uniques.

Le système de notification avancé avec seuil d'alarme comprend des dizaines de situations en temps réel ou post-capture. La création de nouvelles alarmes et le suivi statistique sont réalisés en quelques clics.

Les diagnostics experts ont été conçus par des spécialistes réseau de haut niveau, ingénieurs, consultants seniors et des formateurs à l'analyse de paquets qui disposent d'une longue expérience.

Les règles du système expert d'Etherpeek sont issues du savoir-faire acquis sur de nombreux cas réels. Cette expertise s'applique immédiatement au trafic et aux problèmes que nous rencontrons sur le réseau.

Les mises à jour fournies par l'éditeur augmentent régulièrement la puissance, la richesse fonctionnelle et l'efficacité du logiciel.

UTILISABLE POUR ->

- Analyse, supervision et optimisation.
- Hotline et dépannage.
- Tests et recette logiciels.
- Détection des failles de sécurité.

[WME]

IV. LES RISQUES LIES AU SNIFFER

Les risques que font courir les sniffers sont élevés. En fait, la présence sur un réseau d'un sniffer placé par une personne non autorisée implique déjà que la sécurité a été violée. Etendons notre champ de vision au-delà du LAN pour englober Internet : il est possible alors qu'une personne de l'extérieur ait placé un sniffer sur le réseau (en utilisant un cheval de Troie par exemple) ou qu'un individu en interne manigance quelque chose (n'oublions pas que plus de 50% des attaques proviennent de l'intérieur). Dans les deux cas, le réseau est en situation grave.

De nombreuses attaques par sniffer se sont déjà produites sur Internet. Celles-ci étaient variées en termes de cible et de portée. Dans un extrait d'un rapport de sécurité LANT du Naval Computer & Telecommunication Area Master Station on peut lire : *«En février 1994, une personne non identifiée a installé un sniffer sur de nombreux hôtes et éléments du réseau fédérateur, recueillant plus de cent mille noms utilisateurs et mots de passes valides via Internet...»* [LAN 95]

Naturellement, les instituts et les sociétés privées sont réticents à communiquer le nombre de violations dont ils ont fait l'objet. On compte parmi les victimes connues :

- L'université de l'état de Californie à Stanislaus.
- Un laboratoire de recherche sur les missiles de l'US Army.
- Le centre White Sands Missile Range.

V. LES ATTAQUES PAR SNIFFER

Il faut noter qu'un sniffer écoute tout le trafic qui passe à travers le LAN, et pas seulement le trafic échangé localement entre les hôtes du LAN, comme illustré par la figure 4.8 :

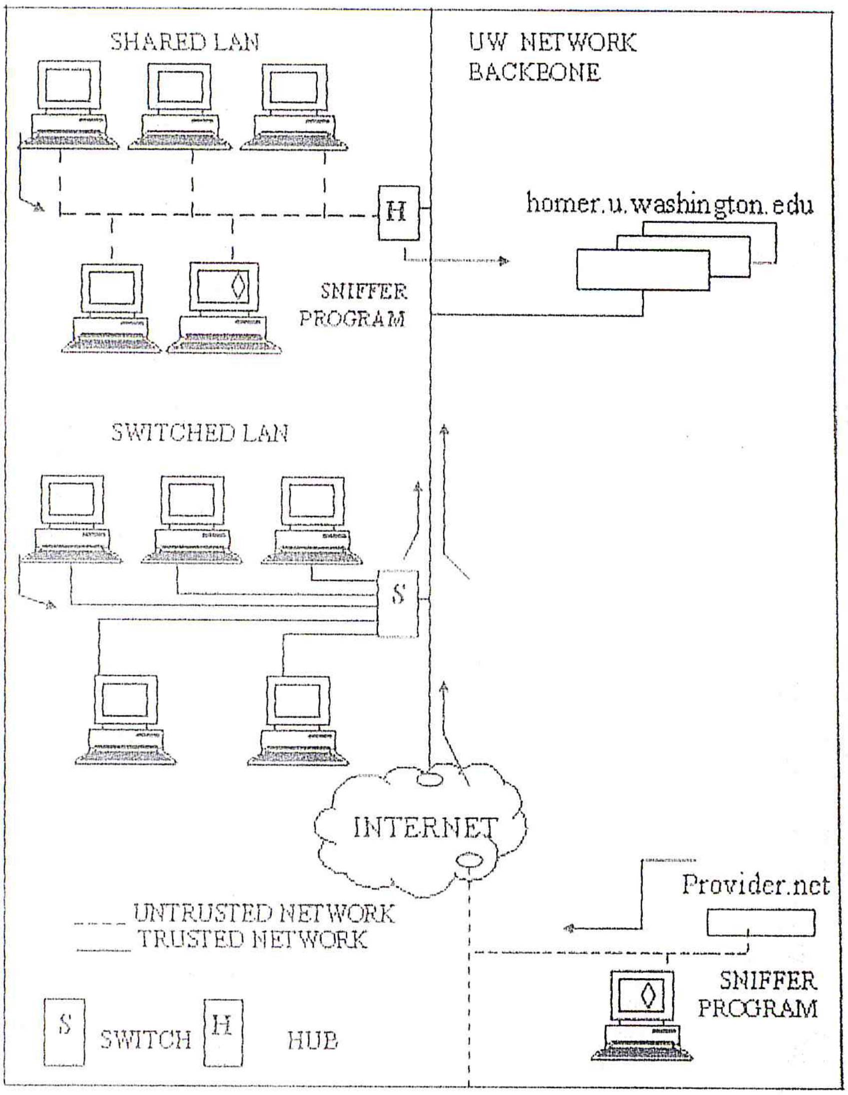


Figure 4.8 : Trafic écouté par un sniffer sur un réseau.

Sur la figure 4.8, le trafic passant par les segments en pointillés est écouté par le sniffer. Une attaque par sniffer se déroule généralement en deux phases : [GRU 98]

V.1. PHASE 1 : Collection des informations

Les informations qui peuvent être collectées par un sniffer sur un réseau local Ethernet sont :

1) Les adresses IP : cela permet à l'intrus de connaître les adresses des hôtes existants sans consulter le Domain Name System (DNS).

2) Les fonctionnalités des hôtes : en analysant quel type d'informations est en train d'être échangé, le sniffer est capable de déterminer, par exemple, l'identité de l'hôte serveur de fichiers ou celle de l'hôte administrateur, etc...

3) Session Telnet, FTP et POP3 : le sniffer peut capturer dans les trois protocoles des passwords et des logins qui sont transférés en texte clair sur le réseau.

4) Sessions rlogin : elles permettent au sniffer de connaître quel utilisateur sur quel hôte a un accès rlogin car l'accès au serveur rlogin est implémenté grâce à la notion de confiance. Pour qu'un utilisateur ait un accès transparent à un hôte, l'adresse IP de sa machine doit être référenciée dans l'un des fichiers **hosts.equiv** -ou **rhosts** le plus souvent- un utilisateur exécutera la commande «rlogin» en spécifiant directement le nom de l'hôte distant : rlogin nom-machine. Il est inutile de préciser un nom d'utilisateur ou un mot de passe puisque le système est authentifié par le serveur «rlogin».
[KAR 98]

5) Noms de la communauté SNMP : presque tout le trafic SNMP (Simple Network Management Protocol) est SNMPvl. [GRA 00] Le mode d'authentification de SNMPvl est très faible, chaque paquet de SNMP-UDP contient en texte clair le nom de la communauté qui est assorti à une version locale dans l'équipement contrôlé. La connaissance du nom de la communauté pour l'accès lecture/écriture permet de contrôler l'équipement du réseau.

6) Lecture de la messagerie électronique : à chaque fois qu'un utilisateur du réseau envoie ou lit un e-mail, cet e-mail transitera sur le réseau en texte clair, le sniffer peut facilement lire cet e-mail.

V.2. PHASE 2 : Attaque

Une fois que l'utilisateur du sniffer réussit à avoir l'information qu'il voulait, il y a quelques attaques simples à exécuter : [GRU 98]

- Réutilisation des mots de passes capturés : c'est l'attaque la plus simple. Une fois que le sniffer a eu le nom-utilisateur et le mot de passe d'un utilisateur en analysant une session Telnet, FTP, ou POP3, il est trivial de les réutiliser.
- Sessions rlogin : en usurpant l'identité d'un hôte dit de confiance grâce au spoofing ou à la modification des tables de correspondance «adresse IP - nom de hôte» tel que décrit dans [BEL 95], l'intrus peut avoir un shell ou exécuter une commande sur un hôte distant.
- SNMPvl : Une fois le nom de la communauté connu, il est simple de l'utiliser pour modifier la table de routage d'un routeur. Cela mènera, par exemple, à une attaque par déni de service.
- NFS, FINGER... : beaucoup de protocoles ont des failles de sécurité, particulièrement les vieilles versions des démons non correctement patchées. La possibilité de capturer tous les paquets sur un segment facilite la découverte de ces vieux serveurs, et mènera à l'attaque appropriée.

V.3. Scénarios d'attaques :

Les attaques suivantes peuvent être réalisées avec l'aide d'un sniffer :

a) **Fermeture abusive d'une connexion** (connection killing)

Dans la figure 4.9 nous avons la situation suivante :
- A et B exécutent une connexion TCP.
- A et B sont sur le même segment.

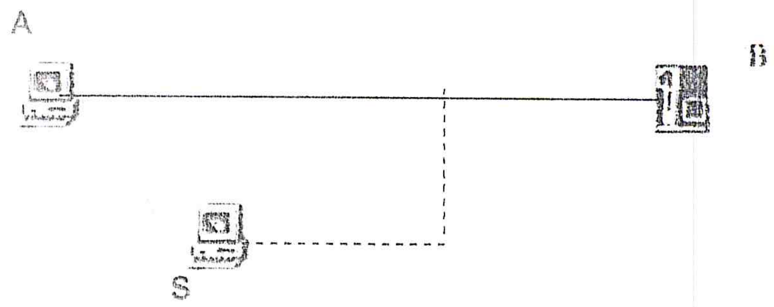


Figure 4.9 : Fermeture abusive d'une connexion.

Concept :

Pour qu'un message RST (utilisé pour fermer les connexions TCP) soit accepté, seul le numéro de séquence devrait être correct.
Nous attendrons une connexion entre A et B, (supposons que nous attendons les paquets adressés à A). Nous pourrons avoir, à l'aide d'un sniffer, le numéro de séquence de A lors de l'établissement de la connexion entre A et B, d'après les paquets d'acquittement (ACK) de B. On envoie ensuite, à partir de S, (et en se faisant passer pour A), un faux paquet RST à B : ce qui fermera la connexion entre A et B.
[CLA 96]

Cette attaque est particulièrement intéressante dans le cas où un attaquant utiliserait un sniffer pour suivre une connexion (FTP, par exemple) en temps réel. Si, par malchance pour lui, l'attaquant n'a pas suivi la connexion dès le début, donc n'a pas réussi à avoir le «nom d'utilisateur» et «le mot de passe», il pourrait toujours fermer abusivement la connexion ; et par conséquent, obliger le client à se connecter de nouveau. Et là, il pourra lire la paire (nom utilisateur, mot de passe).

b) **TCP Hijacking**

Le paragraphe suivant décrit une attaque active contre le protocole TCP qui permet au cracker de rediriger le flux TCP vers sa machine. Les détails de cette attaque sont décrits dans [JON 95].
Dans la figure 4.10 nous avons la situation suivante :

-A et B exécutent une connexion TCP.

(Telnet)

-A et B sont sur le même segment.

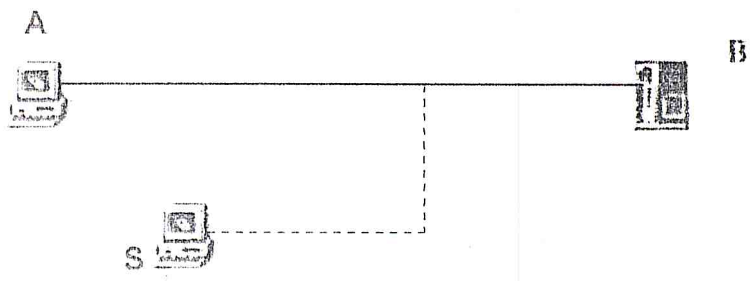


Figure 4.10 : TCP Hijacking.

Concept :

Supposons une session Telnet entre A (client) et B (serveur) : TCP contrôle une connexion en séparant les bons et les mauvais segments TCP grâce aux numéros de séquence et d'acquittement (SEQ/ACK). A ce moment là, le serveur B a confiance dans les paquets du client A si leurs numéros de séquence et d'acquittement SEQ/ACK sont corrects. Ainsi, s'il y avait une possibilité de désynchroniser les paquets d'acquittement (ACK) et de séquencement (SEQ) provenant de A, B arrêterait de croire les vrais paquets de A.

A partir de la machine S, nous nous faisons passer pour A (IP Spoofing), mais en utilisant des numéros SEQ/ACK qui seront considérés comme étant corrects par le serveur B. Il nous est facile de déduire de tels numéros en utilisant un sniffer.

Les nombres SEQ/ACK de A peuvent être désynchronisés par la simple insertion de paquets de données dans le flux du trafic au bon moment. Le serveur B acceptera ces données et mettra à jour les numéros d'acquittement (ACK), A continue d'envoyer ses vieux numéros de séquences (SEQ) vu qu'il ignore les données truquées que nous avons insérées.

De cette manière, on aura réussi à détourner la connexion : le serveur B n'accepte plus les paquets provenant de A (car leurs SEQ/ACK serons faux), et en même temps il accepte les paquets provenant de S ; A est confus et S aura réussi à détourner la session. Cette attaque est appelée TCP Hijacking (généralement détournement d'une session Telnet, mais aussi FTP, rlogin, etc...)

On pourrait se demander pourquoi détourner une session Telnet alors que nous disposons d'un sniffer et que nous pourrions l'utiliser pour «sniffer» les mots de passe et les réutiliser après. Cependant, cette technique est très efficace dans le cas d'un système qui crypte les mots de passe, utilise la technique du «one-time password» ou toute autre technique qui ne nous permet pas d'avoir les passwords en clair.

[JON 95]

VI. PREVENTION CONTRE UNE ATTAQUE PAR SNIFFER

Une attaque par sniffer est considérée par certains experts en sécurité informatique comme étant la plus grave et la plus nuisible envers la sécurité du système, d'où la nécessité de mécanismes de prévention contre une telle attaque.

[BEL 97]

Une des méthodes la plus évidente de protéger un réseau contre les attaques par sniffers est de mettre en place des dispositifs de sécurité assez efficaces (Firewall, Network intrusion Détection System,...) pour contrer toute attaque de l'extérieur ; un cracker qui n'a pas accès à un réseau ne pourra pas y installer un sniffer. Dans un monde parfait la liste des mesures de prévention se terminerait ici ! Mais comme le nombre des failles de sécurité ne cesse de s'accroître et que de nouvelles failles sont découvertes chaque mois, sans oublier que la majeure partie des attaques sont commises de l'intérieur, les mesures citées ci-dessus s'avéreront insuffisantes ; d'où la nécessité de mécanismes plus lourds.

VI.1. Utilisation des switchs :

Une bonne méthode généralement adoptée pour se prémunir contre l'attaque par sniffer consiste à utiliser la technologie «Switchs Ethernet» (Hub intelligent) au lieu des hubs classiques. Un Switch est composé de plusieurs ports, chaque port du Switch doit stocker les adresses des périphériques connectés à ce port. Le switch utilise ces adresses pour envoyer un paquet reçu sur un port au port concerné par l'adresse de destination située dans la trame du paquet, et non pas comme dans le cas des hubs où un paquet reçu et envoyé vers tous les autres ports [KAR 98]. Par conséquent, le réseau sera segmenté et chaque machine du système ne pourra voir que le trafic du segment sur lequel elle se trouve et non pas tout le trafic échangé sur le réseau.

[BEL 97, DRU 00]

VI.2. Utilisation du chiffrement :

Une autre option qui pourrait être combinée avec celles citées ci-dessus est l'utilisation du chiffrement. Il existe plusieurs techniques de chiffrement : [GRA 00, KLA 96]

-SSL : Secure Sockets Layer est intégré dans la plupart des navigateurs Web populaires ; il permet de chiffrer le trafic Web, notamment lors de transactions de commerce électronique (e-commerce) (chiffrage du numéro de carte de crédit).

- PGP : Pretty Good Privacy [ZIM 94] peut être utilisé pour le cryptage du courrier électronique.

- SSH : Secure Shell est devenu de facto le standard pour se connecter aux machines Unix via Internet ; il peut remplacer des protocoles tels que Telnet. Ce produit a été développé par une société finlandaise (cf [WSS 01]).

Ajouter aux techniques citées ci-dessus des mécanismes tels que Kerberos [KOH 93] et la technologie «one-time password». [HAL 94]

VII. CONCLUSION

Il existe plusieurs façons de se prémunir des désagréments que pourraient provoquer l'utilisation d'un sniffer sur votre réseau :

- Utiliser des protocoles chiffrés pour toutes les communications dont le contenu possède un niveau de confidentialité élevé.
- Segmenter le réseau afin de limiter la diffusion des informations. Il est notamment recommandé de préférer l'utilisation de switchs (*commutateurs*) à celle des hubs (*concentrateurs*) car ils commutent les communications, c'est-à-dire que les informations sont délivrées uniquement aux machines destinataires.
- Utiliser un détecteur de sniffer : il s'agit d'un outil sondant le réseau à la recherche de matériels utilisant le mode *promiscuous*.
- Pour les réseaux sans fil il est conseillé de réduire la puissance des matériels de telle façon à ne couvrir que la surface nécessaire. Cela n'empêche pas cependant les éventuels pirates d'écouter le réseau mais réduit le périmètre géographique dans lequel ils ont la possibilité de le faire.

Nous allons passer maintenant au chapitre concernant la conception de notre sniffer. Cette partie éclaircira plus l'architecture et le fonctionnement du logiciel.

Chapitre 5

Conception et réalisation

CHAPITRE 5 : CONCEPTION ET REALISATION

I. INTRODUCTION

La phase la plus importante dans l'élaboration des applications est la conception. Celle-ci produit une architecture simple et nette d'un problème bien défini et étudié au préalable. A cet effet, notre conception, basée essentiellement sur le thème du projet ainsi que l'étude faite dans les chapitres précédents, consiste à bâtir la structure du programme final qui répondra au travail qui nous a été confié et consistant en la réalisation d'un analyseur de trames dans le réseau local Ethernet.

Cet analyseur est basé principalement sur une architecture de capture de trames Ethernet que nous allons définir dans les paragraphes qui suivent. Cette architecture est structurée par niveaux hiérarchiques qui sont en interactions. Sachant que tous les concepts fondamentaux concernant la partie théorique ont été soigneusement étudiés aux chapitres précédents, la partie conception consistera donc à présenter et à préciser tout ce que nous devons faire afin d'atteindre les objectifs visés.

II. LA METHODE DE CONCEPTION

II.1. Introduction :

II.1.1. Qu'est-ce que l'XP ?

L'XP est une approche réfléchie et disciplinée du développement logiciel. Celle-ci met l'accent sur la satisfaction du client : elle permet de livrer le logiciel en temps voulu.

Elle s'appuie sur :

- Une forte réactivité au changement des besoins du client (ou changements technologiques) au plus tôt dans le cycle de vie du logiciel ;
- Un travail d'équipe : combinaison de l'effort des développeurs, de l'encadrement et du client ;
- La qualité du code : elle garantit un gain d'argent pour le client ;
- La qualité des tests effectués au plus tôt ;
- Une intégration permanente.

L'XP est une méthodologie intéressante car :

- Elle est basée sur l'examen des pratiques de développement logiciel ;
- C'est une méthodologie logicielle allégée destinée à réduire le coût du logiciel.

II.1.2. En quoi XP est une méthodologie allégée ?

Une méthodologie logicielle est un ensemble de règles et de pratiques mises en oeuvre pour la création de programmes.

Problèmes actuels : les règles sont trop difficiles à suivre, les procédures complexes et mal comprises et la quantité de documentation à produire hors de contrôle. Pour essayer de rester dans le planning, il est nécessaire de simplifier les règles, garder celles qui contribuent à la qualité et laisser de côté celles qui ralentissent le projet. L'XP est une méthodologie allégée car elle est constituée de peu de règles et de mises en pratique. Le travail en équipe, la réactivité, la concision et l'efficacité s'en trouvent renforcés.

II.2. Règles et mise en pratique de l'XP :

II.2.1. La phase de planification :

II.2.1.a. Les scénarii d'utilisateur (*User Stories*) :

Les scénarii sont utilisés pour réaliser les estimations en temps et en risques à destination du jeu de planification (ils remplacent le cahier des charges) et servent à guider la création des tests fonctionnels. Ils sont semblables aux cas d'utilisation (*Use Cases*) mais ne sont pas limités à l'interface utilisateur. Ces scénarii sont écrits par le client, sous la forme de quelques phrases textuelles suivant la terminologie du client mais sans syntaxe technique. Ils doivent juste apporter un niveau de détail suffisant pour pouvoir estimer le temps nécessaire à leur mise en oeuvre.

Le développement d'un scénario doit durer de 1 à 3 semaines. Si le développement dure moins de 1 semaine, il est nécessaire de regrouper des scénarii car le niveau de détail est probablement trop élevé. S'il dure plus de 3 semaines, le scénario doit être fractionné.

Un projet standard a en moyenne 80 cas d'utilisation à + ou - 20.

Important : Il faut impérativement éviter de considérer tout détail relatif à une technologie spécifique ou à des algorithmes, par exemple.

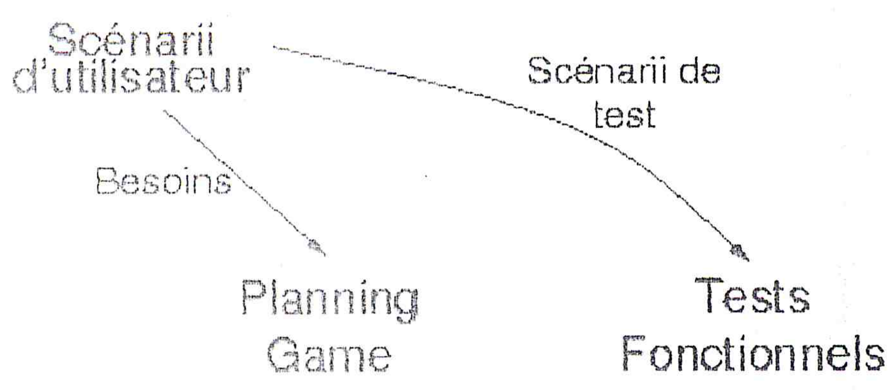


Figure 5.1 : Scénarii utilisateur.

II.2.1.b. Le jeu de planification (*Planning Game*) :

Il s'agit d'un ensemble de règles permettant à toute personne impliquée dans le projet de prendre des décisions. Les règles définissent une méthode permettant de négocier un planning sur lequel tout le monde devra s'accorder.

Le jeu de planification sert à la création du planning de livraison. Ce dernier va être à la base de la planification d'itération, pour chaque itération.

Il permet d'estimer chaque scénario d'utilisateur en terme de jours de programmation, de niveau de risque et de priorité du client.

Les scénarii d'utilisateur sont imprimés sur des cartes. Ces cartes doivent passer entre les mains des développeurs et des clients afin de déterminer l'ensemble des scénarii à implémenter pour la livraison suivante. Tout doit être pensé en vue de la nécessité de livrer au plus tôt un système testable.

Le facteur de charge détermine le nombre de scénarii pouvant être implémentés avant une certaine date et combien de temps sera nécessaire pour finir l'implémentation de cet ensemble de scénarii.
Les itérations peuvent être planifiées en terme de scénarii d'utilisateur à implémenter.

Attention : Ne pas changer l'estimation des scénarii d'utilisateur, même si le planning de livraison final déplaît à l'encadrement. Les estimations sont valides et seront utilisées telles quelles pour les réunions de planification des itérations.

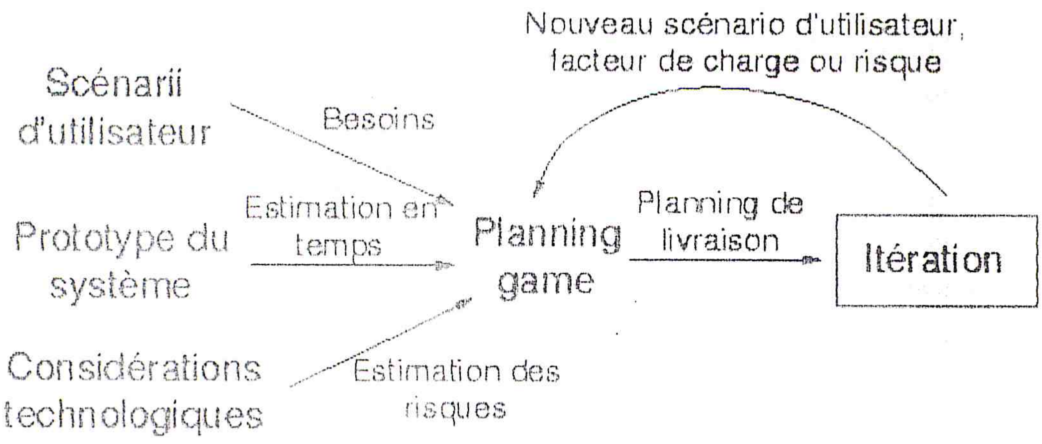


Figure 5.2 : Le jeu de planification.

II.2.1.c. Le planning de livraison :

Après avoir écrit les scénarii d'utilisateur, le planning de livraison est créé à partir du jeu de planification.

Le planning de livraison spécifie les scénarii d'utilisateur devant être implémentés pour chaque version provisoire et les dates de livraison de ces scénarii.

Les scénarii d'utilisateur à implémenter pour l'itération suivante sont décidés par le client pendant la réunion dédiée à la planification de l'itération. Les tâches de programmation individuelle nécessaires à l'élaboration des scénarii d'utilisateur peuvent ainsi être planifiées au sein d'une itération.

Si le facteur de charge se trouve modifié après plusieurs itérations, il est nécessaire de redéfinir un nouveau planning de livraison.

Figure 5.3 : Planning de livraison.

II.2.1.d. Livrer régulièrement des versions :

Cette règle a pour but d'obtenir à temps des réactions de la part du client et d'influencer au plus tôt le développement du système. En effet, plus la livraison de nouvelles fonctionnalités au client est tardive, moins il reste de temps pour corriger ces dernières.

II.2.1.e. Le facteur de charge :

Ce facteur mesure la vitesse du projet.

Facteur de charge = nombre de jours réels pour accomplir une tâche / estimation en jours pour effectuer cette tâche.

Il est utilisé par le planning game pour concevoir le planning de livraison. Calculer ce facteur nécessite la prise en considération de l'expérience des membres du projet et de la technologie à utiliser.

Un facteur de 2 à 3 est parfait pour une première estimation. Il doit être réévalué tout au long du projet.

Si ce facteur change radicalement durant le projet, utiliser le jeu de planification pour renégocier le planning de livraison.

Important : Généralement, ce facteur augmente lorsque le système est placé en production à cause des tâches de maintenance à effectuer.

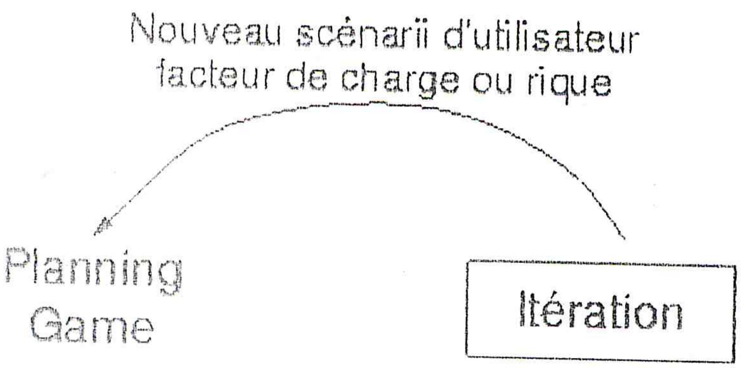


Figure 5.4 : Facteur de charge.

II.2.1.f. Un développement itératif :

Diviser le planning de développement en une douzaine d'itérations d'environ 2 à 3 semaines chacune.
Au début de chaque itération doit être organisée une réunion de planification de l'itération, afin de décider de ce qui va être réalisé.

Avertissement : Il ne faut pas :
- Prévoir les tâches de programmation à l'avance ;
- Aller trop en avant en implémentant ce qui n'est pas prévu pour l'itération courante.

II.2.1.g. La planification d'une itération :

Cette réunion sert à la planification des tâches de programmation à réaliser pendant l'itération suivante.

Celle-ci a lieu à chaque début d'itération. Chaque itération dure de 1 à 4 semaines.
Les scénarii d'utilisateur sont choisis par le client dans le planning de livraison. En premier lieu sont retenus les scénarii les plus capitaux aux yeux du client et les plus risqués pour les développeurs. Les tests fonctionnels ayant échoué sont automatiquement choisis.

Les scénarii d'utilisateur et les tests ayant échoué sont transformés en tâches de programmation. Idéalement, chaque tâche doit durer entre 1 et 3 jours. Les développeurs choisissent leur tâche et évaluent le temps nécessaire à leur accomplissement.
Le facteur de charge sert à déterminer si l'itération est surchargée ou non. Si l'itération se trouve trop chargée, le client choisit les scénarii à reporter vers l'itération suivante.

Important : Les estimations restent valides. Les tâches restantes seront les plus faciles à réaliser.

S'attaquer aux besognes les plus difficiles évite des mauvaises surprises en fin de projet.

II.2.1.h. Déplacer les membres au sein du projet :

Cette règle a pour objectif d'éviter une fuite des connaissances qui pourrait causer un ralentissement de l'avancée du projet.

Chaque itération doit permettre à tous de travailler sur une partie nouvelle du système.

Le travail en binôme en assure la continuité : une des personnes constituant le binôme peut être échangée avec un nouveau partenaire.

II.2.1.i. Réunion matinale quotidienne :

Cette réunion a pour but d'évoquer les problèmes, leurs solutions et aide à se concentrer sur l'esprit d'équipe. Celle-ci doit être courte, pour que tout le monde y vienne et pour éviter de perdre trop de temps. Toute autre réunion ne rassemblera que les personnes nécessaires et les contributeurs. La plupart des réunions n'ont pas à être planifiées : quand il y a peu de personnes impliquées, ces réunions sont généralement spontanées.

Il est envisageable de nuancer l'horaire et la manière dont peuvent être organisées de telles réunions.

Dans le cadre du télétravail, une conférence journalière avec les membres d'un projet via IRC peut être une solution.

II.2.1.j. Redressement du processus quand celui-ci se dégrade :

Les règles initiales peuvent être changées. Cependant, toute l'équipe doit impérativement suivre les mêmes règles.

Il est souhaitable de se réunir pour parler des dysfonctionnements et de la manière de les résoudre.

II.2.2. La phase de conception :

II.2.2.a. Simplicité de la conception :

Préférer la simplicité à la complexité.

Important : Conserver une conception simple est un travail difficile et demande beaucoup de rigueur.
Rappelons-nous qu'il ne faut jamais ajouter de fonctionnalités avant leur planification.

II.2.2.b. Choisir un système de métaphores :

Nommer de manière cohérente les classes et les méthodes. Cela facilite la compréhension et la réutilisabilité du code. Aucune connaissance technique spécifique ne doit être nécessaire à la compréhension du code.

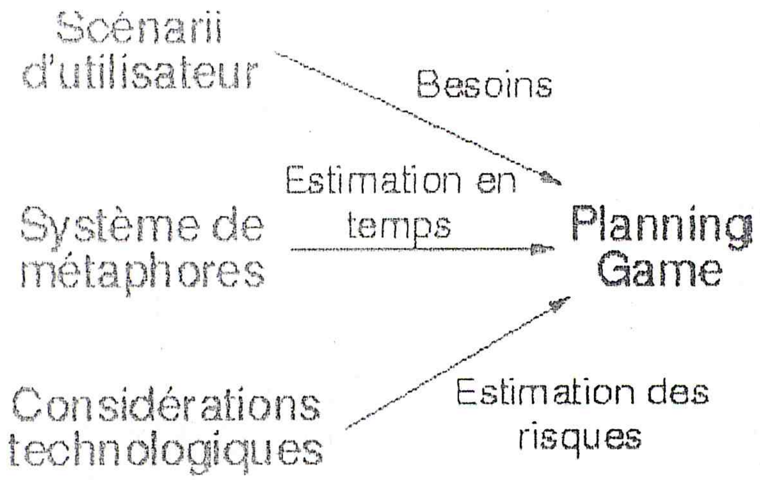


Figure 5.5 : Système de métaphores.

II.2.2.c. Cartes CRC (*Class, Responsabilities and Collaboration*) :

Ces cartes facilitent la contribution de tous les membres de l'équipe à la conception : elles favorisent l'introduction du plus grand nombre de "bonnes" idées dans la conception.

Une carte CRC est utilisée pour représenter l'instance d'un objet. En haut : classe de l'objet. En bas à gauche : responsabilités. À droite de chaque responsabilité : les classes collaboratives. Terme de messages que les objets s'échangent entre eux. Message par message. Mettre dans un document la conception décidée.

Cette méthode permet de s'approcher plus facilement de la technologie objet et de s'éloigner des techniques procédurales.

II.2.2.d. Création d'une solution de pointe (*Spike Solution*) :

Une solution de pointe consiste en un programme d'exploration des solutions potentielles. Des solutions de pointe doivent être créées pour trouver des réponses à des problèmes techniques ou des problèmes de conception difficile. L'objectif est de réduire les risques liés à des problèmes techniques. Et la fiabilité de l'estimation réalisée sur les scénarii d'utilisateur s'en trouve renforcée.

Il est conseillé de construire un système qui ne s'adresse qu'aux problèmes examinés et qui laisse les autres de côté. Quand un problème s'obstine à bloquer le projet, associer un binôme au projet pendant 1 à 2 semaines, en réduisant ainsi les risques potentiels.

II.2.2.e. Ne jamais ajouter de fonctionnalités plus tôt que prévu :

Ne se concentrer que sur ce qui est actuellement planifié. Ne pas penser aux besoins futurs et à l'amélioration de la flexibilité. De cette manière, il est possible d'éviter un ralentissement du projet et un gaspillage des ressources.

II.2.2.f. Ne pas hésiter à refaire :

L'objectif est :

- D'éviter de réutiliser du code qui n'est plus maintenable ;
- De conserver une conception simple : éviter un désordre et une complexité inutile.

Pour cela :

- Eviter la redondance ;
- Eliminer les fonctionnalités inutiles ;
- Rajeunir les conceptions obsolètes.

II.2.3. La phase de programmation :

II.2.3.a. Un client doit être toujours disponible :

La programmation extrême nécessite qu'un client soit toujours disponible pour aider l'équipe de développement, voire pour en faire partie. Il est préférable d'avoir des contacts directs avec le client, sur site. L'idéal est d'associer un ou plusieurs clients à l'équipe de développement.

Attention :

Le client doit être un expert et non un débutant ou un stagiaire.

Les scénarii d'utilisateur doivent être écrits par le client avec l'aide des développeurs pour effectuer au mieux :

- Une estimation en temps ;
- Une évaluation des risques ;
- Une assignation des priorités.

Le client vérifie si les cas d'utilisation couvrent toutes les fonctionnalités désirées. Tous les détails étant exclus des scénarii d'utilisateur, les programmeurs doivent obtenir le plus d'informations possible pour la réalisation d'une tâche : ceci nécessite un engagement à plein temps du client.

Par ailleurs, l'aide du client est nécessaire pour l'élaboration des tests fonctionnels afin de vérifier que le système est prêt à être passé en production. Si le système ne passe pas les tests fonctionnels, le client peut décider ou non de mettre le système en production en fonction du résultat (score) des tests.

Il est préférable que le client soit présent lors des réunions occasionnelles de groupe afin de pouvoir trancher en cas de désaccord entre les experts.

II.2.3.b. Les standards de codage :

Le code doit être formaté suivant des standards de codage afin de :

- Maintenir le code cohérent ;
- Faciliter la lecture et la refonte du code.

Il existe plusieurs standards de codage adaptés à chaque langage et parmi ceux-ci : GNU Coding Standards (C), CoreLinux++ Source code Standards and Guidelines (C++) et Perl Style Guide.

II.2.3.c. La programmation en binôme :

Chaque portion de code destinée à la livraison en production doit être le fait de deux personnes. Cette méthode augmente la qualité du code sans avoir d'impact sur la date de livraison.

II.2.3.d. Des livraisons séquentielles :

1- Le problème actuel :

Chaque développeur agit parallèlement aux autres : il teste son propre code mais la combinaison des codes source n'a pas été testée. C'est pourquoi de nombreux problèmes d'intégration surviennent sans prévenir.

2- Les solutions potentielles :

- Chaque développeur possède des classes spécifiques. Il assure que le code pour chaque classe est intégré et correctement livré.
- Utiliser un intégrateur ou une équipe d'intégration. Cependant, une équipe d'intégration demande trop de ressources quand il s'agit d'intégrer plusieurs fois par semaine.

Attention :
Il existe un risque d'intégration de versions obsolètes.

Il faut donc veiller à toujours travailler par rapport à la version la plus récente du système.

3- Les solutions proposées par l'XP :

L'XP propose de faire effectuer les livraisons de manière séquentielle par les développeurs eux-mêmes profitant de la règle autorisant l'appartenance du code à tous. Ceci implique qu'à un moment donné, un unique binôme intègre, teste et livre dans le dépôt du code source.

Ce processus nécessite un mécanisme de verrous. Par exemple, une machine peut être dédiée à l'intégration.

II.2.3.e. Une intégration fréquente :

Les développeurs doivent intégrer et livrer leur code le plus régulièrement possible (plusieurs fois par jour).

Une intégration fréquente permet :
- d'éviter une divergence et une fragmentation des efforts de développement quand les développeurs communiquent peu entre eux sur ce qui peut être réutilisé et partagé ;
- de détecter au plus tôt les problèmes de compatibilité. Intégrer au plus tôt signifie passer moins de temps à intégrer et éviter le dépassement des délais dû à l'intégration.

II.2.3.f. Le code doit appartenir à tous :

1- Pourquoi partager le code ?

La règle de partage de code permet principalement d'éviter de passer par une personne unique pour effectuer des changements. Par conséquent, une personne peut quitter le projet sans contrarier ce dernier. Cette règle encourage chacun à ajouter de nouvelles idées à tous les segments du projet.

Chaque développeur peut ainsi :
- ajouter des fonctionnalités ;
- corriger des bogues ;
- refondre le code.

2- Comment l'XP propose le partage de code?

Chaque développeur doit concevoir des tests unitaires à destination du code qu'il a développé. Le code livré dans le dépôt des codes source doit être accompagné des tests unitaires. Le code nouvellement ajouté, les bogues corrigées ainsi que le code modifié doivent être passés par une procédure de test automatique. L'ensemble des tests (*test suite*) réalisés doit ainsi couvrir l'intégralité du dépôt des codes source. Avant sa livraison, le code doit passer l'ensemble des tests avec une réussite de 100 %.

De cette manière, chacun peut ajouter une méthode de classe et la livrer dans le dépôt des codes source.

Ceci combiné à une fréquente intégration permet la plupart du temps de masquer les changements mineurs effectués sur une classe.

II.2.3.g. N'optimiser qu'à la toute fin :

La phase d'optimisation ne doit intervenir qu'à la fin. Pour ce faire, utiliser des instruments de mesure et non des évaluations subjectives.

On trouve bon nombre d'outils de mesure d'efficacité dont gprof (inclus dans la suite binutils) et bprof.

II.2.4. La phase de test :

II.2.4.a. Les tests unitaires :

Ils sont une pierre d'angle de la programmation extrême.

L'accomplissement des tests unitaires nécessite de :
1. Récupérer une ossature de test unitaires () : celle-ci aidera à la constitution d'une suite de tests automatisés.
2. Tester toutes les classes du système

Il est préférable de créer les tests préalablement à l'écriture du code. Ceci permet d'avoir très tôt des informations de retour pendant la réalisation.
Les tests unitaires sont livrés dans le dépôt des codes source accompagnés du code qu'ils testent. Tout test unitaire trouvé manquant doit être immédiatement conçu. Un ensemble complet de tests est ainsi disponible au moment voulu.

Ces tests favorisent :
• La possession collective du code : il permettent d'éviter les dommages accidentels ;
• La refonte du code : ils permettent de vérifier si aucun changement n'a été apporté à la fonctionnalité.

Exiger que le code passe tous les tests unitaires garantit que toutes les fonctionnalités marchent toujours.
Une suite de tests de validation et de régression favorise une intégration fréquente.

Attention :
Ajouter de nouvelles fonctionnalités nécessite le plus souvent de changer les tests unitaires.

II.2.4.b. Les tests fonctionnels :

Ils sont un système de tests "boîte noire" qui servent à valider les changements intervenus précédemment à la livraison en production.
Les tests fonctionnels sont générés à partir des scénarii d'utilisateur. Les clients sont responsables de la vérification de leur correction.

Ces tests sont automatisés : le résultat est diffusé à toute l'équipe qui planifie le temps nécessaire à la correction des bogues, pour chaque itération.

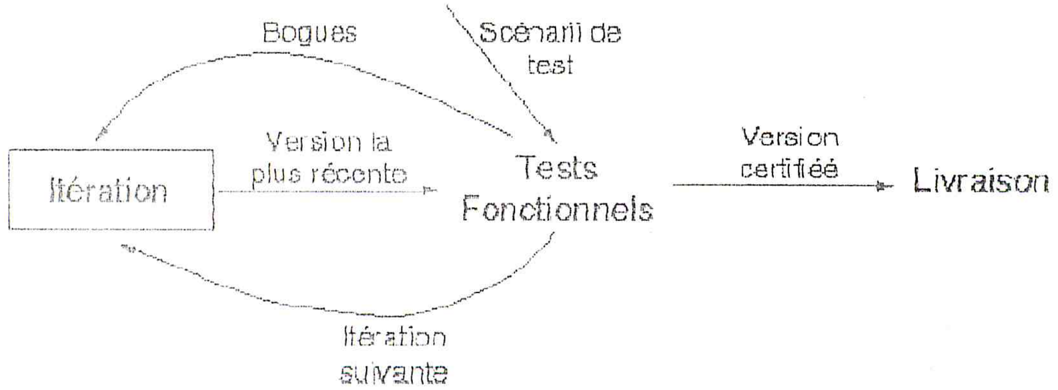


Figure 5.6 : Tests fonctionnels.

II.2.4.c. Le traitement des bogues :

Quand un bogue est trouvé, il faut créer un test pour éviter que celui-ci ne réapparaisse.
La création d'un test fonctionnel avant déboguage aide le client à déterminer les problèmes et à communiquer ceux-ci aux programmeurs.
En cas d'échec des tests fonctionnels, des tests unitaires doivent être créés. Cette phase aide à se concentrer sur le déboguage et offre immédiatement en retour des informations lorsque le bogue a été corrigé.

II.3. Démarrer un projet avec XP :

Il est préférable d'expérimenter l'XP sur un projet qui démarre.
Pour ce faire :
- Collecter les scénarii d'utilisateur et les solutions de pointe concernant tout ce qui semble risqué. Mais essayer de passer le moins de semaines possible sur cette phase ;
- Prévoir un planning de réunions pendant lesquelles sera utilisé le jeu de planification. Ce planning doit être fixé à la fois par l'encadrement, les développeurs et les clients ;
- Démarrer le développement itératif par une réunion de planification d'itération.

[KEB]
[WEX]

III. Concepts de base sur l'architecture :

On a vu dans le chapitre précédent l'architecture globale d'un sniffer (sous Windows). On se base sur une technique de conception en modules qui interagissent entre eux. La décomposition en modules permet de diviser un système complexe à recevoir en sous-systèmes dédiés à des tâches précises, ce qui permet de mieux cerner les problèmes de conception mais aussi d'offrir la possibilité de faire des interventions séparés sur ces modules. En effet, la maintenance s'en trouve facilitée, le fait de pouvoir modifier un module indépendamment des autres si cela est nécessaire. On décrit maintenant ces modules (dans notre conception), leur comportement et les choix architecturaux qui les affectent.

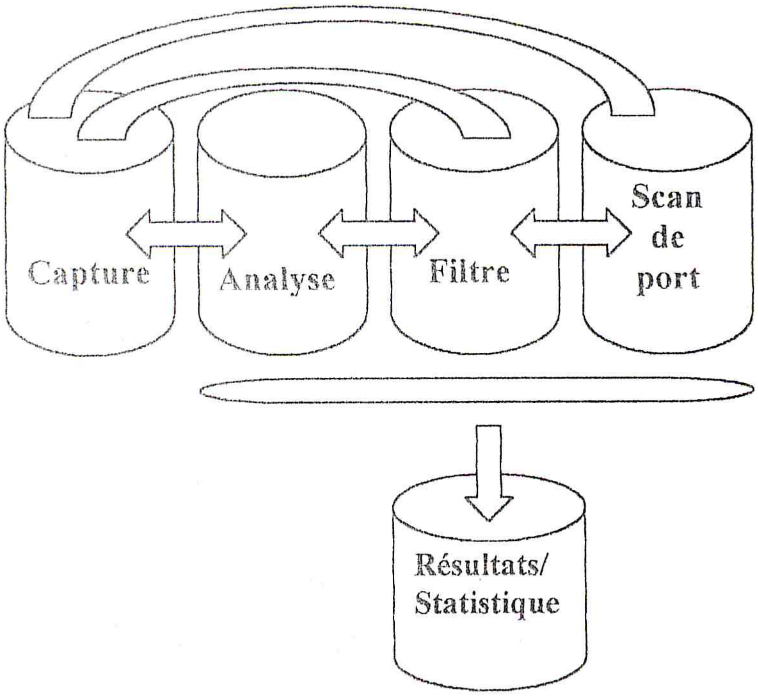


Figure 5.7 : Différents modules composant l'architecture de MekAnal.

Les trois principales étapes suivantes résument l'idée de base :

- Les paquets de données qui circulent sur le segment du réseau local Ethernet sont représentés par le trafic recueilli au niveau de ce réseau.
- Les paquets de données transitent le niveau bas, qui représente le niveau noyau de l'architecture, ceux-ci sont remontés vers les niveaux supérieurs de l'architecture pour des traitements ultérieurs.
- Lorsque les paquets sont traités par les différent modules les résultats des opérations effectuées sur ces derniers sont dirigés vers les organes de sorties se trouvant au niveau de l'utilisateur (Fichier de sorties / monitoring / Rapport…).

Notre logiciel doit réaliser une capture des paquets de données Ethernet, afin de les analyser et les afficher en conséquence, pour d'éventuels traitements.

Pour cela, ce logiciel doit se baser sur l'architecture qu'on a définit précédemment, cette dernière est basée principalement sur le modèle BPF du système d'exploitation UNIX (BPF, pour Berkeley Packet Filter). Elle a pour principal objectif d'analyser le trafic circulant dans le réseau local Ethernet et cela en réalisant une capture d'un échantillon de trames Ethernet via une station (PC, ordinateur,…) qui doit être connecté au réseau LAN-Ethernet, décapsuler les trames capturées de la couche réseau pour pouvoir les lire et les présenter en clair à l'administrateur du réseau. Trois niveaux hiérarchiques principaux constituent notre architecture de capture de trames et d'analyse de réseau Ethernet et qui interagissent entre eux, ces derniers se présentent de haut en bas comme suit :

III.1. Niveau utilisateur : notre programme MekAnal (Analyseur de Mekfouldji) et la bibliothèque libpcap

Le programme MekAnal est la plus haute partie de la pile de capture et dirige l'interaction entre l'utilisateur et le système. Il obtient les entrées de l'utilisateur (par exemple, quels paquets doivent être capturés et de quelle manière doivent être montrés à l'utilisateur) de la ligne de commande et affiche les résultats sur écran. Le MekAnal (Analyseur de Mekfouldji) est exécutable sur une plate-forme Windows.

Il y a des analyseurs sur UNIX qui utilisent une bibliothèque pour le processus de capture du paquet, appelée La bibliothèque de la Capture du paquet, ou bibliothèque de Pcap, ou Libpcap, une interface système indépendante pour la capture du paquet au niveau utilisateur. Libpcap fournit un ensemble de fonctions indépendantes du matériel et du système d'exploitation qu'une application peut les utiliser pour capturer des paquets du réseau. Ces analyseurs n'interagissent pas directement avec le matériel, mais utilisent les fonctions exportées par Libpcap pour capturer les paquets, mettre un filtre et communiquer avec la carte réseau.

Fondamentalement, c'est un système indépendant et peut être facilement porté à n'importe quel système où la bibliothèque Pcap est disponible. Pour cette raison le projet MekAnal (Analyseur de Mekfouldji) inclut un port plein de la bibliothèqué Pcap vers la plate-forme Win32. En outre, la bibliothèque Pcap n'est pas limitée à être utilisé avec MekAnal (Analyseur de Mekfouldji), et on pense que ça peut être très utile aux gens qui veulent convertir des moniteurs et des analyseurs du monde de l'UNIX, et ça peut être aussi une base puissante pour créer de nouveaux outils réseau pour l'environnement Win32.

Par conséquent, on a essayé de maintenir la structure de la version originale dans notre mise en oeuvre. On a modifié la section de Libpcap qui communique avec le noyau en implémentant l'interaction avec le pilote de capture des paquets NDIS. Une caractéristique importante de Libpcap pour Win32 est qu'il y a une version seulement qui travaille sur Windows 95, 98, NT et 2000. Cela peut être obtenu en mettant une bibliothèque de lien dynamique, appelée PACKET.DLL, entre Libpcap et le pilote de capture afin que les appels système au pilote seront les mêmes dans les divers environnements Windows. De cette manière un outil réseau basé sur Libpcap travaillera sur Windows 95 et Windows NT sans aucune modification.

III.2. Niveau kernel (noyau) : le pilote de capture des paquets NDIS

Le rôle essentiel de la partie kernel de la pile de capture est prendre les paquets de la couche lien du réseau et les transférer au niveau application sans modifications. On l'a implémenté comme pilote du kernel (Packet.sys) sous Windows NT et comme pilote de l'interface virtuel (Packet.vxd) sous Windows 95. Les applications peuvent accéder au pilote de capture avec les primitives read/write et peuvent considérer la carte réseau d'une manière ou d'une autre similaire à un fichier normal faisant la lecture ou l'écriture de données provenant du réseau.

L'interaction avec le pilote de capture habituellement traverse la bibliothèque de lien dynamique Packet.dll. La DLL implémente un ensemble de fonctions qui rendent la communication avec le pilote plus simple en évitant l'usage des appels système ou IOCTLs. Le pilote de capture interagit avec les pilotes de l'interface de la carte réseau à travers NDIS qui est une partie du code réseau de Win32.
NDIS est responsable de la gestion de plusieurs cartes du réseau et de la communication entre ces cartes et les portions du logiciel qui implémentent les protocoles.

Un pilote de capture réseau de base peut être assez simple. Il a besoin de lire les paquets seulement du pilote du réseau et les copier à l'application. Cependant, pour obtenir des performances acceptables, des améliorations substantielles doivent être faites à cette structure essentielle. Les plus importantes sont :

• Pour limiter la perte des paquets, le pilote doit être capable de stocker les paquets arrivés dans un buffer car l'application au niveau utilisateur n'est pas prête à les traiter à leur arrivée. Les paquets amortis seront transférés dès que l'application soit prête.

• Pour minimiser le nombre de changements du contexte entre l'application (qui s'exécute en mode utilisateur) et le pilote (en mode kernel), ça devrait être possible de transférer plusieurs paquets du buffer à la application en utilisant un seul appel de lecture.

• L'application doit recevoir seulement les paquets qui l'intéresse, habituellement un sous-ensemble du trafic réseau entier. Une application doit être capable de spécifier le type de paquets qu'elle veut (par exemple les paquets produits par un hôte particulier) et le pilote lui envoie seulement ces paquets. En d'autres termes l'application doit être capable de mettre un filtre sur les paquets qui entrent, en recevant seulement le sous-ensemble qui satisfait le filtre.

Un filtre est tout simplement une fonction avec une valeur de retour booléenne appliquée sur un paquet. Si la valeur de retour est «vrai», le pilote copie le paquet à l'application, autrement le paquet est ignoré. L'implémentation de ces traits et l'architecture du pilote sont inspirées par le BSD Paquet Filtre (BPF) du kernel de l'UNIX. On avait une vue d'ensemble du processus de capture du BPF dans le chapitre précédent.

Considération importante :

On remarque que BPF n'existe pas dans toutes les versions UNIX (c.-à-d. les capacités de filtrer et amortir «buffering» dans le noyau), mais la bibliothèque du pcap peut compenser ce manque. L'architecture est capable de travailler en filtrant les paquets d'une manière compatible avec BPF au niveau utilisateur. Cette solution était adoptée par la première parution du pilote NDIS des paquets. On peut l'implémenter mais elle a des performances limitées pour deux raisons en particulier :

• Le processus de filtrage est fait au niveau utilisateur, afin que chaque paquet soit copié du noyau au buffer de l'application avant de détecter que l'application le veux ou non. Cela gaspille clairement du temps CPU et de la mémoire. Si le processus de filtrage est fait dans le noyau, tous les paquets qui ne sont pas utiles pour l'application de capture seront abandonnés par le système et ne seront pas copiés au niveau utilisateur.

• Il n'y a aucun amortissement de paquets dans des buffers au niveau noyau (kernel). Dans un environnement multi-tâches, l'application de capture doit partager le temps du processeur avec d'autres programmes. C'est possible que l'application de capture ne s'exécute pas quand un paquet arrive. En outre, elle peut faire quelques autres tâches et ne peut pas attendre le paquet. Dans l'absence d'un buffer (kernel), ces situations mènent à la perte du paquet.

Pour ces raisons, l'implémentation du processus de filtrage et d'amortissement des paquets est faite dans le pilote de capture.

III.3. Interaction avec NDIS :

NDIS (Network Pilote Interface Specification) est un ensemble de caractéristiques qui définissent la communication entre une carte réseau (ou, mieux, le pilote qui la dirige) et les pilotes de protocole (IP, IPX...). Le but principal de NDIS est d'agir comme un papier d'emballage qui permet au pilotes de protocole d'envoyer et de recevoir des paquets sur un réseau (LAN ou WAN) sans se soucier d'une carte particulière ou d'un système d'exploitation Win32 particulier.

NDIS supporte trois types de pilotes réseau :

1- Carte de l'interface réseau ou pilotes NIC (Network Interface Card). Les pilotes NIC dirigent directement les cartes de l'interface réseau. Les NIC connectent directement le matériel à leur bord inférieur et à leur bord supérieur présente une interface qui permet aux couches supérieures d'envoyer des paquets sur le réseau, manier des interruptions, réinitialiser le NIC, faire arrêter le NIC, l'interroger et désigner les caractéristiques opérationnelles du pilote. Un pilote NIC ne peut pas communiquer avec les applications de mode utilisateur, mais seulement avec les pilotes intermédiaires NDIS ou les pilotes de protocole. Les pilotes NIC peuvent être soit des miniports ou des pilotes NIC d'héritage complet :

- Les pilotes miniports implémentent seulement les opérations matérielles spécifiques nécessaires pour diriger un NIC, comprenant l'envoi et la réception de données sur le NIC. Les opérations communes à tous les bas niveaux des pilotes NIC, telles que la synchronisation, sont fournies par NDIS. Miniports n'appellent pas les routines du système d'exploitation directement; leur interface au système d'exploitation est NDIS. Un miniport ne se tient pas au courant d'agglutinations. Il passe les paquets simplement à NDIS et NDIS s'assure que ces paquets sont passés aux corrects protocoles.

- Les pilotes NIC complets ont été écrits pour accomplir les opérations matérielles spécifiques et toutes les opérations de synchronisation et d'ajout de queue habituellement faites par NDIS. Les pilotes NIC complets, par exemple, maintiennent leur propre information d'agglutination pour indiquer les données reçues.

2- Les pilotes intermédiaires. Ils connectent entre un niveau supérieur du pilote tel qu'un pilote de transport d'héritage et un miniport. Au niveau supérieur du pilote, un pilote intermédiaire ressemble à un miniport. Au miniport, le pilote intermédiaire ressemble à un pilote de protocole. Un pilote intermédiaire de protocole peut être posé en couche sur un autre pilote intermédiaire bien que de tels découpages en couches puissent avoir un effet négatif sur la performance du système.

Une raison typique pour développer un pilote intermédiaire est d'accomplir une traduction médiatique d'un pilote de transport d'héritage existant et un miniport qui dirige un NIC, à un nouveau type médiatique inconnu au pilote de transport. Par exemple, un pilote intermédiaire peut traduire du protocole LAN au protocole ATM. Un pilote intermédiaire ne peut pas communiquer avec les applications de mode utilisateur, mais seulement avec les autres pilotes NDIS.

3- Pilotes de transport ou pilotes de protocole. Un pilote de protocole implémente une pile de protocole réseau telle que IPX/SPX ou TCP/IP, offrant ses services sur un ou plusieurs NIC. Un pilote de transport entretient avec les clients de la couche application à travers son bord supérieur et connecte un ou plusieurs pilote(s) NIC ou pilote(s) NDIS intermédiaire à son bord inférieur.

La figure 5.8 présente une structure NDIS avec deux piles de capture sur la même carte réseau : l'une avec un pilote NIC et un pilote de protocole, l'autre avec un pilote NIC, un pilote intermédiaire et un pilote de protocole.

2- Les pilotes intermédiaires. Ils connectent entre un niveau supérieur du pilote tel qu'un pilote de transport d'héritage et un miniport. Au niveau supérieur du pilote, un pilote intermédiaire ressemble à un miniport. Au miniport, le pilote intermédiaire ressemble à un pilote de protocole. Un pilote intermédiaire de protocole peut être posé en couche sur un autre pilote intermédiaire bien que de tels découpages en couches puissent avoir un effet négatif sur la performance du système.

   Une raison typique pour développer un pilote intermédiaire est d'accomplir une traduction médiatique d'un pilote de transport d'héritage existant et un miniport qui dirige un NIC, à un nouveau type médiatique inconnu au pilote de transport. Par exemple, un pilote intermédiaire peut traduire du protocole LAN au protocole ATM. Un pilote intermédiaire ne peut pas communiquer avec les applications de mode utilisateur, mais seulement avec les autres pilotes NDIS.

3- Pilotes de transport ou pilotes de protocole. Un pilote de protocole implémente une pile de protocole réseau telle que IPX/SPX ou TCP/IP, offrant ses services sur un ou plusieurs NIC. Un pilote de transport entretient avec les clients de la couche application à travers son bord supérieur et connecte un ou plusieurs pilote(s) NIC ou pilote(s) NDIS intermédiaire à son bord inférieur.

La figure 5.8 présente une structure NDIS avec deux piles de capture sur la même carte réseau : l'une avec un pilote NIC et un pilote de protocole, l'autre avec un pilote NIC, un pilote intermédiaire et un pilote de protocole.

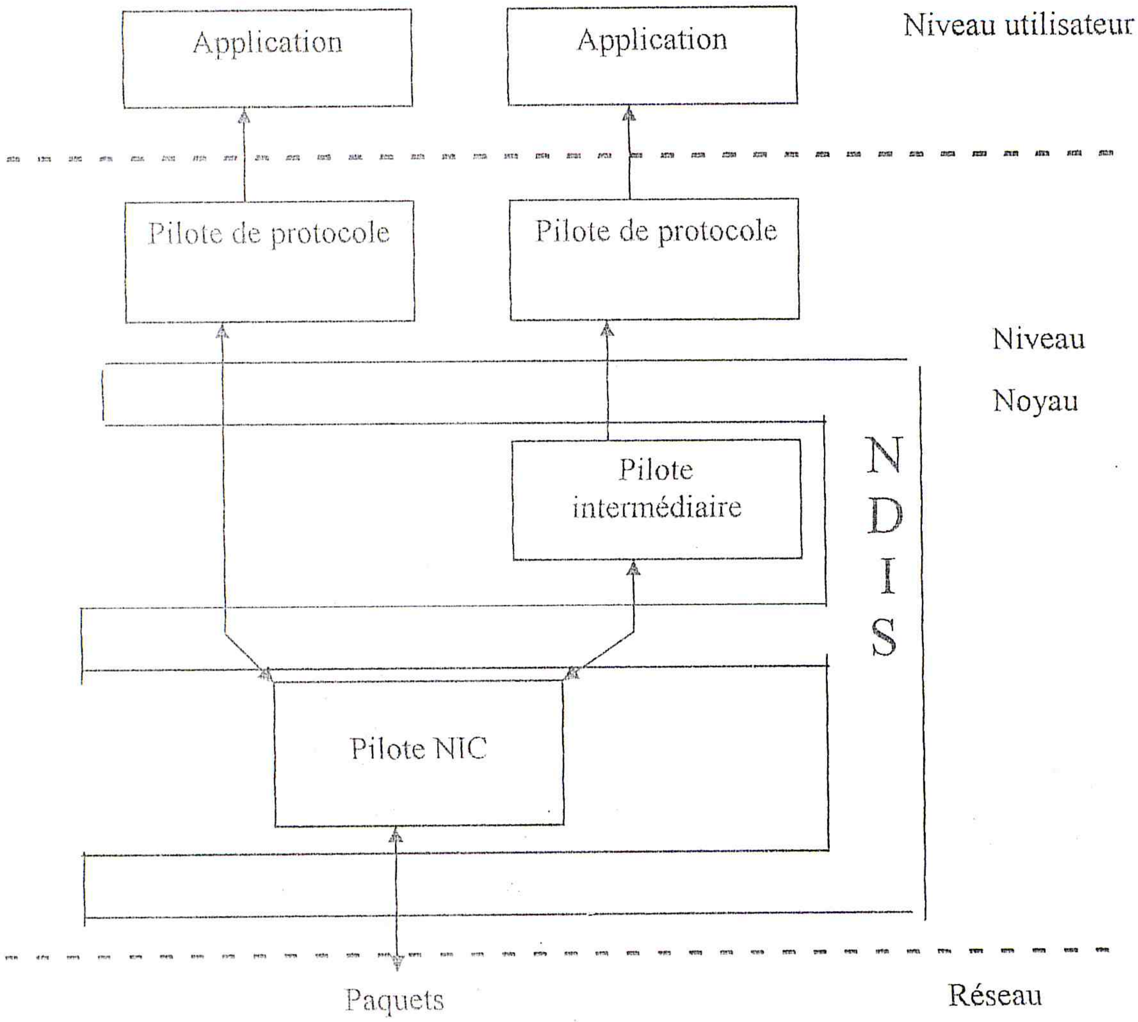


Figure 5.8: Structure NDIS simple.

Le pilote de capture des paquets a besoin de communiquer avec les pilotes du réseau (pour obtenir les données du réseau) et avec les applications de niveau utilisateur (pour leur fournir les paquets), donc il est implémenté dans la structure NDIS comme un pilote de protocole. Cela lui permet d'être indépendant du matériel du réseau, ainsi que travailler avec toutes les interfaces du réseau supportées par Windows. Remarquez cependant que le pilote de capture des paquets travaille pour le moment seulement sur les cartes Ethernet, cartes Loopback et sur quelques connexions WAN à cause des limites imposées par l'architecture du pilote et du filtre.

Remarquez aussi qu'une connexion WAN est vue habituellement par les pilotes de protocole comme un NIC Ethernet, et à chaque paquet reçu une en-tête Ethernet truquée est créé par NDIS. Cela permet aux pilotes de protocole écrits pour Ethernet de travailler sur les connexions WAN sans aucun changement, mais ça implique aussi que des paquets spécifiques comme PPP, NCP et LCP ne seront pas vus par les pilotes de protocole parce que la connexion PPP est virtuelle. C'.à.d que le pilote de paquet ne peut pas capturer ce genre de paquets.

A noter que les divers systèmes d'exploitation Win32 ont des versions différentes de NDIS:

La version de NDIS sous Windows 95 est 3.0, pendant que Windows NT a NDIS 4 et Windows 2000 a NDIS 5. NDIS 4 et 5 sont de NDIS 3, par conséquent, un pilote écrit pour travailler avec NDIS 3 (habituellement) travaille aussi avec NDIS 4 et 5. Le pilote de capture des paquets est écrit pour NDIS 3, mais travaille aussi avec les versions les plus récentes de NDIS. Cela veut dire que l'interaction entre le pilote et NDIS est la même sous Windows 95/98 et sous Windows 2000.
La figure suivante montre la position du pilote de capture des paquets dans l'architecture Win32.

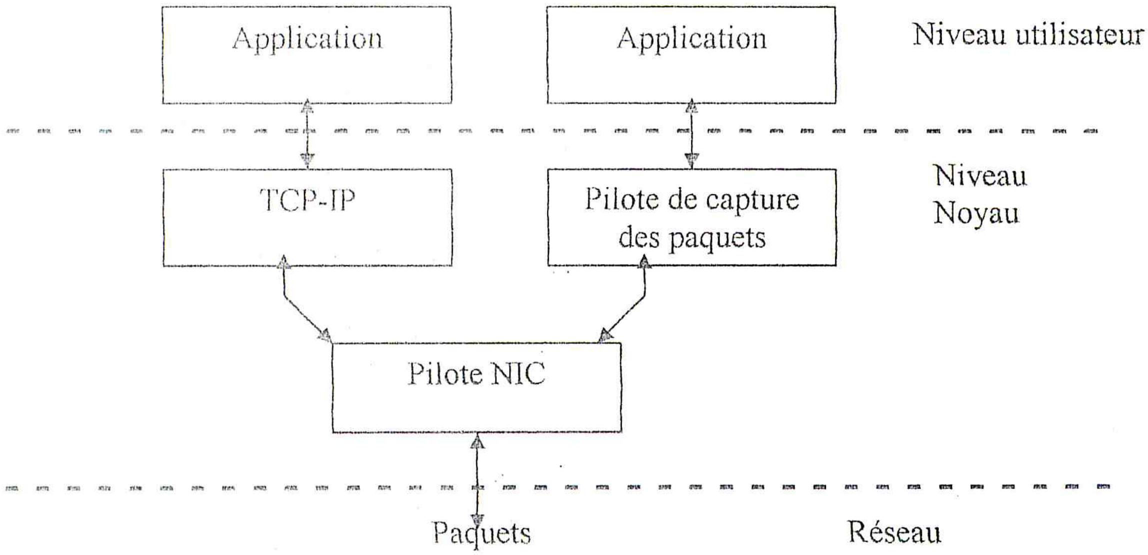


Figure 5.9 : Pilote de Capture des Paquets dans NDIS.

Un pilote de protocole qui communique avec les niveaux inférieurs des pilotes NDIS utilise les fonctions fournies par NDIS. Par exemple, un pilote de protocole doit appeler NdisSend ou NdisSend-Paquets pour envoyer un paquet ou des paquets à un niveau inférieur du pilote NDIS. Les pilotes de niveaux inférieurs, d'un autre côté, communiquent avec le pilote de protocole d'une manière asynchrone. Ils indiquent l'arrivée d'un nouveau paquet en appelant une fonction récursive du pilote de protocole et passer un pointeur à un buffer, sa taille et la taille totale du paquet reçu. Le nom de cette fonction récursive dans le pilote de capture des paquets est Packet_tap. Le comportement du pilote de capture des paquets est cependant tout à fait différent de celui d'un pilote de protocole standard. En fait :

- Le pilote de capture des paquets reçoit et traite tous les paquets qui sont actuellement transférés dans le réseau. Un tel comportement peut être obtenu en plaçant la carte dans le mode promiscuous, c.-à-d la forcer à accepter tous les paquets du réseau.
Un pilote de protocole standard dirige seulement les paquets (unicast et multicast) dirigés à ou venus et ceux qui sont diffusés (broadcast).

- Le pilote de capture des paquets n'implémente pas un protocole, mais il stocke les paquets et les transfère comme ils sont, avec leur timestamp et leur longueur, aux applications.

Un pilote de protocole standard enlève les divers en-têtes des paquets et laisse seulement les données passer aux applications. Le pilote de capture des paquets laisse les en-têtes et les copie aux applications avec les données capturées.

Notez que dans l'implémentation d'UNIX, BPF est appelé avant la pile de protocole, directement, par le pilote de l'interface du réseau. Ce n'est pas possible pour le pilote de capture des paquets qui fait partie de la pile de protocole. C'est pourquoi le pilote de capture des paquets n'est pas capable de capturer les paquets spécifiques PPP, parce qu'il ne travaille pas au niveau hardware, mais au sommet de NDIS. Avoir la préséance sur NDIS impliquerait des changements au niveau noyau (kernel) ou dans les pilotes NIC, ce qui n'est pas possible dans Windows.

III.4. Compatibilité :

- Les systèmes d'exploitation supportés :

Les versions du pilote de capture des paquets existent pour tous les systèmes d'exploitation Win32 principaux : Windows 95, 98, NT4 et 2000.
La version Windows 9x consiste en un pilote de l'interface virtuel (.vxd) qui peut être installé dans Windows 95 et Windows 98. Windows NT4 et Windows 2000, d'un autre côté, ont besoin de pilotes différents (.sys) parce que les processus d'installation et d'initialisation sont différents.

- Compatibilité du matériel :

Le pilote de capture des paquets a été développé d'origine pour travailler avec les cartes Ethernet. Le support pour autre MACs peut être ajouté pendant le développement, mais Ethernet reste le préféré. La raison principale est que tous nos postes de développement ont des cartes Ethernet donc tout nos testes sont faits sur ce type de réseau. Cependant, la situation courante est :

- Windows 95/98: le pilote de capture des paquets travaille sur les réseaux Ethernet correctement. Il travaille aussi sur les liens PPP du WAN, mais avec quelques limitations (par exemple il n'est pas capable de capturer les paquets LCP et NCP).
- Windows NT4/2000: le pilote de capture des paquets travaille sur les réseaux Ethernet correctement.

Nous n'étions pas capables de le faire travailler sur les liens PPP du WAN, à cause d'un problème inévitable (l'amortissement) sur la carte NDISWAN. Des Supports pour FDDI, ARCNET et ATM ont été ajoutés à partir de la version 2.02; cependant nous ne les avons pas testés. N'attendez pas qu'ils travaillent parfaitement.
(Token Ring n'est pas supportée aussi)

IV. PILOTE DE CAPTURE DES PAQUETS POUR WINDOWS

Le Pilote de Capture des Paquets ajoute aux noyaux (kernels) de Windows la capacité de capturer des paquets bruts d'un réseau d'une manière très semblable à celle des noyaux d'UNIX avec BPF. De plus, il fournit des fonctions, qui ne sont pas disponibles dans le pilote BPF original, afin d'aider le développement du test du réseau et les programmes moniteurs. Les buts principaux du pilote de capture des paquets sont : une haute performance de capture, une flexibilité et une compatibilité avec le BPF original.

Le résultat est un pilote de protocole qui peut:
- capturer la trafic brut du réseau et le passer à une application de niveau utilisateur.
- filtrer les paquets entrants qui exécutent le code pseudo-machine BPF. Cela veut dire que l'application de capture peut définir un programme BPF standard et le passer au pilote. Le pilote écartera les paquets entrants qui ne satisfont pas le filtre.
- stocker les paquets dans un buffer quand l'application est occupée ou lorsqu'elle n'est pas assez rapide pour soutenir le flux de paquets qui viennent du réseau.
- rassembler les données de plusieurs paquets et les rendent comme une unité quand l'application fait une lecture. Afin de maintenir les limites du paquet, les paquets sont encapsulés dans une en-tête (le même qui est utilisé par BPF) qui inclut un timestamp, une longueur, et des offsets pour l'alignement des données.
- écrire des paquets bruts au réseau.
- calculer des statistiques sur la circulation du réseau.

IV.1. La structure du pilote :

Ce paragraphe décrira la structure du pilote et les principaux choix architecturaux.

IV.1.1. Architecture de base des diverses versions :

La structure de base du pilote est exposée dans la figure suivante :

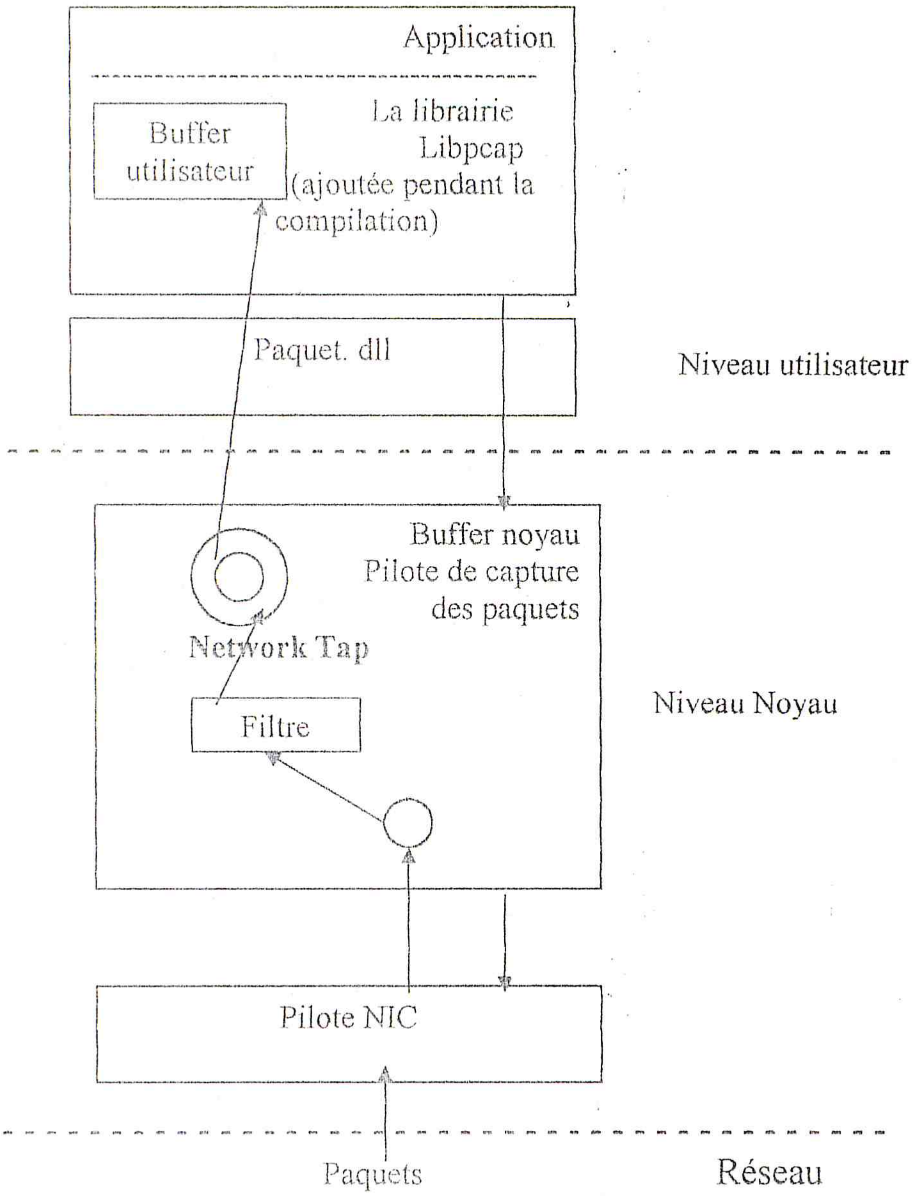


Figure 5.10 : Structure du pilote de capture.

Les flèches vers le sommet représentent le flux des paquets du réseau à l'application de capture. La plus grande flèche qui est entre le buffer de noyau et l'application indique que plus qu'un paquet peut transiter entre ces deux entités dans un appel système de lecture seul. Les flèches vers le fond indiquent la trajectoire des paquets de l'application au réseau. La flèche épaisse entre le buffer de noyau (kernel) et l'application indique que, dans des situations particulières, plus qu'un paquet peut transiter entre ces deux entités dans un appel système d'écriture seul. MekAnal (Analyseur de Mekfouldji) et libpcap n'envoient pas de paquets au réseau, donc ils utilisent seulement la trajectoire du fond au sommet. Cependant, le pilote n'est pas limité à l'usage avec MekAnal (Analyseur de Mekfouldji) et peut être utilisé pour créer de nouveaux outils du réseau. Pour cette raison, il a été inclus la possibilité d'écrire des paquets qui peuvent être exploités à travers un usage direct de la bibliothèque du lien dynamique Packet.dll.

La structure présentée dans la figure 5.10 (c.-à-d. une carte seule et une application seule) est une description simplifiée du pilote de capture des paquets. La vrai structure est plus complexe et peut être vue dans la configuration du pilote avec deux cartes réseaux et deux applications de capture.

Pour chaque session de capture établie entre une carte et un programme de capture, le pilote maintient un filtre et un buffer. L'interface du réseau peut être utilisée par plus qu'une application en même temps. Par exemple, un utilisateur qui veut capturer trafic IP et UDP et les classer dans deux dossiers séparés, peut lancer deux sessions de MekAnal (Analyseur de Mekfouldji) sur la même carte (mais avec des filtres différents) en même temps. La première session mettra un filtre pour les paquets IP (et un buffer pour les stocker), et la deuxième un filtre pour les paquets UDP. C'est aussi possible d'écrire une application qui, en réagissant réciproquement avec le pilote de capture, est capable de recevoir des paquets de plus qu'une interface en même temps.

On remarque que dans les versions les plus anciennes que 2.02, la seule configuration possible dans Windows 95 et Windows 98 était montrée dans la figure 5.10 (c.-à-d. une carte réseau seule et une application de capture seule). C'était dû aux limitations dans l'architecture de ces versions. De la version 2.02, c'est possible dans Windows 95/98, comme dans Windows NT et XP, d'avoir plus d'une demande du pilote, et cela permet d'avoir plus d'applications de capture travaillant en même temps. En outre, il est possible pour une application seule de travailler sur plus qu'une carte réseau.

La structure noyau est tout à fait semblable entre les diverses versions de Windows. Les structures des données internes ne sont pas très différentes, le buffer du paquet et le filtre sont maniés de la même façon. L'interaction avec NDIS est très semblable sous les plates-formes différentes et elle est obtenue par un ensemble de fonctions récursives exportées par le pilote et les fonctions bibliothécaires du groupe NDIS (NdisTransferData, NdisSend...) utilisées par le pilote de capture des paquets pour communiquer avec le pilote NIC. Les dissemblances entre les différentes versions concernent l'interaction avec les autres parties du système d'exploitation (lire et écrire une manutention d'appel des applications du niveau utilisateur, les fonctions de l'horloge...), puisque la philosophie des divers systèmes d'exploitation est tout à fait différente.

IV.1.2. Le processus de filtrage :

Le mécanisme du filtre présent dans le pilote de capture des paquets dérive directement du filtre BPF d'UNIX, donc toutes les choses déjà dites au sujet du filtre BPF sont encore toujours valides. Une application qui a besoin de mettre un filtre sur les paquets entrants peut construire un programme de filtre BPF standard (par exemple à travers un appel à la fonction du pcap_compile de libpcap) et le passer au pilote, et le processus du filtrage sera fait sur le niveau kernel.

Le programme BPF est transféré au pilote à travers un appel IOCTL avec le code du contrôle mis à pBIOCSETF. Une chose très importante est que le pilote a besoin d'être capable de vérifier le code du filtre de l'application. En fait, comme on a dit, les pseudomachines BPF peuvent exécuter des opérations arithmétiques, branchements, etc. Une division par zéro ou un saut à une adresse mémoire défendue, si elle est faite par un pilote, produit inévitablement un écran bleu de mort. Donc, sans aucune protection, un programme de filtre faux ou de défaut pourrait fracasser le système facilement.

Puisque le pilote de capture des paquets peut être utilisé par n'importe quel utilisateur, il pourrait être facile pour une personne malintentionnée de causer des dégâts au système à travers ça. Pour cette raison, chaque programme de filtre venant d'une application est vérifié par la fonction bpf_validate du pilote avant d'être accepté. Si un filtre est accepté, le pilote stocke le programme de filtre et l'exécute sur chaque paquet entrant, en abandonnant ceux qui ne satisfont pas les conditions du filtre. Si le paquet satisfait le filtre, il est copié à l'application, ou mis dans le buffer si l'application n'est pas prête à le recevoir. Si aucun filtre n'est défini, le pilote accepte tous les paquets entrants.

Le filtre est appliqué aux paquets lorsqu'il est encore dans la mémoire du pilote NIC, sans qu'il soit copié au pilote de capture. Cela permet de rejeter le paquet avant toute tentation de copie, ainsi minimisant la charge du système. L'importante caractéristique du processus de filtrage BPF exploité par le pilote de capture est l'utilisation de valeur de retour numérique. Lorsque le programme de filtrage est appliqué aux paquets, le BPF distingue non seulement si le paquet doit être transféré et remonté au niveau application, mais la taille de la partie du paquet à copier.

Cela est très utile pour optimiser le processus de capture car uniquement la portion du paquet sollicité par l'application qui est copiée.
Trois fonctions principales constituent notre programme de filtrage, se présentent comme suit :

* bpf_filter : c'est la fonction de filtrage qui est utilisée par le pilote de capture. Elle implémente le registre machine qui exécute le code de filtrage, elle reçoit deux buffers mémoires : le premier contenant le paquet et le second contient le programme BPF à exécuter. Cette fonction retourne la taille de la portion du paquet à sauvegarder ou «zéro» si le paquet doit être ignoré (le paquet qui ne satisfait pas le filtre).

* bpf_filter_with_2_buffers : cette fonction est très similaire à la fonction précédente, mais elle peut filtrer les paquets dont l'en-tête et les données ont été sauvegardés dans des buffers différents. Elle reçoit trois buffers : le premier contient l'en-tête du paquet, le deuxième contient les données du paquet et le dernier le programme BPF à exécuter.
Cette fonction est lente, mais elle est plus générale que la fonction bpf filter. Elle est sollicitée à cause de la particularité de l'architecture NDIS, et elle est utilisée par le pilote de capture dans des situations particulières où l'en-tête et les données des paquets sont sauvegardés par le pilote sous-jacent dans différents buffers. Cela peut se produire par exemple dans le cas de l'émulation de l'ATM LAN dans lequel l'en-tête est encapsulé par le niveau software et peut être séparée des données.

* bpf_validate : c'est la fonction qui vérifie un nouveau programme de filtrage, retournant «vrai» uniquement dans le cas de programme valide. Cette procédure examine et vérifie que les sauts sont à l'intérieur du bloc de code, que les opérations de mémoire utilisent des adresses valides et que les divisions constantes par « 0» ne sont pas présentes dans le code.

IV.1.3. Le processus de lecture :

Lorsque l'application veut obtenir les paquets du réseau elle exécute un appel de lecture dans le pilote de capture des paquets NDIS. Cet appel peut être «synchrone» ou «asynchrone», parce que le pilote offre les deux possibilités.
Dans le premier cas «Mode synchrone», l'appel de lecture est bloqué et l'application est interrompue jusqu'à ce que les paquets arrivent à la machine. Dans le deuxième cas «Mode asynchrone», l'application n'est pas arrêtée et doit vérifier quand le paquet arrive. La méthode usuelle recommandée pour accéder au pilote est la «Méthode synchrone», à cause des difficultés dans l'implémentation d'amortissement «buffering», mais d'habitude elle n'est pas demandée parce que l'amortissement dans le pilote est plus efficace et propre.

Le pilote de capture supporte des lectures synchronisées (appelées en anglais Timed Read). L'application peut mettre le temps de lecture à travers un IOCTL avec le code PBIOCSRTIMESOUT, si le temps mort est différent de «zéro», chaque appel de lecture exécuté par l'application retournera quand le temps mort expire même s'il n'y a pas de paquets reçus. Après l'arrivage des paquets qui sont acceptés par le filtre, le pilote peut être dans deux situations différentes, qui sont les suivantes :

- L'application est prête à recevoir le paquet, c'est-à-dire elle a exécuté un appel de lecture et elle est actuellement endormie en attente du résultat de l'appel. Dans ce cas le paquet entrant est immédiatement copié dans la mémoire de l'application de capture appelante. L'appel de lecture est achevé et l'application et réveillée.
- Il n'y a pas de lecture pendante en ce moment, c'est-à-dire ; l'application fait autre chose et elle n'est pas bloquée en attente des paquets. Pour éviter la perte des paquets, ceux-ci doivent être sauvegardés dans le buffer du noyau et transférés à l'application une fois qu'elle sera prête à recevoir ces paquets, c'est-à-dire ; à la prochaine lecture.

Le pilote utilise un buffer circulaire pour sauvegarder les paquets. Le paquet est rangé dans le buffer avec une en-tête qui maintient l'information sur le paquet capturé (Timestamp) et la taille du paquet.
De plus, les octets de bourrages (Padding) sont insérés entre les paquets dans le but de les aligner en mots afin d'augmenter la vitesse de copie. La taille du buffer au début de la capture est à «zéro», elle peut être fixée ou modifiée à tout moment par l'application à travers un appel IOCTL. Lorsqu'une nouvelle dimension du buffer est fixée, les paquets actuellement dans le buffer sont perdus. Les paquets entrants sont ignorés par le pilote si le buffer est plein lorsque les nouveaux paquets arrivent. La dimension du buffer du pilote au niveau noyau influe lourdement sur les performances du processus de capture.

En fait, il est probable que l'application de capture, qui a besoin de faire des opérations sur chaque paquet, partageant en même temps le processus avec d'autres tâches, ne sera pas capable de travailler à une vitesse de réseau durant une importante vague de trafic circulant dans le réseau local ou un éclat. Ce problème est plus perceptible sur des machines lentes. Cependant le buffer adéquat dans le pilote peut sauvegarder les paquets quand l'application est occupée, donc il peut compenser la lenteur de l'application et peut éviter la perte des paquets lorsque l'activité du réseau est élevée.

Si le buffer n'est pas plein lorsque l'application exécute l'appel système de lecture, les paquets dans le buffer du pilote sont copiés vers la mémoire de l'application et l'appel de lecture est complété immédiatement. Plus qu'un paquet peut être copié du buffer circulaire du pilote vers l'application en utilisant un seul appel de lecture, cela améliore les performances car le nombre de lectures est minimisé.

En fait, chaque appel de lecture entraîne une commutation de contexte entre l'application et le pilote de capture. A cause de la lenteur des commutations de contextes, le décroissement de leur nombre signifie l'amélioration de la vitesse de capture. Pour maintenir les bordures des paquets, le pilote encapsule les données capturées de chaque paquet avec l'en-tête qui inclue le timestamp et la taille du paquet. L'application doit être capable de déballer proprement les données entrantes. La structure de données utilisée pour améliorer l'encapsulation est la même que celle utilisée par le BPF dans le noyau UNIX, ainsi le format des données retourné par le pilote est le même retourné par le BPF dans UNIX.

Notons que la méthode du buffer circulaire seul comme celle du pilote de capture BPF, comparée avec la méthode du double buffer, permet une meilleure utilisation de la mémoire. La méthode appropriée du double buffer est nécessaire lorsque les buffers sont très petits de taille. Dans ce cas, la fréquence des échanges entre les buffers est élevée et la mémoire est utilisée efficacement. Lorsque les buffers deviennent plus grands, cette méthode gaspille beaucoup d'espace mémoire et elle n'est plus efficace, parce que ces opérations effectuées sur les buffers peuvent prendre beaucoup de temps. Cette méthode de buffer circulaire utilise la mémoire plus efficacement qu'avec de grands buffers parce que la mémoire libre est toujours disponible.

Le pilote de capture manipule le niveau noyau et le buffer entretient avec le niveau utilisateur d'une manière similaire. Il est possible même de choisir n'importe quelle dimension pour le buffer circulaire du pilote, uniquement limitée par la RAM de la machine.
En outre, le buffer utilisé par l'application du niveau utilisateur peut avoir n'importe quelle taille et peut être changée chaque fois que l'application a besoin de le faire. La caractéristique importante est que la taille des deux buffers n'est pas forcément la même. Le pilote de capture détecte la dimension du buffer de l'application et le remplit convenablement et dans un sens correct même s'il a une taille différente du buffer circulaire. Cela est important lorsque le buffer du pilote est grand, parce que un large buffer d'application pourrait être un gaspillage de mémoire. D'un autre côté, ce mécanisme a aussi un inconvénient : puisque la taille des paquets n'est pas fixée, lorsque la dimension du buffer de l'application est plus petite que le nombre d'octets dans le buffer du noyau.

Il peut arriver que le pilote scanne les en-têtes des paquets dans son buffer pour déterminer le taux d'octets à copier. Ce traitement ralenti la procédure de capture, donc de meilleures performances ont obtenues lorsque l'application et le buffer du noyau ont la même taille.
En fait, le pilote détecte si la dimension du buffer de l'application est plus grande que le nombre d'octets dans le buffer du pilote, et si cela est vrai, il copie toutes les données sans scanner. Pour éviter ce problème, avant de commencer à copier, le pilote essaye de déterminer le montant total des données à copier sans scanner le buffer. Si cette opération est réussie, le petit montant du temps de la CPU est sauvegardé.

IV.1.4. Le processus d'écriture :

Le pilote de capture permet aussi d'écrire des données sur le réseau. Cela peut être utile pour tester le réseau ou les protocoles et les applications fonctionnant sous ce réseau. Pour expédier les données, l'application du niveau utilisateur exécute un appel système d'écriture (un appel IOCTL d'écriture sous la plate forme de Windows) sur le fichier du mécanisme du pilote des paquets. Les données sont envoyées vers le réseau comme elles se présentent, sans encapsulation par les protocoles des couches hiérarchiques, donc l'application doit construire une variété d'en-tête pour chaque paquet. L'application n'a pas besoin de générer le champ FCS («Frame Check Sequence» qui représente un champ de 4 octets placé en fin des trame Ethernet, il sert à déterminer si la trame reçue n'a pas été déformée (ou altérée) en cours de transmission) parce qu'il est calculé par le hardware de (carte réseau) et est attaché automatiquement à la fin du paquet avant son émission vers le réseau.

Notons que le taux d'émission de paquets au réseau n'est pas très élevé à cause du besoin d'un appel système pour chaque paquet. L'application du niveau utilisateur peut positionner avec l'appel IOCTL (Code pBIOCSWRITEREP), le nombre de fois qu'un seul paquet peut être envoyé dans le réseau. Par exemple, si cette valeur est mise à 1000, chaque paquet brut du pilote sera transmis 1000 fois dans le réseau en question. Cette caractéristique peut être utilisée pour générer une vitesse élevée du trafic (circulation) parce que la surcharge de la commutation du contexte n'est plus présente, et elle est particulièrement utile pour écrire des outils pour tester les réseaux, les routeurs,etc...

Cette technique existe uniquement dans Windows NT et Windows 2000, tandis qu'elle est simulée au niveau utilisateur dans la librairie Packet.dll dans Windows 95/98. Cela signifie que l'application de capture qui utilise de multiples méthodes d'écriture pourra être exécutée sous le système Win 9x, mais avec une vitesse d'écriture très lente comparée à Windows NTx.

IV.1.5. Quelques fonctions internes du pilote de capture :

En plus des procédures décrites dans les paragraphes précédents, le pilote de capture des paquets offre aussi une série de fonctions secondaires pour accorder toute la compatibilité avec le BPF original, nous décrirons quelques unes :

- Le pilote associe un Timestamp (timbre) pour chaque paquet qui satisfait le filtre. Le Timestamp qui n'est rien qu'une en-tête doit être associé au paquet avant qu'il ne soit mis dans le buffer parce que le processus d'amortissement tend à altérer les variations de temps. En outre, c'est important d'avoir l'en-tête du paquet aussitôt que possible, pour obtenir une référence précise.
  Cependant la prise de l'en-tête est l'une des premières opérations qu'exécutera le pilote lorsque un nouveau paquet arrive. La précision du Timestamp est en microsecondes, une valeur compatible avec l'implémentation UNIX. Le Timestamp est associé par le pilote avec les données du paquet, puis le tout (Timestamp + Données) passent ensemble à l'application lorsque un appel de lecture est exécuté.

- Chaque instance du pilote compte le nombre de paquets reçus du réseau et le nombre de paquets qui ont été éliminés (ignorés). Le paquet est ignoré lorsque l'application n'est pas prête à le recevoir et le buffer du pilote est plein. Dans cette situation le paquet ne peut pas être copié vers l'application de capture des paquets qui se trouve au niveau utilisateur et ne peut pas être sauvegardé dans le buffer et donc le pilote de capture doit le rejeter. Ces valeurs (nombre de paquets reçus et rejetés) sont demandées par la librairie Libpcap et elles sont très utiles pour comprendre les performances du processus de la capture et pour voir aussi combien de paquets ont été perdus.
  L'application peut obtenir à tout moment la valeur de ces deux compteurs à travers l'appel IOCTL avec le code de contrôle mis à pBIOCSETF.

Le gestionnaire ou l'administrateur du réseau n'est pas souvent intéressé par la séquence entière de paquets transitant dans le réseau, mais à leurs valeurs statistiques. Par exemple, l'utilisateur devrait être intéressé par l'utilisation du réseau, le niveau Broadcast, mais aussi des informations raffinées, telles que :

- Le montant du trafic de courrier ;
- Le nombre de requêtes Web par secondes.

Le mode «Statistique» est un mode de travail particulier du pilote de capture de BPF qui peut être utilisé pour faire des statistiques en temps réel sur le trafic du réseau capturé d'une manière simple et avec un impact minimum sur le système. Pour mettre le pilote dans le mode statistique, l'application du niveau utilisateur fait appel de IOCTL avec un code pBIOCSMODE, et la valeur «un» comme paramètre d'entrée. Pour mettre à nouveau le mode de travail (Capture normale), le même IOCTL, mais avec la valeur «zéro» comme paramètre d'entrée, peut être appelé.

Lorsque le mode est à «1», le pilote ne doit pas capturer n'importe quoi, mais il se limite à compter le nombre de paquets et le montant d'octets satisfaisant le filtre BPF définit par l'utilisateur. Ces valeurs passent à l'application dans un intervalle régulier, chaque fois que le Timeout s'expire. La valeur par défaut de ce Timeout est «une seconde» avec l'appel IOCTL (paramètre pBIOCRTIMEOUT). Les compteurs sont encapsulés dans la structure bpf_hdr avant qu'ils ne soient passés à l'application de capture (Programme de capture). Cela permet d'avoir un Timestamp avec une précision de microseconde et une interaction facile avec la librairie Libpcap. Cette librairie est en fait capable de décoder la structure bpf_hdr, et de passer les compteurs à l'application.

IV.2. Pilote des paquets (Packet.dll) et la librairie de capture (Libpcap) :

Pour écrire une application de capture, il est préférable d'utiliser la librairie de capture Libpcap à la place de l'API que nous avons décrit auparavant (Le deuxième niveau constituant notre architecture de capture). La librairie Libpcap utilise aussi bien les fonctions de la librairie dynamique Packet.dll, mais elle fournit un environnement de programmation plus puissant, immédiat et facile à utiliser. Avec Libpcap, des opérations telles que la capture des paquets, la création des filtres de capture ou la sauvegarde de la capture dans un fichier sont implémentées en sécurité et d'une manière plus souple à utiliser. Libpcap est capable de fournir toutes les fonctions sollicitées par un moniteur ou un analyseur standard. Cependant, le programme écrit pour utiliser Libpcap est facilement compilable sous UNIX à cause de la compatibilité qui existe entre la version de Libpcap de Win32 et de celle de l'UNIX.

Cependant L'API Packet.dll offre quelques possibilités qui ne sont pas disponibles dans la Libpcap.
Par exemple : L'analyseur réseau doit avoir besoin d'envoyer des paquets vers le réseau. Le pilote de capture des paquets NDIS et la librairie Packet.dll (utilisant la fonction Packet-Send-Packet ()), les deux, permettent l'envois des paquets vers le réseau. Tandis que la librairie Libpcap ne possède pas une telle caractéristique. Libpcap a été écrite de façon à ce qu'elle soit portable (standard entre les deux systèmes d'exploitation Windows et Unix) et pour offrir aussi une API de capture indépendante du système, donc la librairie ne peut pas exploiter toutes les possibilités offertes par le pilote. Pour cette raison quelques fonctions de l'API Packet.dll seront requises.

IV.3. Structures de données :

La librairie dynamique Packet.dll utilise des structures de données spécifiques pour accéder et dialoguer avec le pilote de carte réseau Ethernet via le pilote de capture «BPF Packet Capture Pilote» destiné à la plate forme de Windows. Parmi les structures de données qu'on va présenter, deux entre elles sont spécifiques au pilote de capture des paquets (PACKET et ADAPTER), tandis que les autres sont déjà définies dans la librairie Libpcap. La deuxième série de ces structures de données est utilisée pour faire des opérations telles que :

- Positionner les filtres ;
- Interpréter les données provenant du filtre ;
- etc...

Les fonctions utilisées par le pilote de capture utilisent la même syntaxe que celle utilisée par le BPF (Berkeley Packet Filtre) pour communiquer avec les applications de capture, ainsi la structure utilisée est la même. Parmi ces structures de données nous décrirons quelques unes :

1. Structure PACKET
2. Structure ADAPTER
3. Structure PACKET OID DATA
4. Structure bpf_insn
5. Structure bpf_program

6. Structure bpf_hdr
7. Structure bpf_stat
8. Structure NetType

1. Structure PACKET : Cette structure est spécifique au pilote de capture, elle est utilisée pour avoir un descripteur de paquet, elle possède les champs suivants : Buffer, Length, Next, ulBytesReceived, bloComplete.

- Buffer : Représente un pointeur vers le buffer du pilote (Packet Capture Pilote) qui contient les données du paquet.
- Length : Indique la taille de ce buffer.
- UlBytesReceived : Ce champ indique la taille de la portion du buffer contenant les données valides.
- bloComplete : Ce champ indique si le paquet contient des données valides après un appel asynchrone. Il est initialisé par les fonctions Packet_Receive_Packet(), Packet_Send_Packet(), Packet_Wait_Packet().

2. Structure ADAPTER : Cette structure décrit la carte réseau, elle est utilisée pour communiquer avec le pilote de capture, elle possède deux champs :

- hFile : Ce champ représente un pointeur vers l'en-tête du pilote, pour communiquer directement avec le pilote. (Dans tous les cas, cette fonction est découragée parce que Packet.dll offre une série de fonctions pour le faire).
- SymbolicLink : Représente une chaîne contenant le nom de la carte réseau actuellement ouverte (Activée).

3. Structure PACKET_OID_DATA : Cette structure est utilisée pour communiquer avec la carte réseau à travers la requête OID et une série d'opérations. Cette structure possède 3 champs, qui se présentent comme suit :

- OID : Ce champ est un identificateur numérique qui indique le type de la fonction query/set à exécuter sur la carte réseau à travers la fonction Packet_Request(). Les valeurs possibles sont définies dans le fichier Include ntddndis.h. Ce champ peut être utilisé, par exemple pour récupérer l'état des compteurs d'erreurs sur la carte, son adresse MAC, la liste des groupes Multicast définis, ainsi de suite.
- Length : Le champ Length indique la taille du champ Data qui contient les informations envoyées/reçues vers/du pilote.

4. Structure bpf_insn : Cette structure est utilisée pour envoyer un programme de filtrage au pilote. Elle possède les champs suivants :

- Code : Référence le code du programme.
- jt et jf : Ces deux champs sont utilisés par le saut (ou jump) conditionnel et sont les décalages (offsets) de l'instruction vrai ou faux cible suivante.

5. Structure bpf_program : Cette structure pointe vers le programme de filtrage, elle est utilisée par la fonction PacketSetBPF pour positionner le filtre dans le pilote. Cette structure possède deux champs :

- bf_len : Ce champ indique la taille du programme de filtrage des paquets.
- bf_insns : Est un pointeur vers la première instruction du programme de filtrage.

La fonction PacketSetBPF positionne un nouveau filtre dans le pilote de capture à travers un appel de IOCTL avec une série de code de contrôle. La structure bpf program est envoyée au pilote durant cet appel.

6. Structure bpf_hdr : Cette structure définie l'en_tête utilisée par le pilote pour délivrer le paquet à l'application de capture. L'en-tête est encapsulé avec les octets du paquet capturé, et cette en_tête est utilisée pour maintenir les informations telles que : La taille du paquet, le timestamp,…

La structure bpf_hdr possède les champs suivants :

- bh_tstamp : Ce champ saisi le timestamp associé avec le paquet capturé. Le timestamp a le même format que celui utilisé par le BPF d'Unix et il est sauvegardé dans la structure TimeVal, qui possède elle-même deux champs :
  - _ tv_sec : Ce sous-champ capture la date (nombre de secondes du 1 /1 /1970).
  - _ tv_usec : La capture en microsecondes.
- bh_caplen : Ce champ indique la taille de la portion capturée.
- bh_datalen : Ce champ indique la taille originale du paquet capturé.
- bh_hdrlen : Représente la taille de l'en-tête qui encapsule le paquet. Ce champ est utilisé uniquement par la version BPF d'Unix. Le BPF est capable de changer la taille de l'en-tête associée au paquet, en insérant une variable offset pour l'alignement (position) des données. Cette optimisation n'est pas encore implémentée dans notre pilote de capture. Pour cette raison, bh_hdrlen contient une valeur fixe, qui est la taille de la structure bpf_hdr.

7. Structure bpf_stat : Cette structure est utilisée par le pilote pour retourner les statistiques de la session de capture. Elle possède deux champs :

- bs_recv : Ce champ indique le nombre de paquets que le pilote a reçu de la carte réseau tout au début de la capture. Cette valeur inclue les paquets perdus par le filtre.
- bs_drop : Ce champ indique le nombre de paquets que le pilote a perdu au début de la capture courante. Fondamentalement, les paquets sont perdus quand le buffer est plein et le pilote les rejette. Ces variables ne sont pas comptées parmi les paquet qui sont directement ignorés ou éliminés par l'interface réseau, cela est dû par exemple au buffer lorsqu'il déborde.

8. Structure NetType : Cette structure est utilisée par la fonction Packet Get NetType () pour avoir les informations sur le type de la carte courante. Elle a deux champs :

- <u>LinkType</u> : Ce champ indique le type de la carte réseau courante.
- <u>LinkSpeed</u> : Indique la vitesse du réseau en Bits par seconde.

Note :

La taille du buffer associée avec la structure PACKET est un paramètre qui peut influencer sensiblement sur les performances du processus de capture. Ce buffer (buffer utilisateur) sauvegarde les paquets reçus du pilote de capture. Le pilote est capable de retourner plusieurs paquets capturés en utilisant un seul appel de lecture (Voir la fonction Packet Receive Packet ()). Le nombre de paquets transférés vers l'application de l'utilisateur en un seul appel est limité uniquement par la taille du buffer associé avec la structure PACKET utilisée pour exécuter la lecture. Par conséquent, utiliser un grand buffer avec la fonction Packet_Init_Packet () peut réduire le nombre d'appels systèmes, améliorant la vitesse de capture. Notons aussi que, lorsque l'application exécute la fonction Packet_Receive_Packet (), elle est souvent non bloquée.

Lorsque cette fonction est invoquée, le pilote copie les données présentes dans son buffer du niveau noyau au buffer de l'application utilisateur, mais l'application est éveillée et reçoit les paquets même s'il n'y a pas suffisamment de données pour remplir son buffer. Dans ce cas, l'application reçoit les paquets immédiatement aussi si elle a mis un grand buffer utilisateur ou lorsque le taux des données sur le réseau est faible.

6. Packet_Free_Packet () :

Cette fonction libère la structure PACKET pointée par lpPacket. Désallouée par le programmeur. Elle a comme paramètres :

VOID Packet_Free_Packet (LPPACKET lpPacket)

7. Packet_Receive_Packet () :

La fonction Packet_Receive_Packet () a les paramètres suivants :

BOOLEAN Packet_Receive_Packet (LPADAPTER AdapterObject, LPPACKET lpPacket, BOOLEAN Sync)

Cette fonction exécute la capture d'une série de paquets de données. Ses paramètres d'entrée sont les suivants :

1- Un pointeur vers la structure ADAPTER identifiant la carte réseau sur laquelle les paquets doivent être capturés ;
2- Un pointeur vers la structure PACKET qui contiendra le paquet ;
3- Un drapeau qui indique si l'opération doit être exécutée dans un mode synchrone ou asynchrone. Si l'opération est synchrone, la fonction bloque le programme jusqu'à ce que la tâche soit complétée (Exemple : les paquets ont été reçus). Si l'opération est asynchrone, la fonction ne doit pas bloquer le programme de capture et la procédure Packet_Wait_Packet () doit être utilisée pour vérifier l'achèvement correct.

Note :

Le pilote de capture a été crée et testé pour travailler sur un appel de système synchrone, par conséquent, le mode synchrone est celui qui est suggéré. Le nombre de paquets reçus avec cette fonction, ne peut pas être connu avant l'appel de la fonction et il est largement variable. Il dépend du nombre de paquets actuellement sauvegardés dans le buffer du pilote, de la taille de ces paquets capturés et de la taille du buffer associé avec le paramètre lpPacket. La figure 5.11 montre le format utilisé par le pilote pour envoyer les paquets à l'application de capture de l'utilisateur.

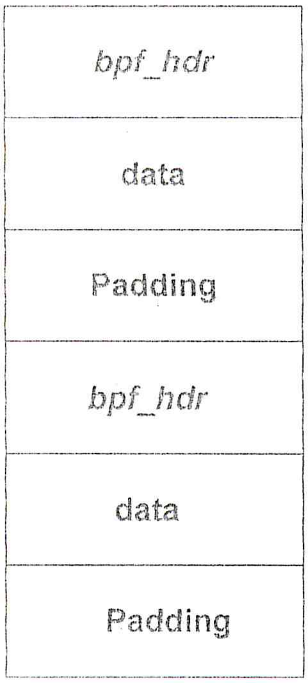


Figure 5.11: Format des paquets envoyés par le pilote.

Le paquet est sauvegardé dans un buffer associé avec la structure IpPacket PACKET. Chaque paquet possède sa remorque consistante dans la structure bpf_hdr qui définie sa taille et saisie son en-tête (timestamp). Le champ Padding (Bourrage) est utilisé pour compléter les données dans le buffer (pour augmenter la vitesse de copie). Les champs bh_datalen et bh_hdrlen des structures bpf_hdr doivent être utilisées pour extraire les paquets du buffer.
La librairie Pcap extrait correctement chaque arrivage de paquets capturés avant qu'elle ne le fasse passer à l'application de l'utilisateur, ainsi l'application qui l'utilise n'a pas de se soucier de cette opération.
[DEG 00]

IV.4. Libpcap et C++ :

La librairie Libpcap a été écrite pour être utilisée dans des programmes C. La raison est que la version originale de libpcap était destinée à la famille des systèmes d'exploitations UNIX, où le C est le langage le plus utilisé. Dans Windows, C++ (avec la classe de librairie MFC) est le langage de programmation le plus utilisé. Pour faciliter la programmation et l'utilisation de cette librairie de capture au programmeur, un nouveau fichier include, appelé Pcap.h, à été crée. Il exporte les fonctions de libpcap dans les conventions d'appels de C++, et doit être inclus dans le fichier C++ du programmeur.

IV.5. Choix du langage et de l'environnement de développement :

Avant la réalisation de n'importe quel logiciel, on passe par une étape délicate qui est le choix du langage de programmation. Pour qu'on puisse garder les aspects de notre architecture il faut choisir le langage le plus adéquat. Vu que l'analyseur est développé dans un environnement réseau, va utiliser la technique multi-thread et ses modules manipulent beaucoup de pointeurs et de références (comme on a vu les fonctions de libpcap qui utilisent beaucoup de pointeurs). On a choisi Visuel C++ (VC++) comme langage de programmation pour développer notre application et on a laissé JAVA. VC++ est un langage orienté objet qui offre un environnement riche d'utilitaires grâce auxquels les développeurs évitent des tâches fastidieuses et se consacrent au programmes principaux.

La fonctionnalité la plus importante de VC++ est le fait qu'il travaille à la fois avec la librairie MFC et le langage C++ par l'interface de programmation des applications Windows (API). L'API Windows, large palette de fonctions implémentées dans un jeu de librairie dynamique apparue avec Windows, a été la base de toute programmation sous Windows.
La librairie MFC (Microsoft Foundation Class) est une librairie qui comprend presque toutes les fonctions implémentées dans l'API Windows et les organise en hiérarchie de classes d'où leur simplicité d'emploi.
Windows autorise l'édition de liens dynamiques avec les modules objet d'un programme. Cela signifie que des bibliothèques spécialement conçues à cet effet peuvent être chargées et liées à l'application de capture (exemple : Libpcap.lib, Packet.dll, etc...).

[AND 94]

V. REALISATION DE MekAnal :

MekAnal (Analyseur de Mekfouldji) est une application (sous windows) qui capture les paquets du réseau. Elle expose tous les paquets transmis sur le réseau local et donne des informations détaillées sur chaque en-tête de paquet. On ne va pas traiter les protocoles de niveau application car notre étude se limite aux trois couches lien, interréseau et transport. Si vous êtes intéressés, vous pouvez ajouter les caractéristiques pour supporter les divers protocoles de niveau application tels que SMTP, FTP,...etc.

V.1. Fonctionnement :

Lorsque votre machine est sur le réseau, des paquets de différentes destinations arrivent. Par défaut (c'-à-d quand la carte réseau est en mode normal), ces paquets sont refusés par la carte réseau puisqu'ils sont destinés aux différents hôtes. Mais si vous voulez, vous pouvez recevoir ces paquets en mettant la carte réseau en mode promiscuous. Dans ce mode, tous les paquets seront acceptés sans tenir compte de l'adresse de destination.
Désormais, vous pouvez analyser les paquets transmis sur votre réseau. Cette manie est utilisée pour déterminer le trafic afin de faire un bon management du réseau entre autres. Vous recevrez les paquets de différentes destinations, si vous utilisez un HUB. Le HUB utilise la technique de diffusion pour transmettre les paquets à tous les hôtes qui lui sont attachés. Cependant, si vous utilisez un SWITCH (un dispositif intelligent), vous ne recevrez donc aucun paquet envoyé à d'autres hôtes sur le réseau.

V.2. Implémentation et résultats:

Voilà un extrait important du code source de l'application qui concerne l'étape du sniffing. Il est accompagné des illustrations montrant l'interaction de l'utilisateur avec l'interface du logiciel.
Lorsque vous faites marcher l'application, la fenêtre principale apparaît. Cliquez sur l'élément du menu Démarrer Capture pour commencer la capture.

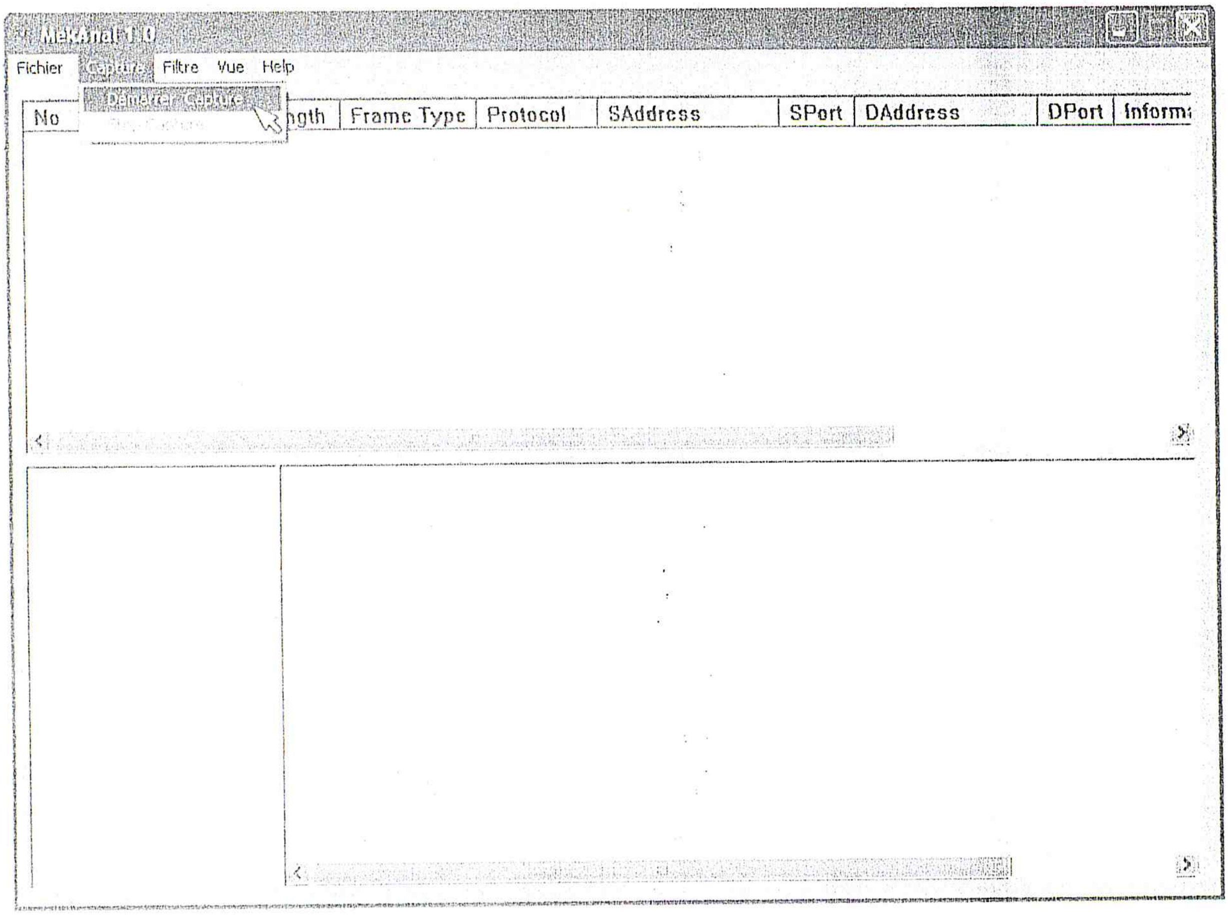


Figure 5.12: Lancement de la capture des paquets.

Tout d'abord, vous devez obtenir la liste des cartes réseau disponibles, ensuite ouvrez la bonne en mode promiscuous. Vous pouvez aussi spécifier la taille du paquet et la valeur du timeout de lecture (dans le programme).

```
// Obtenir la liste des cartes réseau pour capturer les paquets
pcap_findalldevs(&devlist,err);

// Ouvrir la carte en mode promiscuous
hdev=pcap_open_live (devname[index], // nom de la carte
65536, // taille -> capturer le paquet entier
1, // mode promiscuous 1000. // timeout de lecture
err
);
```

Une boite de dialogue s'affiche, maintenant sélectionnez la carte.

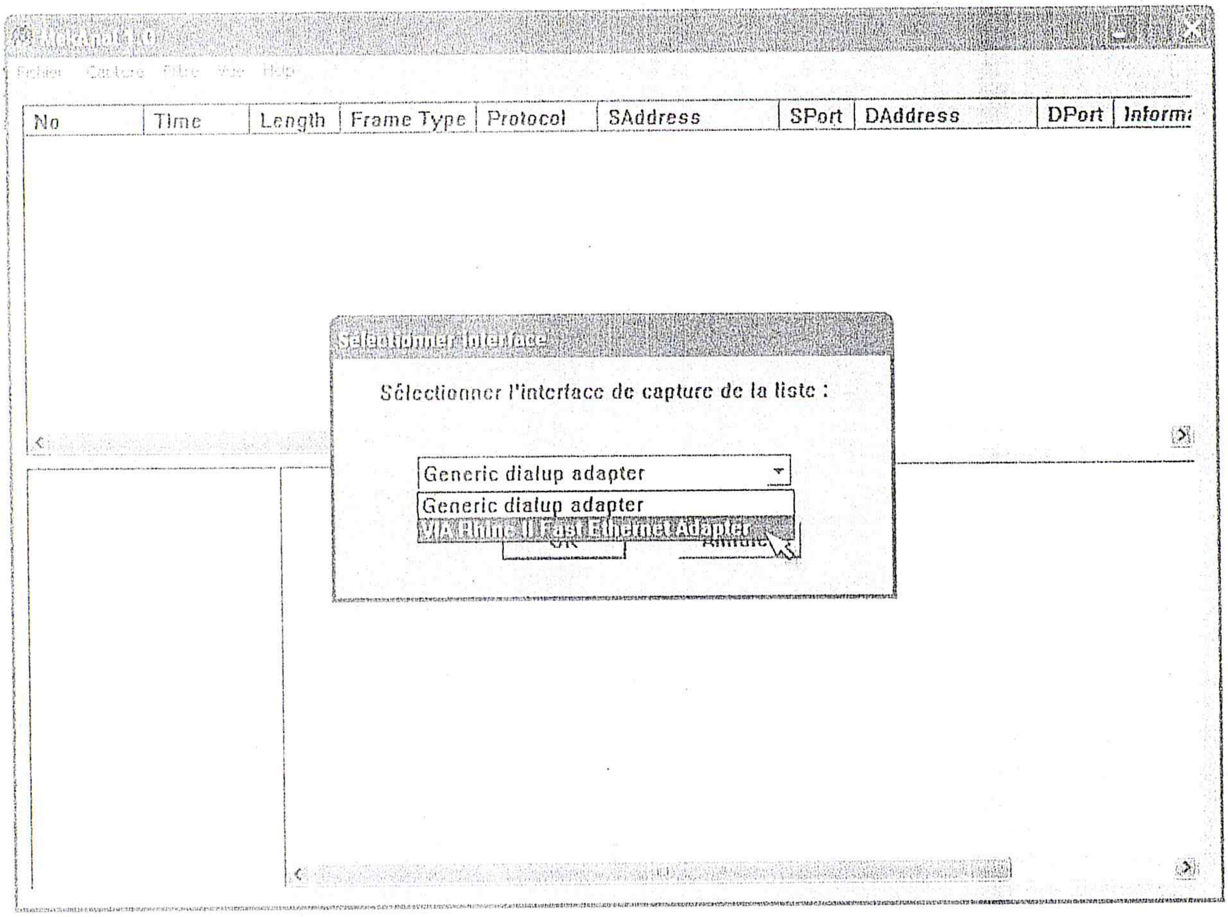


Figure 5.13: Sélection de la carte réseau.

Une fois vous avez ouvert la carte réseau, vous recevrez tous les paquets qui circulent dans le réseau en temps réel. Vous avez sur le contrôle «list box» des colonnes correspondant chacune à une caractéristique de la trame et des entrées ordonnées. A chaque fois une entrée ajoutée signifiant une nouvelle trame capturée, comme le montre la figure suivante :

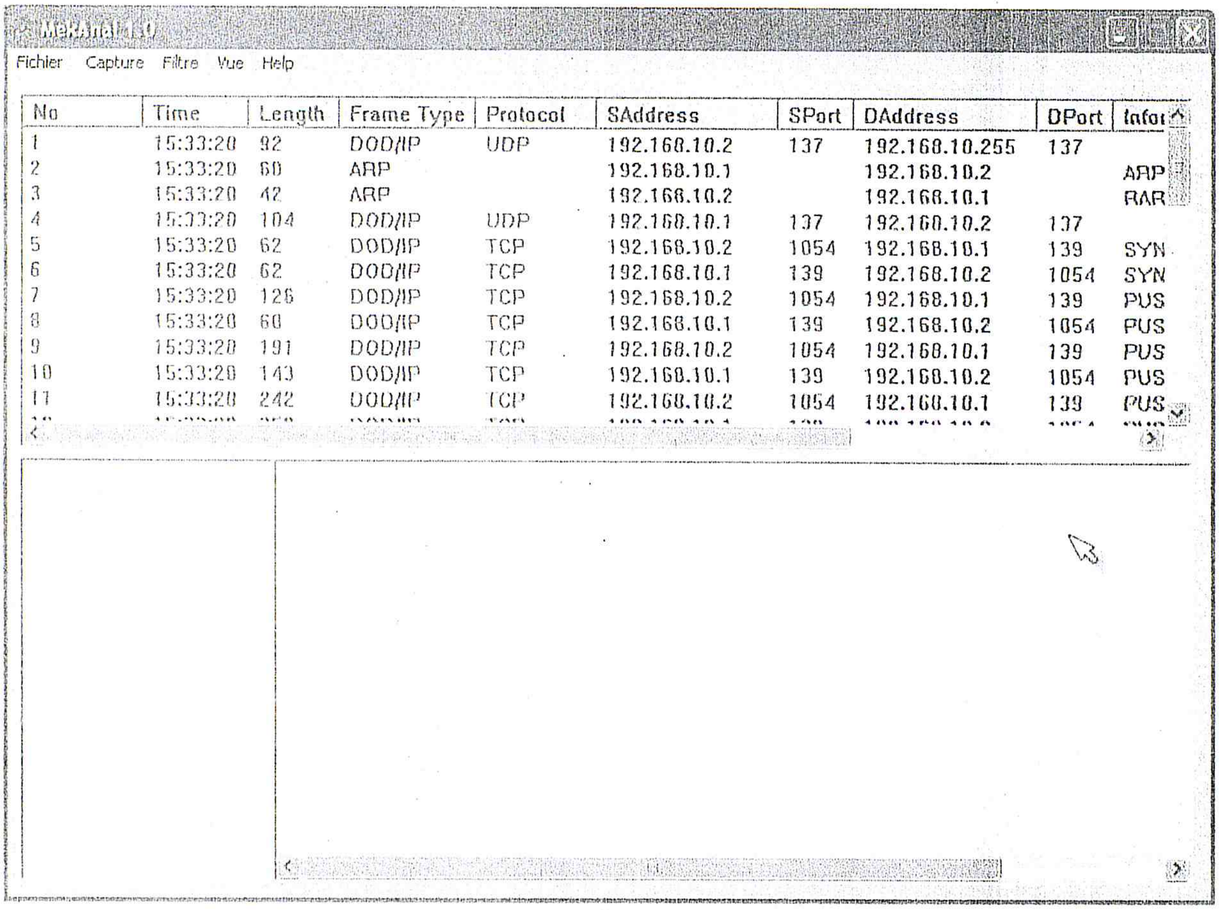


Figure 5.14: Exposition des échanges entre les équipements du LAN.

Les paquets sont affichés sur la fenêtre principale. Cliquez sur le numéro de la trame pour voir davantage de détails sur ses couches TCP/IP.

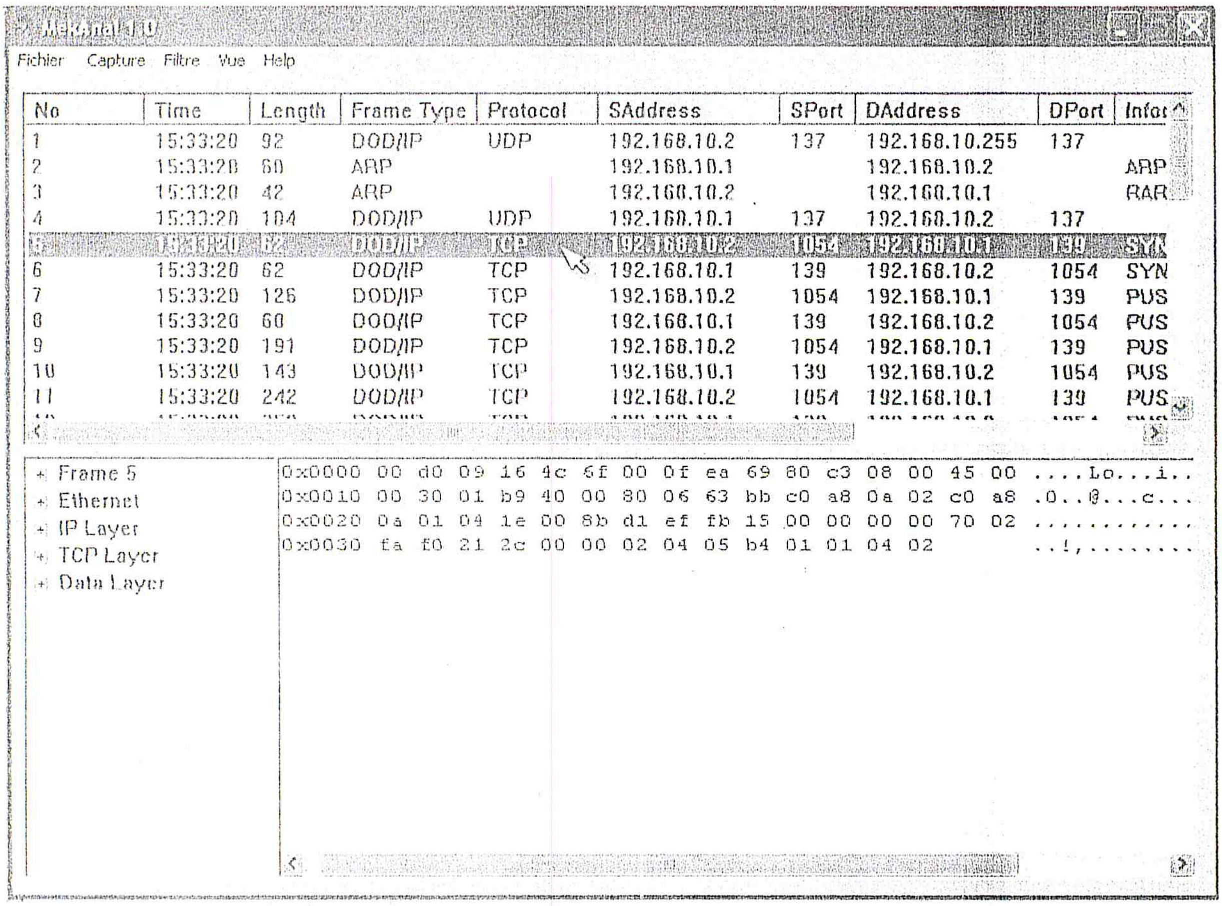


Figure 5.15: Affichage hexadécimal et ASCII de la trame.

Nous pouvons avoir des informations spécifiques concernant chaque couche en cliquant sur l'en-tête dans le contrôle «layer tree» et en déduire ainsi des conclusions.

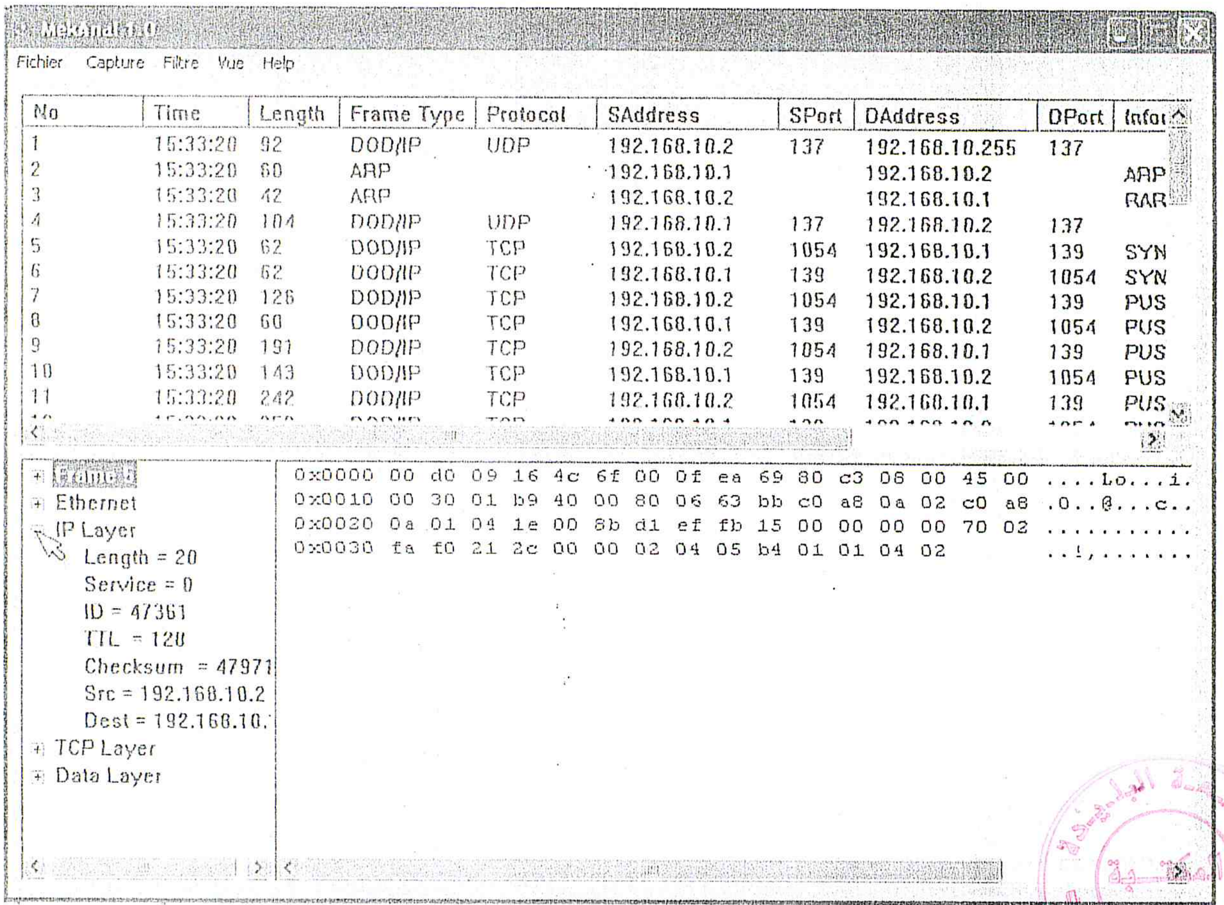


Figure 5.16: Décapsulation de la trame.

Si vous êtes intéressés par des paquets particuliers, comme les paquets provenant/allant de/à une machine précise alors vous avez besoin d'un filtre selon l'adresse IP.

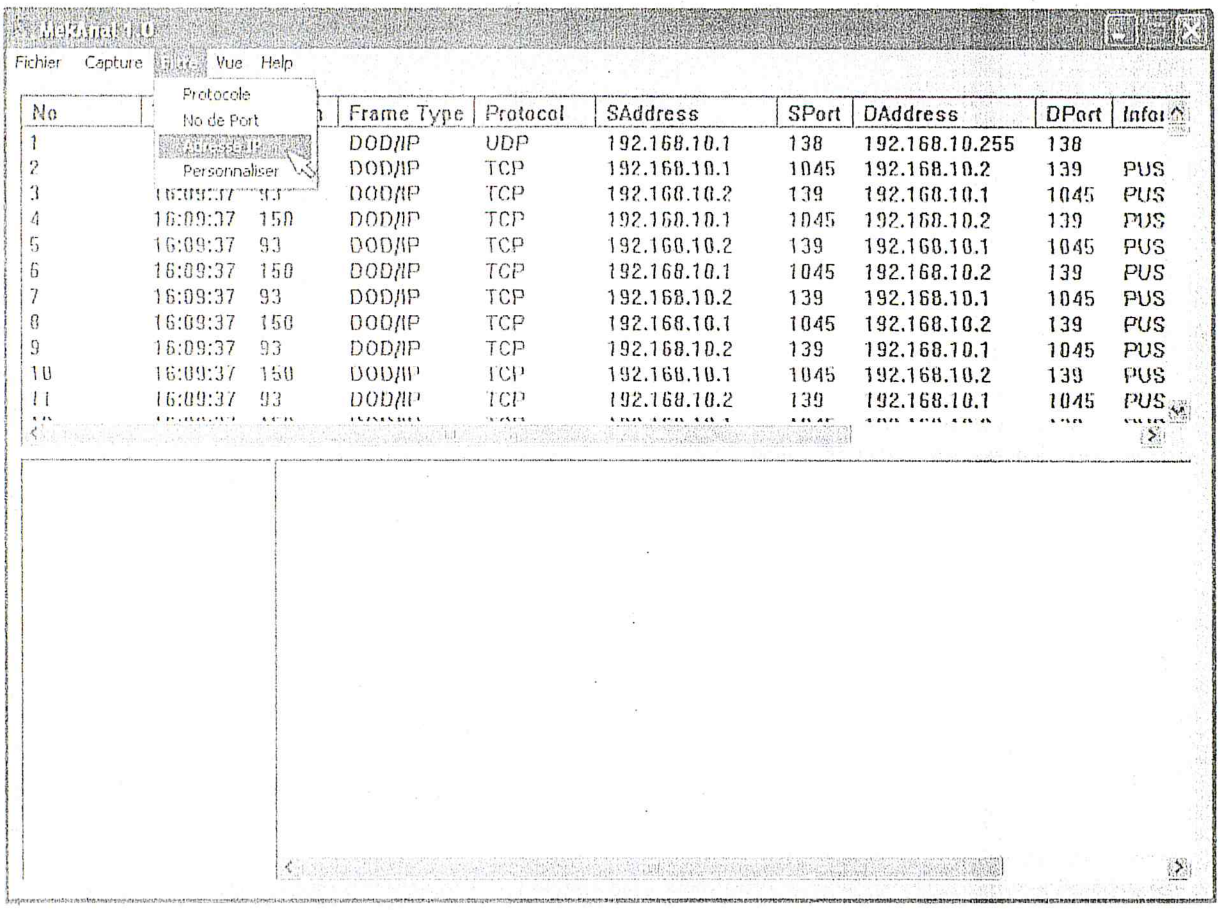


Figure 5.17: Choisir le type du filtre correspondant.

Vous allez activer le filtre, taper l'adresse IP (qui appartient au réseau local que vous analysez) et préciser la direction du paquet.

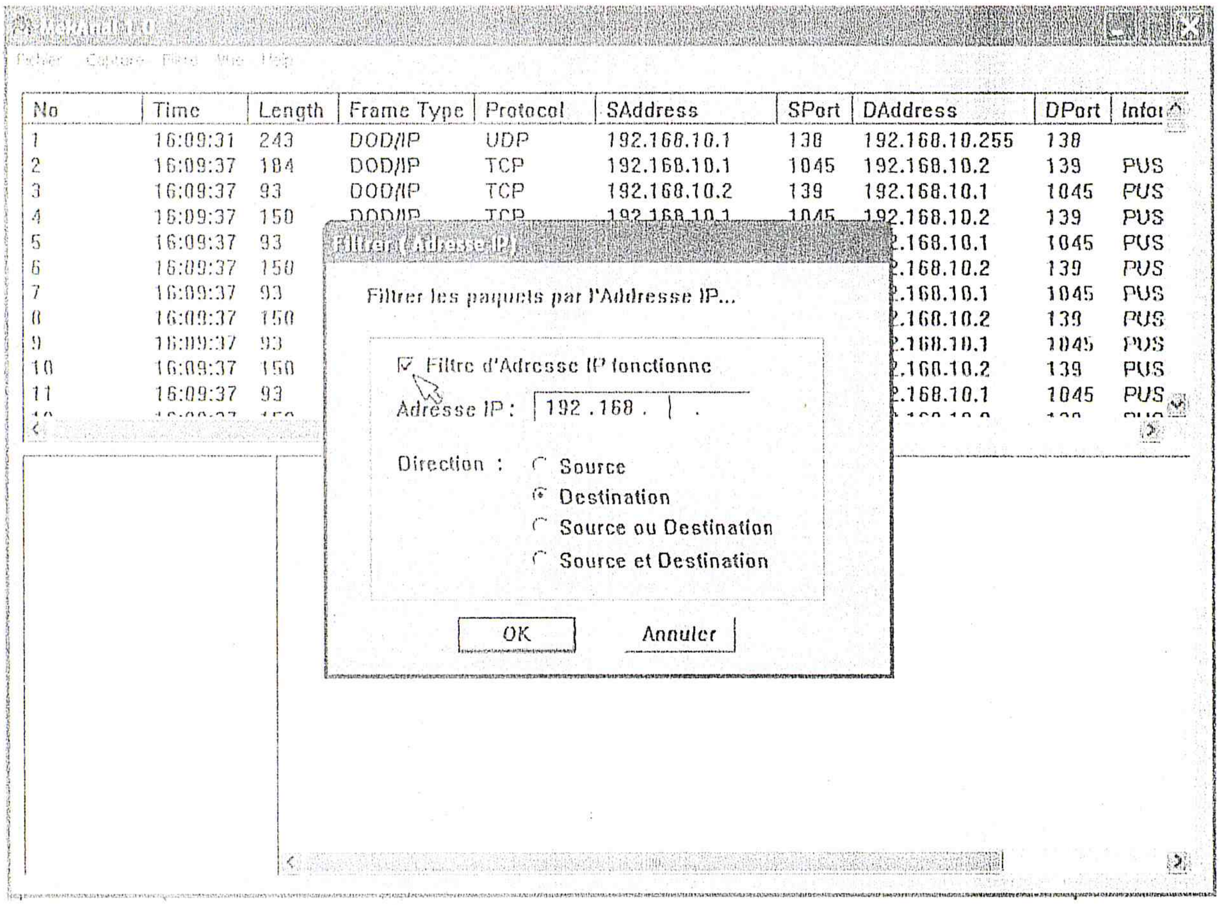


Figure 5.18: Filtrage des paquets par adresse IP.

Si vous trouvez important de contrôler, par exemple, seulement les paquets QUAKE (scan de port 27960) et les paquets ARP en même temps alors vous optez pour le filtre personnalisé. Ensuite vous pouvez écrire l'expression du filtre.

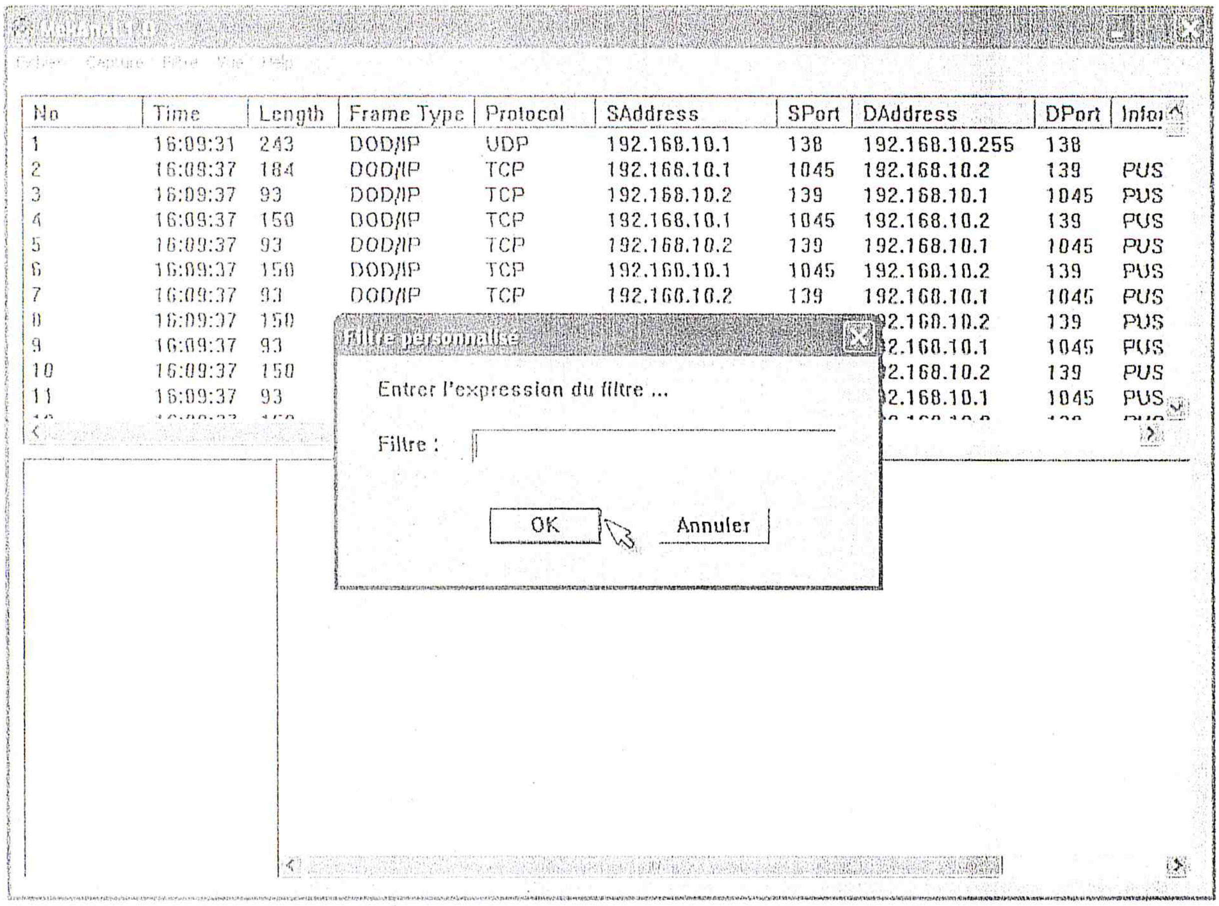


Figure 5.19: Ecrire l'expression du filtre.

Concernant la syntaxe de l'expression, voici un résumé :

Les filtres de Wpcap (version Windows de libpap) sont basés sur une syntaxe de prédicat déclarative. Un filtre est un caractère ASCII contenant une expression de filtrage. pcap_compile() prend cette expression et la convertit en un programme de filtre des paquets au niveau noyau. Seulement les paquets où l'expression du filtre est réalisée «vraie» pour eux seront acceptés.

L'expression consiste en un ou plusieurs primitives qui se composent d'un id (nom ou nombre) précédé d'un ou plusieurs qualificateurs. Il existe trois types différents de qualificateurs :

type

peut être : **host** nom de machine, **net** adresse réseau ou **port** numéro de port. Exemples : 'host foo', 'net 128.3', 'port 20'.

dir

spécifie une direction de transfert particulière and/or de id. Les directions possibles sont : **src**, **dst**, **src or dst** et **src and dst**. Exemples : 'src foo', 'dst net 128.3', 'src or dst port ftp-data'. Pour couches liaison 'null' (c'.à.d protocoles point à point comme slip) les qualificateurs **inbound** and **outbound** peuvent être utilisés pour spécifier une destination désirées.

proto

limite d'aller avec un protocole particulier. Les protos possibles sont : **ether**, **fddi**, **tr**, **ip**, **ip6**, **arp**, **rarp**, **decnet**, **tcp** et **udp**. Exemples : 'ether src foo', 'arp net 128.3', 'tcp port 21'. S'il n'y a pas de qualificateur proto, par exemple 'src foo' signifie '(ip ou arp ou rarp) src foo', 'net bar' signifie '(ip ou arp ou rarp) net bar' et 'port 53' signifie '(tcp ou udp) port 53'. (Dans notre cas les protocoles de TCP/IP)

En plus, il y a des mots clefs qui ne suivent pas les modèles cités : **gateway**, **broadcast**, **less**, **greater** et des expressions arithmétiques. Tous sont décrits en dessous.

Des expressions de filtre plus complexes peuvent être construites en utilisant les mots **and** (ou **&&**; concaténation), **or** (ou **||**; alternation) et **not** (ou **!**; Négation) pour combiner les primitives telles que :

dst host *host*
src host *host*
host *host* // source ou destination, peut être précédée de **ip**, **arp**, **rarp**, ou **ip6**
ether dst *ehost*
ether src *ehost*
ether host *ehost*
gateway *host*
dst net *net*
src net *net*
net *net*
net *net* **mask** *netmask* // syntaxe n'est pas valide pour IPv6
net *net/len* // vraie si l'adresse net est associée au masque de longeur len bits
dst port *port* // port peut être un nombre ou un mot comme **domain**
src port *port*
port *port*
less *length* // **len <=** *length*
greater *length*
ip proto *protocol* // s'il y a plusieurs, ils doivent être séparés d'un antislash (\)
ip6 proto *protocol*
ip6 protochain *protocol* // vraie si le paquet est IPv6 et contient une en-tête de type // *protocol* dans sa chaîne d'en-têtes
ip protochain *protocol* // IPv4
ether broadcast
ip broadcast
ether multicast
ip multicast
ip6 multicast
ether proto *protocol*
vlan *[vlan_id]*
iso proto *protocol* // protocole de paquet OSI, *protocol* peut être un nombre ou l'un // des trois noms *clnp*, *esis*, ou *isis*.

expr relop expr

relop est l'un de >, <, >=, <=, =, !=, et *expr* est une expression arithmétique composée d'entiers constants (exprimés dans la syntaxe du C standard), les opérateurs binaires normaux [+, -, *, /, &, |], un opérateur de longeur, et des accessoires de paquets de données spéciaux. Pour accéder aux données à l'intérieur du paquet, utilisez la syntaxe suivante :
proto [expr : size]

Par exemple, 'ether[0] & 1 != 0' attrape tout le trafic multicast. L'expression 'ip[0] & 0xf != 5' attrape tout les paquets IP avec options. L'expression 'ip[6:2] & 0x1fff = 0' attrape seulement les datagrammes in fragmentés et le fragment zéro des datagrammes fragmentés. Cette vérification est appliquée implicitement aux operations d'index de **tcp** et **udp**. Par exemple, **tcp[0]** toujours veux dire le premier octet de l'en-tête TCP et jamais le fragment intervenant.

Quelques offsets et valeurs de champ peuvent être exprimés par des noms à la place de valeurs numériques. Les offsets de champ de l'en-tête de protocole disponibles sont : **icmptype**, **icmpcode** et **tcpflags**.

Les valeurs suivantes du champ type de ICMP sont disponibles: **icmp-echoreply**, **icmp-unreach**, **icmp-sourcequench**, **icmp-redirect**, **icmp-echo**, **icmp-routeradvert**, **icmp-routersolicit**, **icmp-timxceed**, **icmp-paramprob**, **icmp-tstamp**, **icmp-tstampreply**, **icmp-ireq**, **icmp-ireqreply**, **icmp-maskreq**, **icmp-maskreply**.

Les valeurs suivantes du champ flags de TCP sont disponibles: **tcp-fin**, **tcp-syn**, **tcp-rst**, **tcp-push**, **tcp-push**, **tcp-ack**, **tcp-urg**.

NB : N'oubliez pas les parenthèses dans la combinaison des primitives afin de préciser la priorité des opérations.
[WTC 01]

```
// compiler le filter
pcap_compile(hdev,&fcode,filter,1,netmask);
// maintenant établir le filtre
pcap_setfilter(hdev,&fcode);
```

Une fois que vous avez ouvert la carte et placé le filtre, maintenant vous êtes prêts à recevoir les paquets. Une fois le paquet reçu, l'en-tête contient la longueur, le temps et autre informations sur le réseau. Et pkt_data contient les contenus exacts du paquet en commençant par l'en-tête Ethernet.

```
While(true)
{
pcap_next_ex(hdev,&header,&pkt_data);
// faites ce que vous voulez...
}
```

Afin d'analyser les contenus du paquet, vous devez vous familiariser avec les divers formats d'en-têtes. Vous devez surtout connaître le format des en-têtes suivantes ETHERNET, ARP, IP, TCP, UDP, ICMP et IGMP. On a inclut le fichier protocol.h qui contient les informations du format sur tous ces en-têtes. Si vous voulez autres détails, vous pouvez vous référez au RFCs.

Une fois que vous avez fait le travail, il est temps que vous fermiez carte en sûreté.

```
// fermer le dispositif
pcap_close(hdev);
```

Vous pouvez sauvegarder n'importe quel paquet en cliquant sur l'élément du menu Enregistrer Trame.

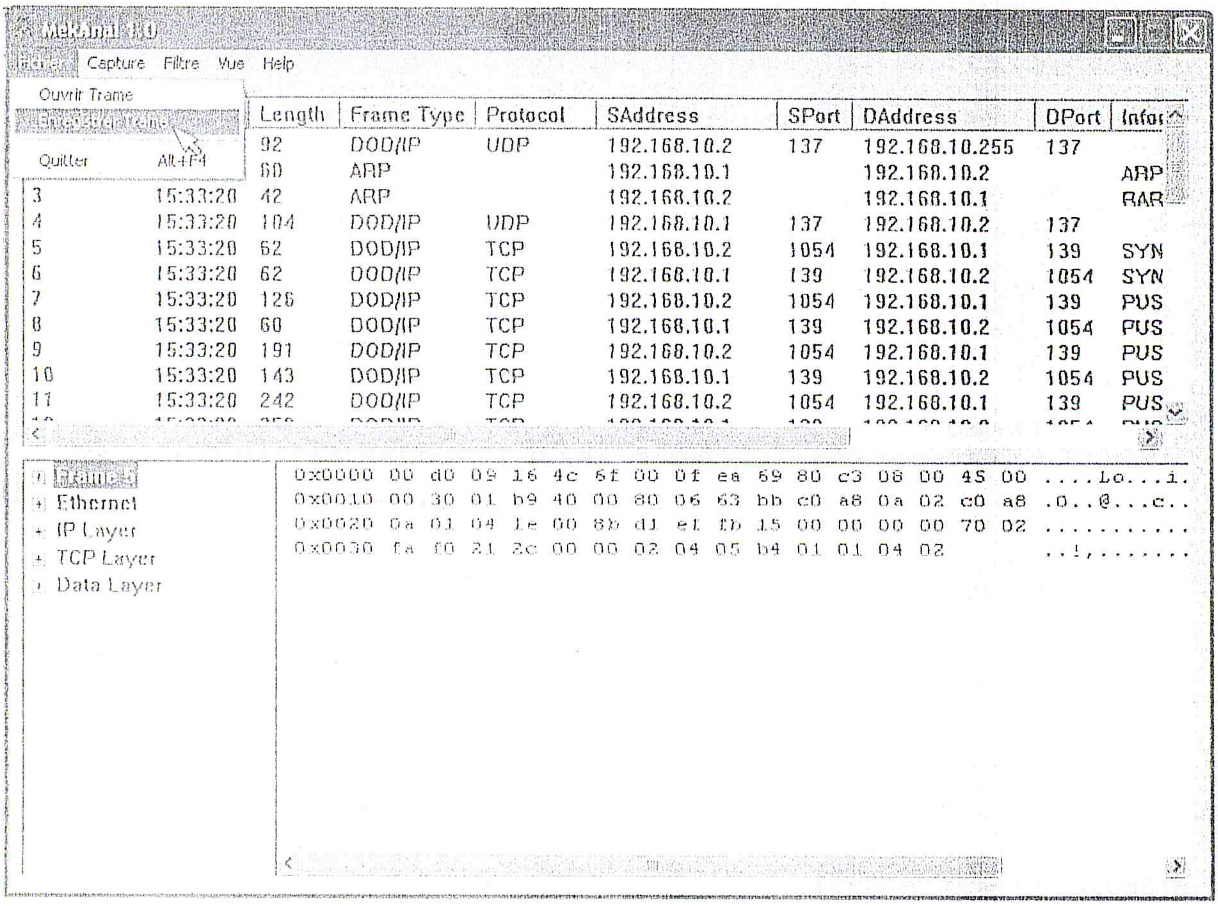


Figure 5.20: Sauvegarde d'un paquet sélectionné.

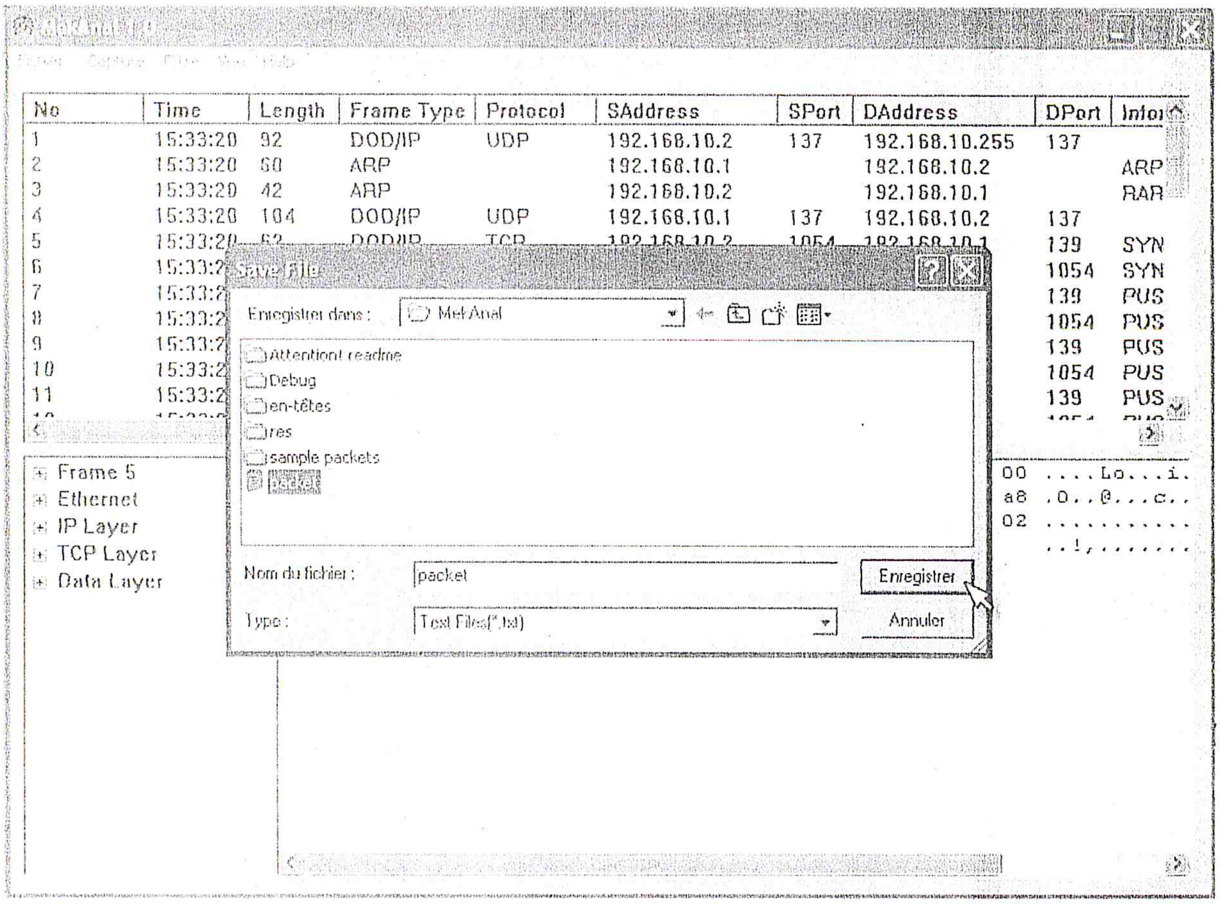


Figure 5.21: Mettre son contenu dans un fichier texte.

Plus tard, vous pouvez ouvrir ce fichier sauvegardé. Si vous n'avez pas de carte réseau ou vous n'êtes pas sur le réseau, on a inclut quelques paquets échantillons dans le dossier sample packets joint au code source. Vous pouvez ouvrir ces dossiers et voir leurs contenus.

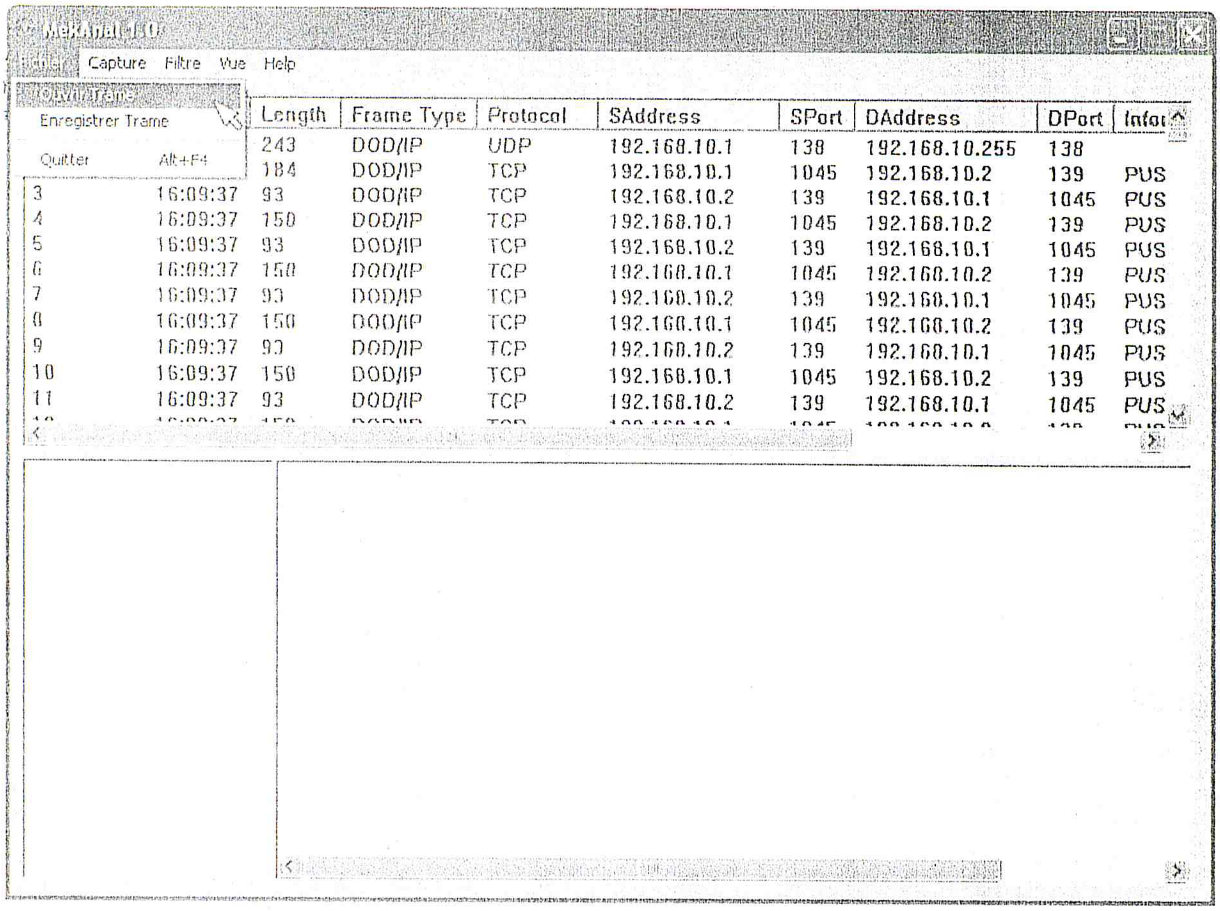


Figure 5.22 : Ouvrir un fichier existant.

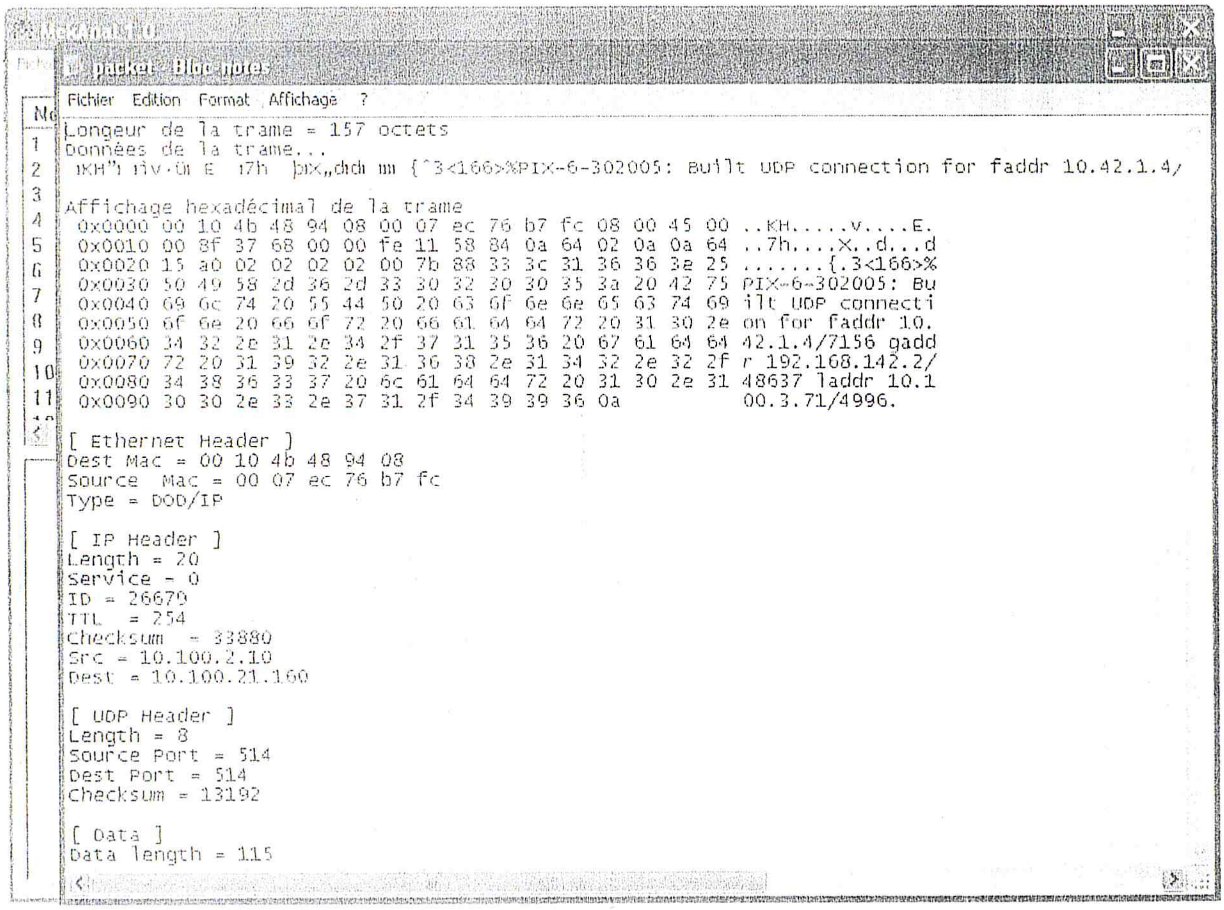


Figure 5.23: Visualiser le contenu de la trame.

VI. CONCLUSION

Nous avons pris comme prétexte dans notre projet de concevoir un sniffer. Cependant, nous aurons bien compris que ce que nous avons réalisé pour extraire des paquets du réseau l'est aussi pour en injecter.

Nous avons présenté une bibliothèque qui rend transparent toute interaction directe avec le système, offrant ainsi une meilleure portabilité. Nous avons vu que la réalisation d'un sniffer de trafic est relativement simple avec cette bibliothèque. Cependant, elle souffre de quelques manques qui sont heureusement comblés par la bibliothèque Nids.

Celle-ci permet de suivre un échange de datagramme UDP, de réassembler une communication TCP à la manière de noyau Linux ou encore de défragmenter les paquets IP. L'utilisation de libnids est capitale si l'information à traiter se trouve au niveau applicatif.

Pour conclure brièvement, disons juste que l'implémentation d'un analyseur du trafic d'un réseau ethernet avec pcap est à la fois triviale et puissante.

CONCLUSION GENERALE

J'ai présenté à travers mon travail une étude visant à contribuer à la conception et à la réalisation d'un logiciel qui fait la capture des trames dans un réseau local Ethernet. Tout au long de la conception du projet, j'ai pu découvrir le monde agréable des réseaux informatiques en général et les réseaux locaux en particulier. Ce monde qui a évolué cette dernière décennie d'une manière fascinante permettant de relier des centaines de millions d'ordinateurs distribués partout dans le monde.

L'étude de la pile de protocoles TCP/IP m'a permis de comprendre des notions très importantes concernant les mécanismes de fonctionnement de très grands réseaux mondiaux tel que Internet ainsi que les réseaux locaux comme Ethernet. Ces notions m'ont offert une vision plus précise sur les protocoles et les applications utilisant l'Internet pour réaliser leurs différents objectifs. Le principe CSMA/CD m'a appris comment accéder au média propre au réseau Ethernet et les règles qui harmonisent la communication entre les stations. L'étude du pilote de capture des paquets Ethernet -BPF Packet Capture Driver- destiné à la plate-forme de Windows, m'a appris le fonctionnement interne de ce pilote, notamment la communication avec le gestionnaire de la carte Ethernet et les différents critères de la capture qui répondent aux besoins initiaux des utilisateurs.

Egalement, l'étude de l'architecture de capture m'a permis de définir en détail les différents niveaux de sa constitution et procéder à l'analyse des paquets Ethernet ; cette dernière étant le but primordial de mon projet et de comprendre le rôle de chaque niveau sans oublier l'interaction entre ces niveaux et les principes de fonctionnement.

L'ajout des différentes librairies de fonctions (Packet.dll et la Libpcap) m'a conduit à apprendre un outil de programmation puissant, le visuel C++ ; ce dernier m'a permis de voir de plus près les différentes techniques de programmation des applications de bas niveau. Ces techniques consistent à dialoguer avec l'interface réseau et le gestionnaire de la carte réseau Ethernet en utilisant des fonctions et des procédures bien spécifiques, qui se trouvent dans les librairies citées ci-dessus.

Plus tard, on peut envisager l'extension de notre application afin d'effectuer une capture et une analyse à distance. Puisque l'un des buts de libpcap est d'être indépendant du matériel, il est possible d'étendre son niveau d'abstraction pour capturer des paquets à travers un équipement distant. Cela nécessite la création d'un serveur de capture et une extension de libpcap capable de communiquer avec lui d'une manière transparente. Une caractéristique très intéressante de Winpcap pouvant être utilisée à distance est le mode statistique car il n'a besoin de transférer qu'une très faible quantité de données dans le réseau. Ainsi, il peut travailler en temps réel sans influence sur le fonctionnement du réseau.

Enfin, l'expérience que j'ai acquise dans le cadre de mon mémoire de fin d'études a été très enrichissante. Elle constitue une base solide qui va me permettre d'aborder des travaux plus complexes et d'apprendre à penser et comprendre en même temps comment exploiter les moyens et les outils disponibles afin de réaliser un travail qui répond au mieux à nos besoins.

BIBLIOGRAPHIE

[AND 94] Mark Andrews,
«Formation Visual C++»,
Microsoft Press,1994.

[BEL 95] Steven M. Bellovin, AT&T Bell Laboratories,
«Using the domain Name System for System break-in»,
Proceedings of the Fifth Usenix UNIX Security Sympsium, Salt Lake City, Juin 1995.

[BEL 97] Mark Bell,
«Protecting Networks With and Without Firewalls
Part 1 - The Sniffer Attack»,
Technical Entreprises, Inc, 1997.

[BFI 97] T. Berners-Lee, MIT/LCS, R. Fielding, UC Irvine, H. Frystyk, MIT/LCS,
Traduction de Valéry G. FREMAUX Ingénieur Professeur / EISTI,
Edition originale : Mai 1996 / Version française : Septembre 1997.

[CLA 96] Brecht Claerhout,
«A Short Overview of IP Spoofing», 1996
http://www.rootshell.com .

[DEG 00] Loris Degioanni,
«Development of an Architecture for Packet Capture
and Network Traffic Analysis»,
Ecole Polytechnic de Torino (Italy) ,2000.

[DEG 01] Loris Degioanni, Paolo Politano, Fulvio Risso and Piero Viano,
«Analyzer : a public domain protocol analyzer»,
Politecnico di Torino, Mars 2001,
http://netgroup-serv.polito.it/analyzer .

[DES 99] Guillaume Desgeorge,
«Synthèse des protocoles courants» 98/99,
guillweb@hotmail.com .

[DRU 00] Jason Drury,
«Sniffers : What are they and How to Protect From Them»,
November 11, 2000 .

[FIN] ftp://ftp.inria.fr/rfc .

[GRA 00] Robert D. Graham,
«FAQ : Network Intrusion Detection Systems»,
Version 0.8.3, March 21, 2000
http://www.robertgraham.com/pubs/network-intrusion-detection.html .

[GRI 97] Richard Grin,
«Introduction à l'utilisation des réseaux informatiques»,
Université de Nice Sophia-Antipolis, 19 décembre 1997.

[GRU 98] Stéphane Grundschober,
«Sniffer Detector»,
IBM Research Division, Zurich Research Laboratory, Global Security Analysis Lab, Juin 1998.

[HAL 94] Haller, N.,
«The S/Key One-time Password System»,
Proceeding of the Symposium on network & Distributed Systems,
Security Internet Society, San Diego, CA, February 1994.

[HAY 02] Matt Hayden,
«Les réseaux»,
CampusPress, 2002, ISBN 2-7440-1381-1.

[JON 95] Laurent Joncheray,
«A simple Active Attack Against TCP»,
Merit Network, Inc. 24 avril 1995.

[KAR 98] Karanjit S.Siyan,
«TCP/IP»,
Simon & Schuster Macmilan, ISBN 2-7440-0391-3. 1998.

[KEB] Kent Beck,
«Extreme Programming explained»,
Addison Wesley, 0-201-61641-6, 2000.

[KLA 96] Christopher Klaus,
«Sniffer FAQ v 3.00»,
Internet Security Systems, Inc. 1996.

[KOH 93] J. Kohl et C. Neuman,
«The Kerberos Network Authentification Service (V5)»
Request for Comments (RFC) 1510, Network Working Group,
Septembre 1993.

[LAN 95] «Rapport de sécurité LANT du Naval Computer & Telecommunication Area Master Station»,
Juillet 1995.
http://www.chips.navy.mil/chips/archives/95_jul/file14.html.

[MCC 92] Steven McCanne & Van Jacobson,
«The BSD Packet Filter : A New Architecture for User-level Packet Capture».
Lawrence Berkley Laboratory, 19 decembre 1992.

[PNI 99] Pascal Nicolas,
De l'Université d'Angers,
article paru le mardi, 2 novembre 1999, 09:20:50 MET.

[PUJ 98] Guy Pujolle,
« Les Réseaux » deuxième édition revue et augmentée,
Editions Eyrolles, Paris, 1998, ISBN 2-212-08967-8.

[RFC xx] Request for comments,

Les RFC peuvent être obtenue via le site www.ds.internic.net

[STA 98] S. Staniford-Chen,
«Common Intrusion Detection Framwork»,1998
http://www.seclab.cs.ucdavis.edu/cidf/ .

[STE 98] W.Richard Stevens,
«TCP/IP illustré, les protocoles»,1994
traduction de Eric Tyberghier, 1998.

[WCE] http://www.centralweb.fr/ .

[WCO] http://coding.romainl.com/ .

[WET 01] http://www.ethereal.zing.org/ .

[WEX] http://www.extremeprogramming.org/.

[WKL 01] http://www.klos.com/ .

[WLA] http://www.labo-cisco.com/ .

[WME] http://www.mediatalk.fr/ .

[WNE 05] http://Neeho.net/ .

[WOO 00] JoAnne Woodcock,
«réseaux» cote if-620, ex 2, bibliothèque du département des sciences exactes, université Saâd Dahleb de Blida, Algérie 2000.

[WRE 01] http://reptile.rug.ac.be/~coder/sniffit/sniffit.html .

[WSN 01] http://www.sniffer.com/ .

[WSS 01] http://www.ssh.fi/ .

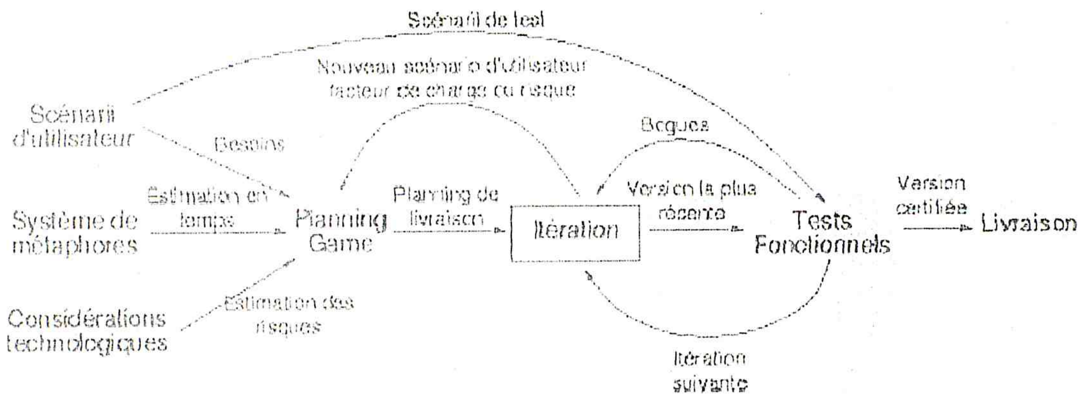
[WTC 01] http://www.tcpdump.org/ .

[ZIM 94] Philip Zimmermann,
«Pretty Good Privacy», août 1994.

Annexe A. Projet basé sur l'XP

A.1.

Figure A-1. Projet basé sur l'XP



1- Quelques principales fonctions de la librairie Pcap :

Packet_Wait_Packet () :

Cette fonction est utilisée pour vérifier l'achèvement de l'opération I/O sur le pilote Paquet Capture Driver. Il est bloqué si l'opération attend d'être complétée par le pilote. La valeur retournée est TRUE si l'opération est réussite, et FALSE dans le cas contraire. Les paramètres de cette fonction sont :

BOOLEAN Packet_Wait_Packet (LPADAPTER AdapterObject, LPPACKET lpPacket)

Packet_Send_packet () :

Cette fonction est utilisée pour envoyer des paquets bruts vers le réseau à travers la carte réseau spécifiée avec le paramètre AdapterObject. << Paquet brut >> signifie que le programme va construire plusieurs en-têtes parce que le paquet est envoyé vers le réseau tel qu'il est. L'utilisateur n'a pas à mettre l'en tête bpf_hdr avant chaque paquet. Et le CRC n'a pas besoin d'être calculé et ajouté au paquet à émettre, parce qu'ils sont mis d'une manière transparente après la fin de la portion de données à envoyer par l'interface réseau. Cette fonction a la même syntaxe que celle de Packet_ Receive_Packet ().

Le comportement de cette fonction est influencé par la fonction Packet_Set_Num_Writes (). Avec Packet_Set_Num_Writes, il est possible de mettre le nombre de temps dont une seule écriture peut être répétée. Si ce nombre est <<1>>, chaque appel de Packet_Send_Packet correspondra à un seul paquet émis vers le réseau. Si ce nombre est plus grand que <<1>>, par exemple <<1000>>, chaque paquet brut écrit par l'application sur le fichier du driver sera émis 1000 fois sur le réseau. Cette caractéristique peut être utilisée pour générer une vitesse de trafic élevée, parce que le surcoût de la commutation de contexte n'est à présent pas long et il est particulièrement utile d'écrire des outils de tests de réseaux, routeurs et serveurs.

Notons que la capacité d'écrire de multiples paquets est présentée en ce moment uniquement dans les versions de Windows NT et Windows 2000 du pilote Packet Capture Driver. Dans Windows 95/98 elle est simulée au niveau de l'utilisateur dans la librairie dynamique Packet.dll. Cela signifie que l'application qui utilise la méthode d'écriture répétée sera exécutée aussi bien sur Windows 9x, mais sa vitesse d'exécution sera très lente, comparée à celle de Windows NTx.
L'optimisation du processus d'émission est restée limitée à un seul paquet à la fois : pour le moment, il ne peut être utilisé (Processus) pour envoyer un buffer de N paquets. La fonction Packet_Send_Packet utilise comme paramètres :

BOOLEAN Packet_Send_Packet (LPADAPTER AdapterObject, LPPACKET pPacket, BOOLEAN Sync)

Packet_Reset_Adapter () :

Cette fonction permet de remettre (réinitialiser) la carte réseau comme paramètre d'entrée (initialiser à être prêt à recevoir les paquets provenant du réseau). Cette fonction retourne la valeur TRUE, si l'opération est exécutée avec succès. Les arguments de cette fonction sont :

BOOLEAN *Packet_Reset_Adapter* (LPADAPTER AdapterObject)

Packet_Set_Hw_Filter () :
Cette fonction positionne le filtrage hardware en arrivage des paquets. Les constantes qui définissent les filtres sont déclarées dans le fichier d'en-tête ntddndis.h. Les paramètres d'entrées sont : La carte sur la quelle le filtre doit être défini, et l'identification du filtre. La valeur retournée est TRUE, si l'opération est réalisée avec succès. On va présenter une liste des filtres les plus utilisés dans la capture des paquets du réseau :

- NDIS_PACKET_TYPE_PROMISCUOUS : Positionne le filtre en mode *promiscuous*. Chaque arrivage de paquet est accepté par la carte réseau.
- NDIS_PACKET_TYPE_DIRECTED : Uniquement les paquets destinés à la carte poste de travail qui sont acceptés.
- NDIS_PACKET_TYPE_BROADCAST : uniquement les paquets de Broadcast sont acceptés.
- NDIS_PACKET_TYPE_MULTICAST : uniquement les paquets de Multicast appartenant au groupe dans lequel cette carte réseau est membre, sont acceptés.
- NDIS_PACKET_TYPE_ALL_MULTICAST : chaque paquet multicast est accepté.

Cette fonction possède les paramètres suivants :
BOOLEAN Packet_Set_Hw_Filter (LPADAPTER AdapterObject, ULONG Filter)

Packet_Request () :
Cette fonction est utilisée pour exécuter l'opération de query / set sur la carte pointé par AdapterObject. Avec cette fonction il est possible d'obtenir ou de définir plusieurs paramètres de la carte réseau, tels que les dimensions des buffers internes, la vitesse de lien ou le compteur des paquets corrompus.
Le second paramètre définis si l'opération est mise à Set (Set = 1) ou à query (query = 1). Le troisième paramètre est le pointeur vers la structure PACKET_OID_DATA (voir la section Structure de données).
La valeur retournée est TRUE, si la fonction est accomplie sans erreurs. Les constantes qui définissent les opérations sont déclarées dans le fichier en-tête ntddndis.h. Cette fonction possède comme paramètres:
BOOLEAN Packet_Request (LPAAPTER Adapeterobject, BOOLEAN Set, PACKET_OID_DATA OidDta)

Packet_Set_Buff () :
Cette fonction est utilisée pour donner au buffer du driver de capture une nouvelle taille associée avec la carte pointé par AdapterObject. Dim est la nouvelle dimension en octets. La fonction retourne la valeur TRUE si l'opération est complétée avec succès, FALSE s'il n'y a pas de mémoire suffisante pour allouer le nouveau buffer. Lorsque la nouvelle dimension est mise, les données dans l'ancien buffer sont écartées et les paquets stockés dans ce buffer sont perdus. Les paramètres de cette fonction sont :
BOOLEAN Packet_Set_Buffer (LPADAPTER AdapterObject, INT Dim)
Remarque :
- La dimension du buffer du pilote affecte fortement sur les performances du processus de capture. L'application de capture a besoin de faire les opérations sur chaque paquet lorsque la CPU est partagée par d'autres taches. Par conséquent, l'application ne peut être travailler sur la vitesse du réseau durant un lourd trafic,

spécialement lorsque la CPU est trop chargée, dû à plusieurs applications. Ce problème est plus perceptible sur des machines lentes.
Le pilote, en d'autres parts, exécute au niveau noyau et est écris explicitement pour capturer des paquets, ainsi c'est plus rapide et le taux de perte des paquets est souvent réduit. Par conséquent, le buffer adéquat dans le driver est capable de sauvegarder au moment où l'application est occupée, compensant ainsi la lenteur de l'application et évitant la perte des paquets durant l'éclatement ou lors d'une activité élevée du réseau.

La taille du buffer est mise à <<0>> lorsque l'instance du pilote de capture est ouverte : Le programmeur doit se rappeler à la mettre à sa propre valeur.
- La librairie *Libpcap* appelle cette fonction et met la taille du buffer à 1MB au début de la capture. Par conséquent, les programmes qui utilisent Libpcap, souvent n'ont pas besoin d'affronter ce problème.

Packet_Set_Bpf () :
Cette fonction associe le nouveau filtre BPF à la carte réseau *AdapterObject*. Le filtre, pointé par fp, est une série d'instructions que le pilote exécutera sur chaque paquet. Cette fonction retourne TRUE si le pilote est positionné avec succès, FALSE si une erreur apparaît ou le programme de filtrage n'est pas accepté. Le pilote effectue une vérification sur chaque nouveau filtre pour éviter l'écroulement du système dû aux problèmes ou aux erreurs des programmes, et il rejette les filtres invalides. Le filtre peut être crée automatiquement en utilisant la fonction *pcap_compile* de la librairie Libpcap.
Les paramètres de cette fonction sont :
BOOLEAN *Packet_Set_Bpf* (LPADAPTER AdapterObject, struct bpf_program *fp)

Packet_Get_Stats () :
Avec cette fonction, le programmeur peut connaître la valeur des deux variables internes du pilote.
- Le nombre de paquets qui ont été reçue par la carte réseau AdapterObject, démarrant au moment auquel il était ouvert avec la fonction Packet_open_Adapter.
- Le nombre de paquets reçus par la carte réseau. Mais qui ont été perdus par le noyau. Le paquet est perdu lorsque l'application du niveau utilisateur n'est pas prête pour la recevoir et le buffer noyau associé à la carte réseau est plein.
Les deus valeurs sont copiées par le pilote dans la structure bpf_stat, fournie par l'application. Ces valeurs sont très utiles pour connaître l'état du réseau et le comportement (l'attitude) de l'application de capture. Elles sont aussi très importantes pour régler la pile de la capture et de choisir la taille des buffers.

En effet :
- La valeur élevée de la variable bs_recv signifie qu'il y a beaucoup de trafic sur le réseau. Si l'application n'a pas besoins de tous les paquets du réseau (par exemple l'application de l'analyseur réseau voudra éventuellement capturer uniquement le trafic qui est généré par des protocoles particuliers ou par un seul hôte), il est meilleur de mettre le filtre BPF sélectif pour minimiser le nombre de paquets que l'application va traiter.
Puisque le filtre travaille au niveau noyau, le filtre approprié augmente les performances de l'application et décroît la charge du système. Dans ce cas, les

paquets qui ne sont pas intéressants n'ont pas besoin d'être transférés au noyau et au niveau utilisateur.

- Si bs drop est grand que <<0>>, l'application est aussi lente et perd des paquets. Le programmeur peut essayer, comme première solution, de mettre un plus grand buffer noyau en utilisant la fonction Packet_Set_Buff (). Souvent, La taille convenable du buffer noyau réduit substantiellement la perte paquets. Une autre solution est d'accélérer le processus de capture en associant un grand buffer usager à la structure Packet utilisée dans l'appel Packet_Receiver_Packet (voir la fonction Packet_Init_Packet ()).
Les paramètres de la fonctions Packet_Get_Stats () sont :

BOOLEAN Packet_get_Stats (LPADAPTER AdapterObject, struct bpf_stat *s)

Packet_Get_Net_Type () :
Cette fonction retourne le type de la carte pointé par *AdapterObject* dans la structure NetType. Le *link_Tyle* du paramètre type peut être convertie par cette fonction vers l'une des valeurs suivantes :

- NdisMedium802_3 : Ethernet (802.3)
- NdisMedium802_5 : Token Ring (802.5)
- NdisMediumFddi : FDDI
- NdisMediumWan : Wan
- NdisMediumLocalTolk : LocalTalk
- NdisMediumArcNetRaw : ARCNET (raw)
- NisMediumArcnet 878-2 : ARCNET (878.2)
- NdisMediumWirlessWan : Various types of NdisWirlessXxx media

Le BPF Packet Capture Pilote en ce moment supporte NdisMediumWan, NdisMedium802_3, NdisMediumFddi, NdisMediumArcnet878-2 et NdisMediumAtm.
Le champ *LinkSpeed* indique la vitesse du réseau en bits par seconde.
La valeur retournée est TRUE si l'opération est exécutée avec succès. Les arguments utilisés dans cette fonction sont :
BOOLEAN *Packet_Get_Net_Type* (LPADAPTER AdapterObject, NetType *type)

Packet_set_Read_Timmeout () :
Cette fonction met la carte *AdapterObject* dans le mode <<*mode*>>. Mode peut avoir deux valeurs possibles :

- MODE_CAPT : c'est le mode de capture standard, qui est mis par défaut après l'appel de la fonction Packet_Open_Adapter ().
- MODE_STAT : C'est le mode statistique, c'est le mode de travail particulier du pilote de capture BPF Packet Capture Pilote qui peut être utilise pour exécuter des statistiques en temps réel sur le trafic réseau. Le pilote ne doit pas capturer n'importe quoi dans le mode statistique et il se limite de compter le nombre de paquets et le taux d''octets satisfaisant le filtre BPF définis par l'utilisateur.

Ces valeurs peuvent être sollicitées par l'application avec le standard Packet_Receive_Packet (), et sont reçus dans des intervalles réguliers, chaque fois le Timeout expire. La valeur par défaut de ce Timeout est <<1seconde>>, mais il peut être positionné en une autre valeur (avec une précision de 1ms) avec la fonction Packet_Set_Read_Timeout (). Les compteurs sont encapsulés dans la structure bpf_hdr avant qu'ils ne soient passés à l'application. Cela permet d'avoir un Timestamp Précis en microseconde pour avoir le même temps d'échelle parmi la capture des données dans ce cas, et une seule capture utilisant Libpcap. Les captures dans ce mode ont un impact très lent avec la performance du système. L'application qui voudra utiliser le Mode Statistique doit exécuter les instructions suivantes :

1. Ouvrir la carte réseau ;
2. Le mettre en mode statistique avec Packet Set Mode ;
3. Mettre le filtre qui définie les paquets qui seront comptés avec Packet_Set_BPF ;
4. Mettre le temps d'intervalle correct avec Packet_Set_Read_Timeout ;
5. Recevoir les résultats avec Packet_Receive_Packet (). Cette fonction retourne le nombre de paquets et le taux des octets satisfaisant le filtre.

2- Comment compiler une application qui utilise directement Packet.dll :

La création d'une application qui utilise le pilote de la capture à travers Packet.dll est entièrement triviale. Les étapes suivantes doivent être respectées pour compiler une telle application :

- Inclure le fichier en-tête Packet32.h au début de chaque programme fichier source qui utilise les fonctions exportées par la DLL. Packet32.h est distribué avec le code source de Packet.dll et elle est indépendante de la plate forme du système.
- Mettre les options de l'éditeur de liens pour inclure le fichier Packet.lib. Packet.lib est généré en compilant le *Packet Driver*.

L'application, ainsi faite, sera capable d'utiliser les fonctions exportées par la DLL et pourra utiliser le pilote pour capturer les paquets du réseau.

3- Les principales fonctions de la librairie dynamique (Packet.dll) :

Packet.dll fournit une série de fonctions qui peuvent être utilisées pour envoyer ou recevoir des paquets du réseau, solliciter et positionner des paramètres de la carte réseau, ouvrir et fermer la carte, manipuler l'allocation dynamique des structures PACKET, établir un filtre BPF, changer la taille du buffer du pilote et finalement extraire les statistiques de la session de capture. On décrit ici quelques fonctions principales :

1. Packet_ Get_ Adapter_ Names () : Cette fonction possède les paramètres suivants :
ULONG Packet_Get_Adapter_Names (PTSTR pStr, PULONG BufferSize) :
Souvent, c'est la première fonction qui doit être utilisée pour communiquer avec le pilote. Cette fonction retourne le nom (ou les noms) de la carte réseau installée dans le système. Ce nom, une fois récupéré, il est transféré vers le buffer utilisateur pour que ce dernier puisse choisir la carte réseau (l'interface réseau) sur lequel on veux lancer le processus de capture. Après le nom de la carte, nous retrouvons la chaîne qui la décrit. Dans le cas où on aurait plusieurs cartes installées dans le système, après chaque nom on a la description de la carte correspondante.

Pstr : Buffer d'allocation du niveau utilisateur. Ce dernier va contenir le(s) nom(s) de(s) carte(s) réseau installée(s) dans le système. Dans le cas où on a plusieurs cartes réseau installés dans le même système, (à partir de la version 2.02) il retourne après les noms de toutes ces cartes, la chaîne qui décrit chacune d'elles.

BufferSize : est la taille du buffer qui va contenir le nom de la carte réseau et la chaîne qui la décrit.

Remarques :

1. Le format du résultat obtenu dans le système d'exploitation Windows NT/2000 est différent de celui obtenu dans le Windows 9x. Windows 9x utilise la méthode du code ASCII pour sauvegardes la chaîne. Tandis que dans le Windows NTx on utilise UNICODE.

2. Après l'appel de la fonction Packet_Get_Adapter_Names () dans le système Windows 9x, pStr contiens la chaîne code ASCII avec les noms des cartes séparés par un seul code ASCII qui est «\0», le double «\0», suivi par la description des cartes réseaux séparés par un seul code ASCII «\0». La chaîne est terminée par un double «\0».

Exemple :

Prenons le cas où on a un ordinateur qui possède trois cartes réseaux de types différents comme suit :

_ Nom de la carte -1- : EL90x
_ Nom de la carte -2- : DF452x
_ Nom de la carte -3- : FR75s

Après l'appel de la fonction Packet Get Adapter Names (), le contenu de l'argument pStr (Buffer de l'utilisateur du niveau utilisateur) est le suivant :

EL90x \0 DF452x \0 FR75s \0\0 Réseau Ethernet \0 Réseau TokenRing \0 Réseau FDDI \0\0.

NB : Dans ce qui va suivre, on prend le cas d'un ordinateur qui possède une seule carte réseau.

2. Packet_ Open_ Adapter () : Les arguments de la fonction Packet Open Adapter sont :
LPADAPTER Packet_Open_Adapter (LPTSTR AdapterName)
Cette fonction reçoit la chaîne contenant le nom de la carte réseau à ouvrir et retourne le pointeur vers l'objet carte proprement initialisé. Les noms des cartes peuvent être obtenus par la fonction Packet Get Adapter Names ().

3. Packet_Close_Adapter () : Elle possède comme paramètre :
VOID Packet_Close_Adapter (LPADAPTER lpAdapter)
Cette fonction libère la structure ADAPTER lpAdapter, et ferme la carte pointé par cette fonction.

4. Packet_Allocate_Packet () :
La fonction Packet_Allocate_Packet () a comme argument :

LPPACKET Packet_Allocate_Packet (void)
Cette fonction alloue une structure de paquet appelée PACKET, et retourne un pointeur vers cette structure. La structure retournée doit être proprement initialisée par appel à la fonction Packet_Init_Packet ().

Remarque :

Le champ buffer dans la structure PACKET n'est pas positionné par cette fonction. Le buffer doit être alloué par le programmeur, et associé à la structure PACKET avec un appel à la fonction Packet_Init_Packet ().

5. Packet_Init_Packet () :
La fonction Packet_Init_Packet possède les arguments suivants :
VOID Packet_Init_Packet (LPPACKET lpPacket, PVOID Buffer, UINT Length)
Cette fonction initialise la structure PACKET. Elle a trois paramètres d'entrée :

1- Un pointeur vers la structure pour être initialisé ;
2- Un pointeur vers le buffer d'allocation de l'utilisateur qui contiendra le paquet;
3- La taille du buffer. C'est la taille maximale du buffer qui sera transférée du pilote à l'application en une seule entité de lecture.

4- Quelques exemples d'utilisation des fonctions de Packet.dll :

Les exemples que nous allons présenter (opérations effectuées sur le pilote de la capture et un dialogue avec la carte réseau), montrent comment écrire un programme de capture de paquet simplifié, dans un réseau local tel que l'Ethernet utilisant le NDIS Packet Pilote.
Ces exemples représentent des programmes de capture de paquets très simples, qui montre l'utilisation de Packet Capture Pilote à travers L'API Packet.dll.
Ces programmes une fois compilés, peuvent être exécutés sous les deux systèmes d'exploitation Windows 95, 98, NT et Windows 2000. Nous allons écrire quelques échantillons de programmes écris en langage visuel C++ qui se présente comme suit :

Programme -1- : Programme qui permet de sélectionner une carte réseau (lorsqu'il y a plusieurs Cartes)
Ce programme permet à l'utilisateur de choisir la carte réseau installé dans la machine, en suite il réalise la capture des paquets à partir de la carte réseau spécifié jusqu'à c e que l'utilisateur enfonce une touche (d'arrêt du programme de la capture), affichant ainsi le contenu des paquets sur l'écran.

```
// **** PROGRAMME DE SELECTION DE LA CARTE RESEAU ****
// Introduire les fichiers include
# Include <stdio.h>
# Include <conio.h>
............
#Include "..\..\Include\Packet32.h "
.............
#define Max_Num_Carte 10
// Prototypes
Void PrintPackets (LPADAPTER lpAdapter) ;
Char List_Carte [Max_Num_Carte] [1024]
```

```
Int main ()
{
//Définir un pointeur vers la structure ADAPTER
LPADAPTER lpAdapter ;
Int   i ;
DWORD dwErrorCode ;

// Les variables Nom_Carte représentent la chaîne qui va contenir la liste des noms
des Cartes réseaux, dans le cas où on a une machine qui possède plus d'un carte.

// Dans le cas du système Windows NT, définir des chaînes UNICODE
WCHAR Nom_Carte[512] ;
WCHAR *temp,*temp1 ;

// Dans le cas du système Windows 95, définir des chaînes en code ASCII
CHAR Nom_Carte_a_[512] ;
CHAR *temp,*temp 1a ;
Int Numero_Carte = 0 ; Open ;
ULONG Taille_Carte ;

// Définir le buffer qui va saisir les données provenant du pilote de la capture
Char *buffer [256000] ;

// La fonction utilisée est Packet_Get_Adapter_Names (), les données retournées
par cette fonction diffère entre Win NT et Win 95
// Dans le système d'exploitation Windows NT

Packet_Get_Adapter_Name (Nom_Carte, & Taille_Carte)
temp = Nom_Carte
temp1 = Nom_Carte ;
While ((*temp != " \0 ") ou (*(temp – 1) = ! " \ 0 ")
{
If (*temp == " \ 0 ")
{
Memcpy (Liste_Carte [i], temp1, (temp-temp1) *2) ;
Temp1 = temp +1
  i++ ;
}
temp ++ ;
}
Numero_Carte = i ;
For (i = 0 ; i < Numero_Carte ; i ++)
Wprintf (L " \ n%d- %s\ n " , i+1, List_Carte [i])
Printf (" \ n ") ;
Else
{
// Dans le système d'exploitation Windows NT
Packet_Get_Adapter_Nam (Nom_Carte_a_, & Taille_Carte)
tempa = Nom_carte_a_;
temp1a = Nom_Carte_a_ ;
```

```
While ((*tempa !=" \0 ") ou (*(tempa-1) = ! " \0 ")
{
If (*temp== "\0 ")
{
Memcpy (Liste_Carte [i], temp1, (temp-temp1)*2) ;
Temp1=temp+1 ;
i++ ;
}

Tempa ++ ;
}
}
Numero_Carte =i ;
For (i=0 ; i<Numero_Carte ; i++)
Printf ("\ n%d- %s\ n", i+1, List_Carte [i]))
Printf ("\ n") ;
}
Do
{
Print ("Sélectionner le numéro de la carte à ouvrir :") ; scanf ("%d", & Open)
Scanf ("% d, & Open) ;
If (Open >Numero_Carte)
Printf (" \n Le nombre de la carte doit être inférieur à %d", Numero_Carte)
}
While (Open >Numero_Carte) ;
..........
}
```

Programme -2- : Programme qui permet l'ouverture de la carte réseau après l'avoir sélectionné

```
.........
{

// Ouverture de la carte réseau sélectionné au préalable
lpAdapter=Packet_Open_Adapter (List_Adaptater [Open-1]) ;
If (! lpAdapter ou (lpAdapter -> hFile ==INVALID_HANDLE_Value))
{
dwErrorCode=GetLastError () ;
Printf (Impossible d'ouvrir le pilote, Code erreur : %1x \ n, dwErrorCode) ;
Return (-1) ;
}
```

Programme -3- : Programme qui permet d'initialiser la carte réseau en mode Promiscuous

Ce programme consiste juste à faire un appel à la fonction qui consiste o mettre la carte réseau sélectionné en mode promiscuous. Cela signifie que la carte réseau acceptera tous les paquets qui circulent dans le réseau et qui sont lus par cette carte (Même ceux qui ne sont pas destinés à cette station).

....

// Appel de la fonction qui permet d'établir un filtre pour la capture après l'ouverture de la carte.
Packet_Set_Hw_Filter (lpAdapter, NDIS_PACKET_TYPE_PRMICUOUS) ;
....

Programme -4- : Programme d'initialisation de la taille du buffer noyau
....
// Initialiser la taille du buffer du pilote de la capture à 512 k
Packet_Set_Buffer (lpAdapter, 512000) ;
....

Programme -5- : Programme qui fait l'allocation et l'initialisation de la structure paquet
Allouer et initialiser une structure PACKET qui sera utilisé pour recevoir les paquets. Notons que le buffer utilisateur sauvegarde uniquement sur 215 k mémoire. Pour de meilleurs performances de capture, une taille de buffer de 512 k (exemple : la même taille du buffer noyau) peut être utilisée.

// Allouer initialiser une structure PACKET qui sera utilisée pour recevoir les paquets.
....
If (lpPacket = Packet_Allocate_Packet ()) == NULL)
{
Printf (" \ n Error : Failed to allocate the LPPACKET structure.") ;
Return (-1)
}
Packet_Int_Packet (lpPacket, (char *) buffer, 256000) ;
....

Programme -6- : Programme qui affiche les statistiques de la capture
....
Packet_Get_Stats (LPPACKET lpPacket) ;
Printf ("\ n\ n%d Paquets reçus. \ n%d Paquets Perdus", stat.bs_recv, stat.bs_drop) ;
Packet_Free_Packet (lpPacket)
....

5- <u>Les principales différences de la version Libpcap entre Windows et UNIX</u> :

- Libpcap pour Win 32 a été étendue pour supporter le <<mode statistique>>, du pilote de capture de paquets, de façon très simple et facile, pour utiliser libpcap pour obtenir des statistiques en un temps réel sur les paquets qui transitent sur le réseau.
- La paire des fonctions spécifique à Windows qui a été crée est :
  - *pcap_setbuff* : Cette fonction est utilisée par l'application de capture pour mettre la dimension du buffer noyau dans le drive. Elle exploite la possibilité du pilote de capture pour changer la taille du buffer du pilote

quand c'est nécessaire. Cette possibilité n'est pas présente dans l'UNIX, lorsque le buffer est alloué d'une manière statique par le BPF et ne peut être changé. Puisque la dimension du buffer du pilote est paramètre important pour la capture, la fonction pour le positionner ou le modifier sans l'utilisation directe de l'appel de PACKET.DLL est très utile.

- *pcap_setmode* : Cette fonction peut être utilisée pour mettre la carte réseau en mode statique. Elle a été rajoutée pour permettre l'utilisation du mode statistique du BPF Packet Capture Pilote de libpcap.

❖ Libpcap pour Win 32 utilise, pour interagir avec le hardware, l'interface fournie par PACKET.DLL. Cette interface est plus complexe à celle offerte par le BPF sous le système UNIX, lorsque la carte réseau est vue comme un fichier standard.

6- <u>les principales routines de la librairie Libpcap</u> :

La librairie de capture de paquet fournit une interface de haut niveau pour les systèmes de capture de paquets. Tous les l paquets sur le réseau, même ceux qui sont destinés à d'autre hôte sont accessibles à travers ce mécanisme. Les principales routines de cette librairie sont :

.P_open_live : Cette fonction est utilisée pour ouvrir la carte réseau et l'initialiser pour capturer des paquets. Premièrement, cette fonction ouvre la carte avec la fonction Packet_Open_Adapter () de Packet.dll. S'il n'y a pas d'erreur, et si le paramètre d'entrée promise est VRAI, la carte est mise en mode promiscuous avec un appel à la fonction
Packet_Set_Hw_Filter (). En suite le type de réseau est déterminé avec l'appel de la fonction Packet_Get_Net_Type (). Le type du réseau est sauvegardé pour être utilisé plus tard par pcap_datalink () de libpcap. En suite pcap_open_live alloue un buffer pour recevoir les paquets du pilote. Notons que ce buffer une fois allouer par pcap_open_live (), après il est utilisé par tous les appel de pcap_read(). Iles alloue uniquement, le temps d'optimiser les performances de capture. La taille de ce buffer est de 256 KB et est fixée (son changement nécessite la recompilation de libpcap. Dans Windows, la taille de ces deux buffers (utilisateur et noyau) peut être différente, parce que le mécanisme de copie du pilote de, capture de paquet est indépendant de la taille du buffer de l'application. Cette caractéristique est très utile lorsque le buffer du pilote est très grand. A ce niveau, la structure PACKET est allouée et initialisée. Elle sera utilisée pour lire les données du pilote. Dans les versions précédentes du libpcap pour Windows, la structure PACKET était allouée et initialisée pour chaque lecture du pilote. Pour améliorer la vitesse de la capture, de la version de 2.01 il y a uniquement une globale structure PACKET, qui est utilisée pour toutes les lectures de libpcap. Finalement, pcao_open_live () utilise la fonction Packet_Set_Buff () de Packet.dll pour allouer un buffer de paquets de 1MB dans le pilote de la capture. L'utilisateur sera capable de changer la taille de ce buffer avec la fonction pcap_setbuff () de libpcap.

.pcap_read () : Cette fonction lit le groupe de paquets du pilot de capture de paquets. Elle commence au début à exécuter Packet_Receive_Packet () pour remplir le buffer de paquets des données provenant du pilote de la capture. A ce moment Pcap_read boucle à travers les paquets.

.pcap_setbuff () : Cette fonction met le nouveau buffer dans le pilote de capture de paquet en appelant la procédure Packet_Set_Buff () de Packet.dll. Cette fonction n'est pas présente dans la version UNIX de Libpcap.

.pcap_setfilter () : Cette fonction met un nouveau filtre de paquets dans le pilote de capture de paquets. Le filtre dans la structure bpf_program, est passé au pilote appelant la procédure Packet_Set_Buff () de Packet.dll.

.pcap_stats () : Cette fonction obtient les statistiques de capture du pilote de la capture de paquets (Nombre de paquets reçus, nombre de paquets perdus par le pilote) en appelant la procédure Packet_Get_Stats de PACKET.DLL.

.pcap_setmode () : Définie le mode de travaille de la carte. Cette fonction peut être utilisée pour mettre une instance du pilote de la capture en mode statistique et est présenté uniquement dans la version de Win32 de libpcap.

7- <u>Quelques fonctions de la librairie Libpcap</u> :

Nous décrirons quelques fonctions de cette librairie :

❖ **Pcap_t *pcap_open_live (char *device, int snplen, int promisc, int to_ms, char *ebuf) :**

Elle est employée pour obtenir un descripteur de capture de paquet pour scruter les paquets sur le réseau.
<u>Device:</u> Est une chaîne qui spécifie quel device doit être ouvert.
<u>Snaplen :</u> spécifie le nombre maximum d'octets à capturer jusqu'à ce qu'une erreur survienne, que EOF est atteinte ou que le timeout de lecture est atteinte (lorsqu'on effectue des capture directement sur le réseau et qu'un timeout non nul est spécifié)
<u>Callback :</u> spécifier la fonction à appeler avec trois arguments : un pointeur de type u_char passe de pcap_dispatch (), un pointeur vers la structure pcap_pkthdr (), (qui précède l'en-tête réseau actuel et les données), et un pointeur type u_char sur les données du paquet capturé. Le nombre de paquets lus est renvoyé.
Zéro est envoyé lorsque EOF est atteint dans un fichier de sauvegarde. Une valeur de retour_1 indique une erreur auquel cas *pcap_perro* () ou *pcap_geterr* () peuvent etre utilisées pour afficher les messages.

❖ **Int *pcap_loop (pcap_t *p, int cnt, pcap_handler callback, u_char *user)**

Est similaire à pcap_dispatch () excepte que celle-ci continue de lire les paquets jusqu'à ce que cnt paquets soient traités ou qu'une erreur survient. Il ne retourne pas s'il y a un

Timeout en cas de capture directe sur le réseau. Plutôt, en spécifiant un timeout non nul à pcap_open_live () et appelant eu suite pcap_dispatch () permet la réception et le traitement de tous les paquets arrivant lors de la survenue du timeout. Un cnt négatif entraîne pcap_loop () à boucler indéfiniment (ou du moins jusqu'à la survenue d'une erreur).

❖ **Void *pcap_dump (u_char *user,struct pcap_pkthdr *h,u_char *sp)**

Emet un paquet vers le fichier de sauvegarde ouvert avec pcap_dump_open ().
Notons que ces arguments d'appels conviennent à l'utilisation avec pcap_dispatche.

- **Int *pcap_compil (pcap_t *p, struct bpf_program *ft, char *str, int optimize, bpf_u_int32 netmask).**

Cette routine est utilisée pour compiler la chaîne str en programme de filtrage.
Programme est un pointeur vers une structure bpf_program et est rempli par pcap_compile().
Optimize : Détermine si une optimisation du code résultant doit être effectuée
Netmask : spécifie le masque réseau du réseau local.

- **Int *pcap_setfilter (pcap_t *p, struct bpf_program *ft)**

Est utilisée pour spécifier un programme de filtrage.
Fp: est un pointeur vers un tableau de structure bpf_program, résultant habituellement d'un appel à pcap_compile (). <<-1>> est renvoyé s'il y a échec, <<0>> si succès.

- **Int *pcap_next (pcap_t *p, struct pcap_pkthdr *h)**

Renvoi un pointeur u_char sur le prochain paquet.

- **Int *pcap_datalink (pcap_t*p)**

Renvoie la couche réseau par exemple : DLT_EN10MB

- **Int *pcap_snapshot (pcap_t*p)**

Renvoie la longueur de la capture spécifiée lors de l'appel à pcap_live ().

- **Int *pcap_is_swapped (pcap_t *p)**

Renvoie VRAI si le fichier de sauvegarde utilise un ordre de rangement de bits différents de celui du système courant.

- **Int *pcap_minor_version (pcap_t *p)**

Retourne le numéro majeur et mineur de la version de pcap utilisé pour écrire le fichier de sauvegarde.

- **Int *pcap_minor_version (pcap_t *p)**

Retourne le numéro mineur de la version de pcap utilisé pour écrire le fichier de sauvegarde.

- **Int *pcap_stats 'pcap_t *p, struct pcap_stat *ps)**

Renvoie <<0>> et rempllit la structure pcap_stat.
Les valeurs représentant les statistiques de paquet de début de l'exécution jusqu'à l'heur de l'appel.
S'il y a erreur ou si la capture de paquet sous-jacente ne supporte pas les statistiques sur les paquets, <<-1>> est renvoyé et un message d'erreur peut être obtenu avec pcap_perror () ou pcap_geterr ().

- **FILE *pcap_file (pcap_t *p)**

Renvoie le nom du fichier de sauvegarde.

❖ Int *pcap_fileno (pcap_t *p)
Renvoie le numéro du descripteur de fichier de sauvegarde.

❖ Void *pcap_perror (pcap_t *p, char *prefix)
Affiche le texte de la dernière erreur de libpcap sur stderr, préfixe par préfixe.

❖ Char *pcap_geterr (pcap_t *p)
Renvoie un texte d'erreur en rapport à la dernière erreur de la librairie.

❖ FILE *pcap_strerror (int error)
Est fournie au cas où strerror (1) ne serait pas disponible.

❖ FILE *pcap_close (pcap_t *p)
Ferme les fichiers associés avec p et désalloue les ressources

❖ FILE *pcap_dump_close (pcap_dumper_t *p)
Ferme le fichier de sauvegarde.

8- Comment compiler une application C++ utilisant libpcap :

Les points suivants doivent être suivis pour compiler une application qui utilise la librairie libpcap :

- Inclure le fichier pcap_ch au début de chaque fichier source qui utilise les fonctions exposées par librairie. Ce fichier peut être trouvé dans le répertoire libpcap\Win32- Include du niveau trois de notre architecture de capture.
- Mettre les options de l'editeur de lien pour inclure le fichier librairie libpcap.lib. Libpcap.lib est générée en compilant le code source de libpcap.
- Mettre les options de l'éditeur de lien pour inclure le fichier librairie winsock (par exemple wsock32.lib). Ce fichier est distribué avec le compilateur et contient des fonctions socket pour windows. Il est nécessaire pour quelques fonctions libpcap.

Ainsi l'application sera capable de reconnaître et d'utiliser les fonctions exportées par libpcap et d'utiliser *NDIS Packet Capture Pilote* pour capturer les paquets. Notons qu'il n'est pas nécessaire d'inclure les fichiers packet32.h et packet.lib pour interagir PACKET.DLL, parce que le code présent dans packet.lib est déjà présent dans libpcap.lib.

Les Ponts :

Les ponts se présentent généralement sous forme de boîtiers empilables disposant d'un nombre réduit d'interfaces : Ethernet, Token-Ring et longue distance (WAN). Les ponts sont des équipements qui travaillent sur les trames MAC indépendamment des protocoles de niveau supérieur qui sont véhiculés. Cette approche a deux conséquences immédiates :

- Les trames 802.2 et 802.5 étant différentes, il est impossible d'interconnecter un segment Ethernet avec un segment Token-Ring à l'aide d'un pont ;
- La technologie est transparente pour l'adressage des protocoles de niveau supérieur, c'est-à-dire que des segments interconnectés par un pont sont dans le même espace d'adressage TCP/IP, IPX....

Les ponts possèdent au moins deux interfaces et agissent au niveau de la couche liaison de données. Leur rôle est de capturer les trames qui arrivent sur l'une de leurs interfaces puis d'analyser leurs adresses MAC pour éventuellement les retransmettre sur une interface menant à leur destination. Les ponts permettent d'éviter l'encombrement du réseau par filtrage d'adresses MAC.

Les Routeurs :

Les routeurs, comme les ponts disposent d'au moins de deux interfaces de communication. Ils agissent quant à eux au niveau de la couche réseau. Leur rôle est d'acheminer les paquets entrants vers leur destination en empruntant le meilleur chemin possible.

Les Commutateurs :

Les commutateurs permettent aux stations de disposer de la totalité de la bande passante à un instant donné. Ceci permet d'augmenter les performances du réseau.

Les Passerelles :

Les passerelles agissent au plus haut niveau de l'architecture OSI. Ils permettent d'interconnecter deux ou plusieurs réseaux qui ont des systèmes complètement hétérogènes.

Les Hubs :

Utilisés principalement dans les réseaux 10 Bit/s. Ils permettent de concentrer plusieurs lignes (ou segments) sur un même bus logique pour un meilleur partage de la bande passante.

Les Répéteurs :

Ils agissent au niveau de la couche physique. Leur rôle est d'amplifier le signal capté sur l'une de leurs interfaces et de le retransmettre vers l'autre.

L'interface entre la carte réseau et l'ordinateur (La couche logicielle entre ces deux Matériels) est réalisée par un pilote de périphérique (Driver, en anglais). Ce logiciel traite les interruptions et envoie des commandes à la carte via des registres spéciaux dits d' "Entrée/Sotie".
Etant donné qu'il existe un pilote pour chaque type de carte, il est nécessaire d'uniformiser son accès par les couches réseaux supérieures. Dans le mode des PC, ces interfaces sont NDIS (**N**etwork **D**river **I**nterface **S**pécification) spécifie par Nouvelle et Packet Driver.

Ces trois standards sont bien sûr incompatibles. La très grande majorité des cartes sont livrées avec ces trois pilotes. Hors du monde des PC, il existe un pilote type de machine : tout d'abord il existe des Unix différents et ensuite des implémentations différentes (CSO sous Pc, HP-UX pour les machines Hewlett-Packard, Solaris pour les machines Sun,etc...).
Une fois l'interface d'accès au pilote standardisée, n'importe quel logiciel réseau peut venir s'ajouter par-dessus. Le principe consiste à ouvrir un lien logiciel au moyen d'interruptions logicielles, de mémoires partagées, de descripteurs de tampons, etc....

L'opération s'appelle bind (liaison) et le lien un SAP (**S**ervice **A**ccess **P**oint). La notion de SAP telle que décrite par le modèle OSI trouve donc son application directe dans les implémentations logicielles. Les logiciels des couches supérieures sont donc LLC (option sur Ethernet), TCP/IP, IPX, LAT,etc...

Au-dessus, les applications y accèdent directement via des SAP : il s'agit de base de données, de logiciels de transfert de fichiers, de partage de disques, de logiciels de test de réseaux.
Par exemple : Dans le mode IP, cela peut être un programme Telnet ou FTP.
Dans la technologie des interfaces réseaux IP utilise NDIS (Network Driver Interface Spécification) pour soumettre les trames à la couche interface réseau.
1- IP par rapport aux technologies de réseau local TCP/IP prend en charge Ethernet, Token Ring, ArcNet, MAN et FDDI
2- IP par rapport aux technologies de réseau étendu.

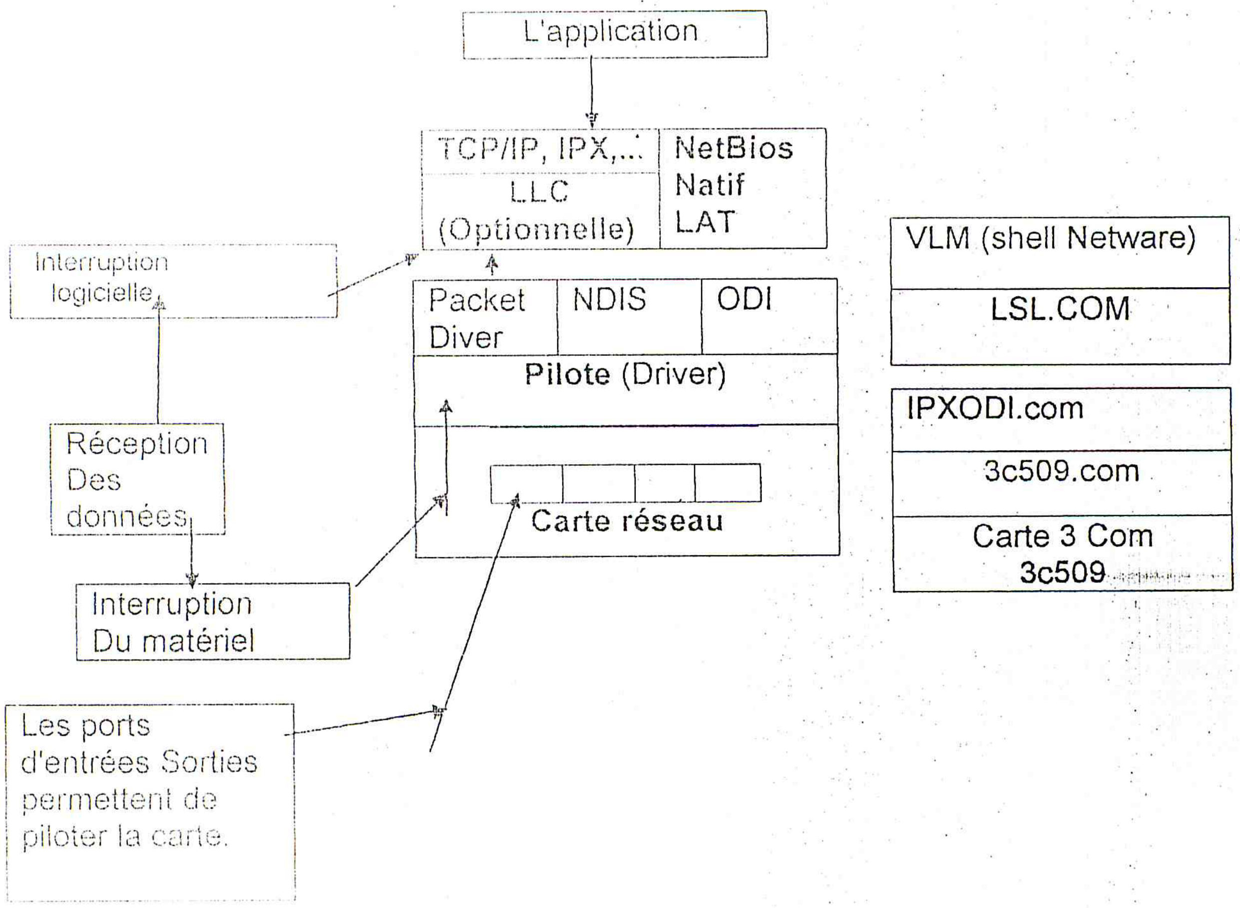


Figure D.1 : Exemple des drivers réseaux.

10BASE-2 : Réseau ethernet utilisant un câble coaxial. Egalement appelé thinnet ou cheapernet. Il admet des segments de réseau de plus de 185 mètres. C'est une topologie en bus qui ne supporte pas la moindre interruption au média.

10BASE-FX : pour le câblage en fibre optique. Le comité IEEE a publié une spécification concernant les réseaux Ethernet câblés en fibre optique 10 Base FX. La principale raison d'utiliser distance maximale d'un segment 10 Base FX est de 2000 mètres.

802 (Comité IEEE 802) : spécifie les efforts pour la normalisation des réseaux locaux qui ont commencé en 1979 sous le patronage de l'IEEE. Le but de la normalisation était de reprendre les couches 1 et 2 du modèle OSI pour les adapter aux particularités des réseaux locaux. En février 1980 le groupe de travail à pris le nom de groupe 802 (80 pour l'année et 2 pour le mois de février).

802.1 : (High Level Interface) traite des architectures des (802.1a), des ponts et du spannig tree (802.1d) et du système Load Protocol (802.c). La 802.1 désigne aussi l'architecture générale du réseau : Format des adresses, technique d'interconnexion des réseaux,...etc

802.12 (100bVG-Any-LAN) : Spécification des réseaux locaux à 100Mbps.

802.2 (LLC) : Spécification de la sous- couche LLC du niveau 2 du modèle OSI.

802.3 (Ethernet CSMA/CD) : Spécification de réseaux Ethernet.

802.5 : Spécification des réseaux Token-Ring.

AUI (Attachment Unit Iterface) : Type de connecteur à 15 broches utilisé pour les matériels Ethernet.

ASCII (American Standard Code for Information Interchange) : Code standard de représentation de données, introduit en 1963 et largement utilisé sur de nombreuses machines. C'est un code sur 7 ou 8 bits, fournissant ainsi 128 profils binaires différents utilisée pour échanger les informations dans les protocoles réseau et dans les bases de données. Il convertit des 1 et des 0 en lettres de l'alphabet, en chiffres ou autres caractères compréhensibles par les humains.

ARP (Adress Resolution Protocol) : ARP permet aux machines de résoudre les adresses sans l'utilisation de table statique. Une machine utilise Arp pour déterminer l'adresse physique destinataire en diffusant (broadcast), sur le sous réseau, une requête ARP qui contient l'adresse IP à traduire. La machine possédant l'adresse IP concernée répond en renvoyant son adresse physique.

Bit (Binary Digit) représente le plus petit élément d'information (état logique), il prend des valeurs binaires <<0>> et <<1>>.

Bps (Bit Par Seconde) : Nombre de bits transmis par seconde.

Broadcast : envoi de données à tous les hôtes d'un réseau.

Carte réseau (Adaptateur) : Un périphérique matériel qui connecte physiquement un ordinateur au réseau.

Collision : Sous CSMA/CD, une collision a lieu quand deux ordinateurs essaient de transmettre simultanément sur le même segment de réseau.

Commutation : Technique dans laquelle chaque connexion entre deux ordinateurs dispose d'un canal dédié à un instant donné.

Commutateur Ethernet : Une solution au problème de congestion en divisant les réseaux en segments, fournissant à chacun une partie de la bande passante.
Le commutateur transfert le trafic entre les segments réseau pour permettre à une station d'un segment, de s'adresser à une station d'un autre segment.

CRC (Cyclic Redundancy Check) : Mécanisme de contrôle d'erreur basé sur le calcul d'un polynôme générateur.

CSMA/CD (Carrier Sense Multiple Access/ Collision Detect) : Une méthode d'accès au support physique propre au réseau Ethernet. Ce protocole définit des règles qui harmonisent la communication entre les stations. La norme IEEE 802.3 définit donc ce protocole qui concerne les couches physiques et liaison de données sous la normalisation de l'ISO. CSMA/CD signifie donc la détection de la porteuse à accès multiples et détection de collision.

Datagramme ou paquet : Paquet de données formant un tout, qui comporte assez d'informations pour être acheminé de la source au destinataire et (éventuellement) des données à transmettre sur le réseau; indépendamment de tout échange antérieur ou ultérieur.

DLL (Dynamic Linked Librairy) est simplement une bibliothèque qui contient des fonctions. Rien ne nous empêche de créer nos propres DLLs. Ces DLLs peuvent être appelées par n'importe quel programme et ceci, indépendamment du langage de programmation.

Ethernet : A l'origine, réseau local (Local Area Network) expérimental de 3 mégabits par seconde sur câble coaxial avec une transmission en bande de base et un protocole de liaison (CSMA/CD). Il a été développé en 1976 par Xerox PARC pour relier des ordinateurs personnels. Il a été adopté en 1980 par DEC, Intel et Xerox comme support de communication standard à10Mbps sur câble coaxial avec une transmission en bande de base et un protocole CSMA/CD. Il réalise effectivement une implémentation de deux couches inférieures du modèle de référence à 7 couches OSI.

Encapsulation : processus de conditionnement des données consistant à ajouter une en-tête de protocole déterminé avant que les données ne soient transmises à la couche inférieure.

FDDI (Fiber Distributed Data Interface) : Réseau local à haut débit (100 Mbps) en anneau à jeton spécifié par l'ANSI.FDDI-II est une version améliorée offrant le support de la voix et de la vidéo. FDDI semble aujourd'hui, la principale solution adaptée à la demande d'interconnexion de réseaux locaux. Les réseaux FDDI utilisent comme support physique la fibre optique (lasers), et ils reposent sur une topologie en double anneau.

FCS (Frame Check Sequence) : est un champ de 4 octets placé en fin de trame Ethernet (Ethernet V2 et IEEE 802.3), il sert à déterminer si la trame reçus n'a pas été déformée (ou altérée) en cours de transmission.

Hub : est un petit boîtier allongé avec plein de prises <<RJ 45>> côte à côte. Chaque ordinateur est connecté à une des prises. Les hubs sont souvent utilisés quand il s'agit de relier quelques ordinateurs pour un petit réseau local. Ainsi quand un ordinateur envois un message, tout le mode l'entend et l'ordinateur concerné trie l'information.

ICMP (Internet Control Message Protocol) : est un protocole qui permet l'envois de messages ICMP dans divers situations. Par exemple, lorsqu'un datagramme ne peut pas atteindre sa destination, les messages ICMP sont émis en utilisant l'en-tête IP de base.

IEEE (Institute of Electricals and Electronics Engineers) : participe à l'ANSI, il est à l'origine des normes des réseaux locaux.

IP (Internet Protocol) : Le rôle du protocole Internet est d'acheminer les datagrammes à travers un ensemble de réseaux interconnectés. Ceci est réalisé en transférant les datagrammes d'un module Internet à l'autre jusqu'à atteindre la destination.

IPX (Internet Packet Exchange) : un protocole utilisé par NetWare IPX/SPX de Novell pour assurer les fonctions d'adressage, routage et aiguillage des paquets.

ISO : International Standard Organisation (chapeaute les organisations nationales).

Ko (Kilo-octets) : Equivalant à 1024 Octets.

LLC (Logical Link Control) : Couche logicielle qui a pour objet d'assurer le transport des trames entres deux stations. Elle se situe au niveau 2 du modèle OSI.

MAC (Medium Access Control) : Couche logicielle qui a pour rôle de structurer les bits d'information en trames adaptées au support physiques des cartes réseaux.

Mo (Méga-octets) : Equivalant à 1 048 576 octets (1024 puissance 2)

Message : Unité d'information qui est transférée par un système de communication de messages (Message switching). Les messages peuvent être de langueurs quelconques, de quelques binaires à un fichier complet, aucune partie du message n'est délivrée au destinataire avant que tout le message ne soit reçu par le nœud réseau qui est adjacent à la destination.

MTU (Maximum Transmission Unit) : Longueur (ou taille) maximale d'une trame ou d'un paquet du protocole réseau.

Monitor : Un périphérique qui reçoit continuellement des données d'un réseau et qui a pour but d'écouter le réseau et de chasser les problèmes.

NDIS (Network Driver Interface Specification) : Spécification faite par Microsoft d'une interface logicielle universelle d'accès aux cartes réseaux (NIC) d'un ordinateur. Cette interface permet au logiciel situé au niveau de la couche 2 du modèle OSI d'utiliser n'importe quelle carte réseau.

NIC (Network Interface Card) : désigne une carte réseau d'un ordinateur.

Nœud : Micro-ordinateur ou un autre système telle qu'une imprimante reliée au réseau.

ODI (Open Data-Link Interface) : Spécification faite par Novell d'une interface logicielle universelle d'accès aux cartes réseaux (NIC) d'un ordinateur. Elle permet au logiciel situé au niveau de la couche 2 du modèle OSI (Liaison de donnée) d'utiliser n'importe quelle carte réseau.

Padding (Remplissage) : utilisé pour étendre une chaîne ou un enregistrement à une longueur donnée.

Pilote (driver ou gestionnaire) : Module logiciel permettant de faire dialoguer un composant électronique et un système d'exploitation. Par exemple, le driver de la carte réseau.

Protocole : Ensemble formel de règles et de conventions qui régit l'échange d'informations entre des unités en réseau.

SAP : Point d'accès au service ; champ de la spécification d'une adresse définie par la nome IEEE 802.2.

UDP (User Datagram Protocol) : Partie de TCP/IP qui prend en charge la délivrance de paquets sans garantie de fiabilité. En d'autres termes, UDP gère la délivrance de paquets sur des liens qui ne sont pas toujours disponibles. Ce protocole suppose l'utilisation du protocole IP comme support de base à la communication.