Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de Bachelor

Développement d'un analyseur réseau sur iPad

Étudiant	:	Meylan Mathieu
Responsable	:	Robert Stephan
Mandant	:	Aginova
Année académique	:	2012 - 2013
Section	:	Télécommunications : Réseaux et services
Date d'édition	:	1er Août 2013

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Aginova
Agile Innovations

iICT
INSTITUTE FOR INFORMATION AND COMMUNICATION TECHNOLOGIES

Remerciements

Je tiens tout d'abord à remercier Dr Stephan Robert, mon mentor lors de ce projet, qui m'a non seulement trouvé un travail de bachelor dans un domaine qui m'intéresse, mais qui a également réussi à m'envoyer poursuivre mon projet les six dernières semaines dans la Silicon Valley en Californie. Cela a été une expérience enrichissante.

J'aimerais également remercier l'entreprise Aginova et tout particulièrement Karl Baumgartner et Thierry Jayet, sans qui ce projet n'aurait pas pu être possible. Karl et Thierry ont toujours été disponible pour m'aider et ont su me guider dans les moments importants.

Je remercie Dr Jon Pierce qui été le point d'ancrage entre la HEIG-VD et San Jose State University pour son accueil et l'organisation mise en place. Je remercie également le bureau des Relation Internationales de la HEIG-VD qui m'ont aidé dans la procédure d'obtention du visa et financièrement pour le voyage.

1 Cahier des charges

Le but de ce projet est d'implémenter une application similaire à « Wireshark » pour les appareils mobiles fonctionnant avec le système d'exploitation iOS (pour les abréviations, veuillez-vous référez au chapitre « Liste des symboles et abréviations utilisées »), plus précisément pour iPad. L'application doit être capable de lire les fichiers au format libpcap standardisé. De plus, l'application doit être spécialisée dans le traitement des paquets à la norme IEEE 802.11 provenant de réseaux sans-fils.

Le but final de projet ne consiste pas seulement en la lecture de fichier au format libpcap mais également la possibilité de lire un flux de paquets, toujours au même format, à partir d'une interface TCP/IP externe. La conception de l'architecture de l'application doit prendre en compte l'évolution possible décrite ci-dessus même si celle-ci ne sera pas implémentée dans le cadre de ce projet.

Il sera manifestement impossible d'implémenter toutes les fonctionnalités de wireshark lors de la durée de ce projet. Par conséquent, une liste des priorités a été établie. L'application délivrée doit être capable de :

1. Afficher la liste des paquets présents dans le fichier formaté au standard libpcap classée de manière chronologique par rapport à leur capture en affichant les informations importantes comme les adresses IP source et destination, le protocole de couche supérieure et la longueur du paquet.
2. Lire et visualiser en détails un paquet spécifique d'une capture en séparant les couches liaison de données, réseaux, transport et application (selon le modèle OSI). Il doit également être possible d'afficher le contenu des champs de chaque protocole.
3. Afficher le contenu brut du paquet en hexadécimal.
4. Comprendre les paquets provenant des réseaux wifi aux normes standardisées IEEE 802.11 au lieu d'Ethernet.
5. Définir des préférences de déchiffrement des paquets chiffré 802.11 en spécifiant pour un SSID le type d'encryption et la clé dans le but d'afficher le contenu du paquet directement en clair.
6. Comprendre et traiter des filtres basiques sur les protocoles de la norme 802.11, UDP et TCP.
7. Emettre des données de statistiques sur toutes les interfaces spécifiques sans-fil d'une capture. Les statistiques sont le nombre de paquets émis et reçus ainsi que le nombre d'octets qui ont transités d'une interface à l'autre.

2 Table des matières

3 Résumé

Avant de commencer ce travail de bachelor il y a un objectif que je m'étais fixé. Je voulais principalement faire quelque chose de nouveau, me plonger un sujet que je n'avais jamais appris en cours et non pas simplement appliquer ce qu'on a étudié à l'école à plus grande échelle. En plus de cela, je trouve qu'il manque plusieurs domaines techniques à la formation télécommunications de la HEIG-VD. Une de ces domaines est le wifi. Le monde tend vers de plus en plus d'appareil connectés sans fils avec l'explosion des smartphones. Des experts prévoyaient un dépassement du nombre d'ordinateurs par les smartphones en 2015. C'était un peu pessimiste car le dépassement s'est fait déjà en 2011.[1] La conclusion de cela est que les réseaux sans fils sont très importants.

On peut dire que j'ai doublement accompli l'objectif que je m'étais en fixé lors de ce travail de bachelor. D'abord en apprenant une nouvelle technologie, la programmation sous iOS en Objective-C, et puis également on comblant un gap en apprenant énormément sur la norme WLAN IEEE 802.11.

Passons maintenant en revue les différents objectifs fixés cette fois-ci par le cahier des charges. Lors de la rédaction de celui-ci, le mandant a tracé un fil conducteur à suivre pendant toute la durée du projet. Les points fixés étaient directement inspirés des fonctionnalités les plus utilisées de l'application Wireshark. J'ai pu implémenter les quatre points principaux de manière complète:

1) Afficher la liste des paquets présents dans un fichier au format libpcap lu en affichant les informations importantes.

2) Lire et visualiser en détails un paquet spécifique d'une capture en séparant les différentes couches selon le modèle OSI. Il doit également être possible d'afficher le contenu des champs de chaque protocole.

3) Afficher le contenu brut du paquet en hexadécimal. De plus l'affichage des paquets en ascii a été ajouté afin d'avoir du texte lisible par l'utilisateur au premier coup d'œil.

4) Comprendre les paquets provenant des réseaux wifi et non Ethernet.

Une fois les quatre principaux points du cahier des charges accomplis et avec ça une base solide pour le développement posée, le mandant et moi avons discuté d'ou mettre la priorité pour la suite du projet. Nous avons décidé de ne pas continuer sur le fil directeur du cahier des charges en ajoutant de nombreuses nouvelles fonctionnalités, mais plutôt d'implémenter plus de protocoles afin de pouvoir lire un

[1] http://gizmodo.com/5882172/the-world-now-buys-more-smartphones-than-computers

fichier de capture en partant de la couche la plus basse et de pouvoir remonter jusqu'aux protocoles de couches les plus hautes.

L'application est capable de comprendre les protocoles Radiotap, LLC (IEEE 802.2), IP, ICMP, TCP, UDP ainsi que la norme wifi IEEE 802.11. Les trois derniers points du cahier des charges n'ont donc pas pu être accompli avec ce changement de priorité.

4 Introduction

Le but de ce travail de bachelor est de développer un analyseur de paquets sur la plateforme iOS pour iPad. Lorsque l'on parle d'analyseurs de paquets, le logiciel gratuit universellement reconnu auquel on pense immédiatement est Wireshark. Wireshark est l'analyseur de protocoles réseau le plus avancé au monde. Il permet de voir ce qui se passe sur un réseau jusqu'à un niveau microscopique et est devenu un standard autant au niveau industriel que pédagogique.[2]

Si l'on se penche sur la plateforme mobile iOS par contre, il n'existe pas un équivalent aussi complet que Wireshark. La seule application qui s'approche un minimum de ce qui est réalisé dans de projet s'appelle *pcap touch*. Pcap touch permet de voir des fichiers de capture de paquets au format libpcap. Les extensions supportées sont .cap et .pcap. Pcap touch supporte également le streaming de paquets en direct depuis un ordinateur linux grâce au serveur Pcap touch.[3] Cette application est tout de même bien différente de ce projet de bachelor puisqu'elle ne supporte que le protocole Ethernet de la couche liaison de données et ne comprend pas la norme IEEE 802.11 WLAN.

Un analyseur de paquets orienté vers les réseaux sans fils est donc un produit innovant sur la plateforme iOS puisqu'il n'existe rien d'exactement similaire. Ce genre de produit pourrait être utile à des administrateurs réseaux afin d'analyser ce qui se passe sur un réseau lors de la recherche d'un problème. Une autre possibilité d'utilisation est d'envergure sécuritaire. Un professionnel effectuant un audit de sécurité sur un réseau pourrait utiliser cette application afin de vérifier ce qui est visible par une personne physiquement présente sur le site dans le but de déceler des failles. Il n'est pas exclut que des personnes moins bien intentionnées ne voie en cette application un outil permettant de trouver des failles dans un tout autre but.

[2] http://www.wireshark.org/about.html
[3] http://pcaptouch.net/

5 Environnement de développement

5.1 iOS et Xcode

Afin de pouvoir développer des applications iOS, il faut utiliser Xcode, l'environnement de développement intégré (IDE) d'Apple, ainsi que le iOS Software Development Kit (SDK). Le développement d'application se fait dans le langage Objective-C ou éventuellement en C/C++.

Apple applique une politique très restrictive sur les développeurs d'application iOS. Il est en effet nécessaire de posséder un MAC tournant avec le système d'exploitation MAC OS. Cela est dû à la disponibilité uniquement sur MAC OS d'Xcode, du SDK et avec lui d'un des seuls compilateurs Objective-C. Je parle d'un des seuls compilateurs Objective-C car il est néanmoins possible de développer du code pour iOS sur un autre système d'exploitation en utilisant soit un compilateur ciblé pour le système en question, ou en faisant de la « cross-compilation » grâce à des outils tels que cocotron (http://www.cocotron.org).

Apple veut que l'on utilise son système d'exploitation, MAC OS, ainsi que son IDE, Xcode, pour développer des applications et c'est ce que j'ai fait pour ce projet. Xcode est le pilier central d'un développeur iOS. Il fournit tout ce dont on a besoin pour créer des applications. Il comprend un éditeur de source, un éditeur graphique d'interface utilisateur de type « drag-and-drop » ainsi que de nombreuses autres fonctionnalités. Xcode est disponible au téléchargement sur l'App Store Mac gratuitement pour tout possesseur d'un mac.

Figure 1 – Comment commencer à développer

https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/GetToolsandInstall.html

Dans cette section, je vais expliquer les différentes vues et fonctionnalités utiles d'Xcode sans rentrer dans un niveau de détail inutile au contexte de ce projet. Xcode comprend un éditeur de code (voir numéro 1 sur la figure ci-dessous) comme fenêtre principale. Cette vue se transforme en éditeur d'interface utilisateur lorsque l'on ouvre un fichier de type XIB ou Storyboard qui sont des constructeurs d'interface.

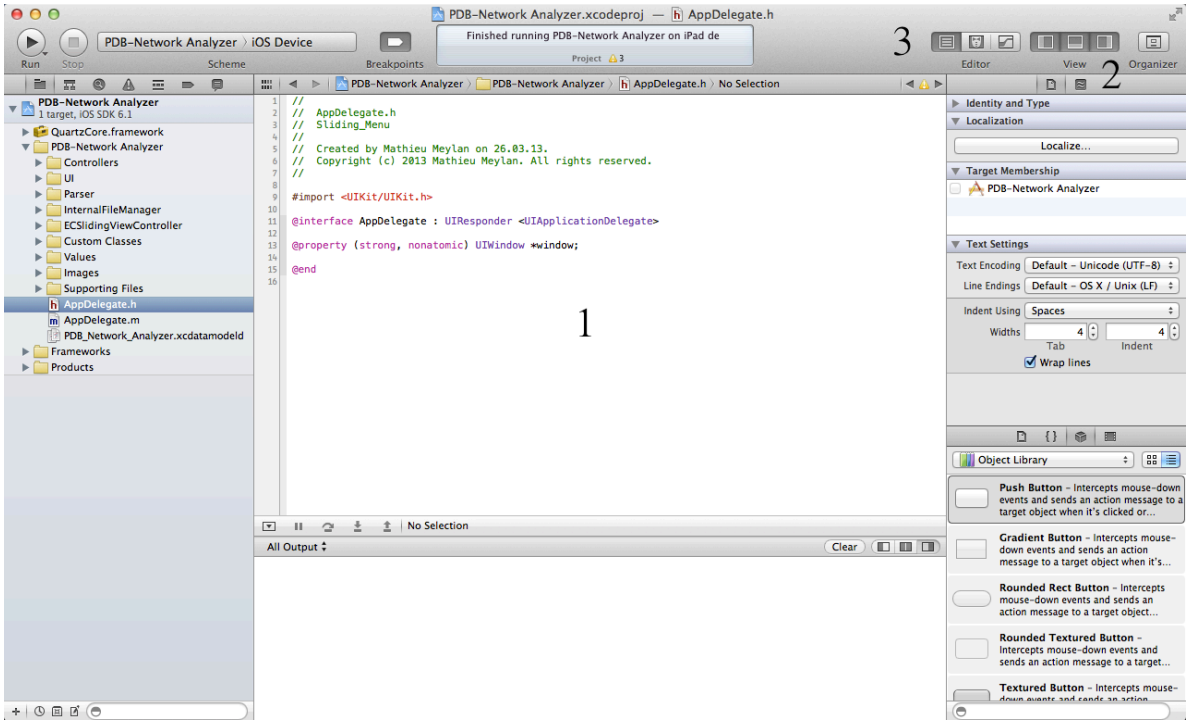

Figure 2 - Xcode

L'éditeur de code peut être utilisé de trois manières différentes grâce au bouton « Editor » (voir numéro 3 sur la capture ci-dessus). La vue standard est une simple vue avec le fichier ouvert prenant tout l'espace écran de l'éditeur. La deuxième possibilité, « Assistant Editor », permet de diviser la fenêtre de l'éditeur en deux afin d'ouvrir deux fichiers.

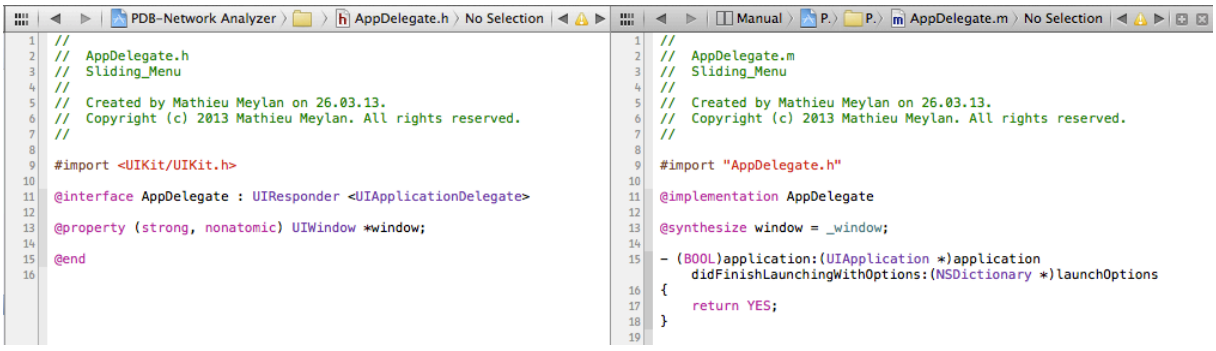

Figure 3 - Editeur de mode assistant

Ce mode d'affichage de l'éditeur est utile lorsque l'on développe une nouvelle classe et que l'on doit écrire les déclarations des fonctions dans le fichier d'en-tête et le corps de ces fonctions dans le fichier m. Cette vue est également très utilisée lorsque l'on relie des éléments disposés sur l'interface graphique au code de l'application, comme des boutons, des labels et autres. La troisième vue est un éditeur de version permettant la gestion des différentes versions des fichiers lorsque l'on utilise un outil de contrôle de source configurable avec Xcode.

Xcode dispose de trois volets rétractables permettant l'interaction avec l'éditeur et sont accessible à travers les boutons *View* (voir numéro 2 sur la figure ci-dessus). Le volet de gauche est certainement le plus utilisé. Il permet d'ouvrir des fichiers dans l'éditeur. Il nous montre l'arborescence des classes et permet également l'accès aux avertissements et aux erreurs générées par le compilateur. La fonctionnalité de débogage est également accessible à travers ce volet.

Le volet du bas est la zone de débogage et va afficher les différentes sorties de l'application lors de l'exécution ainsi que les éventuelles erreurs et exceptions levées.

Le volet de droite est surtout utilisé lors de la conception de l'interface utilisateur car il contient tous les éléments graphiques de la librairie, comme des vues, des boutons, des labels, etc. Ces objets peuvent être directement tirés sur l'éditeur (principe de drag-and-drop).

5.2 Matériel et outils de développement

Lors de la phase de développement, il y a deux choix de cible possibles pour l'exécution de l'application. On peut utiliser le simulateur iPad ou iPhone fourni avec Xcode ou alors développer et directement exécuter sur une cible matérielle. L'utilisation d'un simulateur est la manière la plus simple et gratuite et permettra de tester la plupart des fonctionnalités d'une application.

Pour développer directement sur un iPad ou un iPhone, il est nécessaire de s'inscrire au programme de développeur d'Apple et de payer une licence. La procédure d'acquisition d'une licence de développement Apple est expliquée dans l'annexe « Licence de développement iOS ».

5.3 Librairies

Cette section parle des librairies natives du SDK iOS qui ont été utilisées lors du développement de l'application. Ces librairies sont inclues par défaut dans tout projet d'application iOS sur Xcode et il n'est pas nécessaire de les télécharger ni même de les importer. Toutes les librairies externes utilisées dans ce projet sont décrites dans le chapitre « Conception/Réalisation » de ce rapport.

Les librairies des bases iOS utilisées sont les suivantes :

UIKit/UIKit.h : Le framework UIKit fournit les classes dont on a besoin pour construire et gérer les interfaces utilisateurs sous iOS. Elle permet notamment de la gestion des évènements, le dessin des vues et tous les contrôles nécessaires à une interface destinée à un écran tactile.[4]

[4] http://developer.apple.com/library/ios/#documentation/uikit/reference/UIKit_Framework/_index.html

Foundation/Foundation.h : Le framework « Foundation » fournit la couche de base de toute classe Objective-C. En plus de prévoir tout un set de classes primitives utiles, il introduit plusieurs paradigmes qui ne sont pas couvert par le langage Objective-C. Les objectifs principaux de Foundation sont les suivants:[5]

- Fournir une collection de classes basiques utiles.
- Rendre le développement logiciel plus simple en introduisant des conventions de codage consistantes.
- Prévoir une couche d'indépendance du système d'exploitation dans le but d'améliorer la portabilité.
- Assurer le support des chaînes de caractères Unicode.

CoreGraphics/CoreGraphics.h : CoreGraphics est basé sur une API du langage C utilisant le moteur de dessin avancé Quartz (voir ci-dessous). Ce framework fournit un rendu bas niveau et léger en deux dimensions. Ce framework est utilisé notamment pour la gestion des couleurs, la gestion et la création d'images et le traitement complet de fichier PDF.[6]

CoreData/CoreData.h : Le framework CoreData fournit des solutions généralisées et automatisées pour toutes les tâches communes associées à la gestion du cycle de vie d'un objet.[7]

Ces quatre framework peuvent être utilisées sans aucun ajout particulier au projet. Il est quand même recommandé de les importer quand on en a besoin grâce à la directive `#import <Framework/FrameworkHeader.h>`.

J'ai également besoin d'une autre librairie Apple qui n'est pas incluse par défaut dans le projet. Il s'agit de QuartzCore.

QuartzCore/QuartzCore.h : QuartzCore donne accès à des API de manipulation d'image et de vidéos.

J'utilise QuartzCore et notamment la classe CAAnimation pour manipuler des images, comme par exemple des flèches indiquant qu'un menu est déroulé ou pas. Pour ajouter cette librairie au projet, lors de la première importation de cette classe, il

[5] https://developer.apple.com/library/mac/#documentation/cocoa/reference/foundation/ObjC_classic/_index.html

[6] http://developer.apple.com/library/ios/#documentation/CoreGraphics/Reference/CoreGraphics_Framework/_index.html

[7] http://developer.apple.com/library/mac/#documentation/cocoa/Conceptual/CoreData/cdProgrammingGuide.html

suffit de cliquer sur le projet dans le navigateur et dans le menu *Build Phases* la section *Link Binary with Libraries* permet d'ajouter QuartzCore. [8]

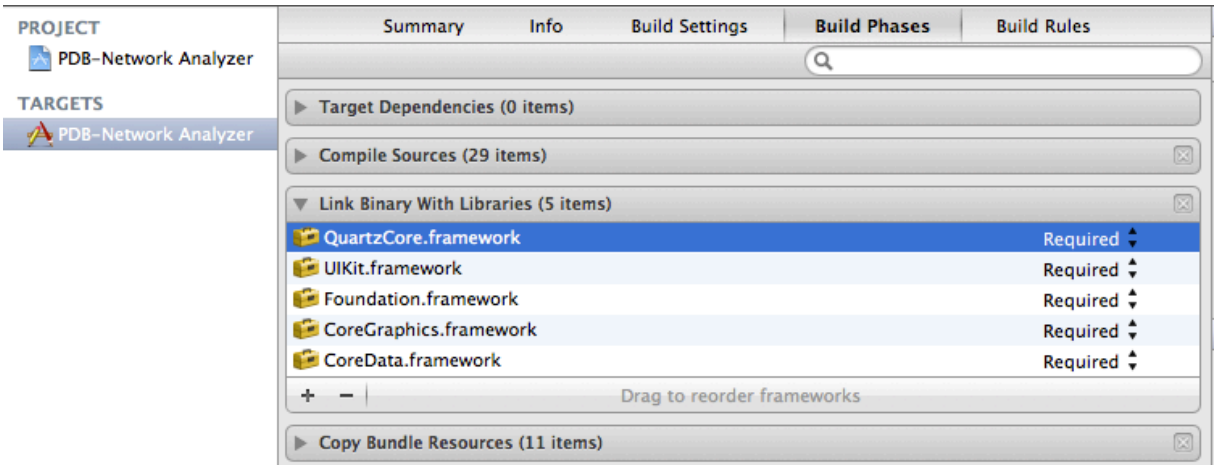

Figure 4 - Ajout de QuartzCore

5.4 Conventions de codage

Lors de ce projet, j'ai suivi les conventions de codage et de nommage recommandées par Apple.[9] Les noms ainsi que les commentaires utilisés dans le code sont en anglais. Cela a été décidé car l'entreprise mandante, Aginova, est une entreprise internationale et les développeurs qui pourraient reprendre le projet parlent tous anglais mais pas forcément français. De plus, l'anglais est en quelque sorte considéré comme un standard lors de développement.

Trouvez-ci dessous un résumé de la manière de nommer les différents éléments du code Objective-C :

- Classe : `@interface MaClasse : SurClasse @interface`
- Propriété : `@property Type* nomPropriete;`
- Méthode : `– (void)maMethodeAvecPremierParametre:(id)param1 etDeuxiemeParametre:(id)param2;`
- Variable d'instance : `Type _maVariableInstance;`

Les en-têtes des protocoles ont été la plupart du temps emprunté à la librairie *libpcap* et les conventions **C** ont été conservées pour ces fichiers-là. Ces fichiers d'en-têtes ne contiennent que des définitions de structures, d'énumérations, de constantes ou de définitions et peuvent être trouvées suivant les conventions suivantes :

[8] https://developer.apple.com/library/mac/#documentation/GraphicsImaging/Reference/QuartzCoreRefCollection/_index.html

[9] Les conventions de codage recommandées par Apple sont spécifiés à l'adresse suivante : https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CodingGuidelines/Articles/NamingBasics.html#//apple_ref/doc/uid/20001281-BBCHBFAH

- Structure :
```
typedef struct ma_structure {
        uint8_t premier_champ;
        uint8_t deuxieme_champ;} mon_type;
```
- Définition :
```
#define VALEUR_ZERO 0
```
- Enumération :
```
enum mon_enumeration {
        VALEUR_ZERO = 0,
        VALEUR_UN = 1};
```

6 Fonctionnalités

L'objectif technique de ce projet de bachelor consiste à développer une application mobile conçue pour la plateforme iPad du système d'exploitation iOS d'Apple.

L'application, une fois lancée, comprend en premier plan deux boutons et une barre d'état. Le tableau vide au centre est le packet viewer et se remplira lors de l'ouverture d'une capture.

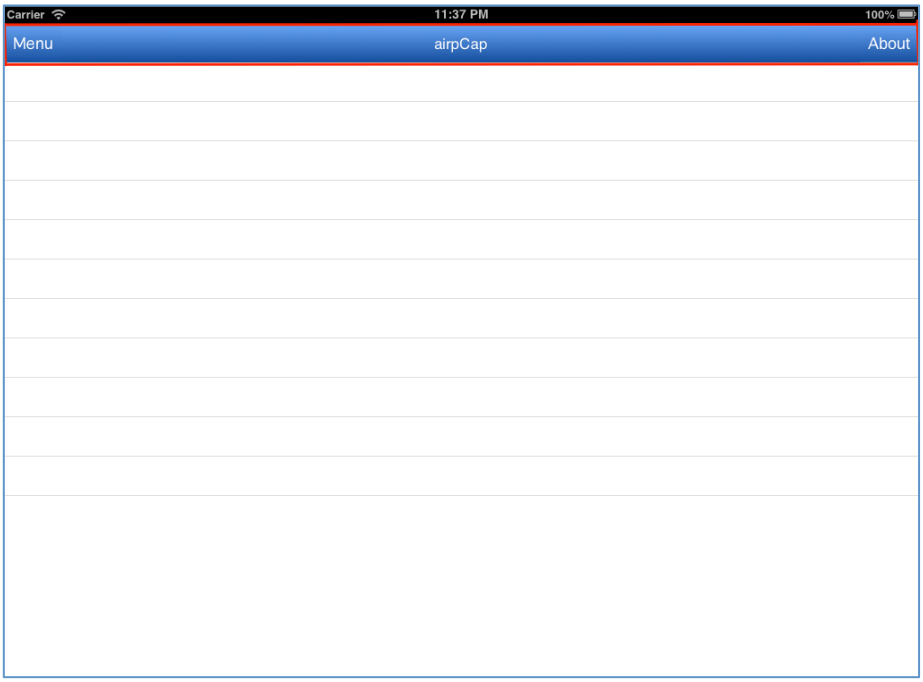

Figure 5 – Application une fois lançée

Le bouton « Menu » permet, comme son nom l'indique, d'ouvrir le menu de l'application qui va autoriser l'utilisateur à gérer ses fichiers. Les fonctionnalités du bouton Menu seront expliquées en détails plus tard dans cette section. L'autre bouton, « About », va afficher dans une bulle une simple phrase d'explication sur le développement de ce projet. En plus de cela, lorsqu'une capture est ouverte, les données de l'en-tête du fichier pcap seront affichées, comme le nom du fichier, le « magic word », la version du format pcap ainsi que le numéro du protocole de la couche liaison de données.

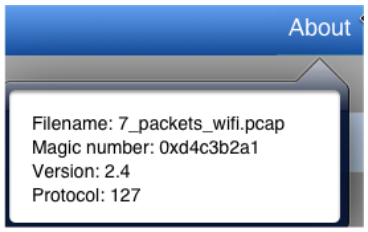

Figure 6 - About

La fonctionnalité principale permet de lire les fichiers à l'extension .pcap formaté au standard libpcap 2.4. Une fois que les fichiers pcap sont ajoutés à l'environnement de l'application à travers iTunes, ils sont référencés dans la deuxième section du menu.

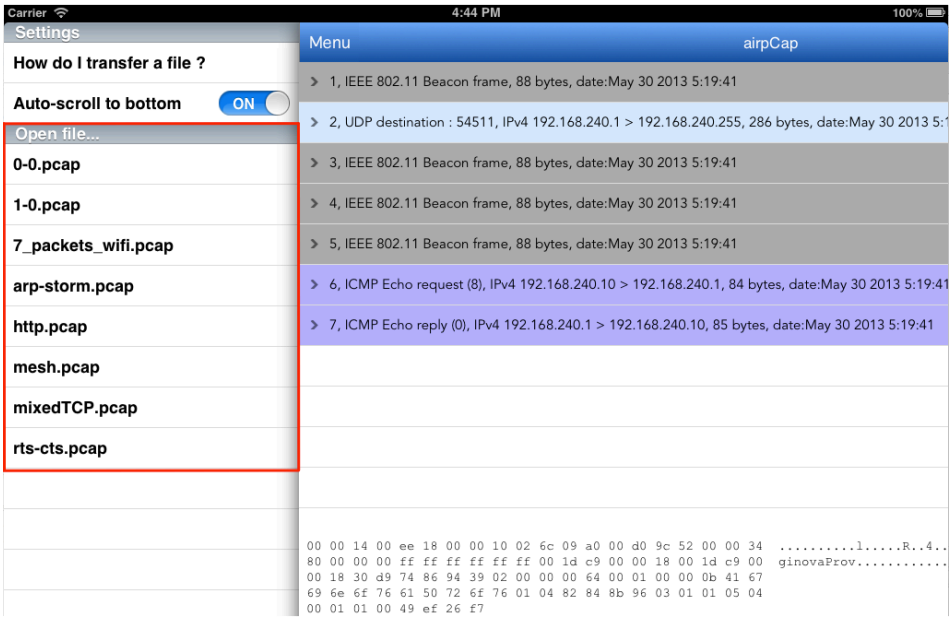

Figure 7 - Ouvrir un fichier

Il est possible de directement gérer la liste de fichiers présents dans le répertoire de l'application. Si l'on ajoute un fichier alors que l'application est toujours en marche, il suffit de tirer le menu vers le bas afin de rafraîchir la liste et de voir notre nouvelle capture s'afficher dans le menu. La suppression de fichier est également possible grâce à un glissement du doigt sur le fichier ciblé.

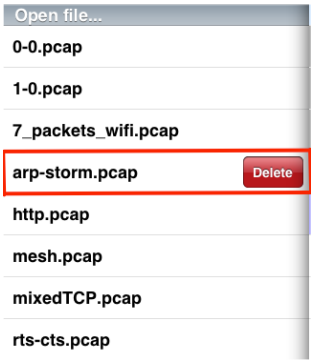

Figure 8 - Supprimer une capture

La première section du menu comporte deux fonctionnalités mineures mais très utiles. La première ligne ouvre une bulle d'aide expliquant comment ajouter un fichier pcap à l'application. La deuxième ligne comporte un interrupteur qui va, si activé, faire défiler les paquets lors de l'ouverture d'un fichier. Cette fonctionnalité est

pratique pour les captures contenant une grande quantité de paquets car elle évitera à l'utilisateur de devoir faire défiler tous les paquets.

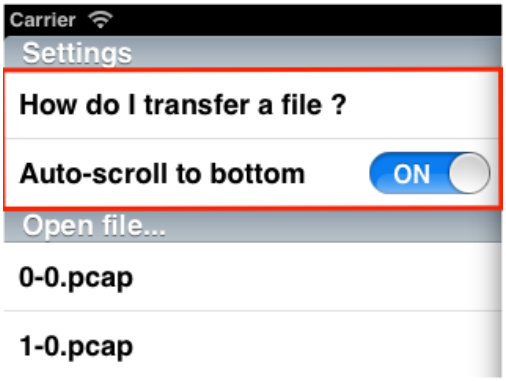

Figure 9 - Aide & Autoscroll

Lors de la sélection d'un fichier dans le menu, le parseur va afficher la liste des paquets lue à partir d'un fichier au format libpcap en affichant les informations importantes. Chaque paquet représente une cellule dans le « packet viewer » (vue de l'application affichant le contenu d'un fichier pcap). Un simple clic permet de développer ces cellules afin de voir le contenu du paquet en question. Une fois étendu, le contenu du paquet est organisé en couches selon le modèle OSI. Chaque couche peut elle même également être étendue afin de voir plus en détails son contenu. Ce comportement se répète jusqu'à qu'on arrive à un simple champ. Chaque ligne pouvant être étendue comporte un indicateur de développement qui sera tourné une fois la ligne développée.

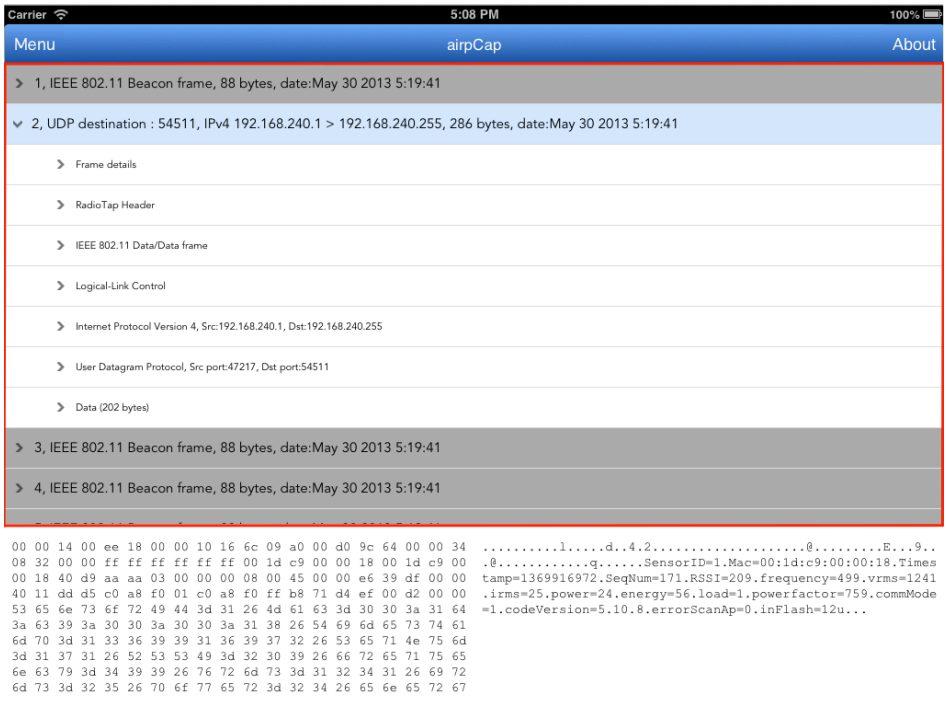

Figure 10 - Packet Viewer

Pour avoir un aperçu d'ensemble simple et efficace immédiatement lors de l'ouverture d'une capture, la ligne descriptive des protocoles les plus communs et importants possède une couleur significative propre à son protocole. On voit par exemple que le paquet UDP est affiché en bleu et un paquet ICMP, par exemple, serait affiché en violet.

En plus de cela, afin que l'expérience utilisateur soit la plus aisée et agréable possible, chaque ligne descriptive va résumer les champs importants de chaque paquet. Pour prendre un exemple, afin de connaître l'adresse source et destination d'un paquet IP, l'utilisateur n'aura pas besoin de dérouler le paquet jusqu'aux champs concernés, mais il obtiendra l'information du premier coup d'œil sur la ligne descriptive.

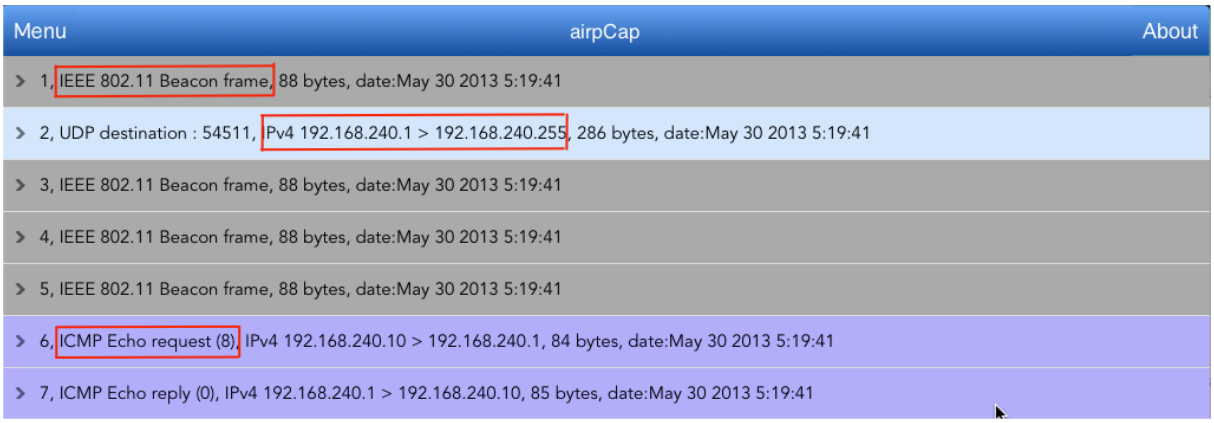

Figure 11 - Informations importantes

Lorsque l'utilisateur clique sur un paquet dans la liste du packet viewer, les données brutes sont automatiquement montrées en dessous du packet viewer. La représentation hexadécimale et *ascii simplifié* des octets sont affichées.

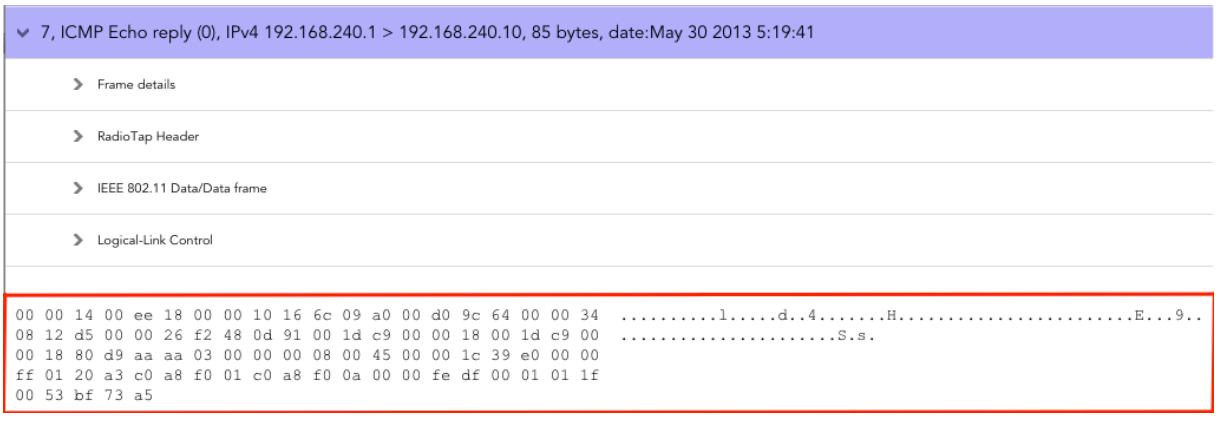

Figure 12 - Paquet brut

Il est important de noter que seuls les octets réels du paquet qui ont transité sur le réseau sont montrés. Tout l'overhead ajouté par le format libpcap au début du

paquet et au début du fichier est supprimé. Lorsque l'utilisateur sélectionne un protocole, un champ ou un groupe de champs dans le packet viewer, les octets concernés sont directement mis en évidence dans la représentation hexadécimale et ascii du paquet.

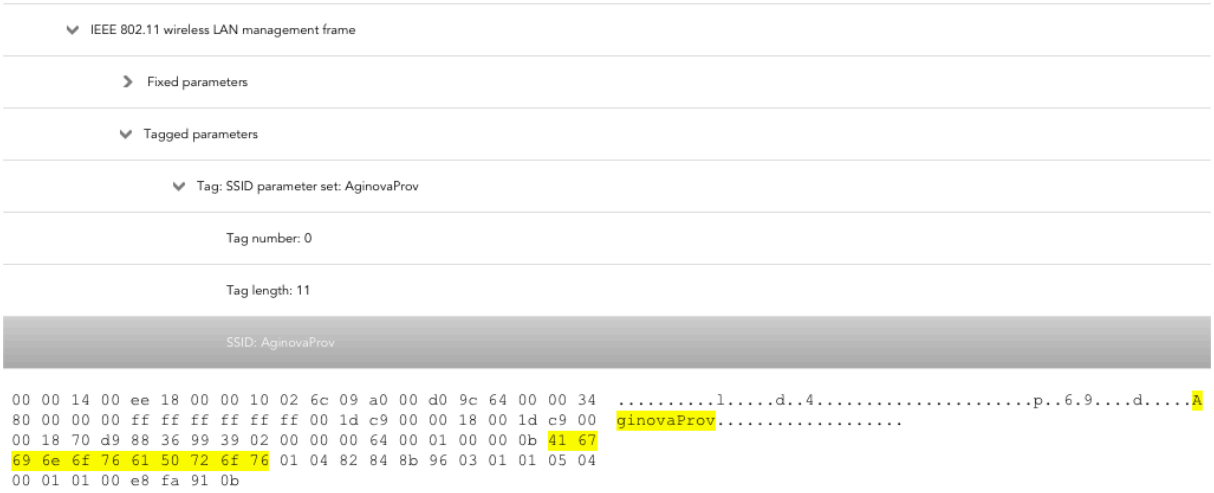

Figure 13 - Mise en évidence d'octets

Avoir la représentation en hexadécimale est standard, mais la représentation ascii simplifiée est, à mon avis, extrêmement utile et permet à une personne de voir directement des données qui ont du sens, des mots lisibles. Je parle d'ascii simplifié car seuls les caractères numéro 48 à 126 de la table ascii sont affichés. Tous les autres sont remplacés par des points. On ne sélectionne que ces caractères là car on veut que les données en représentation ascii soient lisibles sans avoir besoin de réfléchir.

Le but de ce projet était de construire une application comprenant prioritairement les paquets venant de la couche liaison de données à la norme IEEE 802.11. Les protocoles et normes supportés sont les suivants :

- Radiotap
- IEEE 802.11 Data, Management & Control (wifi)
- IEEE 802.11 Management Data Tags
- Logical Link Control (LLC)
- Internet Protocol version 4 (IPv4)
- Internet Control Message Protocol (ICMP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

7 Analyse

Ce chapitre concerne toute l'analyse préliminaire au codage sur le format des fichiers pcap, le format des protocoles ainsi que la manière dont chaque champ doit être interprété. Chaque protocole expliqué dans ce chapitre a été implémenté dans l'application. Pour ce qui est de la description sur le parsing en lui-même des captures pcap ainsi que de chaque couche des paquets, elle est expliquée en détails dans le chapitre « Conception/Réalisation ».

Le but de ce projet n'étant pas d'implémenter un analyseur de paquets connaissant tous les protocoles existant, seuls les protocoles définis comme prioritaires par le mandant, Aginova, sont implémentés. Cette application a comme focalisation les réseaux sans fils 802.11 et non pas les liens câblés communiquant avec Ethernet.

Tout au long de ce chapitre, des protocoles et des en-têtes seront décrits. Les différents « champs » (terme utilisé dans ce rapport pour représenter un certain nombre de bits ayant une signification précise décrite dans la documentation du protocole) seront décrits dans l'ordre dans lequel ils apparaissent dans la trame. Pour UDP par exemple, les champs seraient décrits dans l'ordre *Port Source, Port Destination, Longueur, Checksum* et c'est l'alignement dans lequel ils sont physiquement transmis sur une ligne de transmission.

7.1 Format Libpcap

Le format libpcap est le format de capture principal utilisé par tcpdump/windump, Wireshark et bien d'autres outils de réseaux. Ce format est disponible sur la plupart des plateformes UNIX. Un format appelé WinPcap est disponible pour Windows.

Ce format de fichier très basique permet de sauvegarder les données capturées sur un réseau. La librairie libpcap est un standard de la capture de données sur un réseau sous UNIX et il est devenu une sorte de « dénominateur commun » pour les fichiers de capture réseaux dans le monde open source. Libpcap et la version de Windows, WinPcap, utilisent le même format.

Libpcap n'est pas utilisable que pour Ethernet, comme on le pense souvent, mais il est adapté à de nombreux types de réseaux. Tous les types de réseaux supportés par libpcap sont référencés sur le site officiel de tcpdump:

http://www.tcpdump.org/linktypes.html

L'extension des fichiers libpcap proposée par wireshark est : « .pcap ». C'est pour cela que dans ce rapport je parle parfois de *fichier pcap*, en rapport avec l'extension

proposée. Il faut bien se rendre compte que les fichiers pcap mentionnés sont simplement des fichiers au format libpcap avec une extension en « .pcap ». [10]

7.1.1 Format des fichiers

Il existe plusieurs versions du format libpcap, dont certaines variantes que l'on pourrait qualifier de « sauvages ». Ces versions sont des versions adaptées par certaines personnes pour leurs protocoles. J'ai utilisé et je vais décrire ici uniquement la version utilisée le plus communément, la version **2.4**. C'est la dernière version et le format n'a pas changé depuis au moins la version libpcap 0.4 de 1998. Aucun changement n'est prévu jusqu'à la version PCAPng (PCAP next generation) dont on en parlera dans la section PCAPng de ce chapitre.

Figure 14 - Format d'un fichier libpcap

http://wiki.wireshark.org/Development/LibpcapFileFormat

Chaque fichier au format libpcap comprend un en-tête global contenant des informations globales et est suivi par zéro ou plus enregistrements de paquets, comme présenté dans la figure ci-dessus.

Figure 15 - En-tête globale

http://wiki.wireshark.org/Development/LibpcapFileFormat

L'en-tête global est composé de la manière suivante (dans le même ordre que présenté ci-dessous) :

Global Header	
Champ	**Taille**
Magic Number	**4 octets**

Le « numéro magique » est utilisé pour déterminer l'ordre des octets. L'application écrivant le fichier au format libpcap insère 0xa1b2c3d4 avec son ordonnancement d'octet natif. L'application lisant la valeur va ainsi déterminer si son ordonnancement d'octets est identique, en lisant 0xa1b2c3d4, ou si c'est inversé : 0xd4c3b2a1. Si c'est inversé, l'application lisant saura qu'elle devra inversé tous les autres champs également afin de retrouver l'ordre initial.

Ce champ est extrêmement important car il va déterminer l'ordre dans lequel on va lire le reste du fichier. L'application doit en tenir compte et garder en mémoire cette valeur.

[10] Informations sur le format pcap : http://wiki.wireshark.org/Development/LibpcapFileFormat

Version Major **2 octets**

Le numéro de version « majeur » du format du fichier. Si le format est à la version actuelle de 2.4, alors l'application lira « 2 ».

Version Minor **2 octets**

Le numéro de version « mineur » du format du fichier. Si le format est à la version actuelle de 2.4, alors l'application lire « 4 ». La combinaison de version major et version minor nous donne donc la version utilisée par le fichier.

This zone **4 octets**

Le temps en secondes corrigé entre GMT (UTC) et la timezone locale aux timestamp de l'en-tête des paquets (voir description de l'en-tête des paquets). Par exemple si les timestamp des paquets sont enregistrés au format Central European Time (Amsterdam, Berne, …) soit GMT + 1:00 alors Thiszone vaut -3600. En pratique, toutes les captures sont faits en GMT ce qui implique que Thiszone vaut toujours 0.

Sigfigs **4 octets**

En théorie, ces bits valent la précision des timestamp de la capture. En pratique, tous les utilitaires réseaux mettent 0.

Snaplen **4 octets**

Ces 4 octets représentent la longueur du « Snapshot » pour la capture. En général, 65535, mais l'utilisateur peut limiter cette longueur. Pour plus d'informations, voir les longueurs de paquets dans la description de l'en-tête des paquets.

Network **4 octets**

Ce champ est très important car il nous informe du type de la couche liaison de données sur laquelle les paquets ont été capturés. Par exemple, si notre capture a sniffé des paquets sur un lien Ethernet, Network vaut 1.

La liste à l'adresse suivante référence tous les types d'en-têtes de la couche liaison de données : *http://www.tcpdump.org/linktypes.html*

Une fois l'en-tête global lu, on passe aux enregistrements des paquets. Chaque paquet lu sur le réseau se voit ajouté un en-tête propre au format libpcap. Il est important de noter que cet en-tête ajouté avant chaque paquet n'a rien à voir avec le paquet en lui-même. C'est un overhead ajouté par libpcap. Cet en-tête n'a jamais été transmis sur le canal de transmission.

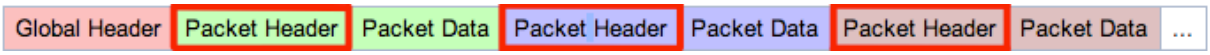

Figure 16 - En-têtes des paquets

http://wiki.wireshark.org/Development/LibpcapFileFormat

Les en-têtes des paquets sont composés de la manière suivante :

Packet Header	
Champ	**Taille**
Timestamp seconds	**4 octets**

Ces 4 octets représentent le timestamp auquel le paquet a été capturé. Un timestamp est une valeur en seconde depuis le 1er janvier 1970 à 00:00:00 GMT. Ceci est également appelé UNIX Time. Si timestamp n'est pas basé sur GMT, le champ Thiszone de l'en-tête globale permet de faire les ajustements.

Timestamp microseconds	**2 octets**

Offset en micro secondes par rapport aux premiers 4 octets lu (donc sur le timestamp). Si cette valeur atteint les 1 000 000, donc une seconde, « Timestamp seconds » doit être incrémenté à la place !

Incl length	**2 octets**

Nombre d'octets du paquet capturé qui sont stockés dans le fichier. Il peut arriver que ce champ soit plus court que « Orig length » lorsque le programme ayant écrit le fichier n'a pas stocké la totalité du paquet dans le fichier pcap. Ce champ ne doit évidemment pas dépasser « Orig length » ou « Snapshot » !

Orig length	**4 octets**

Taille du paquet en octets tel qu'il a été capturé sur le réseau. Si ce « Orig length » est plus grand que « Incl Length » cela veut dire que « Incl length » a été limité par « Snapshot » de l'en-tête globale et que le paquet n'est pas présent en totalité dans le fichier en cours de lecture.

Le premier en-tête du protocole de liaison de données ainsi que les données en un bloc de taille « Incl length » suivent l'en-tête ajoutée par libpcap. Le format des données dépend du protocole spécifié dans le champ « Network » de l'en-tête global du fichier et le traitement à appliquer peut être déterminé grâce à celui-ci.

7.1.2 Désavantages du format libpcap

On l'a mentionné plusieurs fois, le format libpcap est en soi très simple. C'est une raison pour laquelle il est devenu si populaire. Malheureusement, il manque plusieurs éléments qui pourraient s'avérer très utile :

- Un compteur du nombre de paquets dans la capture. Cela serait très utile pour pouvoir simplifier et paralléliser le parsing et le traitement des fichiers.
- Des commentaires utilisateurs, pour donner des informations sur les points importants d'une capture comme une coupure de connexion, par exemple.
- Une précision de capture à la nanoseconde.
- Des informations sur les interfaces, comme le fabricant du matériel, par exemple.

Les éléments ci-dessus sont simplement des exemples de différents ajouts qui seraient utile aux développeurs traitant le format libpcap. Un nouveau format est en développement et pourrait combler ces lacunes : le format « libpcap Next Generation ».

7.2 PCAP Next Generation

Un « draft » du format pcapng (Pcap Next Generation) a été rédigé et publié sur le site *http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html*. Cette ébauche se conforme à la section 10 de la RFC 2026. Ce nouveau format vise à atteindre les objectifs suivants :

- Extensibilité : Les tiers partis devraient être capables d'enrichir une capture grâce à des extensions d'informations ajoutés qui peuvent être ignorés par des outils qui ne sont pas capable de les comprendre. C'est les commentaires mentionnés à la section précédente.
- Portabilité : Une capture devrait contenir des informations permettant d'être lue indépendamment du réseau, du matériel et du système d'exploitation qui l'a effectuée.
- Il devrait être possible de fusionner ou d'ajouter des données à la fin d'un fichier et que celui-ci soit toujours lisible.

Afin d'atteindre ces objectifs, la structure du format libpcap actuel a été repensée et réorganisée. Le format 2.4 libpcap est celui traité lors de ce projet car il est stable et que pcapng n'est pour l'instant qu'une ébauche. [11]

[11] Le nouveau format pcap next gen est présenté sur le document suivant : *http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html.*

7.3 Radiotap

Radiotap est un standard « de facto » (de facto = de fait, dans la pratique) pour l'injection et la réception de trames IEEE 802.11. IEEE 802.11 est un ensemble de normes mis au point pour les réseaux sans fil. Les normes concernant 802.11 seront expliquées plus en détails dans la section suivante.

Les en-têtes radiotap ont été ajoutés afin de fournir des informations supplémentaires concernant les trames. Conçue à la base pour les systèmes NetBSD, le format des en-têtes radiotap apporte plus de flexibilité que les autres formats disponibles et permet au développeur de pilotes de spécifier un nombre arbitraire de champs grâce à la présence d'un bitmask dans l'en-tête. Lorsque je parle de bitmap ou de bitmask, c'est un champ de bit (généralement 32 bits) pour lequel chaque bit a une signification précise définie dans la définition du protocole. Dans le cas de radiotap, chaque bit du bitmap de l'en-tête indique la présence d'un champ spécifique plus tard dans la trame. Radiotap est assez flexible pour permettre l'ajout de nouveaux champs sans pour autant rendre inutilisable les parseurs existant et c'est son atout principal.[12]

Le format de capture radiotap commence par un en-tête radiotap ayant la forme suivante :

```
struct ieee80211_radiotap_header {
        u_int8_t        it_version;     /* set to 0 */
        u_int8_t        it_pad;
        u_int16_t       it_len;         /* entire length */
        u_int32_t       it_present;     /* fields present */
} __attribute__((__packed__));
```

Figure 17 - En-tête

radiotap http://www.radiotap.org/

Radiotap Header	
Champ	**Taille [octets]**
IT Version	**1 octet**

Le premier octet représente la version majeure de l'en-tête radiotap en utilisation. Actuellement, il est toujours égal à zéro. L'ajout de support pour les différents champs de radiotap ne change pas le numéro de version.

IT Pad **1 octet**

Ce champ n'est actuellement pas utilisé.

[12] Le standard Radiotap est expliqué sur la page officielle : http://www.radiotap.org/

IT Len **2 octets**

Ce champ représente la longueur des données radiotap avec l'en-tête comprise. Cette information est utile notamment pour permettre la localisation du début de la trame 802.11.

IT Present **4 octets**

C'est sur ce champ que se base tout le mécanisme de radiotap. Comme cité précédemment, c'est un bitmask spécifiant les champs de données suivant l'en-tête. C'est sur la valeur lue ici que l'on va se baser pour parser les données qui suivent.

Lorsque le bit numéro 31 du champ IT Present est défini pour parle de la présence d'un « extended bitmask » suivant celui en cours de lecture avant les données radiotap. Un nombre indéterminé de bitmask supplémentaire peut-être chaînés en spécifiant simplement le bit 31 de chaque bitmask à un, indiquant la présence d'un bitmask suivant.

7.3.1 Données radiotap

Les champs de données défini et suggéré sont listés avec leur numéro de bit. A cause de l'utilisation du bit 31 de chaque trame pour le chaînage de bitmask, les bits 31, 62, etc. (n * 32 – 1) sont réservés. L'espace de nommage spécifié aux fournisseurs ainsi que l'espace de nommage radiotap réserve également les bits 29 et 30. Les bits aux positions 29, 39 et 31 sont donc réservés et inutilisables.

La figure suivante liste la signification de chaque bit défini du champ *IT Present* de l'en-tête. Lorsque le bit en question est égal à un, cela indique la présence du champ correspondant dans la table ci-dessous.

bit number	field
0	/TSFT
1	/Flags
2	/Rate
3	/Channel
4	/FHSS
5	/Antenna signal
6	/Antenna noise
7	/Lock quality
8	/TX attenuation
9	/dB TX attenuation
10	/dBm TX power
11	/Antenna
12	/dB antenna signal
13	/dB antenna noise
14	/RX flags
19	/MCS
20	/A-MPDU status
21	/VHT

Figure 18 - Données radiotap définies

Les données correspondantes aux bits d'IT Present sont les suivantes : [13]

Radiotap Data Fields	
Champ	**Numéro de bit (flag)**
TSFT	**0 (0x1)**

Taille : 8 octets
Valeur en microsecondes du timer MAC « Time Synchronization Function » 64 bits 802.11 lors de l'arrivée du premier bit du MPDU.

Flags **1 (0x2)**

Taille : 1 octet
Ce champ est une bitmap représentant certaines valeurs, comme la présence d'un préambule, l'encryption WEP, ainsi que d'autres (voir figure ci-dessous).

Mask	Meaning
0x01	sent/received during CFP
0x02	sent/received with short preamble
0x04	sent/received with WEP encryption
0x08	sent/received with fragmentation
0x10	frame includes FCS
0x20	frame has padding between 802.11 header and payload (to 32-bit boundary)
0x40	frame failed FCS check

Figure 19 – Flags

Parmi ces flags, il y a spécialement deux qui vont être très important pour l'élaboration de parseur de trames wifi. Le premier est le « frame includes FCS 0x10 ». Si ce bit est à 1, cela signifiera que la trame comportera comme en-queue quatre octets de Frame Check Sequence. La plupart du temps ce sera un CRC. Il faudra alors que le parseur s'occupe de ces octets afin de ne pas les interpréter comme des données.

Le deuxième flag important est le « frame has padding between 802.11 header and payload (to 32-bit boundary) ». La présence de ce flag va déterminer si il y a du padding après l'en-tête 802.11. Dans l'affirmative, on devra passer sur des bits de padding en cas d'un en-tête de longueur non multiple de quatre octets (to 32-bit boundary).

[13] http://www.radiotap.org/defined-fields

Rate **2 (0x4)**

Taille : 1 octet
Taux de transmission/réception en valeur de 500 Kbps. Si, par exemple, cet octet est égal à 4, cela signifiera un taux de transmission de 2 Mbps.

Channel **3 (0x8)**

Le champ canal est divisé en plusieurs parties contenant des informations différentes toute reliée au canal utilisé lors de la transmission de la trame.

1ère partie
Taille : 2 octets
Fréquence de transmission/réception en MHz.

2ème partie
Taille : 2 octets
Bitmap représentant diverses informations sur le canal, comme la modulation, la fréquence, etc. (voir figure ci-dessous).

Mask	Meaning
0x0010	Turbo Channel
0x0020	CCK channel
0x0040	OFDM channel
0x0080	2 GHz spectrum channel
0x0100	5 GHz spectrum channel
0x0200	Only passive scan allowed
0x0400	Dynamic CCK-OFDM channel
0x0800	GFSK channel (FHSS PHY)

Figure 20 - Channel flags

Ces informations nous sont utiles car elles permettent de déterminer le type de canal grâce à des combinaisons des caractéristiques de celui-ci :

- Turbo Channel | OFDM Channel → Channel A
- 2 GHz spectrum | CCK Channel → Channel B
- 2 GHz spectrum | Dynamic CCK → Channel G
- 5 GHz spectrum | OFDM Channel | Turbo Channel → Channel TA
- 2 GHz spectrum | Dynamic CCK | Turbo Channel → Channel TG

FHSS **4 (0x10)**

Le champ FHSS est divisé en plusieurs parties :

1ère partie
Taille : 1 octet
« Hop set »

2ème partie
Taille : 1 octet
« Pattern for frequency-hopping radios »

Antenna signal **5 (0x20)**

Taille : 1 octet
Puissance du signal RF (Radio Frequency) à l'antenne, en décibels de 1mW.

Antenna noise **6 (0x40)**

Taille : 1 octet
Bruit du signal RF à l'antenne, en décibels de 1mW.

Lock quality **7 (0x80)**

Taille : 2 octets
Qualité du code lock de Barker. Cette valeur est appelée « Qualité du signal » dans les fiches techniques.

TX attenuation **8 (0x100)**

Taille : 2 octets
Puissance de transmission exprimée en tant que distance sans unité par rapport à la puissance maximum lors de la calibration en usine. En termes plus compréhensibles, c'est l'attenuation du signal de transmission. 0 est la puissance maximum, donc avec une attenuation nulle.

dB TX attenuation **9 (0x200)**

Taille : 2 octets
Attenuation de la puissance de transmission exprimée en dB.

dBm TX power **10 (0x400)**

Taille : 1 octet
Puissance de transmission exprimée en dBm (décibels à partir d'une référence de 1 milliwatt). Cette valeur représente le niveau de puissance absolu mesuré à l'antenne.

Antenna **11 (0x800)**

Taille : 1 octet
Index de l'antenne RX/TX pour ce paquet. La première antenne est 0.

dB antenna signal **12 (0x1000)**

Taille : 1 octet
Puissance du signal RF à l'antenne en décibel.

dB antenna noise **13 (0x2000)**

Taille : 1 octet
Bruit RF à l'antenne en décibel.

RX flags **14 (0x4000)**

Taille : 2 octets
Bitmap représentant les propriétés des trames reçues. Les flags suivants sont définis :

mask	meaning
0x0001	reserved [was FCS failed but this is a regular flag]
0x0002	PLCP CRC check failed
0xfffc	reserved for future expansion

Figure 21 - RX Flags

MCS **19 (0x80000)**

Le champ MCS est divisé en trois parties distinctes.

1ère partie : MCS known
Taille : 1 octet
Bitmap contenant les informations listées dans la figure ci-dessous :

flag	definition
0x01	bandwidth
0x02	MCS index known (in mcs part of the field)
0x04	guard interval
0x08	HT format
0x10	FEC type
0x20	STBC known
0x40	Ness known (Number of extension spatial streams)
0x80	Ness data - bit 1 (MSB) of Number of extension spatial streams

Figure 22 - MCS known

2ème partie
Taille : 1 octet
Bitmap contenant les informations listées dans la figure ci-dessous :

flag	definition
0x03	bandwidth - 0: 20, 1: 40, 2: 20L, 3: 20U
0x04	guard interval - 0: long GI, 1: short GI
0x08	HT format - 0: mixed, 1: greenfield
0x10	FEC type - 0: BCC, 1: LDPC
0x60	Number of STBC streams
0x80	Ness - bit 0 (LSB) of Number of extension spatial streams

Figure 23 - MCS Flags

3ème partie
Taille : 1 octet
Les taux MCS sont référencés à la page suivante :
http://en.wikipedia.org/wiki/IEEE_802.11n-2009#Data_rates

A-MPDU status **20 (0x100000)**

La présence de ce champ indique que la trame a été reçue en tant que part d'un a-MPDU. Le concept d'a-MPDU est de joindre plusieurs MPDU regroupés sous les mêmes en-têtes.

1ère partie
Taille : 4 octets
Numéro de référence généré par l'outil de capture. Ce numéro est le même pour chaque trame inférieure d'un A-MPDU. Il permet de fusionner plusieurs captures. Attention, ce numéro n'est pas garanti d'être unique entre les différents canaux.

2ème partie
Taille 2 octets
Bitmap contenant les informations listées dans la figure ci-dessous :

0x0001	driver reports 0-length subframes
0x0002	frame is 0-length subframe (valid only if 0x0001 is set)
0x0004	last subframe is known (should be set for all subframes in an A-MPDU)
0x0008	this frame is the last subframe
0x0010	delimiter CRC error
0x0020	delimiter CRC value known: the delimiter CRC value field is valid
0xffc0	reserved

Figure 24 - A-MPDU flags

3ème partie
Taille : 1 octet
CRC de contrôle du A_MPDU.

4ème partie
Taille : 1 octet
Réservé.

VHT **21 (0x200000)**

Le champ VHT est divisé en plusieurs parties parties d'information distinctes listées ci-dessous :
1ère partie
Taille : 2 octets
Bitmap contenant les informations suivantes :

flag	definition	notes
0x0001	STBC known	In `flags` part of the field.
0x0002	TXOP_PS_NOT_ALLOWED known	In `flags` part of the field.
0x0004	Guard interval	In `flags` part of the field.
0x0008	Short GI NSYM disambiguation known	In `flags` part of the field.
0x0010	LDPC extra OFDM symbol known	In `flags` part of the field.
0x0020	Beamformed known/applicable	In `flags` part of the field. This flag should be set to zero for MU PPDUs.
0x0040	Bandwidth known	In `bandwidth` part of the field.
0x0080	Group ID known	In `group_id` part of the field.
0x0100	Partial AID known/applicable	In `partial_aid` part of the field. This flag should be set to zero for MU PPDUs.
0xfe00	(unused)	

Figure 25 - VHT known flags

2ème partie
Taille : 1 octet
Bitmap contenant les informations suivantes :

flag	definition	notes
0x01	STBC	Space-time block coding. Set to 0 if no spatial streams of any user has STBC. Set to 1 if all spatial streams of all users have STBC.
0x02	TXOP_PS_NOT_ALLOWED	Valid only for AP transmitters. Set to 0 if STAs may doze during TXOP. Set to 1 if STAs may not doze during TXOP or transmitter is non-AP.
0x04	Guard interval.	Set to 0 for long GI. Set to 1 for short GI.
0x08	Short GI NSYM disambiguation	Valid only if short GI is used. Set to 0 if NSYM mod 10 != 9 or short GI not used. Set to 1 if NSYM mod 10 = 9.
0x10	LDPC Extra OFDM symbol	Set to 1 if one or more users are using LDPC and the encoding process resulted in extra OFDM symbol(s). Set to 0 otherwise.
0x20	Beamformed	Valid for SU PPDUs only.
0xc0	(unused)	

Figure 26 - VHT flags

3ème partie

Taille : 1 octet

bitmask	definition	notes
0x1f	Bandwidth	
0xe0	(unused)	

Figure 27 - VHT bande passante

Cet octet encode la bande passante. Pour une capture, la bande passante totale du transmetteur n'est généralement pas connue et pour cette raison la bande passante sera seulement enregistrée en tant que 20, 40, 80 ou 160. Les valeurs de bande passante de ce champ sont les suivantes :

value	total bandwidth (MHz)	sideband	sideband index
0	20		
1	40		
2	40	20L	0
3	40	20U	1
4	80		
5	80	40L	0
6	80	40U	1
7	80	20LL	0
8	80	20LU	1
9	80	20UL	2
10	80	20UU	3
11	160		
12	160	80L	0
13	160	80U	1
14	160	40LL	0
15	160	40LU	1
16	160	40UL	2
17	160	40UU	3
18	160	20LLL	0
19	160	20LLU	1
20	160	20LUL	2
21	160	20LUU	3
22	160	20ULL	4
23	160	20ULU	5
24	160	20UUL	6
25	160	20UUU	7

Figure 28 - VHT valeurs bande passante

4ème partie

Taille : 4 octets

Ces quatre octets encodent MCS et NSS (Network switching subsystem) pour jusqu'à quatre utilisateurs.

bitmask	definition	notes
0x0f	NSS	Number of spatial streams, range 1-8.
0xf0	MCS	MCS rate index, range 0-9.

Figure 29 - MCS et NSS

Si le champ NSS d'un utilisateur est égal à zéro, alors l'utilisateur n'est pas présent et le MCS et le codage (5ème partie) ne sont pas valides et doivent être ignorés.

5ème partie

Taille : 1 octet non signé

Cet octet encode le FEC pour jusqu'à quatre utilisateurs.

bitmask	definition	notes
0x01	Coding for user 0	Set to 0 for BCC. Set to 1 for LDPC.
0x02	Coding for user 1	Set to 0 for BCC. Set to 1 for LDPC.
0x04	Coding for user 2	Set to 0 for BCC. Set to 1 for LDPC.
0x08	Coding for user 3	Set to 0 for BCC. Set to 1 for LDPC.
0xf0	(unused)	

Figure 30 - VHT coding

6ème partie

Taille : 1 octet

Cette partie contient l'ID du groupe. Cet ID est utilisé pour différencier les différentes SU PPDUs et MU PPDUs.

7ème partie

Taille : 2 octets

Contient le AID partiel. Ce champ est uniquement utilisable pour les SU PPDU.

Radiotap Namespace **29 (0x20000000)**

Ce bit est réservé pour le domaine de nom radiotap.

Vendor Namespace	**30 (0x40000000)**

Ce champ est réservé pour le domaine de nom du fournisseur.

Extended Bitmask	**31 (0x80000000)**

Ce bit spécifie la présence d'une bitmap supplémentaire suivant directement ce champ. Les données se trouvent après toutes les bitmaps chaînées.

On a pu remarquer que radiotap est assez complexe mais surtout très complet et permet un ajout d'informations important par dessus des trames 802.11.

Les données qui seront ajoutées après l'en-tête radiotap sont très flexible et se basent sur la bitmap « IT Present », dernier champ de l'en-tête. L'avantage est qu'il est très facile de rajouter des champs sans que les parseurs actuels ne soient rendus inutilisables.

7.4 IEEE 802.11 Wireless Local Area Network

[14]Le but du standard IEEE 802.11, développé par le « Institute of Electrical and Electronics Engineers », est de définir les spécifications pour la couche « Medium Access Control (MAC) » et la couche « Physical Layer (PHY) » pour la connectivité sans fils des stations fixes, portables et en mouvements d'un réseau local. Pour faire plus simple, le but est de spécifier la façon dont l'information est transportée d'un point à l'autre en utilisant l'air comme média. Mon travail dans ce projet a été de rechercher et apprendre le format des trames aux normes 802.11 afin de pouvoir développer un parseur de telles trames. Pour cela, je me suis basé sur le document officiel publié par l'institut EEE, publication 2012, disponible à l'adresse http://standards.ieee.org/about/get/802/802.11.html.

Ce document comptant près de 2700 pages, il n'est évidemment pas possible d'expliquer en détail la norme dans ce rapport. Pour cette raison, je ne vais ici que parler des champs les plus importants des trames à la norme 802.11.

Tout en-tête de trame 802.11 est constitué selon une forme bien spécifique, définie ci-dessous.

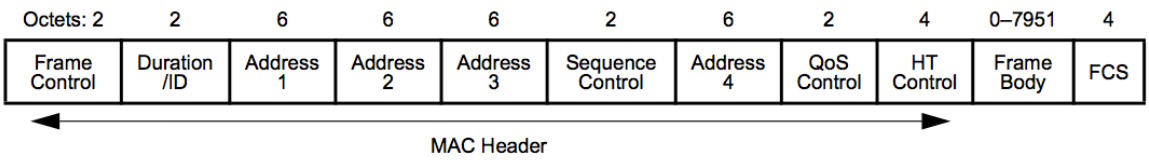

Figure 31 - Trame 802.11

[14] Toute la documentation sur IEEE802.11 vient du standard publication 2012 http://standards.ieee.org/about/get/802/802.11.html.

Le terme « station » ou STA est largement utilisé dans ce chapitre et il est important d'en donner une définition précise. Une station est simplement un appareil ayant la capacité de comprendre le protocole 802.11. Il peut s'agir d'une access point, d'un téléphone portable, d'un ordinateur portable, etc.

La forme spécifiée ci-dessus est le format complet de l'en-tête MAC d'une trame 802.11 et tous les champs spécifiés ne sont pas obligatoirement présents dans l'en-tête. Comme on le verra plus tard, chaque trame possède un type et un sous-type qui vont définir exactement la présence ainsi que la signification de chaque champ de l'en-tête.

802.11 General frame format	
Champ	**Taille (octets)**
Frame Control	**2 octets**

Le champ frame control consiste en une collection de sous-champs : Protocol, Version, Type, Subtype, To DS, From DS, More fragments, Retry, Power Management, More Data, Protected Frame et Order.

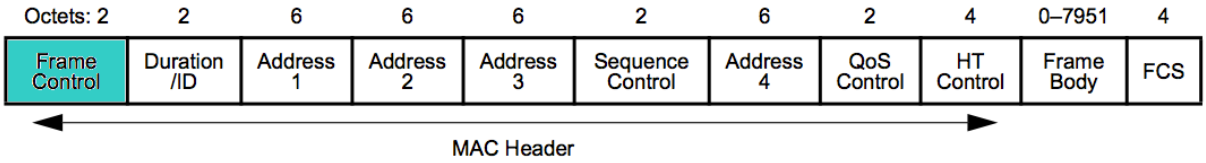

Figure 32 - Frame control

Tous ces champs sont expliqués ci-dessous et vont déterminer la présence ainsi que la signification des champs suivant frame control. Frame control est un champ obligatoire qui se retrouvera dans l'en-tête d'absolument toutes les trames wifi 802.11.

Les sous-champs de frame control sont montrés dans la figure ci-dessous :

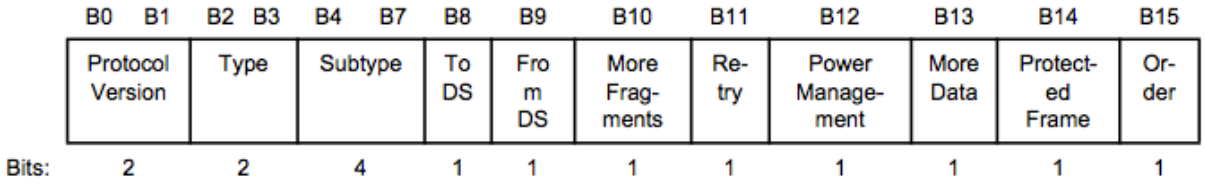

Figure 33 - Frame control sous-champs

Protocol version : Pour cette révision du standard, la version est toujours 0. Toutes les autres valeures sont réservées. Le numéro de révision sera incrémenté uniquement quand une incompatibilité fondamentale existera entre la nouvelle révision et l'ancienne.

Type et Subtype : Le type et le sous-type sont les champs les plus importants de toutes trames wifi car ils déterminent totalement la signification et l'usage de la trame. Il existe quatre types de trames : Management, Data, Control et Reserved. Pour chacun de ces

types, de nombreux sous-types existent.

Une fois le type déterminé, on peut se pencher sur le sous-type. C'est le sous-type qui va faire la distinction entre une trame Probe request, Beacon, Association, Authentification et autres. Tous ces exemples ci-dessus sont des sous-types de trames de type Management.

En plus de cela, le bit de poid fort (most significant bit, MSB) du sous-type est appelé le champ QoS (Quality of Service field). Ce bit spécifie la présence du champ QoS plus tard dans la trame, comme on le voit sur la figure ci-dessus représentant l'en-tête MAC.

Le tableau ci-dessous montre toutes les possibilités de combinaison type – sous-type formant l'ensemble de toutes les trames wifi possibles dans la norme actuelle :[15]

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0000	Association request
00	Management	0001	Association response
00	Management	0010	Reassociation request
00	Management	0011	Reassociation response
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	0110	Timing Advertisement
00	Management	0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101	Action
00	Management	1110	Action No Ack
00	Management	1111	Reserved
01	Control	0111	Control Wrapper
01	Control	1000	Block Ack Request
01	Control	1001	Block Ack

[15] Format spécifié dans Std 802.11-2012 (voir réf.) page 382 - 383

01	Control	1010	Ps-Poll
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	ACK
01	Control	1110	CF-End
01	Control	1111	CF-End+CF-Ack
01	Control	0000 - 0110	Reserved
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack + CF-Poll (no data)
10	Data	1000	QoS Data
10	Data	1001	QoS Data + CF-Ack
10	Data	1010	QoS Data + CF-Poll
10	Data	1011	QoS Data + CF-Ack + CF-Poll
10	Data	1100	QoS Null (no data)
10	Data	1101	Reserved
10	Data	1110	Qos CF-Poll (no data)
10	Data	1111	Qos CF-Ack + CF-Poll (no data)
11	Reserved	0000 – 1111	Reserved

Comme cité précédemment, chaque bit dans le sous-type va indiquer une modification spécifique aux données basique de la trame. Le bit 4 est égal à 1 pour les trames qui incluent +CF-Ack, le bit 5 informe de l'inclusion de +CF-Poll, le bit 6 indique qu'il n'y a pas de corps pour cette trame et le bit 7 est mis à 1 pour les trames de type QoS.

To DS et From DS : La signification de la combinaison des valeurs des sous-champs To DS et From DS sont montrés dans la figure ci-dessous :

To DS and From DS values	Meaning
To DS = 0 From DS = 0	A data frame direct from one STA to another STA within the same IBSS, a data frame direct from one non-AP STA to another non-AP STA within the same BSS, or a data frame outside the context of a BSS, as well as all management and control frames.
To DS = 1 From DS = 0	A data frame destined for the DS or being sent by a STA associated with an AP to the Port Access Entity in that AP.
To DS = 0 From DS = 1	A data frame exiting the DS or being sent by the Port Access Entity in an AP, or a group addressed Mesh Data frame with Mesh Control field present using the three-address MAC header format.
To DS = 1 From DS = 1	A data frame using the four-address MAC header format. This standard defines procedures for using this combination of field values only in a mesh BSS.

Figure 34 - From DS et To DS

La signification de ces champs n'est pas essentielle à la conception d'un parseur de trames 802.11. Ces deux champs auront un impact sur les trames de type Data uniquement, dans lesquelles ils changeront la signification des différents champs d'adresses.

More Fragments : Le flag More Fragment est un simple bit dans toutes les trames de type Management ou Data qui va indiquer que la trame possède un autre fragment à suivre.

Retry : Le flag Retry est un bit qui va indiquer que la trame est une retransmission d'une trame précédemment envoyée qui a été perdue.

Power Management : Power Management est un bit utilisé pour indiquer le mode de gestion de la puissance.

More Data : More Data est un flag d'un bit utilisé pour indiquer que d'autres trames destinées à cette station sont présentes dans le buffer de l'access point. Le bit à 1 indique qu'au moins une autre trame est dans le buffer prête à être envoyée à cette station. Ce flag n'est valable que pour les trames de type Data ou Management.

Protected Frame : Le flag Protected Frame est mis à 1 lorsque le corps de la trame (les données) ont été passées à travers un algorithme d'encapsulation cryptographique. En résumé, si les données sont protégées par WEP, WPA/WPA2 ou tout autre type de cryptographie, ce bit est mis à 1. Ce bit n'est interpreté que pour les trames de type Data et Management.

Order : Le bit Order est utilisé pour deux raisons :

- 1 dans une trame non QoS utilisé pour indiquer que la trame contient un MSDU.
- 1 dans une trame QoS afin d'indiquer la présence d'un champ HT Control.

Duration/ID **2 octets**

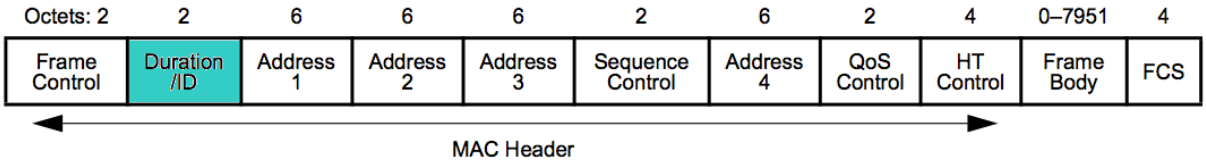

Figure 35 - Duration/ID

Le contenu du champ Duration/ID varie en fonction du type et du sous-type de la trame dépendant si la trame est transmise durant le CFP (Contention Free Poll), et avec des capacités de qualité de service. Par contre ce champ sera également toujours présent dans n'importe quel en-tête d'une trame 802.11. Les valeurs de ce champ sont expliquées dans le tableau ci-dessous.

Bits 0–13	Bit 14	Bit 15	Usage
0–32 767		0	Duration value (in microseconds) within all frames other than PS-Poll frames transmitted during the CP, and under HCF for frames transmitted during the CFP
0	0	1	Fixed value under point coordination function (PCF) within frames transmitted during the CFP
1–16 383	0	1	Reserved
0	1	1	Reserved
1–2007	1	1	AID in PS-Poll frames
2008–16 383	1	1	Reserved

Figure 36 - Contenu du champ Duration/ID

Address fields **6 octets par champ**

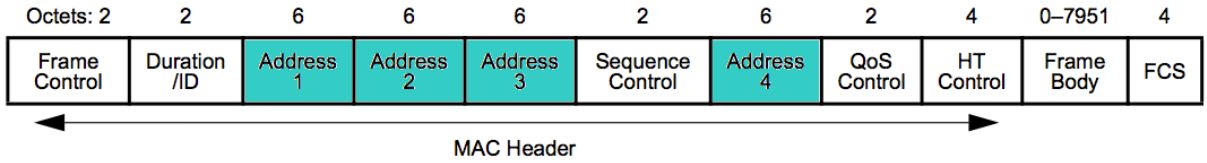

Figure 37 – Address fields

Il y a quatre champs d'adresses dans le format de trames MAC. Ces champs sont utilisés pour indiquer le « Basic Service Set Identifier (BSSID) », « Source Address (SA) », « Destination Address (DA) », « Transmitting STA Address (TA) » et « Receiving STA address ». Il y aura au moins un champ adresse dans chaque trame, mais il est possible que les trois autres ne soient pas présentes dans certaines en-têtes 802.11.

Chaque adresse est un champ de 48 bits défini par le standard IEEE 802-2001 plus connu sous le nom d'adresse MAC ou d'adresse physique. L'adresse est composée de deux éléments, le « Organization Unique Identifier (OUI) » et le « Network Interface Controller (NIC) ». Le premier élément, faisant 3 octets, définit l'identifiant du constructeur de l'interface à laquelle l'adresse appartient. Le deuxième élément différencie les différentes interfaces d'un même constructeur.

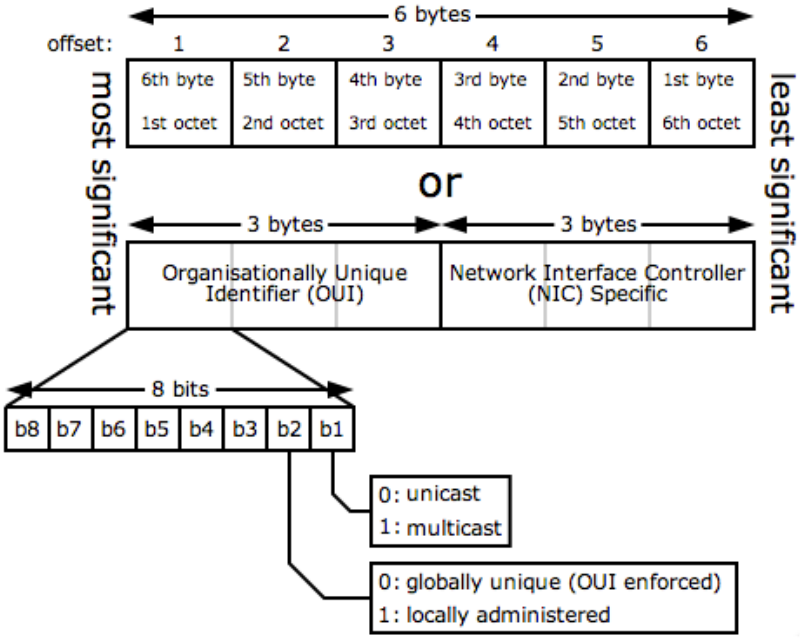

Figure 38 - Adresse MAC
http://fr.wikipedia.org/wiki/Adresse_MAC

La présence de chaque adresse ainsi que sa signification sera expliqué plus bas lors de la présentation individuelle de chaque type de trame.

BSSID : Le BSSID est une adresse MAC permettant d'identifier de manière unique chaque BSS au sein d'un réseau WLAN. La valeur de ce champ dans une infrastructure BSS est l'adresse MAC utilisé par l'AP du BSS. Dans le cas d'une infrastructure IBSS, le BSSID est gére localement avec 46 bits générés aléatoirement.

DA : Le champ DA contient l'adresse MAC individuelle d'une interface ou d'un groupe étant le récipiendaire final de la trame en question.

SA : Le champ SA contient l'adresse MAC individuelle d'une interface ou d'un groupe ayant émis la trame.

RA : Le champ RA contient l'adresse MAC individuelle d'une interface ou d'un groupe identifiant le récipiendaire immédiat des données contenues dans la trame.

TA : Le champ TA contient l'adresse MAC individuelle d'une interface ou d'un groupe ayant transmis les données contenues dans la trame.

Sequence Control **2 octets**

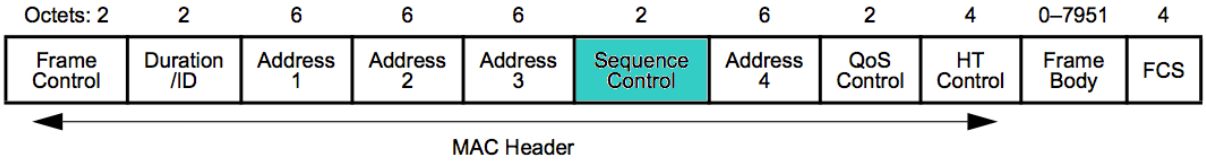

Figure 39 - Sequence control field

Le champ sequence control consiste en deux sous-champs, le numéro de fragment ainsi que le numéro de séquence de la trame, constitué de la manière suivante :

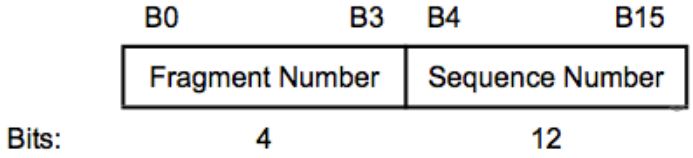

Figure 40 - Sequence Control subfields

Le champ sequence control n'est pas présent dans les trames de contrôle.

Fragment number : Le numéro de fragment est un champ de 4 bits indiquant le numéro de chaque fragment d'un MSDU et qui est incrémenté pour chaque fragment. Ce numéro est mis à 0 pour chaque trame non fragmentée.

Sequence number : Le numéro de séquence est un champ de 12 bits indiquant le numéro de séquence d'un MSDU. Chaque trame émise par une station se voit assignée un numéro de séquence. Seules les trames de contrôle n'ont pas de numéro de séquence assigné étant donné que ce champ n'existe pas.

Lors de l'utilisation de trames fragmentées, le numéro de séquence est le même pour chaque fragment. Autrement, le numéro est incrémenté jusqu'à 4095, puis repart de zéro pour les trames suivantes.[16]

[16] Les informations sur le champ de sequence control, fragment number ainsi que sequence number proviennent de http://technet.microsoft.com/en-us/library/cc757419(v=ws.10).aspx

QoS Control **2 octets**

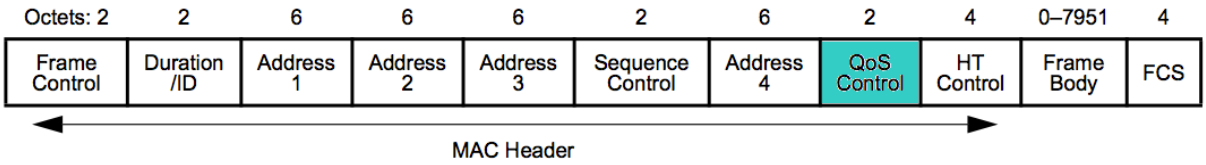

Figure 41 - QoS Control

Le champ QoS control est un champ utilisé pour le contrôle de la qualité de service offerte par la norme 802.11. Ce champ est présent dans toutes les trames de type data qui ont le flag QoS à 1. Chaque champ QoS control comprend cinq ou huit sous-champs définis pour la station envoyant la trame. L'utilisation de ces champs dépend de chaque sous-type de trames.

Les sous-champs définis sont le TID, EOSP et Ack Policy, A-MSDU Present, TXOP Duration Request, Mesh Control present, Mesh Power Save Level, Queue size, AP PS Buffer State. La signification détaillée de ces champs n'est pas pertinente au contenu de ce rapport et ne sera pas expliqué dans ce document.

HT Control **4 octets**

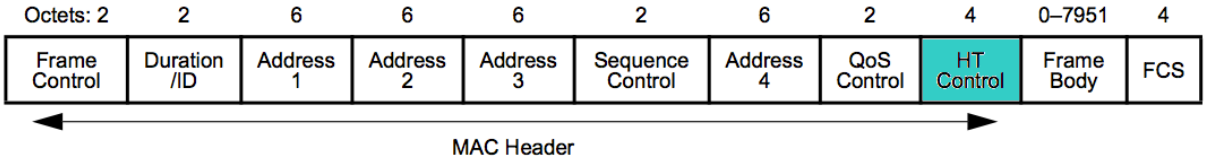

Figure 42 - HT Control field

Le champ ht control est toujours présent dans les trames de type « Control Wrapper » ainsi que dans les trames Data QoS et les trames Management qui ont le flag Order à 1. Le flag Order est le dernier bit du champ frame control.

Le format du champ HT control est défini dans la figure ci-dessous :

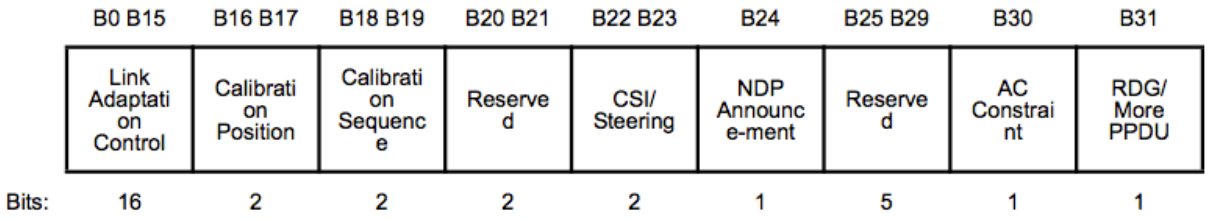

Figure 43 - HT Control field

Les différents éléments du champ HT Control sont très spécifiques aux trames 802.11 et ne sont pas importants à la conception d'un parseur.

Frame Body **0 – 7951 octets**

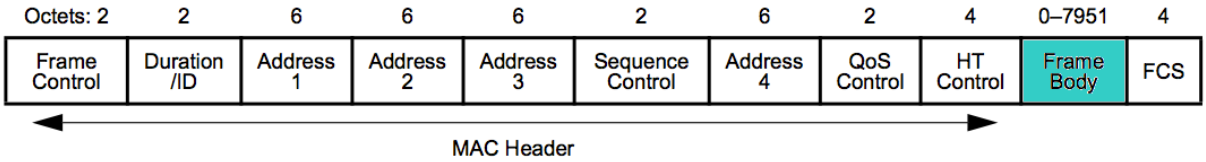

Figure 44 - Frame Body

Le corps des trames 802.11 a une longueur variable allant de 0 à 7951 octets et contient les informations spécifiques aux type et sous-type individuelle des trames wifi. Si le Mesh Control Field le spécifie, un overhead pour l'encryption de la trame est présent avant les données.

Le reste des données dans le corps de la trame dépend du type et du sous-type. Il peut s'agir de tag 802.11 pour une trame management, il peut s'agir d'un en-tête LLC précédent des protocoles plus haut-niveau comme IP ou il peut ne pas y avoir de données du tout. On peut déterminer les données qui seront contenues dans frame body en se basant sur le type et le sous-type du champ frame control. Les différences et les détails de chaque type et sous-type sont expliqués dans la section suivante.

Frame Check Sequence **4 octets**

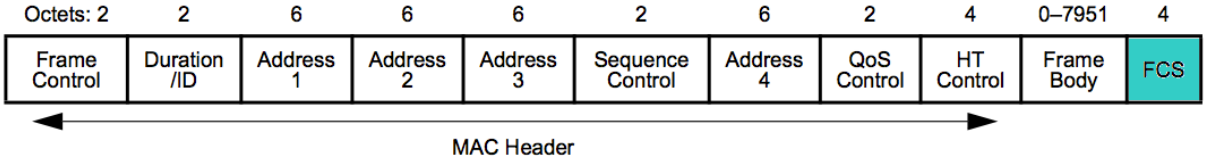

Figure 45 - FCS field

Le champ FCS est les 4 octets de contrôle placés à la fin des trames 802.11. Ce champ est présent si le flag « FCS is present » est à 1 dans l'en-tête radiotap (voir le chapitre concernant Radiotap). Ce champ contient un CRC 32 bit calculé sur tous les champs de l'en-tête MAC ainsi que sur le corps de la trame. Le polynôme générateur de degré 32 utilisé pour le calcul du FCS est le suivant :

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Le FCS est le complément à 1 de la somme (modulo 2) des étapes suivantes :

a) Le reste de $x^k \times (x^{31} + x^{30} + \cdots + x^2 + x + 1)$ divisé (modulo 2) par $G(x)$, avec k le nombre de bit dans les champs utilisés pour le calcul.
b) Le reste de la multiplication du contenu des champs par x^{32} puis divisé par $G(x)$.

Le champ FCS est transmis en commençant par le coefficient le plus haut.

Tous les champs que l'on peut trouver lors de l'analyse d'une trame wifi à la norme 802.11 ont été décrit dans le tableau ci-dessus et ne seront pas réexpliqué lors de l'analyse en détails de chaque trame spécifique. Il s'agit maintenant de déterminer quels champs seront présents et quels champs ne le seront pas en se basant sur le type et le sous-type d'une trame. On sait que les champs Frame Control et Duration/ID se retrouveront dans chaque trame, maintenant nous allons parler de chaque sous-type individuel.

7.4.1 Control frames

Les trames de type Control (champ type = 0x1) sont les plus simple a interpréter. Chaque sous-type de trame de contrôle possède un format bien défini de longueur fixe et ne comporte pas de données. Les trames de contrôles facilitent l'échange de données entre les stations.[17]

7.4.1.1 Trame RTS

Les trames « Request-To-Send » et « Clear-To-Send » fournissent un mécanisme de réduction de collisions pour les access points et les stations cachées. Une station envoie un « Request-To-Send » avant de transmettre afin de s'assurer que le canal est libre. C'est ce qu'on appelle la première étape du « two-way handshake » se produisant entre une station et un access point.

Le format des trames « Request-To-Send » est le suivant :

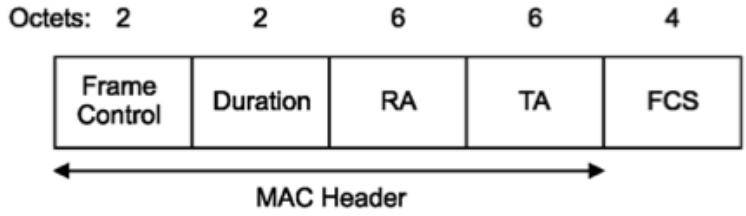

Figure 46 - RTS frame

Pour toutes les trames RTS envoyées par des stations non QoS, le champ Duration est le temps en microsecondes nécessaire avant de pouvoir envoyer des données.

7.4.1.2 Trame CTS

Les trames « Clear-To-Send » sont envoyées par un point d'accès en tant que réponse aux trames « Request-To-Send » afin d'informer une station qu'elle peut commencer à transmettre. Ces trames sont donc la deuxième phase du « two-way handshake ».

Le format des trames « Clear-To-Send » est le suivant :

[17] http://en.wikipedia.org/wiki/IEEE_802.11#Control_Frames

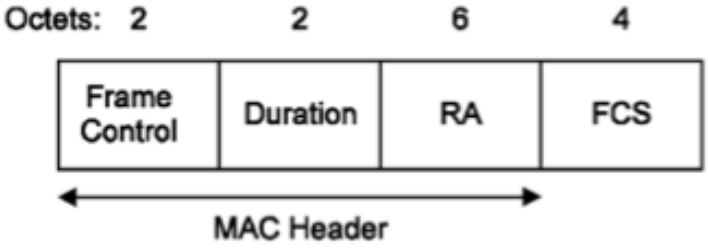

Figure 47 - CTS frame

Lorsqu'une trame CTS suit une RTS, le champ RA de la trame CTS est copié depuis le champ TA de la trame RTS reçue.

7.4.1.3 Trame Ack

Les trames « Acknoledgment » sont envoyées par une station après avoir reçu une trame de données si aucune erreur n'est survenue. Si la station ayant envoyé la trame de données ne reçoit pas d'accusé de réception (sous la forme d'un Ack), la station retransmettra la trame.

Le format des trames « Acknoledgment » est le suivant :

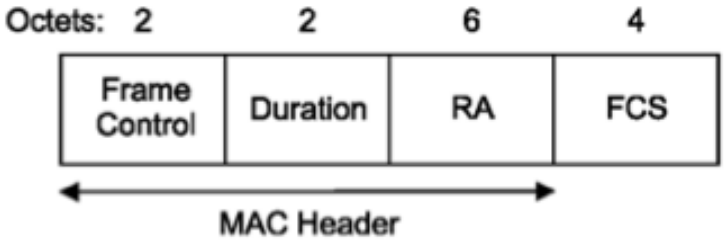

Figure 48 - Ack frame

Le champ RA d'une trame de contrôle Ack est directement copié à partir de l'adresse de la trame à laquelle cet Ack répond.

7.4.1.4 Trame PS-Poll

« Power-save poll (PS-Poll) » est une trame permettant l'élection d'une trame stockée dans un buffer après qu'une station se réveille.

Le format des trames « PS-Poll » est le suivant :

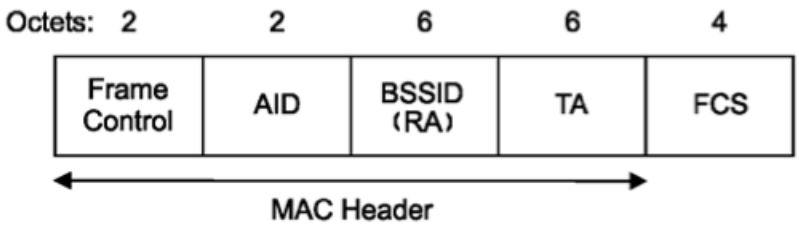

Figure 49 - Trame PS-Poll

Le champ Duration/ID des trames PS-Poll a une différente signification des autres types de trames traitées précédemment. AID est la valeur assignée à une station transmettant la trame par le point d'accès qui a établit l'association avec la station.

7.4.1.5 Trame CF-End

Lorsque la période contention-free (sans conflit de transmission) se termine, l'access point envoie une trame « Contention-Free End (CF-End) » afin de libérer les stations des règles d'accès imposées par la période Contention-Free.

Le format des trames « CF-End » est le suivant :

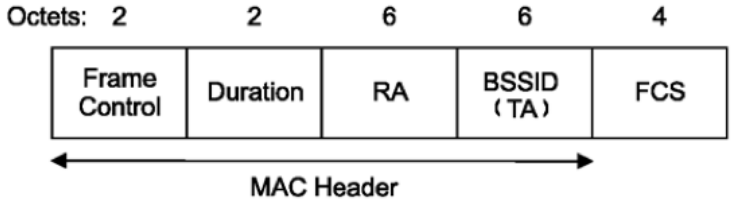

Figure 50 - Trame CF-End

Le champ BSSID est l'adresse du point d'accès. Le champ RA contient l'adresse de diffusion multicast du groupe.

7.4.1.6 Trame CF-End-CF-Ack

Lorsque la période de contention-free se termine, une trame CF-End est transmise par l'AP. Si l'access point doit également accuser la réception de données, il enverra une trame CF-End-CF-Ack qui accomplira la fonction d'une trame CF-End et d'une trame CF-Ack en même temps.

Le format des trames « CF-End-CF-Ack » est le suivant :

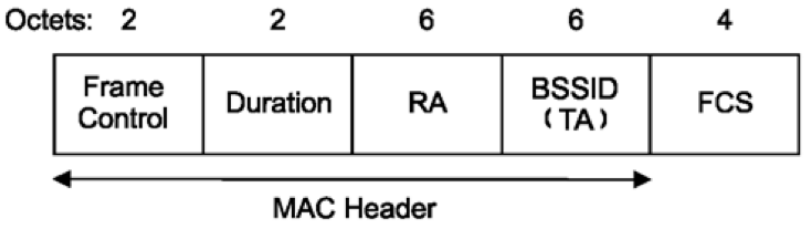

Figure 51 - Trame CF-End-CF-Ack

7.4.2 Management frames

Les trames de type management (type = 0x0) permettent la maintenance de la communication sur un réseau WLAN. Les différentes trames de type management

permettent par exemple la découverte d'un réseau et encore l'association et l'authentification permettant de s'y connecter.[18]

Le format des trames de type management est défini dans la figure ci-dessous. Les champs frame control, duration, address1, address2, address3 et sequence control sont présent dans tous les sous-types de trames management. La taille maximale des données encapsulées dans une trame de management est de 2304 octets, sans compter les octets de contrôle placés à la fin de la trame.

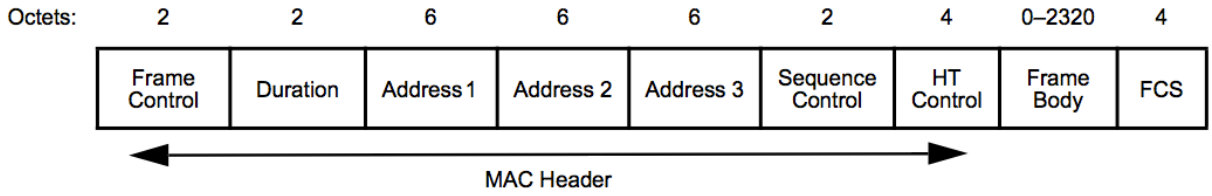

Figure 52 - Management header

Le champ **Address1** est utilisé afin de faire correspondre les adresses lors de la réception de trames. Ce champ est utilisé pour la destination de la trame et est donc utilisé par l'AP pour faire la correspondance entre une source envoyant une trame destinée à une station ou à un groupe multicast. Ce champ est le DA expliqué plus haut dans la section concernant le format général d'une trame 802.11.

Le champ **Address2** est toujours l'adresse source de la station transmettant la trame, c'est le SA.

Le champ **Address3** peut avoir plusieurs significations. La plupart du temps ce champ sera le BSSID, à l'exception de quand une station est une « Mesh STA », c'est-a-dire qu'elle est interconnectée avec toutes les autres stations. Dans ce cas là, address3 est l'adresse de transmission (TA).

Par contre, il est notable que le champ **QoS Control** n'est pas présent dans les trames management par rapport au format général des trames 802.11. Cela est dû au fait que l'on n'a pas besoin de qualité de service pour les trames de gestion, ce qui rend l'utilisation de ce champ inutile.

Comme pour les autres types, le champ **HT Control** sera présent uniquement si le flag Order est présent dans frame control.

[18] http://en.wikipedia.org/wiki/IEEE_802.11#Management_Frames

7.4.2.1 Management frame body

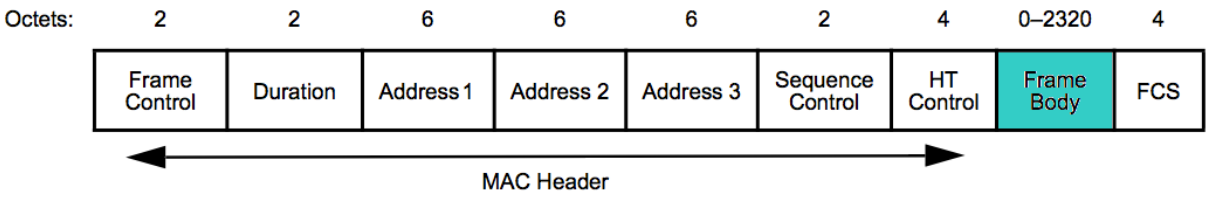

Figure 53 - Management data

La différence entre les différents sous-types des trames de management vient des données qui vont suivre l'en-tête. Le corps des trames management consiste en des champs et des éléments définis de manière précise pour chaque-sous type existant. Tous les champs et éléments sont officiellement obligatoires à moins qu'il n'en soit défini autrement.

Il existe donc deux type de structures qui peuvent suivre l'en-tête d'une trame management : Des champs ou des éléments.

Les champs sont simplement les données brutes insérées dans l'ordre précis dans lequel la norme l'impose. Lorsqu'une trame management possède des champs, ces champs sont obligatoires et doivent apparaître dans l'ordre précisé par la norme. Il existe énormément de champs différents et comme cité précédemment, ce n'est pas le but de ce travail de bachelor de comprendre le moindre détail de la norme 802.11 qui est extrêmement précise et complexe. Je vais donc expliquer les différents champs importants qui sont implémentés par le parseur 802.11 de l'application.[19] « Quels champs apparaissent quand » est expliqué dans la section concernant chaque sous-type individuel des trames de management de ce chapitre. Ces champs sont référencés comme « Fixed Parameters » dans ce rapport.

802.11 Management Frame Data Fields	
Champ	**Taille (octets)**
Timestamp	**8 octets**

Le timestamp représente la valeur du timer de la « Timing Synchronization function » de la source d'une trame.

Beacon Interval	**2 octets**

Ce champ représente le nombre d'unité de temps (TU) entre chaque transmission de Beacon frame (voir explication sur chaque sous-type). Une unité de temps TU est égal à 1 seconde / 1024e-6.

[19] Toutes les informations sur les champs des données de trames management viennent du document officiel IEEE Std 802.11 version 2012 disponible à http://standards.ieee.org/about/get/802/802.11.html à partir de la page 437.

Authentication algorithm number **2 octets**

Ce champ indique le numéro d'un algorithme d'authentification utilisé pour un handshake d'authentification. Les valeures suivantes sont définies en tant que numéro d'algorithme d'authentification :

- 0 = Open System
- 1 = Shared Key
- 2 = Fast BSS Transition
- 3 = Simultaneous Authentication of Equals (SAE)
- 65535 = Vendor Specific Use

Tous les autres numéros d'algorithme d'authentification sont réservés.

Authentication transaction sequence number **2 octets**

Ce champ indique l'état actuel de progression dans une transaction en plusieurs étapes.

Capability Information **2 octets**

Le champ capability information contient un certain nombre de sous-champs qui sont utilisés pour indiquer les capacités optionnelles demandées ou annoncées par une station ou un AP. Les sous-champs spécifiés pour ce champ sont les suivants :

B0	B1	B2	B3	B4	B5	B6	B7
ESS	IBSS	CF Pollable	CF-Poll Request	Privacy	Short Preamble	PBCC	Channel Agility

B8	B9	B10	B11	B12	B13	B14	B15
Spectrum Mgmt	QoS	Short Slot Time	APSD	Radio Measurement	DSSS-OFDM	Delayed Block Ack	Immediate Block Ack

Figure 54 - Capability Information

Listen Interval **2 octets**

Ce champ est utilisé pour indiquer à l'access point la fréquence à laquelle une station écoute les beacon lorsqu'elle est en « power save mode ».

Status Code **2 octets**

Ce champ est utilisé dans la trame management de réponse pour indiquer le succès ou l'échec d'une opération demandée. Il existe plus de cents codes qui peuvent être utilisé avec

ce champ. Les plus fréquents sont les suivants :

Status code	Name	Meaning
0	SUCCESS	Successful
1	REFUSED, REFUSED_REASON_UNSPECIFIED	Unspecified failure
2		TDLS wakeup schedule rejected but alternative schedule provided
3		TDLS wakeup schedule rejected
4		Reserved
5		Security disabled
6		Unacceptable lifetime
7		Not in same BSS
8–9		Reserved

Figure 55 - Status codes

Association ID Code **2 octets**

Dans une infrastructure BSS, le champ AID est une valeur assignée par l'AP lors de l'association. Ce champ représente l'ID d'une station. Dans une opération mesh BSS, ce champ est l'ID d'un voisin avec lequel la station est associée.

Current AP Address **6 octets**

Ce champ représente la MAC adresse de l'access point avec lequel la station est actuellement associée.

Reason code **2**

Ce champ est utilisé pour indiquer la raison pour laquelle une notification non désirée de type Disassociation, Deauthentication a été générée. Il existe environ 70 raisons et donc codes définis par la norme. Le plus fréquents sont les suivants :

Reason code	Name	Meaning
0		Reserved
1		Unspecified reason
2		Previous authentication no longer valid
3		Deauthenticated because sending STA is leaving (or has left) IBSS or ESS
4		Disassociated due to inactivity
5		Disassociated because AP is unable to handle all currently associated STAs

Figure 56 - Reason codes

Le deuxième type de structure que l'on va trouver après les champs de données lors de la lecture des données d'une trame management s'appelle les éléments d'information. Les éléments ont une forme plus générique qui simplifie nettement l'analyse des données.

En effet, le premier élément de chaque élément est son ID. L'ID d'un élément l'identifie de manière unique. L'octet suivant est la longueur de l'information en question. Savoir la longueur de l'information est toujours pratique lors de l'élaboration d'un parseur. Vient ensuite l'information de longueur spécifiée par l'octet précédent.[20]

Ces champs sont référencés comme « Tagged Parameters » dans ce rapport.

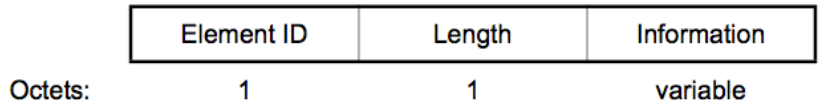

Figure 57 – Element

Il existe plus de deux cents ID et donc type d'informations différentes. Uniquement les plus importants types d'élément sont expliqués ci-dessous.

802.11 Management Frame Data Elements	
Champ	**Taille des données (octets)**
SSID	**0-32 octets**

Le SSID (Service Set Identifier) identifie l'identité d'un ESS ou d'un IBSS. En clair, c'est le nom d'un réseau sans fil.

Supported Rates **1-8 octets**

Cet élément spécifie jusqu'à huit vitesses opérationnelles supportées par le réseau. Chaque octet est une vitesse en unité de 500 kilobits par seconde.

Extended Supported Rates **1-255 octets**

Cet élément spécifie jusqu'à 255 vitesses opérationnelles supportées par le réseau. Cet élément est utile pour les réseaux qui supportent plus que huit vitesses. C'est un complément à l'élément Supported Rates. Chaque octet est une vitesse en unité de 500 kilobits par seconde.

[20] Toutes les informations sur les champs des données de trames management viennent du document officiel IEEE Std 802.11 version 2012 disponible à http://standards.ieee.org/about/get/802/802.11.html à partir de la page 474.

DSSS **1 octet**

Cet élément contient le numéro du canal radio utilisé par les stations.

TIM **4 – 254 octets**

L'élément TIM content quatre champs : DTIM Count, DTIM Period, Bitmap Control, Partial Virtual Bitmap.

	Element ID	Length	DTIM Count	DTIM Period	Bitmap Control	Partial Virtual Bitmap
Octets:	1	1	1	1	1	1–251

Figure 58 - Elément TIM

DTIM Count : Indique combien de Beacon frame vont apparaître avant le prochain DTIM.
DTIM Period : Indique l'intervalle de beacon entre des DTIM successives

Le Bitmap control et Partial Virtual Bitmap sont utilisés pour diverses informations concernant le trafic sur le réseau.

CF **6 octets**

Cet élément contient un ensemble de paramètres nécessaire au support du PCF (Point Coordinated Function). PCF est une fonction optionnelle qui permet aux access points de coordiner les transmissions pour les AP et les stations. Les champs contenus dans l'élément CF sont le CFPCount, CFPPeriod, CFPMaxDuration, CFPDuRemaining.

	Element ID	Length	CFP Count	CFP Period	CFP MaxDuration (TU)	CFP DurRemaining (TU)
Octets:	1	1	1	1	2	2

Figure 59 - Elément CF

FH **5 octets**

Cet élément contient un set de paramètres permettant la synchronisation entre les stations. Les diverses informations contenu dans ce champ sont le Dwell Time (en TU), Hop set, Hop pattern et Hop index.

	Element ID	Length	Dwell Time (TU)	Hop Set	Hop Pattern	Hop Index
Octets:	1	1	2	1	1	1

Figure 60 - Elément FH

IBSS **2 octets**

L'élément IBSS permet le support des IBSS. Un IBSS, Independant Basic Service Set, est un réseau WLAN 802.11 ad-hoc. C'est-à-dire que les clients sont interconnectés indépendamment de tout access point.

Country **6 – 254 octets**

Cet élément fournit les informations nécessaires à une station pour s'identifier dans un domaine régulé dans lequel la station se trouve afin de configurer sa couche physique. Chaque pays possède des régulations différentes et ce paramètre fournit, d'après le pays, le numéro du premier canal, le nombre de canal utilisables ainsi que la puissance maximum de transit.

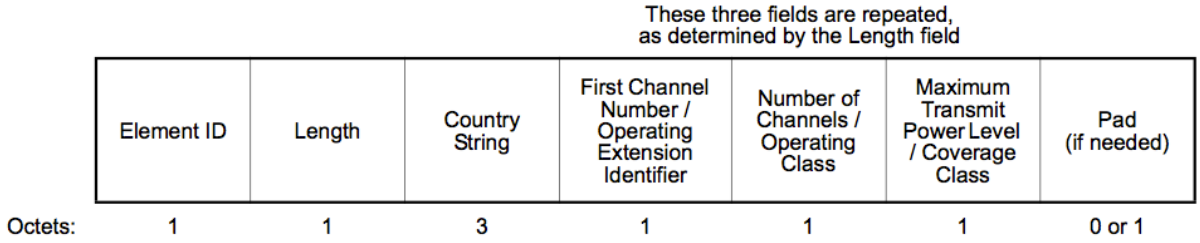

Figure 61 - Elément Country

Challenge **1– 253 octets**

Cet élément contient le texte de challenge utilisé durant l'échange d'authentification.

BSS Load **5 octets**

Cet élément contient des informations concernant la population actuelle d'utilisateurs ainsi que le niveau de trafic sur le BSS. Cet élément est souvent utilisé lorsqu'une station fait du roaming.

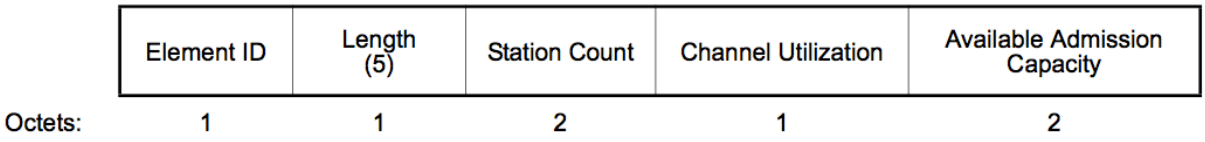

Figure 62 - Elément BSS Load

Vendor Specific **1 – 255 octets**

Cet élément est différent des autres car son contenu n'est pas défini par la norme. Néanmoins il est très souvent utilisé. Vendor Specific est utilisé par n'importe quelle organisation possédant un OUI afin de transporter des informations dans un format défini dans le but d'atteindre une interopérabilité plus facilement malgré la présence d'information non standard. Le format de l'élément est décrit dans la figure ci-dessous. Les trois premiers octets ou plus sont réservés afin d'identifier l'organisation grâce à son OUI. Appelons la longueur de ce champ *j*. Les octets suivants sont les données spécifique au vendeur et seront de taille *n* – *j* ou *n* est la taille maximale de l'élément : 255.

	Element ID	Length	Organization Identifier (see 8.4.1.31)	Vendor-specific content
Octets:	1	1	*j*	*n–j*

Figure 63 - Elément Vendor Specific

Il y a généralement plusieurs éléments de type Vendor Specific dans chaque trame.

Tout ces éléments et ces champs vont être retrouvé dans la plupart des trames management. Maintenant, il est nécessaire de savoir quels champs et éléments sont obligatoire et dans quel ordre ils vont apparaître. Pour cela, il faut se pencher sur le format individuel de chaque sous-type des trames management.

7.4.2.2 Beacon frame

Les beacons transportent les informations sur un réseau et sont transmises périodiquement pour annoncer la présence du réseau WLAN. C'est ces trames là que tous nos appareils utilisent pour générer la liste des réseaux sans-fils disponibles. C'est l'access point qui va générer en transmettre les beacons d'un réseau BSS (infrastructure). Dans le cas d'un IBSS, la distribution des trames beacons se fait entre les stations.

Les beacons possèdent obligatoirement les champs suivants (dans l'ordre spécifié) :[21]

Ordre	Champ / Elément	Notes
1	Timestamp	
2	Beacon Interval	
3	Capability	
4	SSID	Si c'est un réseau de type Mesh, le SSID est un SSID de type wildcard.
5	Supported rates	

7.4.2.3 ATIM frame

Les trames management de sous-type ATIM sont utilisées dans les réseaux ad-hoc (IBSS) afin de pouvoir dresser une « carte » du trafic. Ce sous-type de trame ne comporte aucune donnée dans son corps.

[21] Format spécifié dans Std 802.11-2012 (voir réf.) page 419

7.4.2.4 Association Request frame

Ce genre de trame est envoyée par une station désirant se connecter à un WLAN et va permettre à l'access point d'allouer des ressources et de synchroniser cette station. Les Association transportent les informations suivantes (dans l'ordre spécifié) :[22]

Ordre	Champ / Elément	Notes
1	Capability	
2	Listen Interval	
3	SSID	
4	Supported Rates	

7.4.2.5 Association Response

Ces trames sont envoyées par un access point à une station afin de l'informer de l'acceptance ou de la rejection de la demande d'association envoyée précédemment. Les association responses vont manifestement être envoyé après la réception d'une association request. En cas d'association positive (status code d'association réussie), la trame contiendra l'identifiant de l'association ainsi que les vitesses supportées.

Ordre	Champ / Elément	Notes
1	Capability	
2	Status Code	
3	AID	
4	Supported Rates	
6	ECDA Parameter	Vient après un éventuel Extended Supported Rates (s'il y a plus de 8 vitesses supportées) qui serait placé en 5ème position si présent.

7.4.2.6 Reassociation Request

Lorsqu'une station passe en dehors de la portée du champ de l'access point avec lequel elle est associée et qu'elle trouve un autre point d'accès avec une signal plus fort, c'est une Reassociation Request qui va être envoyé au nouvel access point.

[22] Format spécifié dans Std 802.11-2012 (voir réf.) page 423

Celui-ci va coordiner le transfert de données restantes dans les buffers de l'ancien point d'accès jusqu'au nouveau.

Les Reassociation Requests transportent les informations suivantes :

Ordre	Champ / Elément	Notes
1	Capability	
2	Listen Interval	
3	Current AP Address	
4	SSID	
5	Supported Rates	

7.4.2.7 Reassociation Response

Cette trame suivra bien sûr une reassociation request et est envoyée par l'access point ayant reçu la trame avec le status code spécifiant si la réassociation est acceptée ou pas.

Les reassociation responses transportent les informations suivantes :

Ordre	Champ / Elément	Notes
1	Capability	
2	Status Code	
3	AID	
4	Supported Rates	
6	ECDA Parameter	Vient après un éventuel Extended Supported Rates (s'il y a plus de 8 vitesses supportées) qui serait placé en 5ème position si présent.

7.4.2.8 Probe Request

Lorsqu'une station a besoin d'obtenir des informations à propos d'une autre station, c'est une probe request qui est envoyée. Par exemple, une station enverrait une probe request pour déterminer les access points qui sont à portée.

Les probes request transportent les champs/éléments suivants :

Ordre	Champ / Elément	Notes
1	SSID	Si c'est un réseau de type Mesh, le SSID est un SSID de type wildcard.
2	Supported Rates	

7.4.2.9 Probe Response

Lors de la réception d'une probe request, la station en question va répondre grâce à une probe response en envoyant les informations suivantes :

Ordre	Champ / Elément	Notes
1	Timestamp	
2	Beacon interval	
3	Capability	
4	SSID	Si c'est un réseau de type Mesh, le SSID est un SSID de type wildcard.
5	Supported rates	

7.4.2.10 Authentication frame

L'authentification commence lorsqu'une station envoie une trame management de sous-type authentication contenant sont identitié à l'access point. La suite dépend du mode d'authentification. En cas de « Open Authentication », un seul échange ne se passe entre la station et l'access point, qui va répondre par l'acceptance ou le refus d'authentification. Avec une authentification de type « Shared Key », l'access point va répondre à cette trame avec un challenge text que la station demandant l'authentification devra encrypter et renvoyer à l'access point. Celui-ci pourra ensuite vérifier le texte en le décryptant avec sa propre clé et le résultat de cette opération permettra, ou pas, l'authentification de la station.

Les trames authentication transportent obligatoirement les informations suivantes :

Ordre	Champ / Elément	Notes
1	Authentication algorithm number	
2	Authentication transaction sequence number	

7.4.2.11 Deauthentication frame

Ce genre de trame est envoyé par une station souhaitant terminer la connexion sécurisée avec une autre. La seule information obligatoire dans les trames de deauthentication est le « **Reason code** ». Il est très commun de trouver des « **Vendor Specific** » également,

7.4.2.12 Action frame

Les trames Action sont utilisées pour démarrer une action dans une cellule. Ces trames sont en quelques sortes spéciales. Elles permettent à une station d'avertir l'AP, ou alors l'AP d'avertir les stations que « ceci va se passer », « Il faut que vous fassiez ceci ». [23] C'est un espère d'ordre ou de requête demandé à être exécuté ou avertissant d'une action. Le seul champ obligatoire dans les action frames est « **Action** », qui va contenir le code de l'action qui doit ou qui va être effectuée. Il est très commun de trouver des « **Vendor Specific** » également.

Il existe une variante de la trame Action s'appelant « Action No-Ack » qui ne demande pas l'envoi d'accusé de réception.

7.4.2.13 Timing Advertisement frame

L'élément Timing Advertisement spécifie des champs décrivant la source de temps correspondant à un standard de temps étant soit une horloge externe, un offset entre un temps standard et un timer ou une dérivation d'une estimation d'un offset. Cette information peut être utilisée par un STA la recevant afin d'aligner son horloge sur celle du STA envoyant la trame.

Les informations obligatoires se trouvant dans le corps de la trame sont les suivantes :

Ordre	Champ / Elément	Notes
1	Timestamp	
2	Capability	

7.4.3 Data Frames

Le but des trames de type Data (type = 0x2) est manifestement de transporter des données.[24] Les réseaux sans fils possèdent de nombreuses fonctions mais l'idée principale est de transporter de l'information d'un point à l'autre en utilisant l'air comme média de transmission. Ce que l'on a pu voir jusqu'à maintenant, les

[23] Information citée de la source suivante : http://my.safaribooksonline.com/book/certification/9781118075234/chapter-4-802dot11-management-frames/action_frames
[24] Introdution au data frames http://www.wi-fiplanet.com/tutorials/article.php/3442991

nombreuses trames de contrôle ou de management, tout cela sert à finalement pouvoir transférer des données grâce au trames Data entre plusieurs stations.

Grâce aux MAC adresses, les trames Data peuvent être envoyée à une station particulière, ou à un groupe en utilisant du multicasting. Dans des BSS (infrastructure), les trames passent toujours à travers l'AP pour atteindre leur destination. L'AP prendra le contenu de la trames et l'enverra encapsulé dans une nouvelle trame Data. Dans les IBSS (ad-hoc), les trames seront transférées directement d'un utilisateur à un autre sans intermédiaire.

Le corps des trames de données transporte des informations de protocoles de couches supérieures tel que des paquet UDP ou TCP. Le format des trames Data est le suivant :

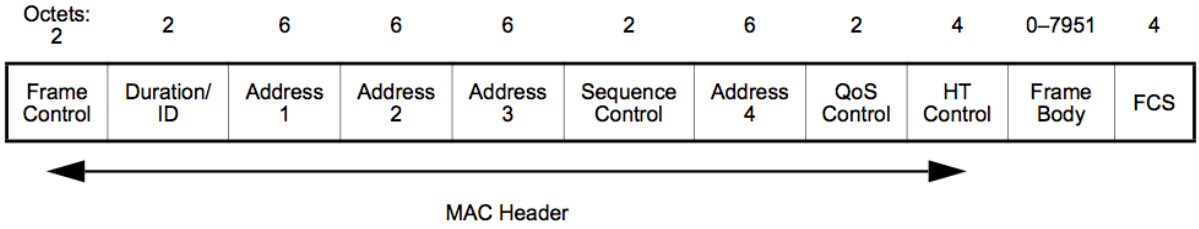

Figure 64 - Data frame

Les trames avec le flag QoS à 1 dans le champ subtype de frame control sont appelées des QoS data frames. Chacun de ces sous-type de trames data ont QoS dans leur nom et possèderont le champ QoS Control dans leurs en-têtes. Une STA supportant la qualité de service utilisera toujours des trames de type QoS pour communiquer avec d'autres stations supportant la qualité de service.

Le contenu des quatre champs d'adresse dépend des valeurs de To-DS et From-DS dans frame control. Le tableau ci-dessous décrit précisément la signification de chaque adresse dans tous les cas possibles.

To DS	From DS	Address 1	Address 2	Address 3		Address 4	
				MSDU case	A-MSDU case	MSDU case	A-MSDU case
0	0	RA = DA	TA = SA	BSSID	BSSID	N/A	N/A
0	1	RA = DA	TA = BSSID	SA	BSSID	N/A	N/A
1	0	RA = BSSID	TA = SA	DA	BSSID	N/A	N/A
1	1	RA	TA	DA	BSSID	SA	BSSID

Figure 65 - Champs Address

En ce qui concerne le corps des trames Data, il existe deux possibilités :

- MSDU (ou un fragment), MAC Service Data Unit, comprenant une en-tête et une en-queue de sécurité si la trame est protégée.
- A-MSDU avec un en-tête et une en-queue de sécurité si la rame est protégée. Un A-MSDU est un MSDU directement opéré au niveau de la couche MAC, contrairement au MSDU standard transmis à la couche supérieur (généralement LLC, Logical Link Control). C'est des trames agrégées, c'est-à-dire qu'il s'agit d'une trame avec un seul en-tête contenant plusieurs trames destinées au même client.[25]

La taille du payload (Frame Body) montré sur la figure ci-dessus n'est possible que dans le cas d'utilisation de A-MSDU. Dans le cas contraire, la taille maximale sera de 2338 octets pour une encryption de type CCMP et 2342 octets pour du TKIP.[26]

7.4.3.1 Null Data frames

On peut se demander pourquoi est-ce que l'on enverrait des trames de Data, dont la définition même est de transporter des données, avec un payload nul. Et bien certains constructeurs utilisent ce genre de trames pour faire passer des indications de contrôles spéciaux, souvent liés au « power save mode ». Une autre fonction des trames de data vide est le scanning actif. Une station peut envoyer une trame nulle à un AP afin que celui-ci la mette dans le buffer ainsi que toutes les autres trames lui étant destinées, et pendant ce temps la station peut faire un scan des autres fréquences. Une station fera cela car durant un scan elle ne peut recevoir de trames sur un autre canal.

7.5 802.2 Logical Link Control

La norme IEEE 802.2, ou LLC (Logical Link Control), est un protocole de communication de données se plaçant au dessus de la sous-couche MAC de la couche liaison de données du modèle OSI. [27]

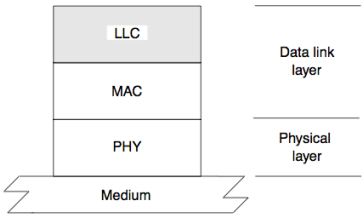

Figure 66 - Vue d'ensemble des couches 1 et 2

[25] Les A-MSDU expliqué à : http://ergodicthoughts.blogspot.com/2012/02/difference-between-mpdu-msdu-ampdu-and.html
[26] IEEE Std 802.11 version 2012 page 414
[27] Toute la documentation sur IEEE 802.2 vient du standard publication 1998 http://standards.ieee.org/about/get/802/802.2.html.

La sous-couche LLC offre des mécanismes de multiplexage permettant à des protocoles de plus haut niveau d'être transportés sur un réseau multipoints. LLC est également très répandu pour sa contrôle de flux et ses mécanismes de gestion d'erreurs.[28]

LLC fait vraiment le lien entre le média (PHY) et la sous-couche MAC de la liaison de données avec la couche réseau.

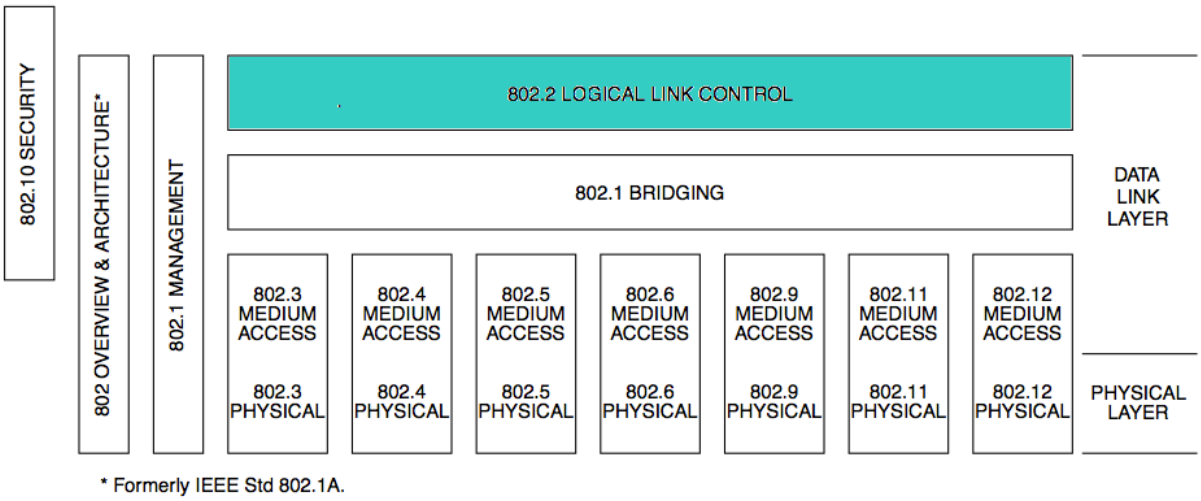

Figure 67 – LLC

Pour satisfaire un grand nombre potentiel d'application, LLC inclut trois types d'opérations de contrôle de liaison de données. Le premier type d'opération est un lien de données sans connexion pour une communication le plus simple possible avec un protocole avec un minimum de complexité. Ce type d'opération peut être utile lorsque les protocoles de plus haut niveau fournissent déjà des services de séquencement et de sécurité sur les données, ainsi on évitera d'être redondant. Les applications ne nécessitant la garantie de livraison des données, comme des applications multimédia temps réels, peuvent également être intéressée par l'utilisation de ce service. Le second type d'opération offert par LLC fournit un service en mode connexion incluant un séquencement des trames, un service d'accusé de réception, une gestion des erreurs compréhensive. Le troisième type d'opération est en quelque sorte un mélange des deux premiers puisqu'il fournit un mode sans connexion avec accusés de réception des trames, ce qui permet à une station de d'envoyer des données et de demander une réponse en même temps. Bien que ce type soit sans connexion, la réception des données est garantie par le séquencement des trames.[29]

[28] http://en.wikipedia.org/wiki/Logical_link_control
[29] Texte cité du standard IEEE 802.2, édition 1998, page 2

Le standard international identifie quatre « classes » distincte d'opérations LLC. La class I fournit un mode sans connexion seul. La classe II fournit un mode avec connexion et un mode sans connexion. La classe III fournit un mode sans connexion avec accusé de réception plus un mode sans connexion. Class IV fournit un mode sans connexion avec accusé de réception plus un mode sans connexion plus un mode avec connexion.

7.5.1 Format des PDU LLC

Toutes les trames LLC doivent se conformer au format présent ci-dessous.[30]

DSAP address	SSAP address	Control	Information
8 bits	8 bits	8 or 16 bits	M*8 bits

Figure 68 - Format des trames LLC

802.2 PDU format	
Champ	**Taille (octets)**
DSAP address	**1 octet**

Chaque PDU doit contenir deux champs address de chacun un bit. Le Destination Service Access Point (DSAP) identifie le ou les service access point pour qui l'information LLC est destinée. Ce champ comporte sept bits d'information permettant d'identifier le service access point. Le bit de poids faible, s'il n'est pas mis à 1, signifie que la trame est une « Individual DSAP ». Dans le cas contraire, c'est une « Group DSAP».

SSAP address **1 octet**

Le Source Service Access Point identifie le service access point spécifique d'ou l'information LLC a été initiée. Ce champ comporte sept bits d'information permettant d'identifier le service access point. Le bit de poids faible, s'il n'est pas mis à 1, signifie que la trame est une « Command ». Dans le cas contraire, c'est une « Response ».

Control **1 ou 2 octets**

Ce champ consiste en un ou deux octets utilisés pour désigner une fonction de commande ou de réponse. Ce champ comprend le numéro de séquence quand le type de la trame en a besoin. Les premiers bits vont déterminer la longueur de ce champ. Si le champ commence par « 0 » ou « 10 » alors ce sera un mot de deux octets. Si par contre les premiers bits sont « 11 », le champ ne fera qu'un octet. Le format de ce champ est expliqué dans la figure ci-

[30] Std IEEE 802.2, édition 1998, page 39

dessous :

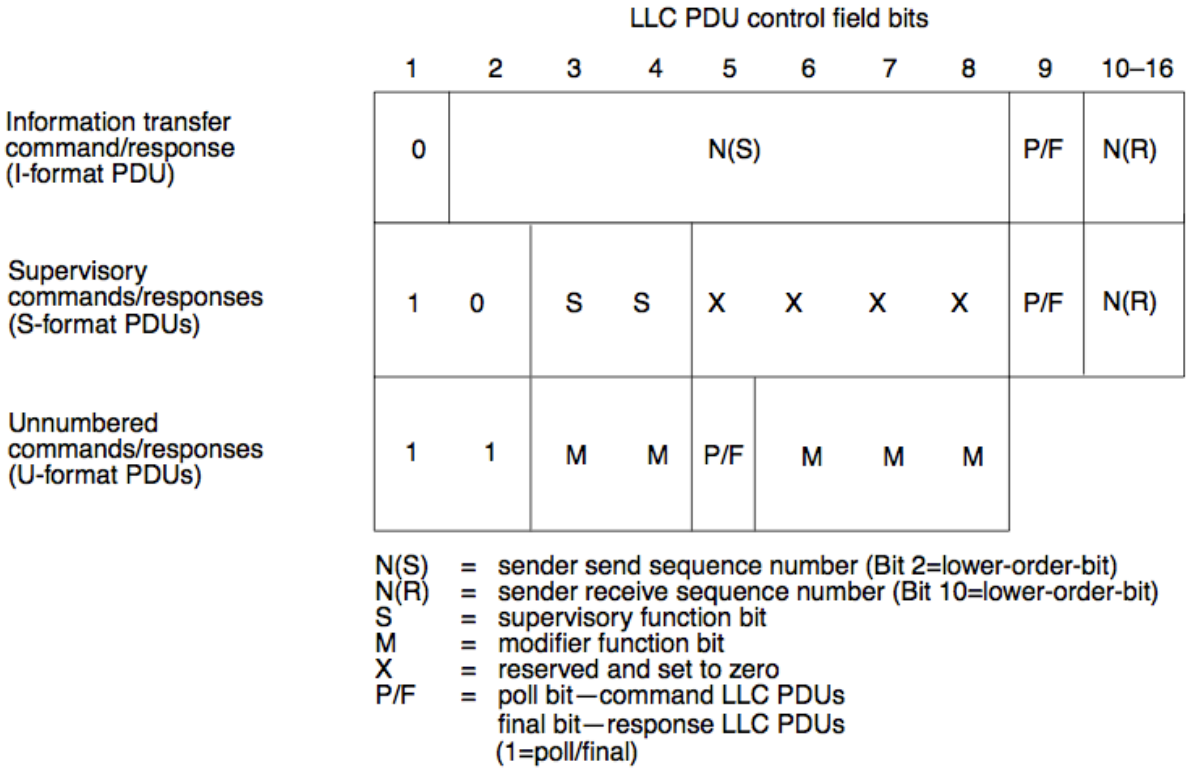

Figure 69 - Control field

Information field **0 – M octets (variable)**

Ce champ est défini de manière vague dans la norme. La définition est la suivante :
« The information field shall consist of any integral number (including zero) of octets. »[31]

7.6 802.2 SubNetwork Access Protocol

Le format détaillé dans le chapitre 802.2 LLC est le format LLC standard défini par l'IEEE. Le format observé dans toutes les captures étudiées et tous les exemples trouvés est légèrement différent. Il s'agit du format nommé SNAP, Subnetwork Access Protocol. SNAP est un standard pour les transmissions de datagrammes IP sur des réseaux aux normes 802. C'est en fait une extension de l'en-tête LLC permettant l'utilisation d'IP et d'ARP sur des réseaux 802.2. [32]

[31] Citation de la page 41, document Std. IEEE 802.2 LLC, édition 1998
[32] SNAP est spécifié par la RFC 1042 : http://www.ietf.org/rfc/rfc1042.txt.

7.6.1 Format des PDU SNAP

L'en-tête SNAP possède une différence par rapport à l'en-tête LLC qui lui permet de supporter IP. Elle possède deux champs de longueurs fixes qui remplacent le champ Information de l'en-tête LLC de longueur variable.

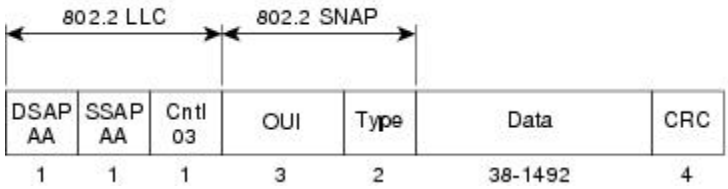

Figure 70 - Extension SNAP à LLC

http://astorinonetworks.com/2011/08/04/meet-the-ethernets/

Extension SNAP à LLC	
Champ	**Taille (octets)**
OUI	**3 octets**

Organizationnaly Unique Identifier est un numéro assigné par l'IEEE, identifie uniquement un fabricant ou une organisation. Ce champ est l'identifiant de l'organisation qui a émis le type (champ suivant).

Type	**2 octets**

Type, ou Ethertype, est le type du protocole encapsulé dans les champs Data. L'ethertype de LLC/SNAP est utilisé pour les protocoles qui ne dépendent pas de l'IEEE mais utilisent tout de même la sous-couche LLC. La liste des ethertypes autorisés est émise par l'IEEE RAC EtherType Field Approval Authority.

Voici quelques exemples des plus utilisées :

- IP (0x0800)
- ARP (0x0806)
- AppleTalk (0x809b)

7.7 Internet Protocol Version 4

Internet Protocol, ou plus communément IP (La version 4 du protocole est connu sous le nom d'IPv4), est un protocole de la couche réseau du modèle OSI. IP permet de transmettre des bloques de données appelés datagrammes d'une source à une destination, ou les sources et les destinations sont des hôtes identifié par des adresses de taille fixes, appelées adresses IP. Internet Protocol fournit également la possibilité de fragmenter puis réassembler des datagrammes plus long qui ne pourrait pas autrement être transmis sur certains réseaux. IP ne comprend par contre

aucun mécanisme de garantie de livraison end-to-end, ni de contrôle de flux, ou de séquencement des paquets. IP est défini par la RFC 791.[33]

7.7.1 Format de l'en-tête IP

L'en-tête IP possède une taille minimale de 20 octets mais peut varier à cause des options qui peuvent être spécifiés.

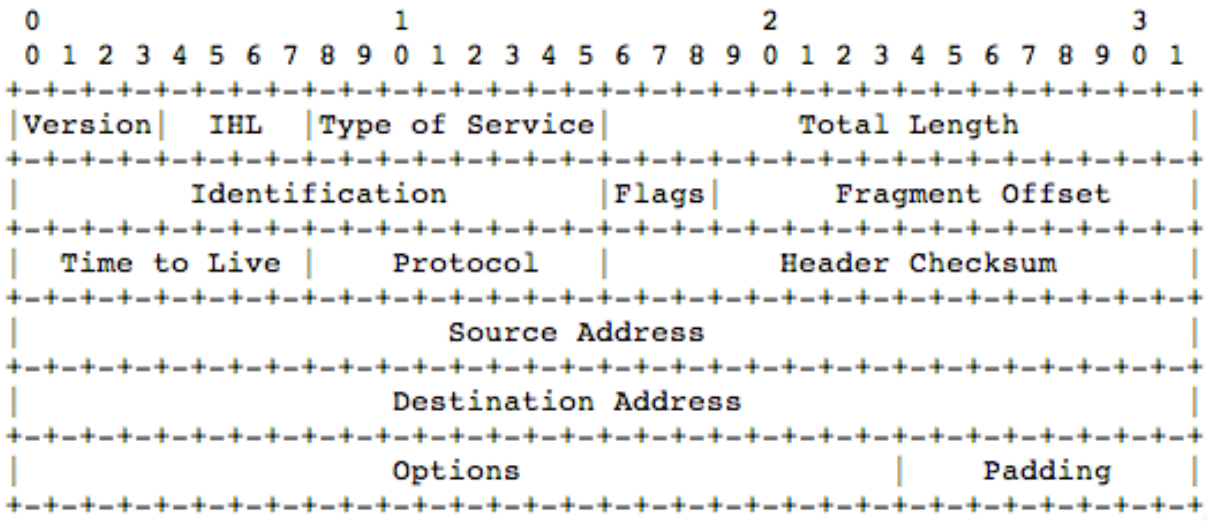

Figure 71 - En-tête IP

http://www.ietf.org/rfc/rfc791.txt page 11

IP header	
Champ	**Taille (octets)**
Version	**1/2 octet (4 bits)**

La version du protocole. Ce document décrit la version numéro 4.

IHL	**1/2 octet (4 bits)**

Internet Header Length nous informe de la longueur de l'en-tête IP en unité de 32 bits. C'est grâce à ce champ que l'on peut déterminer ou commence les données. La valeur minimum pour ce champ est 5, puisque l'en-tête IP fait au moins 20 octets.

Type of Service	**1 octet**

Ce champ fournit une indication sur les paramètres de qualité de service désirés. Certains réseaux offrent un service de priorité, traitant les trames notées comme prioritaire avant le reste du trafic. Le choix de quelle trame est prioritaire sur quelle trame est un compromis entre trois paramètres: délai, fiabilité et débit. Ce champ est divisé de la manière suivante :

[33] http://www.ietf.org/rfc/rfc791.txt

- Bits 0 – 2 : Priorité
  - 111 – Network Control
  - 110 – Internetwork Control
  - 101 – CRITIC/ECP
  - 100 – Flash Override
  - 011 – Flash
  - 010 – Immediate
  - 001 – Priority
  - 000 – Routine
- Bit 3: 0 = Normal Delay, 1 = Low Delay
- Bit 4: 0 = Normal Throughput, 1 = High Throughput
- Bit 5: 0 = Normal Reliability, 1 = High Reliability
- Bits 6 – 7: Reservés

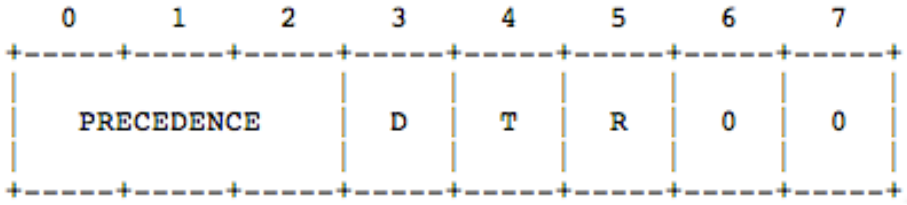

Figure 72 - Type of Service

Total length **2 octets**

Ce champ indique la taille du datagramme en octets, en comptant l'en-tête IP et les données. Ce champ faisant deux octets permet d'avoir des datagrammes d'une taille jusqu'à 65 535 octets. Néanmoins, des datagrammes de cette taille ne sont pas utilisable sur la plupart des réseaux et des hôtes. Tous les hôtes supportant IP supportent des datagrammes de taille allant au moins jusqu'à 576 octets, c'est pourquoi il est recommandé d'envoyer uniquement des paquets de cette taille à moins d'avoir l'assurance que la destination supporte des tailles plus grande.

Identification **2 octets**

Valeur d'identification assignée par l'envoyeur du paquet afin d'aider l'assemblage des fragments d'un datagramme.

Flags **3 bits**

Ce champ comporte trois bits et ainsi trois flags.

- Bit 0 : réservé, doit être égal à 0
- Bit 1 : (DF) 0 = May Fragment, 1 = Don't Fragment.
- Bit 2 : (MF) 0 = Last Fragment, 1 = More Fragments.

Ces bits permettent donc la gestion de la fragmentation. Certains paquets ont besoin d'être transmis en entier et ne doivent pas être fragmentés, ca peut être le cas dans des réseaux VPN avec calcul du CRC avant de l'encryption par exemple. Le flag DF permet de faire cela.

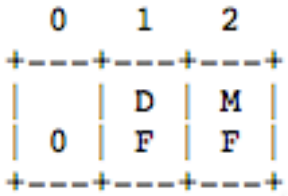

Figure 73 – flags

Fragment offset **13 bits**

Ce champ indique ou se trouve un fragment dans le datagramme. Cette valeur est mesurée en unité de 8 octets. Le premier fragment a un offset de zéro.

Time to Live **1 octet**

Cette valeur représente le nombre de fois maximum qu'un datagramme peut reste dans un système internet. Si ce champ possède une valeur de zéro, alors le datagramme est détruit. Ce champ est décrémenté à chaque fois qu'il arrive sur une machine.

Protocol **1 octet**

Ce champ indique le protocole de niveau suivant utilisé dans les données suivant cette en-tête. Les valeurs identifiant les protocoles sont spécifiée dans la section "Assigned Numbers" [9] de la RFC 791 Internet Protocol.

Voici quelques numéros de protocoles communs :

- TCP (0x06)
- UDP (0x11)
- ICMP (0x01)

Header Checksum **2 octets**

Ce champ est un checksum sur l'en-tête IP seulement. Etant donné que certains champs comme time-to-live changent à chaque bond sur une station, le checksum est recalculé et vérifié à chaque fois que l'en-tête est modifiée.

L'algorithme du checksum est le suivant :[34]
« The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero. »

[34] Citation de l'algorithme donné dans la RFC 791 (IP), page 14 http://www.ietf.org/rfc/rfc791.txt

Source Address **4 octets**

Adresse IP ayant émit le datagramme.

Destination Address **4 octets**

Adresse IP de la destination.

Options **Variable (0 – N)**

Les options peuvent apparaître ou pas dans un datagramme. Il existe deux possibilités concernant la forme de chaque option.

1) Un seul octet représente le type de l'option
2) Un octet avec le type de l'option, un octet avec la longueur de l'option (*l*) et *l* octets de données.

Type de l'option : 1 bit représente le "copied flag" (1 = option a été copié, 0 = non copié), 2 bits sont l'"option class" et les 5 derniers bits sont le numéro de l'option. Les "option classes" sont : 0 = control, 1 = reserved for future use, 2 = debugging and measurement, 3 = reserved for future use.

Le tableau ci-dessous liste toutes les options IP possible avec leurs classe, leurs numéro, leurs longueur ainsi que leurs descriptions.

Class	Number	Length	Description
0	0	-	**End of Option List**. Cette option ne compte qu'un octet et indique la fin des options pour indiquer qu'il n'y en a plus qui va suivre.
0	1	-	**No Operation**. Cette option ne fait absolument rien, il est utilisé entre des options afin d'éventuellement aligner des mots de 32 bits.
0	2	11	**Security**. Cet option fournit la possibilité aux hôtes d'envoyer des paramètres de sécurité et de restrictions.
0	3	Var.	**Loose Source Routing**. Fournit un moyen pour une source d'un datagramme internet de donner des informations de routing au gateways afin de guider le datagramme jusqu'à destination.
0	9	Var.	**Strict Source Routing**. Fournit un moyen pour une source d'un datagramme internet de donner des informations de routing au gateways afin de guider le datagramme jusqu'à destination.
0	7	Var.	**Record Route**. Fournit un moyen pour une source d'un datagramme internet de donner des informations

			de routing au gateways afin d'enregistrer la route du datagramme jusqu'à destination.
0	8	4	**Stream ID**. Cet option fournit un moyen de transporter l'identifiant SATNET de flux à travers des réseaux qui ne supportent pas les flux.
2	4	Var.	**Internet Timestamp**.

7.8 Internet Control Message Protocol

Internet Control Message Protocol est un protocole de la couche réseau du modèle OSI faisant partie de la suite IP utilisé par les systèmes d'exploitation de machines en réseau afin d'envoyer divers messages d'erreurs. Bien qu'étant encapsulé dans IP, ICMP diffère des autres protocoles comme TCP ou UDP dans le sens ou il n'est pas typiquement utilisé pour échanger des données entre deux systèmes ou applications.[35] Il existe plusieurs versions d'ICMP, mais la version traitée dans ce document est ICMP conçu pour IPv4, également appelé ICMPv4. ICMP est défini dans la RFC 792.[36]

7.8.1 Format des paquets ICMPv4

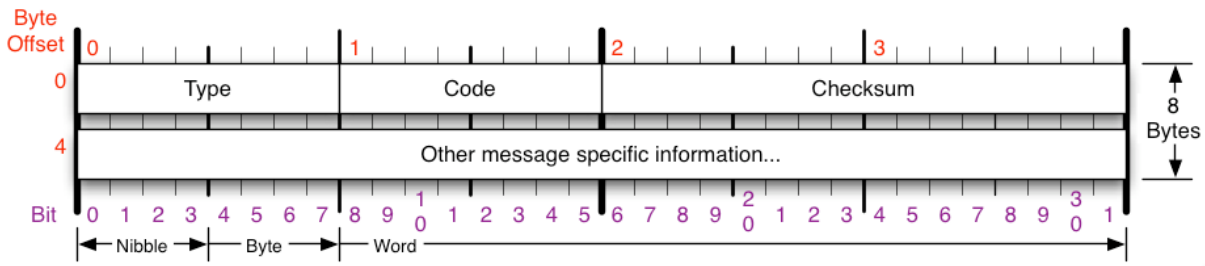

Figure 74 - En-tête ICMP

http://www.siongboon.com/projects/2006-03-06_serial_communication/

Internet Control Message Protocol	
Champ	**Taille (octets)**
Type	**1 octet**

Type de message ICMP. Chaque type possède une variété de codes donnant ensemble une signification au message ICMP. Les combinaisons type – code sont décrites dans un tableau plus bas.

[35] https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol
[36] http://www.ietf.org/rfc/rfc792.txt

Code **1 octet**

Code d'un message ICMP.

Other message specific **variable**

D'après le type et le code, il est possible qu'ICMP ajoutent des informations dans ce champ. Par exemple lors d'un message "Destination Unreachable", ICMP va ajouter l'en-tête ainsi que les 64 premiers bits de la trame qui n'a pas pu arriver à destination. Les différences entre chaque type ne seront pas expliquées dans ce rapport, mais sont disponible dans la RFC 792.

Les combinaisons type / code d'ICMP sont présentées ci-dessous :[37]

Type	Code	Description
0 – Echo Reply	0	Echo reply (used to ping)
1 et 2	-	Reserved
3 – Destination Unreachable	0	Destination network unreachable
	1	Destination host unreachable
	2	Destination protocol unreachable
	3	Destination port unreachable
	4	Fragmentation required, and DF flag set
	5	Source route failed
	6	Destination network unknown
	7	Destination host unknown
	8	Source host isolated
	9	Network administratively prohibited
	10	Host administratively prohibited
	11	Network unreachable for TOS
	12	Host unreachable for TOS
	13	Communication administratively prohibited
	14	Host Precedence violation

[37] Types et codes pris de https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

	15	Precedence cutoff in effect
4 – Source Quench	0	Source quench (congestion control)
5 – Redirect Message	0	Redirect Datagram for the Network
	1	Redirect Datagram for the Host
	2	Redirect Datagram for the TOS & network
	3	Redirect Datagram for the TOS & host
6	-	Alternate Host Address
7	-	Reserved
8 – Echo Request	0	Echo request (used to ping)
9 – Router Advertisement	0	Router advertisement
10 – Router sollicitation	0	Router discovery/selection/sollicitation
11 – Time Exceeded	0	TTL expired in transit
	1	Fragment reassembly time exceeded
12 – Parameter Problem : Bad IP header	0	Pointer indicates the error
	1	Missing a required option
	2	Bad length
13 – Timestamp	0	Timestamp
14 – Timestamp Reply	0	Timestamp reply
15 – Information Request	0	Information request
16 – Information Reply	0	Information reply
17 – Address Mask Request	0	Address Mask Request
18 – Address Mask Reply	0	Address Mask Reply
19	-	Reserved for security
20 – 29	-	Reserved for robustness experiment
30 – Traceroute	0	Information Request
31	-	Datagram Conversion Error
32	-	Mobile Host Redirect
33	-	Where-Are-You (à la base pour IPv6)
34	-	Here-Am-I (à la base pour IPv6)
35	-	Mobile Registration Request

36	-	Mobile Registration Reply
37	-	Domain Name Request
38	-	Domain Name Reply
39	-	SKIP Algorithm Discovery Protocol
40	-	Photuris, Security failures
41	-	ICMP for experimental mobility protocols
42 – 255	-	Reserved

7.9 Transmission Control Protocol

TCP, Transmission Control Protocol, est un des piliers de la suite de protocoles IP plus communément appelée TCP/IP. TCP est un protocole de la couche transport du modèle OSI et a été crée dans le but de fournir un service host-to-host extrêmement fiable entre des hôtes interconnectés sur un réseau. TCP est un protocole orienté connexion, fiable sur toute la ligne (end-to-end) et est conçu pour fonctionner dans une hiérarchie de protocoles supportant les applications multi-réseaux. Très peu d'hypothèses ne sont posées concernant les protocoles en dessous de TCP, on part du principe que l'on va recevoir simplement un datagramme possiblement non fiable d'une couche inférieure et qu'il est nécessaire d'assurer la fiabilité end-to-end.[38]

Afin de garantir un service fiable du début à la fin d'une transmission, TCP établit tout d'abord une connexion entre les deux hôtes concernés grâce à un « three-way handshake » grâce aux trames [SYN] – [SYN-ACK] – [ACK] (les détails concernant les trames TCP sont expliqués dans la section « Segments TCP » de ce chapitre). En plus de la connexion, TCP implémente un service de détection d'erreur grâce à des segments séquencés ainsi qu'un checksum. En plus de ces services, TCP possèdent non seulement un contrôle de flux mais également un contrôle de congestion. Le contrôle de flux end-to-end va permettre d'éviter que l'hôte émettant les données ne le fasse trop vite et ne surcharge le récepteur. Ce mécanisme est effectué grâce à une fenêtre coulissante. Le contrôle de congestion permet d'éviter des chutes des effondrements de congestion qui ralentirait le réseau.

TCP se trouve directement au dessus d'Internet Protocol lorsque le datagramme possède le champ « Protocol » de son en-tête IP égal à 0x6. TCP est défini dans la RFC numéro 793.

[38] Les informations concernant le protocole TCP ont été receuillie directement à la source, la RFC numéro 793 : http://www.ietf.org/rfc/rfc0793.txt

7.9.1 Segments TCP

Les PDU TCP sont appelés des segments et consistent en une entête et une section de données. L'en-tête TCP consiste en dix champs obligatoires et des options facultatives.

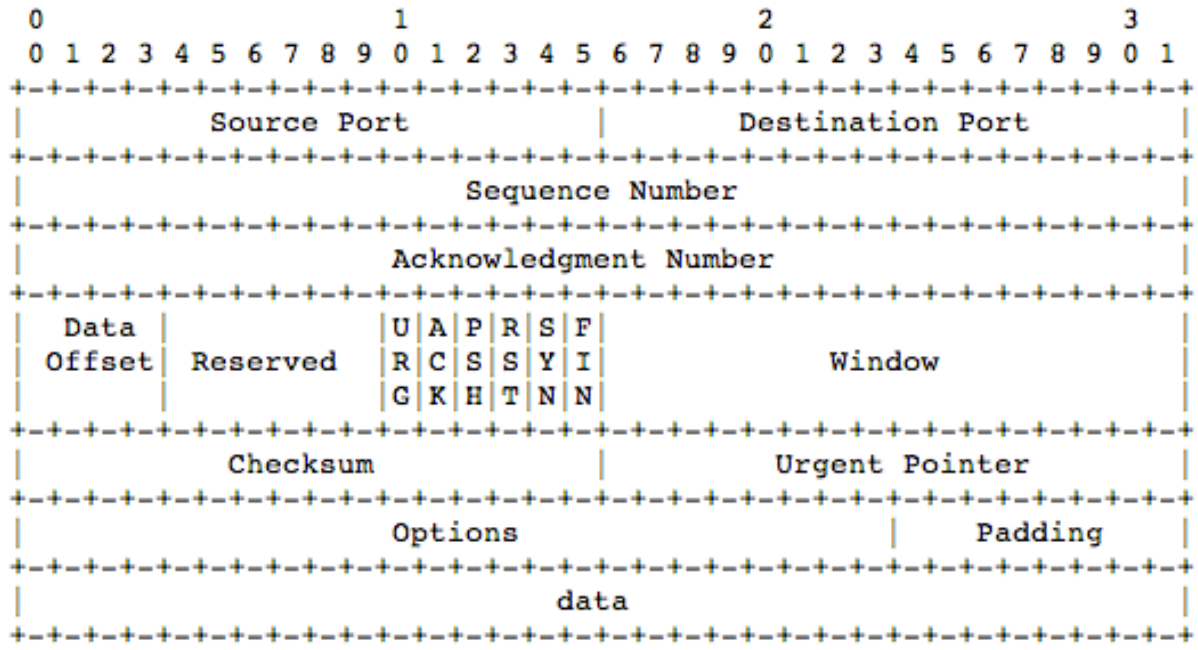

Figure 75 - En-tête TCP

Les adresses contenues dans les protocoles à partir de la couche transport du modèle OSI sont utilisées pour identifier des « hôtes » au sein d'une même machine. Dans le cas de TCP, on n'appelle plus cela des adresses mais des ports. Chaque port identifie une application (ou plusieurs) et permet de faire communiquer plusieurs applications sur une même interface d'une machine.

Transmission Control Protocol	
Champ	**Taille (octets)**
Source Port	**2 octets**

Numéro du port source envoyant le segment TCP.

Destination Port	**2 octets**

Numéro du port destination qui va recevoir le segment.

Sequence Number **4 octets**

Numéro de séquence du premier octet de données dans ce segment (à moins que le flag SYN ne soit utilisé). Si SYN est présent alors le numéro de séquence est le numéro de séquence initial (ISN) et le premier octet de données vaut ISN+1

Acknowledgment Number **4 octets**

Si le bit de contrôle ACK est fixé alors ce champ contient la valeur du prochain numéro de séquence que l'émetteur du segment s'attend à recevoir. Une fois que la connexion est établie, ceci est toujours envoyé.

Data offset **4 bits**

Le nombre de mots de 32 bits dans l'entête TCP. Ceci indique donc ou exactement les données commencent. La longueur d'une entête TCP sera toujours un multiple de 32 bits. Si ce n'est pas le cas, du padding sera ajouté afin de compléter l'en-tête.

Reserved **6 bits**

Cet espace est réservé pour des utilisations futures.

Control bits **6 bits**

Chaque bit de ce champ est très important et défini le rôle du segment TCP. Les flags sont décrits ci-dessous allant de gauche à droite[39] :

URG: Urgent Pointer field significant. Ce flag est utilisé pour informer la station réceptrice que certaines données dans le segment sont urgentes et devraient être prioritaire. Si ce flag est défini, le champ Urgent Pointer compte combien d'octets sont urgent à partir du premier byte.

ACK: Accusé de réception de données.

PSH: Pour comprendre le flag PSH, il est nécessaire de comprendre comment les buffers TCP marchent. TCP présente un simple socket aux couches supérieures sur lequel on peut lire ou écrire. Ce socket masque la complexité des communications par paquets. Afin d'autoriser les applications de lire ou écrire sur ce socket à n'importe quel moment, des buffers sont implémentés des deux côtés d'une connexion TCP, dans les deux sens. Ces buffers permettent de simplifier la gestion de taille des segments. Par contre, on ne veut pas attendre à chaque fois qu'un segment soit rempli au maximum pour être envoyé. C'est là que le flag PSH intervient. Il va permettre de "pousser" les données hors du buffer et d'envoyer le segment, au lieu d'attendre que plus de données n'arrivent. En résumé, l'application envoyant le segment va utiliser le flag PSH pour déclencher l'envoi des données et l'hôte

[39] Descriptions des flags expliquées http://packetlife.net/blog/2011/mar/2/tcp-flags-psh-and-urg/

recevant le segment sera informé que les données doivent être poussée à l'application destinataire.

RST: Annule la connexion après la réception d'une erreur.

SYN: Initie une connexion.

FIN: Ferme et termine une connexion.

Window **2 octets**

Ce champ est utilisé par la "fenêtre glissante" pour le contrôle de flux. Il s'agit du nombre d'octets de données que l'envoyeur est prêt à accepter.

Checksum **2 octets**

Le checksum TCP est le complément à un de la somme des tous les mots de 16 bits contenus dans l'entête et les données. Si un segment contient un nombre impair d'octets sur lesquels appliquer le checksum, un dernier octet égal à zéro est ajouté. Ce padding n'est par contre pas transmis mais ajouté par l'algorithme effectuant le checksum si besoin est.

Urgent Pointer **2 octets**

Ce champ communique l'offset par rapport au numéro de séquence dans le segment. Ce pointeur indique le début des données suivant les données urgentes. Ce champ doit être interprété uniquement si le flag URG est défini.

Options **Variable**

Les options peuvent apparaître ou pas dans un segment. Il est nécessaire que le champ option soit un multiple de 8 bits. Toutes les options sont inclues dans le checksum. Il existe deux possibilités concernant le format de chaque option.

1) Un seul octet représente le type de l'option
2) Un octet avec le type de l'option, un octet avec la longueur de l'option (*l*) et *l* octets de données.

La longueur de l'option compte le type ainsi que lui-même. Il faut donc soustraire 2 octets afin d'avoir la taille exacte des données de l'option.

Le tableau ci-dessous liste toutes les options IP possible avec leurs classe, leurs numéro, leurs longueur ainsi que leurs descriptions.

Type	Longueur	Description
0	-	**End of Option List**. Cette option ne compte qu'un octet et indique la fin des options pour indiquer qu'il n'y en a plus qui

		va suivre.
1	-	**No Operation**. Cette option ne fait absolument rien, il est utilisé entre des options afin d'éventuellement aligner des mots de 32 bits.
2	4	**Maximum Segment Size**. Si cette option est présente, elle indique la taille de segment maximale que la station concernée peut recevoir. Cette option peut uniquement être utilisée lors de la requête de connexion initiale (SYN). Si elle n'est pas utilisée, toutes les tailles de segment sont autorisées.

7.10 User Datagram Protocol

UDP, User Datagram Protocol, fait partie de la suite de protocoles IP en tant que protocole de la couche de transport du modèle OSI. Avec UDP, les applications peuvent envoyer des messages, appelés datagrammes, à d'autres hôtes sur un réseau IP sans avoir de communications ou de connexion établie au préalable. UDP n'utilise pas de handshake comme TCP et ne garantit aucune fiabilité afin d'avoir un overhead de taille minimal avant chaque datagramme. Il n'y a donc aucune garantie de livraison ou de doublons. UDP fournit simplement un checksum pour l'intégrité des données ainsi que des numéros de ports pour l'adressage de la source et de la destination du datagramme.

UDP est destiné à des usages ou la détection et correction d'erreurs n'est soit pas nécessaire ou faîte au niveau de l'application. Les services temps réels utilisent très souvent UDP car ils n'ont pas le temps d'effectuer des détections d'erreurs et retransmissions. UDP est standardisé et référencé par la RFC numéro 768.[40]

7.10.1 Datagrammes UDP

Le format des datagrammes UDP est très simple et léger puisqu'il ne comporte que huit octets et c'est un des avantages du protocole, n'ajoutant qu'un faible overhead avant les données afin de simplifier au plus les transmissions.

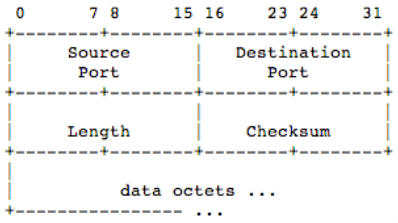

Figure 76 - Header UDP

[40] Le format des datagrammes ainsi que les informations sur UDP viennent de la RFC http://www.ietf.org/rfc/rfc768.txt

User Datagram Protocol	
Champ	**Taille (octets)**
Source Port	**2 octets**

Port source envoyant le datagramme. Ce champ est souvent sans aucune signification. Il est généralement utilisé comme port destination lors de l'envoi d'une réponse.

Destination Port **2 octets**

Numéro de port auquel est destiné le datagramme. Ce champ est important lors de l'écriture d'un parser car c'est sur celui-ci que l'on va se baser pour déterminer le protocole applicatif encapsulé dans UDP si le port est connu.

Length **2 octets**

Longueur en octets du datagramme incluant l'en-tête UDP ainsi que les données. La valeur minimale de ce champ est donc huit.

Checksum **2 octets**

Checksum classique sous la forme d'un complément à 1 sur l'en-tête et les données avec du padding si nécessaire pour rendre le datagramme multiple de deux octets.

8 Conception / Réalisation

8.1 Paradigme « Protocol and Delegate »

Lorsque l'on programme une application pour iOS, il est très fréquent d'utiliser le paradigme que j'appellerais *delegate*. Un delegate est en fait simplement une instance de classe qui est contenue dans une propriété d'une autre classe. Cette manipulation permet de pouvoir envoyer des messages à un objet afin de mettre à jour une table ou un champ texte, par exemple. En pratique, on utilise souvent un *protocole* pour représenter un delegate. Pour faire l'analogie avec Java, un protocole Objective-C est une sorte d'interface (Java), à la seule différence qu'il rajoute une option supplémentaire : il est possible de spécifier des méthodes optionnelles.

Ce modèle de conception est similaire au type « Observateur – Observé », qui permet d'avertir une vue lorsque son modèle, auquel la vue est « abonnée », se met à jour.

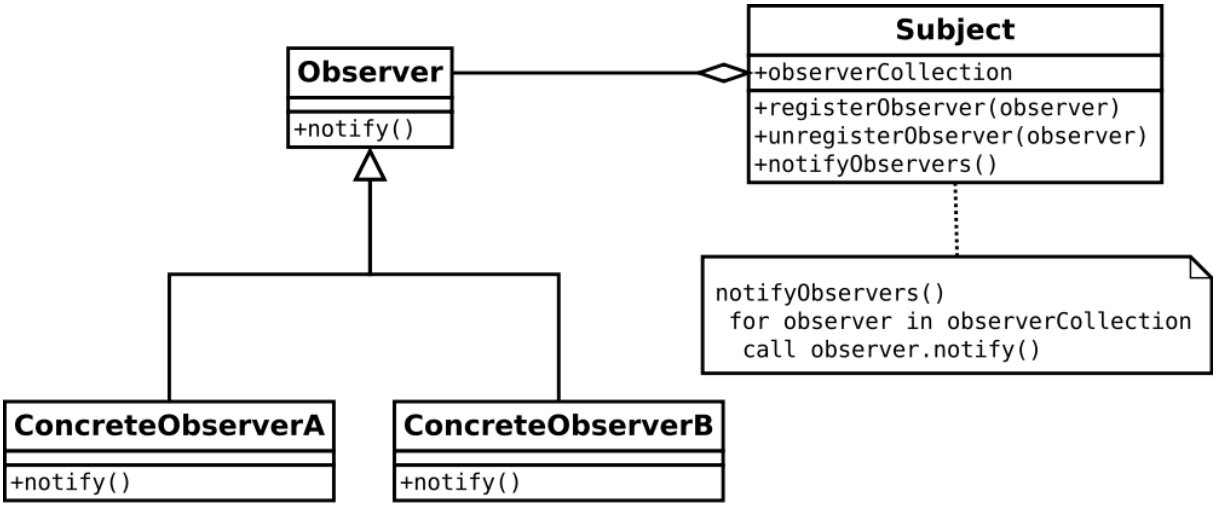

Figure 77 – Observateur - Sujet

http://fr.wikipedia.org/wiki/Observateur_(patron_de_conception)

La figure ci-dessus représente le design pattern Observateur – Observé et si on le compare avec l'utilisation faite avec iOS, la classe *Observer* et en fait notre protocole et le *Subject* aura une propriété de type Observer. Lorsqu'une modification est effectuée, la méthode *notifyObservers* va envoyer un message à toutes les propriétés delegate afin de les informer qu'une modification du modèle est survenue et qu'ils doivent agir en conséquence.

Il est important de comprendre le concept de delegate pour comprendre l'architecture de l'application qui utilise ce modèle à plusieurs points critiques, comme pour la lecture de flux ou pour le packet viewer.

8.2 Interface utilisateur

L'application développée dans le cadre de ce travail de Bachelor est conçue pour iPad. L'avantage principal qu'à un iPad par rapport à un iPhone, ou respectivement une tablette par rapport à un Smartphone, est la dimension de l'écran. C'est sur cette taille que l'on va baser la construction de l'apparence de l'application.[41]

	 iPad with Retina display	 iPad 2	 iPhone 5	 iPhone 4S
Taille *[mm]*	241.2 x 185.7	241.2 x 185.7	123.8 x 58.6	115.2 x 58.6
Résolution *[pixels x pixels]*	2048 x 1536	1024 x 768	1136 x 640	960 x 640

On voit sur la figure ci-dessus qu'un iPad est environ cinquante pourcent plus grand et trente pourcent plus large qu'un iPhone dernière génération. Cette différence de taille nous donne pas mal de place supplémentaire.

Afin de rendre l'application simple d'utilisation et intuitive, on se base sur la taille de l'écran dans le but de pouvoir afficher les informations demandées le plus clairement possible. Le but est de diviser l'écran de notre application en différents modules permettant une navigabilité intuitive.

Un modèle graphique d'application sous iOS très répandu actuellement est basé sur l'utilisation d'onglets et d'une barre de navigation. Regardons par exemple l'application native « Contacts » installée par défaut sur tous les mobiles tournant sous iOS.

[41] Détails sur le matériel récupéré sur www.apple.com/iphone et www.apple.com/ipad.

Figure 78 - Application "Contact"

La barre de navigation ainsi que la barre d'onglet nous font perdre chacune environ *1/12* de l'écran, soit au total *1/6* de la totalité l'écran à disposition.

L'idéal serait de trouver un moyen d'économiser cet *1/6* d'écran en ayant la même facilité de navigation dans l'application. La solution la plus élégante trouvée se fait grâce à un menu « caché » sous la page affichée à l'utilisateur. Ce menu sera accessible soit via un bouton ou encore en faisant glisser un doigt afin de « pousser » la vue principale. Un exemple de ce type de menu est présent plus tard dans ce chapitre du rapport.

8.3 Conception & Réalisation de l'interface utilisateur

Voyons tout d'abord l'organisation générale de l'application. A noter que l'application n'est disponible qu'en mode paysage. Après discussion avec Aginova, le but principal de ce projet est que l'application ainsi que les différents outils soient fonctionnels. Le design est secondaire et sera, hors projet, dessiné par un professionnel. Pour l'instant l'objectif cherché est simplement de faire une interface simple, navigable et intuitive.

Dans cette section, des ébauches représentant le design prévu de l'application conçues lors de la phase de conception sont présentées. Ces ébauches représentent ce qui avait été initialement prévu pour le design lors de la conception de l'application précédent la phase de développement. A chaque point, une

comparaison sera faite entre ce qui avait été initialement prévu et ce qui est réellement implémenté. Il sera intéressant d'observer les différences et les avantages des modifications effectuées lors de la phase de développement afin de justifier de tels changements.

La figure ci-dessous montre la manière dont les différents modules de l'application et le design général étaient planifiés.

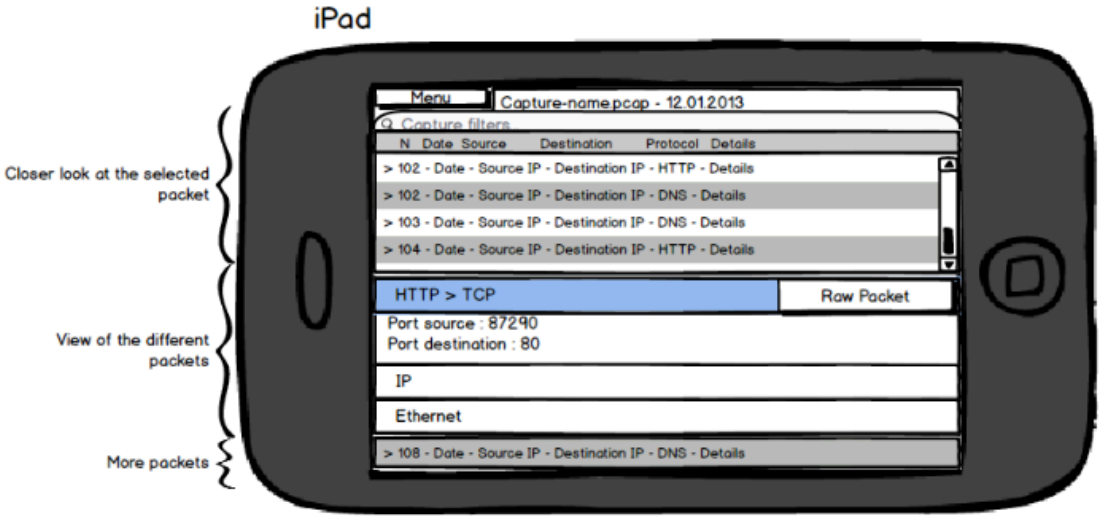

Figure 79 - Organisation des modules de l'application

Le bouton « Menu » en haut à gauche permet l'ouverture du menu qui se cache sous le packet viewer. Lors d'un click sur ce bouton, le menu va pousser les autres vues et s'afficher à partir de la gauche.

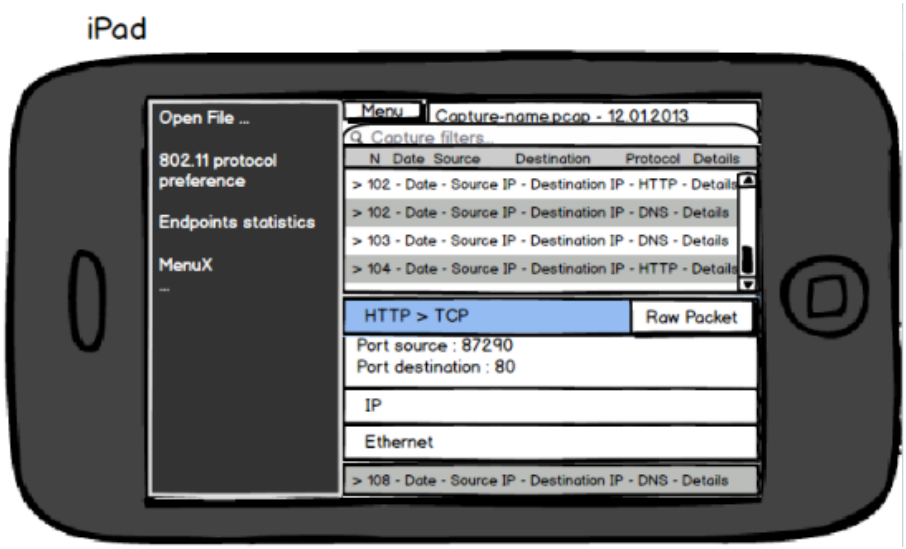

Figure 80 - Affichage du menu

Les figures suivantes sont des captures d'écrans de la dernière version de l'application développée montrant la même vue que celles ci-dessus.

Figure 81 - Organisation des modules de l'application après développement

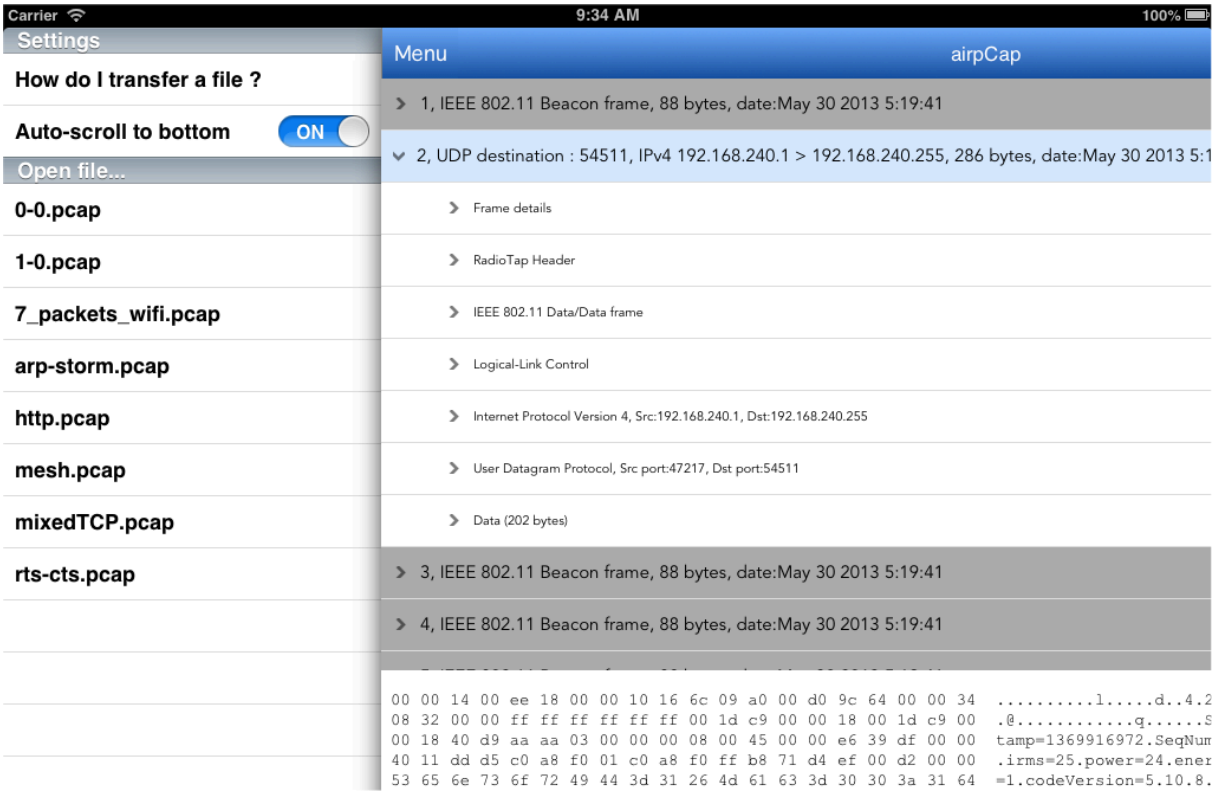

Figure 82 - Affichage du menu après développement

On voit qu'entre ce qui était initialement prévu et ce qui a finalement été implémenté, il y a eu des changements. Chaque module de l'interface utilisateur va maintenant être passé en revue et les différences seront expliquées et justifiées.

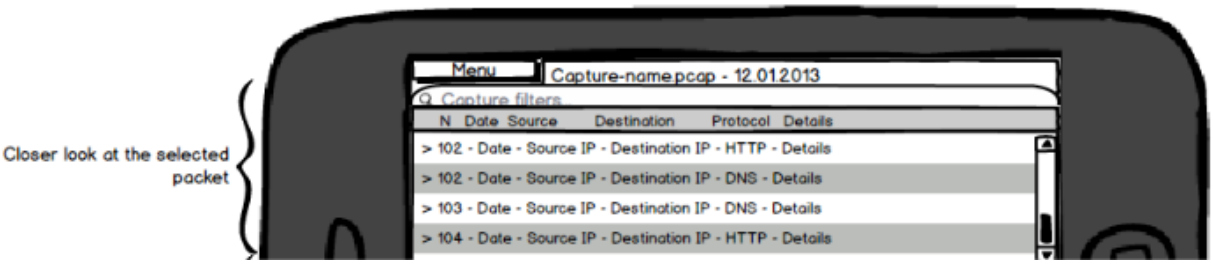

Figure 83 - Packet Viewer & Filtres

1. La section principale permet de visionner tous les paquets contenus dans la capture, c'est le « packet viewer ». Diverses informations importantes provenant de protocoles présents dans le paquet sont affichées. Les informations affichées peuvent évidemment différer suivant la capture. Il y a malgré tous certains champs récurrents qui se retrouveront obligatoirement affiché à l'écran quelque soit le contenu du paquet. Par exemple, la numérotation des paquets qui est importante. Bien que très simple, elle apporte une lisibilité implicite accentuée. Les adresses IP source et destination, si disponibles, les différents protocoles encapsulés dans le paquet ainsi que le timestamp formaté sont des informations utiles qui permettent d'avoir un coup d'œil rapide sur le contenu d'un paquet sans pour autant l'étendre.

   Sur la capture ci-dessus on aperçoit également la barre de filtrage permettant d'afficher uniquement certains paquets. La fonctionnalité de filtrage n'étant pas implémentée, cette barre n'a plus lieu d'être et a été supprimée, comme on le voit sur la capture ci-dessous. Au lieu de cela, l'application affiche des informations sur l'opération en cours dans une barre d'état. Le bouton « About » a également été ajouté par rapport à la conception initiale, permettant d'afficher des informations générales sur l'application ainsi que la capture en cours de lecture.

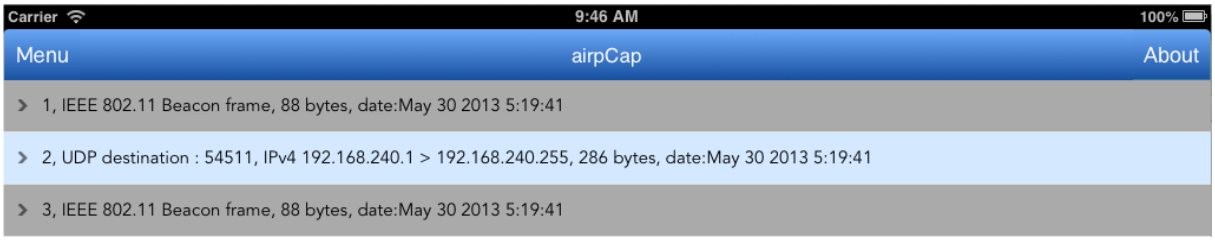

Figure 84 - Packet Viewer & barre d'état après développement

2. Toutes les entrées de la table contenant les paquets peuvent être étendues en cliquant dessus. Cela permet afin de voir tous les champs et les informations sur l'entrée sélectionnée.

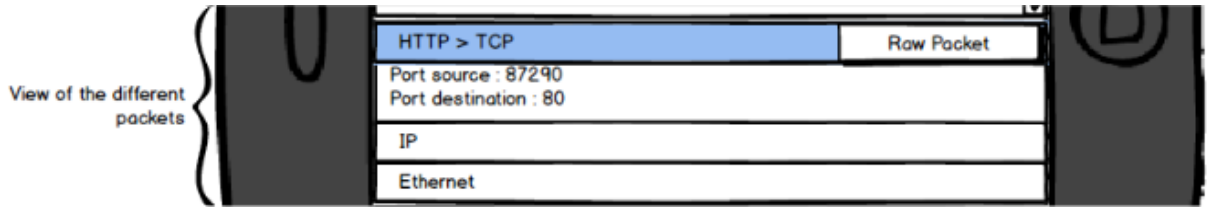

Figure 85 - Détails du paquet sélectionné

Les informations sont séparées par couche. En effet, chaque encapsulation de chaque protocole (protocole applicatif, couche réseau, couche liaison de données) ajoute un en-tête et éventuellement un en-queue. On aura donc les champs du même protocole réunis et séparés des champs des autres protocoles. Cette partie n'a pas changé par rapport à la conception initiale mise à part le fait que les couches sont affichées de la plus basse à la plus haute car cela semblait plus logique lorsque l'on lit un paquet. De plus certains protocoles verront leur cellule d'une couleur spécifique afin d'avoir une interface plus intuitive.

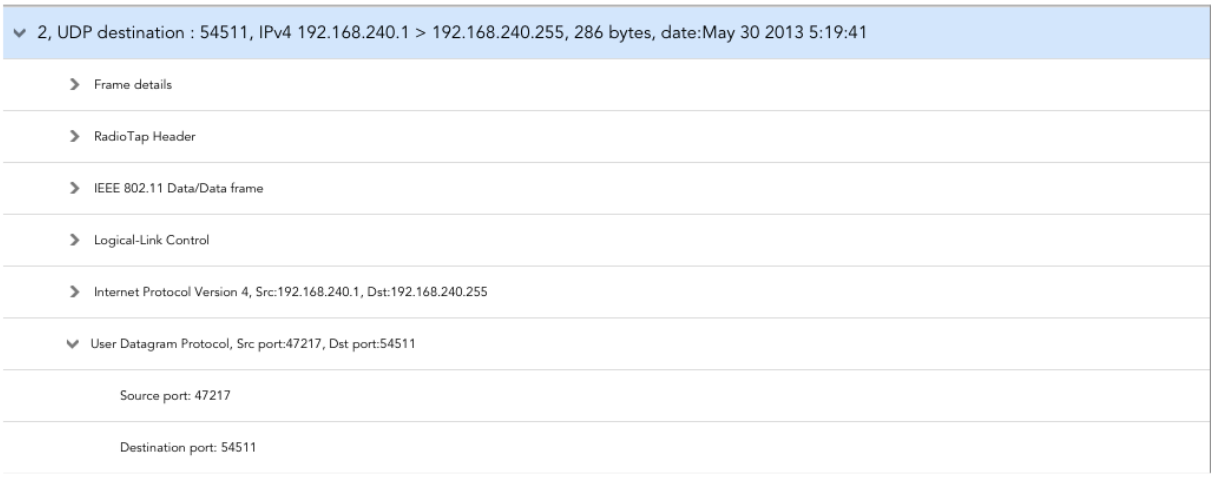

Figure 86 - Détails du paquet sélectionné après développement

3. Dans le cahier des charges, il est également spécifié d'avoir la possibilité d'afficher la représentation hexadécimale du paquet et cette fonctionnalité était initialement prévue par l'utilisation d'un bouton dédié.

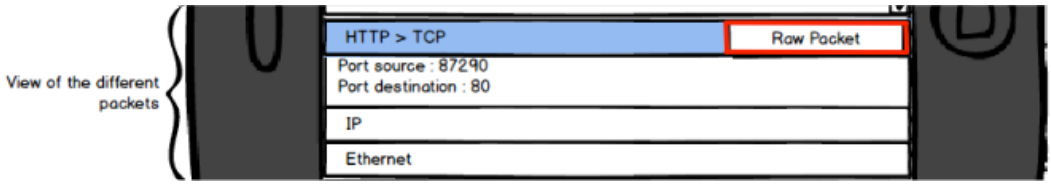

Figure 87 - Paquet brut

Lors de l'utilisation de ce bouton, il était prévu qu'une nouvelle vue s'affiche et que l'on voit les octets de chaque couche séparément. Ce module est celui qui a le plus changé par rapport à la planification initiale.

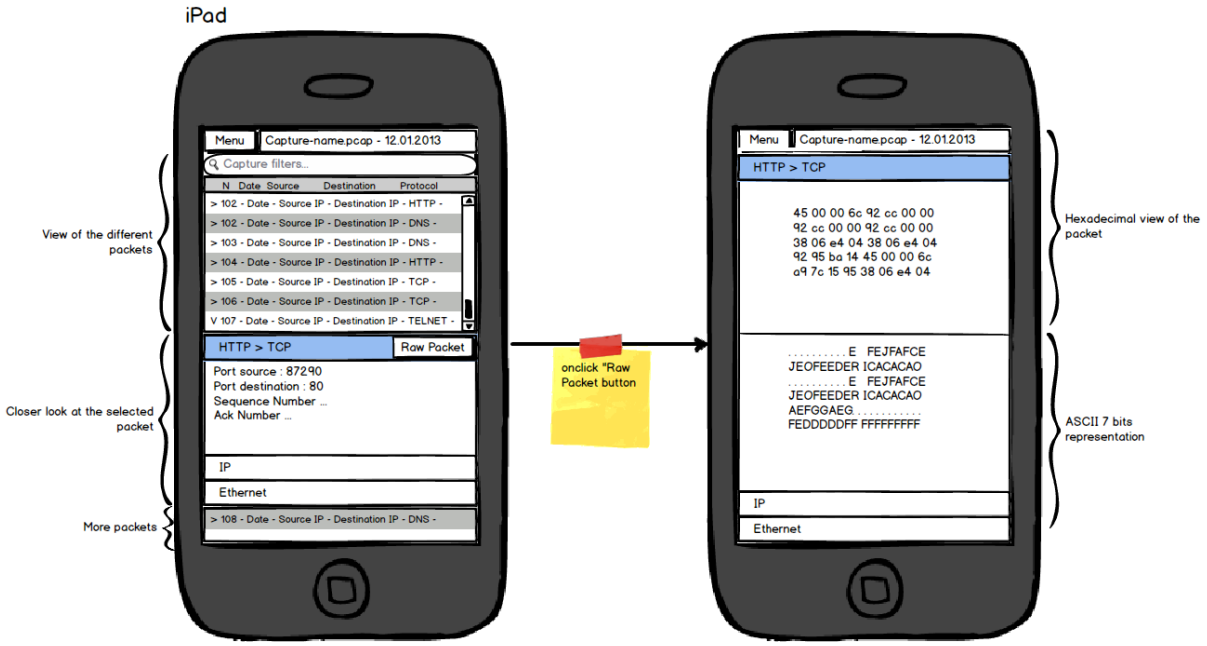

Figure 88 - Vue du paquet brut

Lors du début du développement, le packet viewer prenait initialement tout l'espace de l'application comme prévu dans la conception. Allouer autant d'espace au packet viewer s'est vite avéré ne pas être optimal, au contraire. En effet, l'interface donnait l'impression de ne pas utiliser la totalité de l'espace fournit par l'écran de la tablette. De plus lorsque l'on veut voir le contenu brut du paquet ou d'un champ il faut au moins faire deux clics : cliquer sur le bouton affichant le contenu brut et cliquer sur le bouton pour revenir au packet viewer. Dans une application mobile on va chercher à limiter le nombre d'actions qu'un utilisateur va prendre pour arriver à ses fins et les clics nécessaires pour afficher le contenu d'un paquet sont trop encombrants.

Pour ces deux raisons, l'écran a été divisé horizontalement afin d'ajouter la représentation hexadécimale et ascii directement sur la page principale. Ainsi, aucun clic n'est nécessaire pour voir les octets et on affiche plus d'informations sur la vue principale, supprimant la sensation de gaspillage d'espace affichable. Afin d'avoir une expérience utilisateur encore plus agréable et facile, cliquer sur un champ ou une couche va mettre en évidence directement les octets concernés dans les représentations ascii et hexadécimale du paquet.

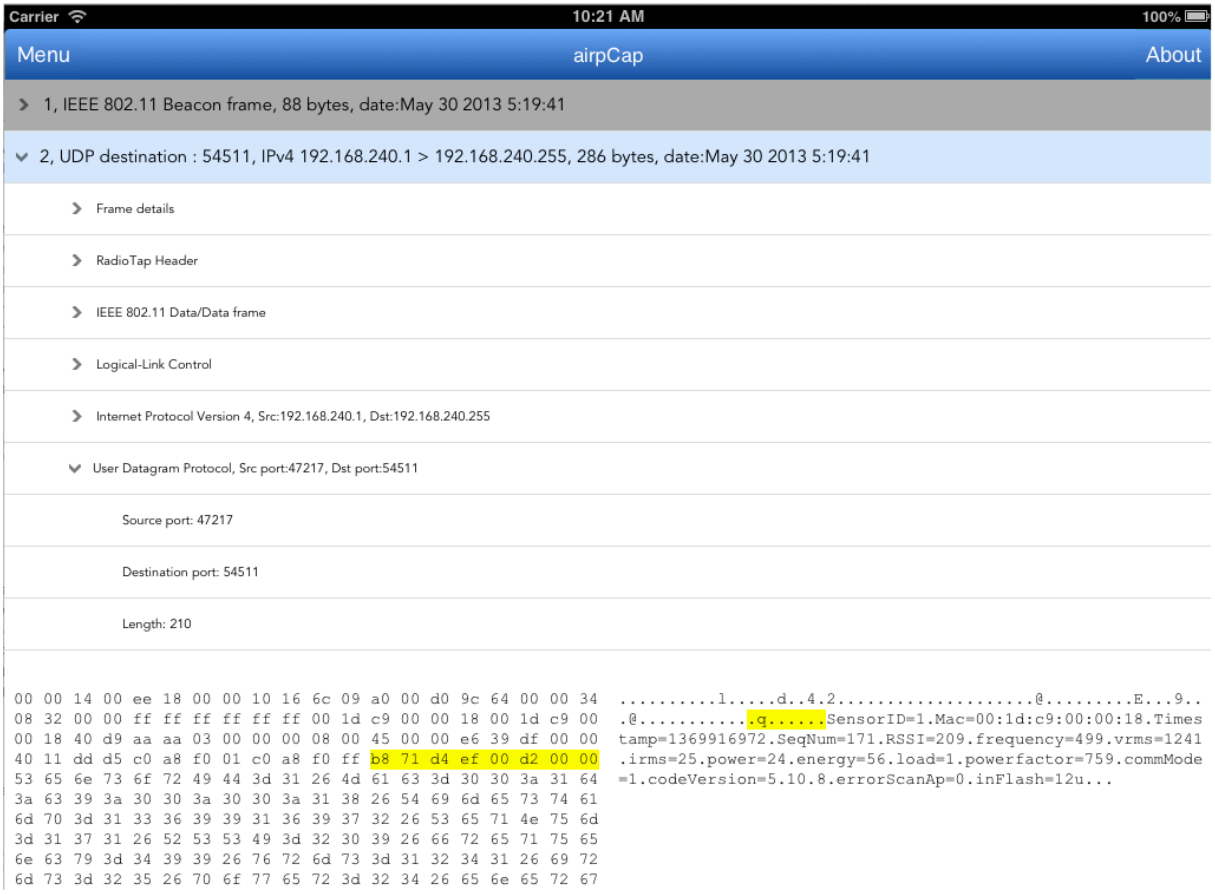

Figure 89 - Vue du paquet brut après développement

4. Le menu principal de l'application va permettre l'accès aux autres divers paramètres configurable par l'utilisateur. Lors de la phase de conception, la configuration des clés de déchiffrement pour les trames wifi ainsi que l'affichage des statistiques étaient deux fonctionnalités pouvant être utilisées à travers ce menu. Ces fonctionnalités n'ayant pas été implémentées, elles ont pour l'instant été retirées du menu. Le but principal du menu est de permettre l'ouverture des fichiers de capture pcap. Chaque fichier qui a été ajouté à l'application via iTunes sera affiché dans le menu et un simple clic sur le nom de ce fichier lance l'ouverture et le parsing de celui-ci.

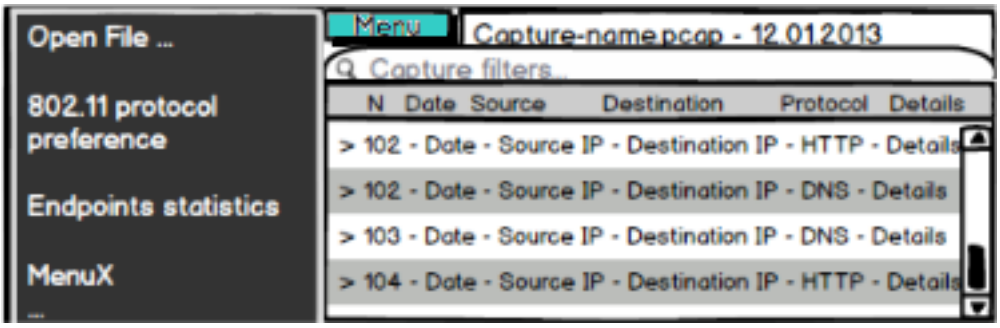

Figure 90 - Menu de navigation

Mis à part les deux fonctionnalités supprimées du menu, celui implémenté lors du développement est similaire à ce qui avait été planifié.

Figure 91 - Menu de navigation après développement

8.3.1 Menu de navigation

On a parlé précédemment de l'économie de place que l'on pouvait facilement réaliser grâce à une architecture en couche et un menu de navigation cacher sous la fenêtre principale. Voyons en image ce que ça donne.

Figure 92 - Menu en couche

Ce modèle d'application nous fait gagner beaucoup de place et est, à mon avis, plus esthétique. Le défaut d'implémenter un menu de ce type là est que, nativement, il n'existe pas dans le panel d'objets de développement d'Xcode. Beaucoup de personnes se sont déjà intéressées au problème et ont développés des librairies très complètes permettant d'avoir ce design.

Librairie	Avantages	Désavantages
JTRevealSidebar[42]	Support les storyboard et les segues, compatible avec ARC, très simple à utiliser.	Développée à la base pour iOS 5, aucune documentation sur le support des dernières versions. Support du changement d'orientation par assuré.
ECSliding[43]	Supporte iOS 5 et 6, iPad, compatible avec storyboard et segues et compatible avec ARC. Le changement d'orientation est supporté	
JWSlideMenu[44]	Supporte ARC et iOS 6.	Librairie très peu documentée et rarement mise à jour. Aucune information sur le changement d'orientation.

Mon choix s'est porté sur **ECSliding**, disponible sur le site[45] :

https://github.com/edgecase/ECSlidingViewController

Les caractéristiques de la librairie ECSliding qui ont influencé ce choix sont les suivantes :

- Possible de copier, utiliser et modifier le code librement (MIT Licence).
- iOS 5 ou plus – Un support des versions d'iOS les plus récentes est nécessaire
- Support iPad – Manifestement, le support pour iPad est indispensable.
- Support portrait et paysage – Bien que le support portrait et paysage ne soit pas un point essentiel pour ce projet, cette librairie est évolutive et gère parfaitement toutes les orientations pour les évolutions futures de notre application.

[42] https://github.com/mystcolor/JTRevealSidebarDemo
[43] https://github.com/edgecase/ECSlidingViewController/blob/master/README.md
[44] https://github.com/jeremieweldin/JWSlideMenu
[45] La document de ECSliding est disponible à l'adresses suivante :
https://github.com/edgecase/ECSlidingViewController/blob/master/README.md

La première chose à faire pour utiliser la librairie ECSliding est évidemment de l'ajouter au projet. Il est nécessaire d'ajouter les fichiers **ECSlidingViewController.h**, **ECSlidingViewController.m**, **UIImage+ImageWithUIView.h** et **UIImage+ImageWithUIView.h**.

Attention, il n'est pas nécessaire de télécharger et d'importer ces fichiers lors de l'importation du projet sur un nouvel ordinateur, cette manipulation n'a plus besoin d'être faite. Veuillez vous référez à l'annexe « Importer le projet » pour savoir comment installer l'application dans votre environnement de développement.

Afin de créer une application avec un menu d'utilisation basé sur ECSliding, il est nécessaire d'organiser l'application en diverses classes bien définies. La classe principale, s'occupant de gérer le storyboard et le lancement de l'application, doit hériter de *ECSlidingViewController*. Appelons la *MainViewController* pour l'exemple. C'est sur cette classe que l'application doit démarrer, car elle va définir quelle est la vue du dessus, celle qui est visible par l'utilisateur. La vue du dessus est donc celle contenant les informations et non pas le menu. On appelle les vues de la couche supérieure des *TopViewController*.

On voit ci-dessous une partie du code de la classe principale qui va instancier la TopViewController affichée à l'utilisateur au démarrage de l'application.

```
@implementation MainViewController

- (void)viewDidLoad
{
    [super viewDidLoad];

    UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Storyboard" bundle:nil];
    self.topViewController = [storyboard
        instantiateViewControllerWithIdentifier:@"FirstTop"];
}
```

Figure 93 - Classe principale

Chaque TopViewController qui sera affichée via le menu doit être une vue et donc une classe dérivée de n'importe quel sous-interface de *UIViewController*. Dans l'exemple ci-dessous, on utilise simplement une UIViewController. Dans les vues du dessus, il nous faut instancier la « *UnderLeftViewController* » qui n'est autre que le menu de navigation. Le nom est explicite, ce sera un composant se trouvant sous la vue affichée à l'utilisateur, donc caché la plupart du temps.

Afin d'avoir également la possibilité d'afficher le menu de manière intuitive, on va ajouter un gestionnaire de gestuels qui nous permettra de détecter quand l'utilisateur fait glisser son doigt de gauche à droite, ce qui provoquera l'affichage du menu.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    if (![self.slidingViewController.underLeftViewController isKindOfClass:
        [MenuViewController class]]) {
        self.slidingViewController.underLeftViewController  = [self.storyboard
            instantiateViewControllerWithIdentifier:@"Menu"];
    }

    [self.view addGestureRecognizer:self.slidingViewController.panGesture];
    [self.slidingViewController setAnchorRightRevealAmount:300.0f];

    self.view.layer.shadowOpacity = 0.75f;
    self.view.layer.shadowRadius = 10.0f;
    self.view.layer.shadowColor = [UIColor blackColor].CGColor;

}
```

Figure 94 - Instanciation du menu et gestionnaire de gestuels

La dernière étape pour les vues du dessus est de lier le bouton « Menu » à une méthode affichant le menu.

```
- (IBAction)showMenu:(id)sender {
    [self.slidingViewController anchorTopViewTo:ECRight];
}
```

Figure 95 - Méthode affichant le menu

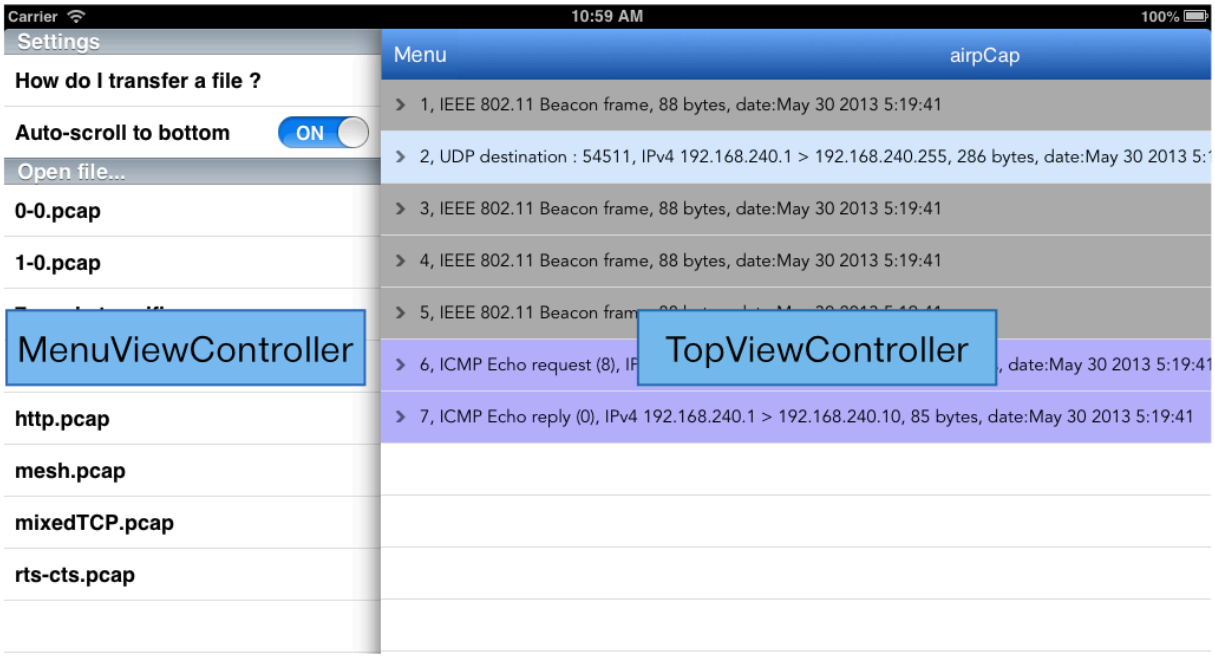

Figure 96 - MenuViewController

La vue du dessous, « MenuViewController » sur la figure ci-dessus, est unique, contrairement aux différentes « TopViewController » que l'on peut ajouter à besoin. C'est le menu nous permettant de naviguer à travers les différentes vues. On aura logiquement le même menu sur toutes les pages. Afin de pouvoir administrer un

menu dynamique (à cause des fichiers ajoutés ou supprimés), la meilleure solution est d'utiliser la classe *UITableView.*

Une instance d'UITableView est faite pour afficher et éditer une liste hiérarchique d'informations. Une UITableView est constituée d'une ou plusieurs sections, qui chacune sont constituées de lignes. Chaque ligne est une cellule qui doit être de type *UITableViewCell* ou une sous-classe de celle-ci. Lorsque l'on implémente une UITableView, il est indispensable de définir qui va jouer le rôle de *datasource* et de *delegate*.

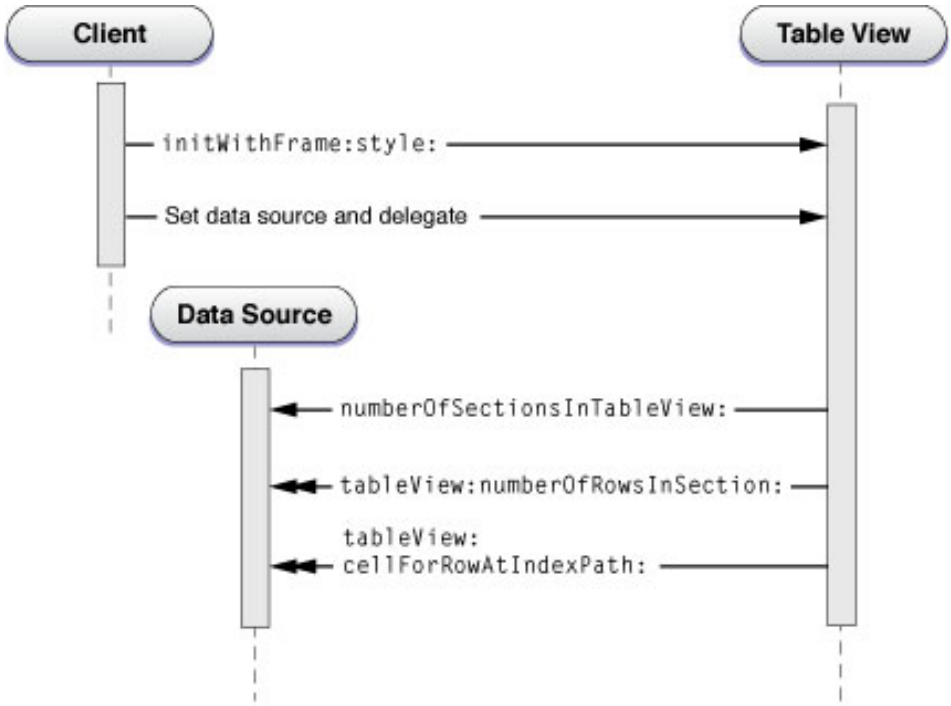

Figure 97 - Diagramme de création d'une UITableView[46]

L'objet assigné en tant que datasource de la table doit impérativement implémenter le protocole *UITableViewDataSource*. Ce protocole est, si l'on fait l'analogie avec le paradigme « Modèle Vue Contrôleur », le modèle. Le datasource doit informer la table du nombre de sections présentes, ceci grâce à la méthode :

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
```

Ainsi que le nombre de lignes, grâce à la méthode :

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

46 http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/tableview_iphone/CreateConfigureTableView/CreateConfigureTableView.html

Et surtout la cellule et son contenu pour chaque ligne. C'est la méthode ci-dessous qui retourne la cellule appropriée et son contenu pour chaque « indexpath » (combinaison section, ligne) :

- `(UITableViewCell *)tableView:(UITableView *)`*tableView* `cellForRowAtIndexPath:(NSIndexPath *)`*indexPath*

Ces méthodes sont automatiquement appelées par l'instance d'UITableView et doivent donc être implémentées. Le même système est utilisé pour l'objet delegate dont sa classe doit implémenter le protocole *UITableViewDelegate*. Ce protocole permet principalement la gestion des actions sur les cellules. Par exemple, la méthode suivante est appelée lorsque l'utilisateur clique sur une cellule :

- `(void)tableView:(UITableView *)`*tableView* `didSelectRowAtIndexPath:(NSIndexPath *)`*indexPath*

La classe UITableView est largement utilisée dans le monde iOS et également dans ce projet. En effet, pas seulement le menu est implémenté grâce à cette classe, mais également l'affichage des paquets de manière hiérarchique. Ce concept est ainsi très important. Le packet viewer est cependant une table un peu différente du menu et sera expliqué en détails dans la section « Packet Viewer » de ce chapitre.

8.4 Packet Viewer : Affichage des paquets

Le terme « packet viewer » utilisé dans ce rapport fait référence à la table permettant l'affichage des paquets lu dans un fichier pcap. C'est également le nom de la classe implémentant la fonctionnalité. Le challenge lors de la conception d'une table permettant d'afficher une grande quantité d'information que l'on va lire dans un fichier est de la rendre claire et facile d'utilisation. Pour atteindre ce but, il ne faut pas afficher la totalité des informations directement, mais les organiser et les classer.

Pour l'application de ce projet, l'organisation de la quantité d'information affichée dans le packet viewer est faite par couche selon le modèle OSI. On veut pouvoir grouper les champs d'un même protocole et grouper tous les protocoles d'un même paquet. Et pourquoi ne pas étendre ce concept à un niveau abstrait indéfini afin de pouvoir grouper n'importe quel groupe de champs à souhait.

Le schéma initial voulu, représenté sur la figure ci-dessous, est de grouper les champs tant qu'il y a plus d'une information contenue à la fois jusqu'à ce qu'on arrive à un champ atomique et ceci à un niveau sans limite.

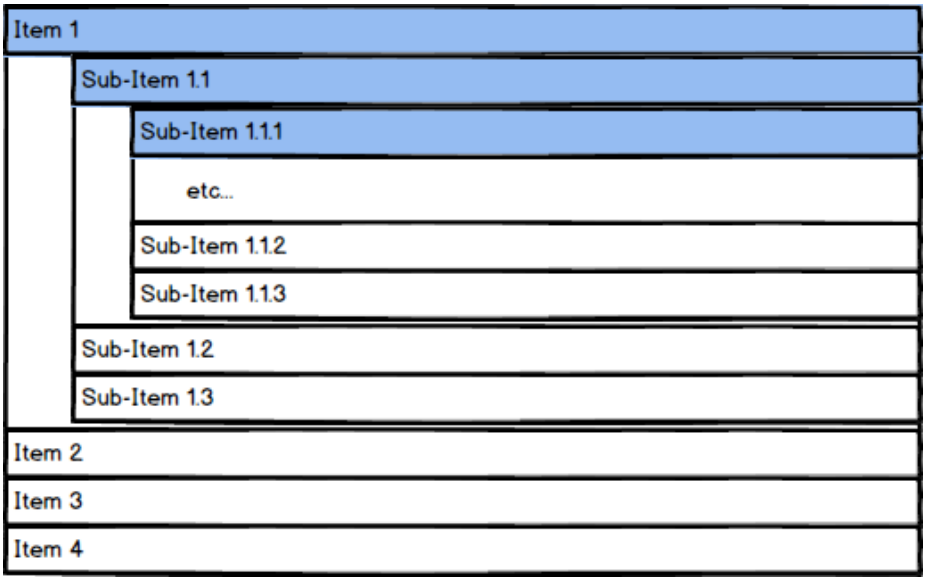

Figure 98 - Concept initial

Afin de pouvoir programmer un tableau abstrait de ce type, quelque soit le langage, il faut une structure de données dynamique. Le paradigme d'arbre est adapté et a été choisit afin de réaliser le packet viewer le plus dynamiquement et modulable possible. En informatique, les structures de données en arbre sont largement utilisées pour implémenter des structures hiérarchiques avec une racine, des sous arbres et des feuilles. Le concept d'arbre peut être défini de manière récursive en une collection de nœuds ou chaque nœud est lui-même une structure de données.

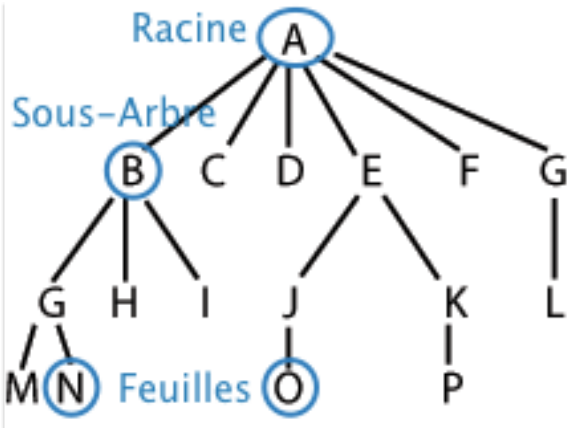

Figure 99 - Concept d'arbre

http://fr.wikipedia.org/wiki/Arbre_binaire

La racine est le nœud principal contenant tous les premiers sous arbres. Dès qu'un nœud est le dernier d'une branche et ne contient qu'une information, il est considéré comme une feuille. Le paradigme d'arbre s'adapte parfaitement à ce que l'on veut faire avec le packet viewer et l'interface utilisateur affichant le contenu des paquets. La racine est le fichier pcap avec comme sous arbres principaux chaque paquet

contenu dans la capture qui eux-mêmes vont comprendre un sous arbre par couche. Par exemple, un sous arbre 802.11, un sous arbre IP, un sous arbre TCP. Ce processus se répétera jusqu'à arriver à des champs atomiques, comme le numéro de port source ou destination dans TCP, qui représentent les feuilles de l'arbre. Si l'on dessine l'arbre d'une capture voilà ce que ça donne :

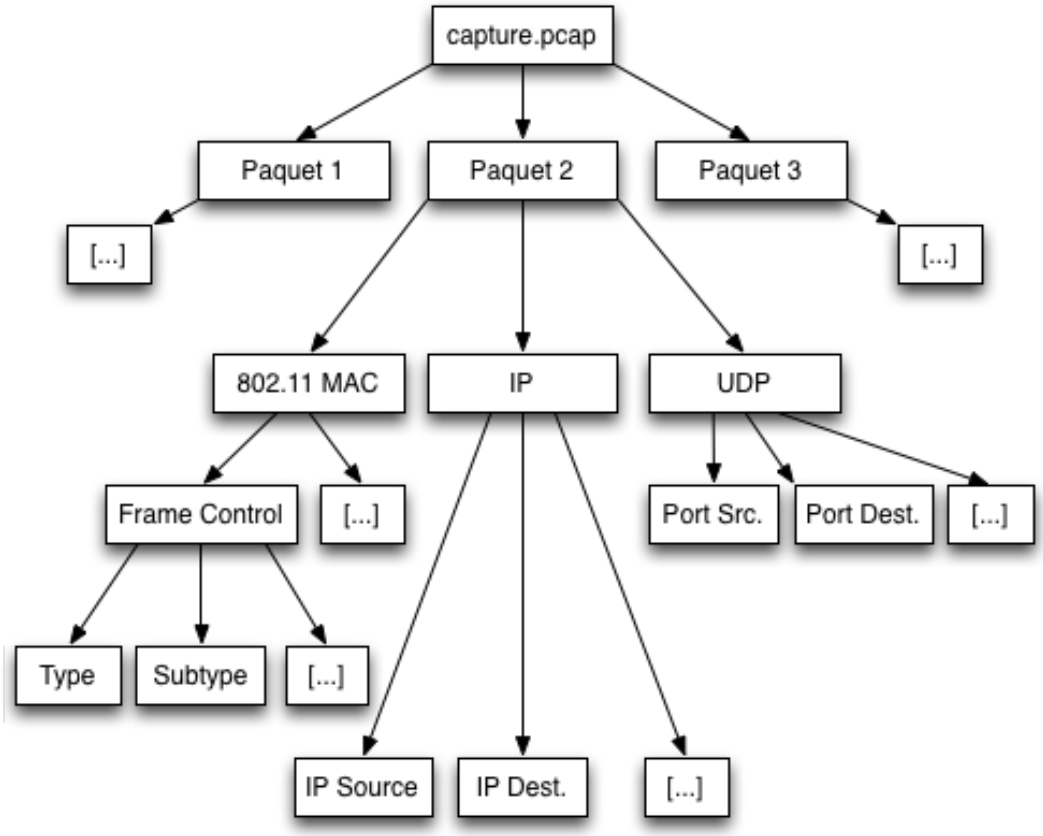

Figure 100 - Arbre d'un fichier pcap

Afin d'implémenter une structure de ce genre pour représenter graphiquement les paquets, deux points délicats sont à traiter. Comment insérer des paquets dans l'arbre et comment ensuite les afficher dans un tableau graphique en Objective-C. Avant de s'intéresser à ces deux problématiques, regardons d'abord comment la structure de donnée est organisée. Le tableau utilisé pour afficher les paquets est une UITableView.

La classe PacketViewer contient l'instance de UITableView et joue le rôle de delegate et de datasource. Le diagramme ci-dessous représente la classe PacketViewer simplifiée. Pour des questions de lisibilité, uniquement les propriétés, variables d'instances et méthodes liées à l'affichage des paquets sont référencées. Pour des informations sur les normes décrivant les classes dans les diagrammes de ce rapport, veuillez-vous référer à l'annexe « Normes et diagrammes ».

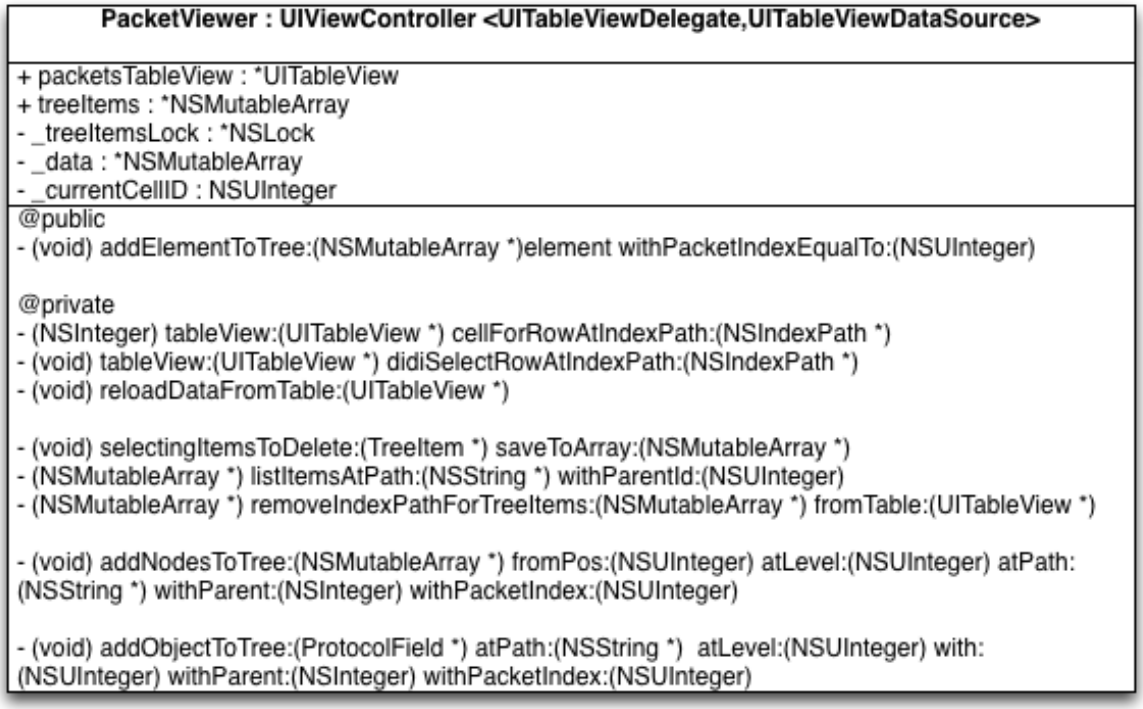

Figure 101 - UML PacketViewer simplifié

PacketViewer est le delegate et le datasource de la UITableView affichant les paquets. Les différentes méthodes nécessaires à la gestion d'une UITableView sont donc présentes dans PacketViewer :

cellForRowAtIndexPath, *didSelectRowAtIndexPath*, *numberOfSectionInTableView* et *numberOfCellsInSection*.

Les cellules affichées sont stockées dans la propriété treeItems. Il faut bien faire la distinction entre les cellules affichées et les cellules non affichées. Lorsque l'on montre un paquet dans le packet viewer, pas tous les champs ne sont visibles dès le début. Le principe est de cliquer sur une cellule afin d'étendre un groupe et de voir les champs contenus à l'intérieur. Lorsqu'un groupe n'est pas étendu, les champs contenus dans ce groupe ne sont pas affichés et donc pas alloués en mémoire dans le tableau de cellules treeItems. La variable d'instance _data contient absolument toutes les cellules, affichées ou pas, et l'algorithme de la méthode listItemsAtPath va faire la traduction entre _data et treeItems afin d'ajouter dans treeItems des nouvelles cellule lorsque l'utilisateur étend une cellule ou d'en supprimer lorsqu'il rétrécit une cellule. Le fonctionnement de listItemsAtPath sera expliqué plus tard dans ce chapitre, il est d'abord nécessaire de comprendre la structure des éléments de l'arbre.

La propriété packetsTableView est la table affichant les paquets et cette table est composée de cellules. Pour créer un arbre et l'afficher dans une UITableView, on ne

peut pas utiliser les UITableViewCell standard. Il faut en créer une sous-classe : TreeTableViewCell. Chaque cellule affichée dans le packet viewer sera donc une instance de TreeTableViewCell.

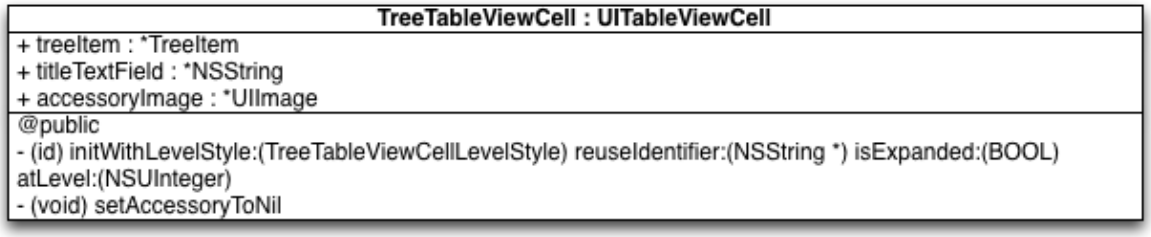

Figure 102 - Diagramme UML de TreeTableViewCell

Les objets de type TreeTableViewCell sont uniquement utilisés pour décrire les cellules ainsi que les éléments graphique des cellules. Chaque cellule sera indentée selon sa profondeur dans l'arbre. Si une cellule n'as pas atomique, c'est-à-dire qu'elle possède des branches, un indicateur de développement (accessoryImage) est affiché. L'indicateur va faire une rotation de 45 degré lorsque l'on développe la cellule puis revenir dans son état normal lors du regroupement des champs.

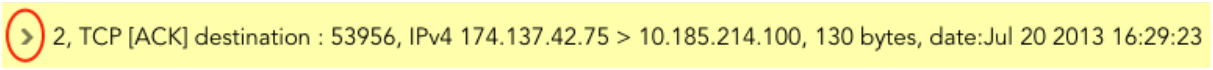

Figure 103 - Accessory image

La propriété la plus intéressante dans cette classe est treeItem, objet de la classe TreeItem. C'est le nœud de l'arbre qui va contenir non seulement les propriétés structurelles permettant la gestion de l'arbre ainsi que les informations concernant l'élément du paquet concerné. Pour résumer, les cellules TreeTableViewCell sont la vue graphique interfaçant les TreeItems dans le packet viewer.

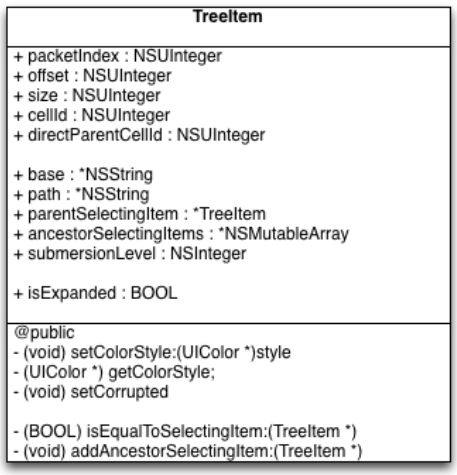

Figure 104 - Diagramme UML de TreeItem

Chaque nœud de l'arbre est de type TreeItem, qu'il s'agisse d'un champ de flags, d'un champ atomique, d'un paquet, d'un protocole, etc. Les nœuds sont référencés par un chemin similaire à celui qu'on trouve dans les systèmes UNIX avec comme racine « / ». Le chemin d'un nœud est constitué de la propriété « path / base ». Ce chemin permet de retrouver le nœud dans l'arbre. L'exemple ci-dessus montre le chemin qu'aurait le champ adresse ip destination d'un paquet.

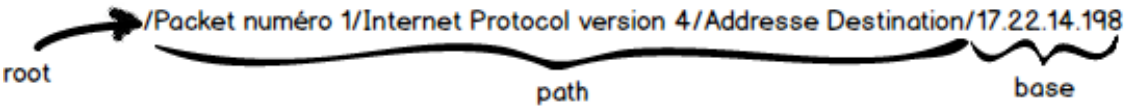

Figure 105 - Format du chemin des nœuds

Afin de distinguer les cellules qui pourraient avoir le même chemin, chacune disposent d'un ID unique générés grâce à la variable d'instance _currentCellID de PacketViewer. Ce cas est très rare, mais il peut arriver dans le cas ou un protocole permette l'utilisation du même champ plusieurs fois dans le même en-tête et que par pure hasard ces champs aient la même valeur. Chaque cellule connaît également l'index du paquet duquel elle vient dans le tableau contenant tous les paquets. Ce tableau est stocké dans l'instance de PcapParser, voir chapitre « Traitement de fichier iOS / Classe PcapParser ». La profondeur de la cellule permettant l'indentation une fois affichée dans le packet viewer est contenue dans la propriété submersionLevel.

Maintenant qu'on a vu le format des cellules et des nœuds, on peut s'intéresser aux problématiques d'implémenter une structure de données en arbre. La première est comment insérer des nœuds dans cette arbre. Un algorithme a été conçu afin d'ajouter des nœuds dans l'arbre automatiquement à partir d'un tableau formaté d'une manière bien précise. La méthode addElementToTree de PacketViewer est un raccourci pour ajouter un nœud directement à la racine de l'arbre. L'algorithme faisant tout le travail est contenu dans addNodesToTree (classe PacketViewer). C'est un algorithme récursif permettant d'insérer un nœud à n'importe quel niveau de l'arbre en lui passant un simple tableau de type NSMutableArray et en lui indiquant ou exactement dans l'arbre on veut l'insérer (position, niveau, chemin).

Le format du tableau contenant les éléments est bien spécifique et doit être respecté pour que l'algorithme d'insertion puisse fonctionner correctement. Ce format est le suivant :

NSMutableArray (ProtocolField , [NSMutableArray | ProtocolField, ...])

Chaque tableau, instance de NSMutableArray, y compris le premier tableau passé en paramètre à la méthode d'insertion de nœuds, doit contenir en tout cas un objet ProtocolField. Ce champ sera la ligne sur laquelle on pourra cliquer pour afficher les

champs contenus dans le tableau. Ensuite, on peut soit avoir à nouveau une table NSMutableArray comportant le même format ou alors un autre ProtocolField. La classe ProtocolField contient une propriété NSString contenant la description qui sera affichée à l'écran ainsi que d'autres propriétés permettant de gérer l'affichage brut des octets. Chaque fois que l'on a un objet de type ProtocolField, ce sera en fait une feuille dans l'arbre. Cette classe sera expliquée plus en détails dans le chapitre « Affichage brut des octets ».

L'algorithme d'insertion prenant en paramètre le format décrit ci-dessus est le suivant : (A noter que la méthode addObjectToTree appelée dans le pseudo-code ci-dessous est simplement une méthode instanciant un objet de type TreeItem et l'ajoutant au tableau de cellules.)

Algorithme d'insertion de nœud dans PacketViewer

```
//method adding nodes in the tree
define addNodesToTree:(1) fromPos:(2) atLevel:(3) atPath:(4)
withParent:(5) withPacketIndex:(6)
  param (1) nodes: NSMutableArray *
  param (2) pos: NSUInteger
  param (3) lvl: NSUInteger
  param (4) path: NSString *
  param (4) parentIndex: NSInteger
  param (5) pi: NSUInteger

BEGIN

//Basic case, the object at the current index is a string, we simply
//add it as an item
IF nodes[pos] isMemberOfClass ProtocolField THEN
  addObjectToTree : nodes[pos] atPath : path atLevel :lvl with :0
                    withParent :parentIndex witPacketIndex :pi

//Encapsulated case, it s an array, so a new level of indentation
ELSE IF nodes[pos] isMemberOfClass ProtocolField THEN
  //We add the first element, it must be a ProtocolField as specifie
  //by the format
  addObjectToTree : nodes[pos][0] atPath : path atLevel :lvl + 1
                    with :nodes[pos].count
                    withParent :parentIndex witPacketIndex :pi

  //We construct the new path for objects encapsulated in the array
  DECLARE newPath
  IF path = root («/») THEN
    newPath = path+nodes[pos][0]
```

```
  ELSE
    newPath = path+/+nodes[pos][0]
  END IF

  //Recursive call to compute the remaining of the array
  addNodesToTree :nodes[0] fromPos :1 atLvl :lvl+1 atPath :newPath
      withParent :_data.count - 1 withPacketIndex :pi
END IF

//if there is more data in the current array, we recursively call
//this method
IF pos + 1 < nodes.count
  addNodesToTree :nodes fromPos :pos+1 atLvl :lvl atPath :path
      withParent :_data.count - 1 withPacketIndex :pi
END IF
END
```

Cette algorithme et la clé de l'interface graphique implémentée dans l'application car il fait le lien entre les protocoles et l'affichage de ceux-ci à l'utilisateur. Il suffit que chaque protocole retourne un simple tableau au format spécifié ci-dessus en organisant de la manière voulue les champs, les éventuelles couches et groupes de champs, et l'application sera capable de l'afficher.

Maintenant qu'une méthode permet d'insérer correctement des nœuds dans l'arbre, il faut se pencher sur le problème de savoir quelles cellules afficher dans le packet viewer. Rappelons que toutes les cellules sont stockées dans la variable d'instance _data de PacketViewer. La table n'affiche pas tous les éléments de _data, mais uniquement les éléments qui doivent être montré à l'écran, ceux contenus dans la propriété treeItems. Lors de l'ouverture d'une capture, les seuls nœuds que l'on va afficher sont les éléments attachés à la racine, ceux décrivant les paquets. Pour afficher les éléments désirés, comme par exemple ceux uniquement à la racine, il suffit d'appeler listItemsAtPath avec le chemin des éléments recherchés puis d'enregistrer le résultat dans treeItems et de rafraîchir la table. C'est ce que fait l'application lorsqu'un fichier est pasé : Elle liste tous les éléments attaché à la racine, donc tous les paquets.

```
554     [_treeItemsLock lock];
555     [self setTreeItems:[self listItemsAtPath:@"/" withParentId:0]];
556     [_treeItemsLock unlock];
```

Figure 106 - Récupération des éléments à la racine

Après rafraîchissement, on obtient les éléments à la racine de l'arbre :

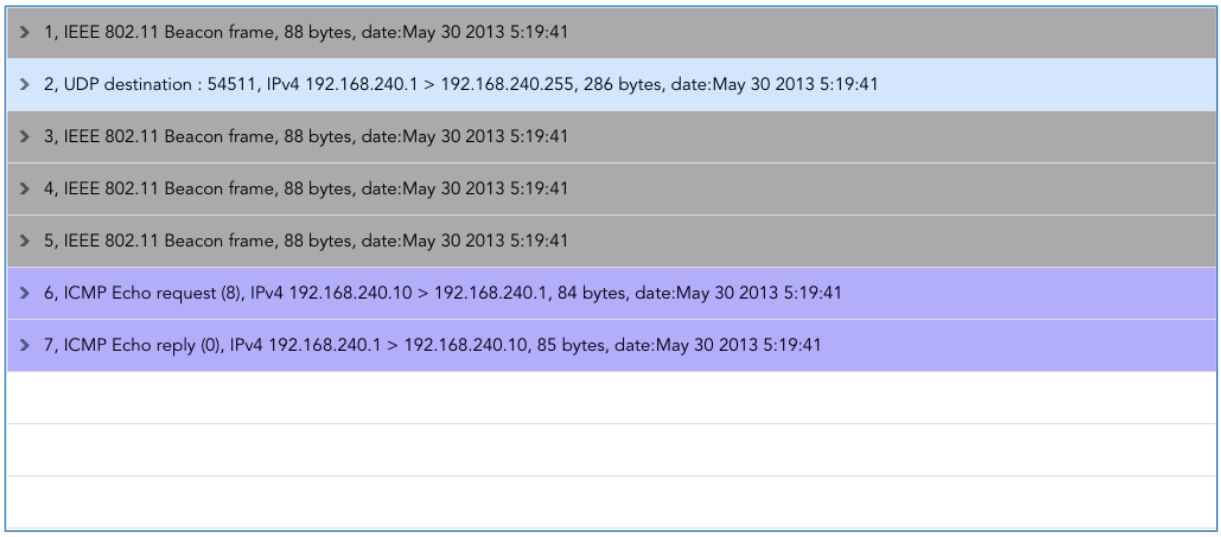

Figure 107 – Nœuds affichés

Sur la capture ci-dessus par exemple, uniquement sept nœuds sont affichés dans la table, alors que _data doit certainement en contenir plusieurs centaines ! Lorsque l'on clique sur un des paquets, le deux disons, on veut voir tout ce qui est compris dedans. Ces cellules doivent ainsi être ajoutées au tableau. Par contre, si on reclique une fois sur la ligne du paquet deux après, cette fois on veut que les lignes soient supprimées et que la ligne se contracte. C'est au moment du clic, dans la méthode didSelectRowAtIndexPath que l'on va effectuer ces deux traitements.

8.5 Système de fichier iOS et transfert de fichier

8.5.1 Système de fichier iOS

La fonction principale de l'application de ce travail de Bachelor est de pouvoir lire et interpréter des fichiers au format libpcap. Maintenant, il est primitif de savoir ouvrir et lire un fichier avant de pouvoir commencer à en interpréter le sens. Pour être capable d'ajouter des fichiers sous iOS, il est nécessaire de s'intéresser un peu plus en détail à la base du système de fichier.

Le système de fichier sous iOS est basé sur celui d'UNIX. Tous les disques attaché à l'appareil, qu'ils soient physiquement sur la machine ou accessible à travers le réseau, ajoutent de l'espace à une collection de fichiers unique. Le système de fichier utilise des dossiers afin de créer une organisation hiérarchique des fichiers.

Chaque application est censée être un « bon citoyen » et mettre les fichiers dans les endroits appropriés, c'est pour cela qu'avant de commencer à coder, il est nécessaire d'avoir une bonne compréhension du système de fichier utilisé par iOS.

Le système de fichier iOS est orienté vers chaque application tournant indépendamment. Apple interdit l'accès direct au système de fichier aux utilisateurs

d'iOS pour « garder le système simple » et les applications doivent également suivre cette convention.[47]

Chaque application est considérée comme une « île ». Les interactions entre l'application et le système de fichier sont limitées principalement aux répertoires à l'intérieur de la *sandbox* de l'application. Une sandbox est un espace réservé et isolé du reste du système de fichiers pour des raisons de sécurités.

Lors de l'installation d'une nouvelle application, le code installeur va créer un répertoire « home » pour l'application, il va placer l'App dans ce répertoire et il va créer différents autres répertoires clés. Ces dossiers constituent la vue primaire de l'App sur le système de fichier (voir figure ci-dessous).

A cause de sa présence dans une sandbox, l'application n'a généralement pas la possibilité de créer ou d'accéder à des fichiers en dehors de l'espace de son répertoire « home ». Une exception à cette règle existe lorsque l'application utilise des interfaces publiques afin d'accéder aux contacts ou à la musique de l'utilisateur.[48]

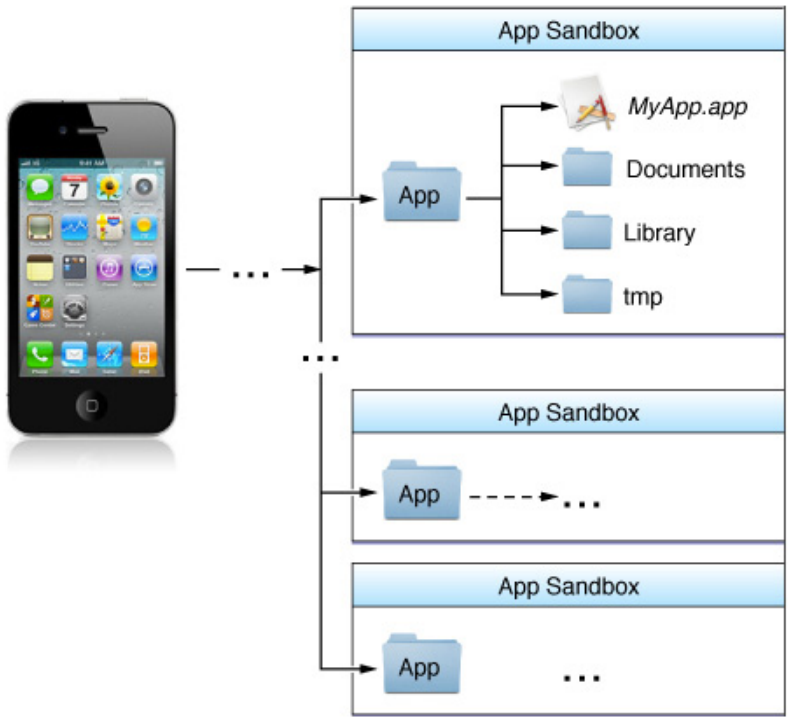

Figure 108 - île de l'application

[47] Citation d'Apple de l'article « File System Programming Guide » provenant de https://developer.apple.com.

[48] Toutes les informations concernat le système de fichier d'iOS ont été trouvées à l'adresses suivante : http://developer.apple.com/library/mac/#documentation/FileManagement/Conceptual/FileSystemProgrammingGUide/FileSystemOverview/FileSystemOverview.html

https://developer.apple.com

Pour des raisons de sécurités, l'application dispose d'un nombre d'endroits limité pour écrire des données. Le tableau ci-dessous résume « l'univers » de l'application duquel elle ne peut sortir.

Système de fichier de l'App

Répertoire	Description
<Application_Home>/Documents/	Ce répertoire doit être utilisé pour stocker les documents utilisateurs important et les données de l'application. Le contenu de ce répertoire peut être accédé par l'utilisateur.
<Application_Home>/Library/	Ce répertoire est utilisé pour les fichiers qui ne sont pas des données utilisateur. Le contenu de ce répertoire et sauvegardé par iTunes.
<Application_Home>/tmp/	Ce répertoire doit être utilisé pour les fichiers temporaires qui n'ont pas besoin de persister entre deux exécutions de l'application. L'App devrait supprimer le contenu de ce répertoire lorsqu'il n'est plus utile.

Les fichiers de capture sont placé dans le répertoire : *<Application_Home>/Documents/.* Lorsque l'on est en environnement de développement et que les tests se font encore sur le simulateur, les fichiers que l'on désire ajouter à l'application doivent être directement copiés dans le répértoire du simalteur sur le disque dur de l'ordinateur de développement. Ce répértoire est le suivant : */Users/<username>/Library/Application Support/iPhone Simulator/<ios version>/Applications/<Application_Home>/.*

Une fois au bout de ce chemin, on se trouve à la racine de la sandbox de l'application et il suffit de copier les fichiers voulus à l'endroit désiré. Lorsque le développement se fait directement sur l'appareil ou que l'application est en distribution, on a alors plusieurs solutions pour le transfert de fichiers à une application. La manière implémentée lors de ce projet est de transférer les captures directement à travers iTunes. Lorsque l'on branche son appareil à l'ordinateur, l'onglet « Apps » d'iTunes permet de transférer des fichiers, en cliquant sur l'application avec laquelle on souhaite partager des fichiers.

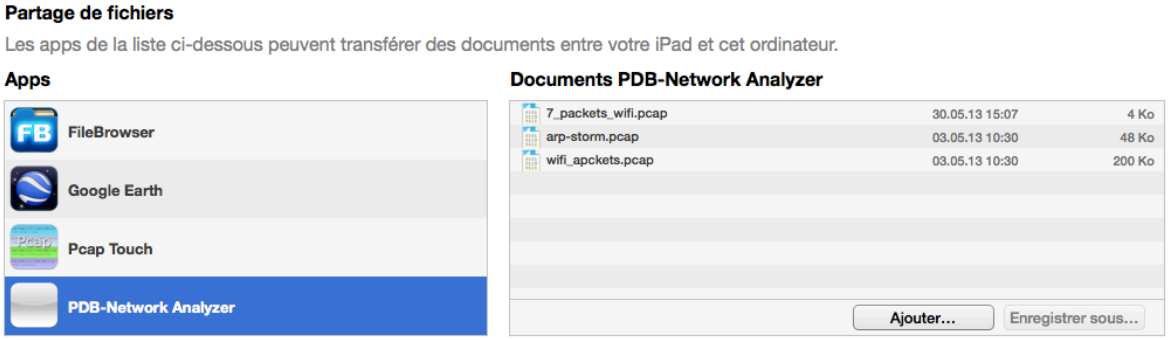

Figure 109 - Partage de fichiers iTunes

Pour que cela soit possible il faut avertir l'application que le partage de fichier est toléré en spécifiant la clé *UIFileSharingEnabled* à *YES* dans le fichier info.plist (voir figure ci-dessous.

Une fois que les fichiers sont transférés dans la sandbox de l'application et qu'ils sont ainsi disponibles à l'utilisation de celle-ci, c'est dans le code que le traitement va s'effectuer. Il y a trois moyens de référencer un fichier en Objective-C sous iOS :

- URL basé sur le chemin : file://localhost/Users/Steve/myFile.txt
- URL basé sur la référence du fichier : file:///.file/id=76383821.213312/
- URL basé sur le chemin en string : /Users/steve/myFile.txt

La classes NSURL permet la gestion des URL de manière standardisée par les RFC numéros 1808, 1738 et 2732.[49]

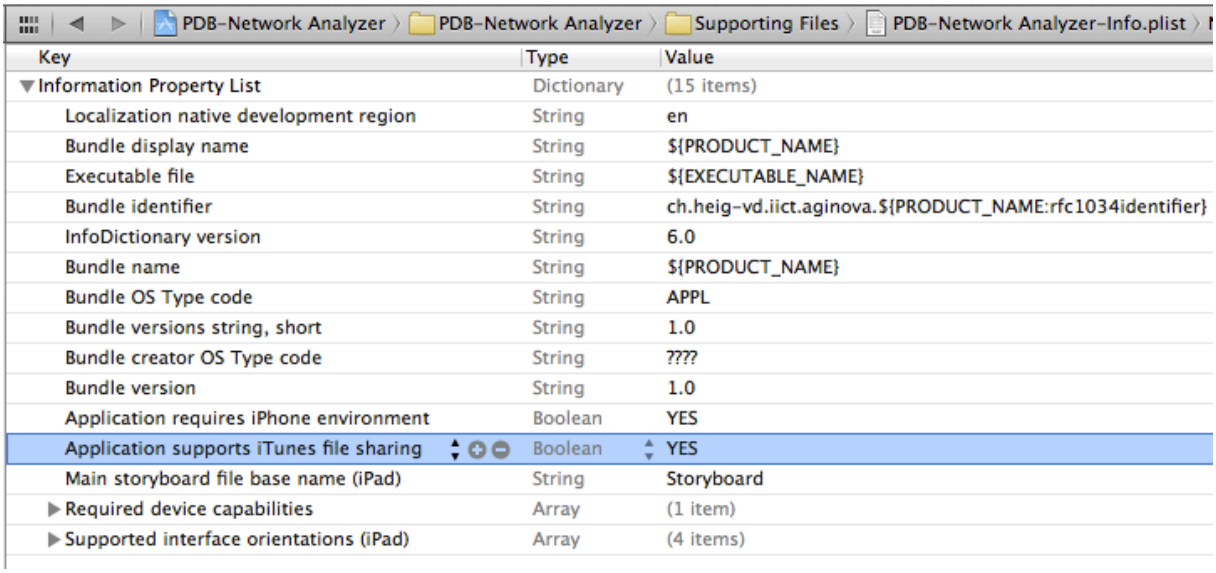

Figure 110 -info.plist

[49] http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSURL_Class/Reference/Reference.html

C'est la classe *InternalFileManager\InternalFileManager.h* qui s'occupe de localiser les fichiers et qui permet à l'utilisateur de les supprimer ou les ouvrir. La classe InternalFileManager va instancier un objet de type *NSFileManager* qui est la classe permettant d'effectuer de nombreuses opérations génériques sur le système de fichier. Une fois instanciée, on peut envoyer un message *contentsOfDirectoyAtURL* afin de récupérer les URLs de toutes les captures téléchargées sur l'appareil par l'utilisateur.

```
- (id) init
{
    self = [super init];
    if (self) {
        fileManager = [NSFileManager defaultManager];
        documentFolderURL = [[fileManager URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask] objectAtIndex:0];
    }
    return self;
}

- (NSArray *) getLocalFilesInformation
{
    NSArray *propertiesWanted = [[NSArray alloc] initWithObjects:NSURLNameKey, NSURLAttributeModificationDateKey, nil];

    return [fileManager contentsOfDirectoryAtURL:documentFolderURL includingPropertiesForKeys:propertiesWanted options:
        NSDirectoryEnumerationSkipsHiddenFiles error:nil];
}

- (BOOL) deleteFileAt:(NSURL *) url {
    return [fileManager removeItemAtURL:url error:nil];
}
```

Figure 111 - Utilisation de NSFileManager

C'est également l'objet *fileManager*, instance de NSFileManager, qui permet de supprimer des fichiers (voir figure ci-dessus) grâce à la méthode *removeAtURL* appelée *deleteFileAt*.

La classe *MenuViewController* qui est, rappelons-le, le menu caché sous la vue principale de l'interface utilisateur, possède comme propriété une instance d'InternalFileManager lui permettant de pouvoir ajouter dans le menu un lien pointant vers chaque fichier sous la forme d'un objet de type *NSURL* par fichier. Le contenu de la cellule est le nom du fichier extrait de l'objet NSURL. Lorsque l'utilisateur clique sur une des « cellule-fichier » dans le menu, on va instancier un objet de type *PacketViewer* qui est la vue principale à notre application et qui permet l'affichage du contenu des fichiers pcap. Lorsque l'objet PacketViewController est créé on lui donne le lien du fichier à ouvrir sous forme de l'objet NSURL.

```
- (void) openFileAt:(NSURL *)url {
    //Instanciation of the Packet Viewer
    UIViewController *newTopViewController = [self.storyboard instantiateViewControllerWithIdentifier:@"PacketViewer"];

    //we set the file being read parameter (it's the file URL) and the auto scroll parameter
    [(PacketViewer *)newTopViewController setFileBeingShown:url];
    [(PacketViewer *)newTopViewController setAutoScroll:[_autoScroll isOn]];

    //And we add the newly created packet viewer as the top view controller
    CGRect frame = self.slidingViewController.topViewController.view.frame;
    self.slidingViewController.topViewController = newTopViewController;
    self.slidingViewController.topViewController.view.frame = frame;
    [self.slidingViewController resetTopView];
}
```

Figure 112 - Envoi de l'URL à la vue principale

L'utilisateur a également la possibilité de supprimer un fichier de la liste en faisant glisser le doigt sur la ligne du menu visée.

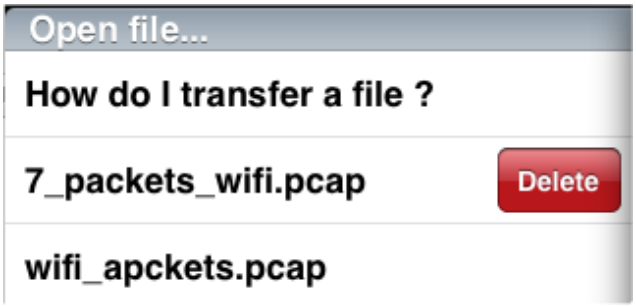

Figure 113 - Supprimer un fichier

Non seulement le fichier sera supprimé du système de fichier grâce au file manager de InternalFileManager, mais l'entrée dans le menu sera également directement retirée afin de ne pas tenter d'ouvrir un fichier qui n'existe plus. Cette gestuelle permettant d'afficher le bouton de suppression est rendu possible grâce à l'implémentation de la méthode *commitEditingStyle* qui va être appelée automatiquement par le datasource de la table UITableView. Cette méthode est également appelée lorsque l'on veut ajouter ou modifier une cellule, c'est pour cela que le paramètre *editingStyle* va nous indiquer si l'utilisateur veut supprimer une cellule, donc un fichier, en étant égal à *UITableViewCellEditingStyleDelete* (voir figure ci-dessous).

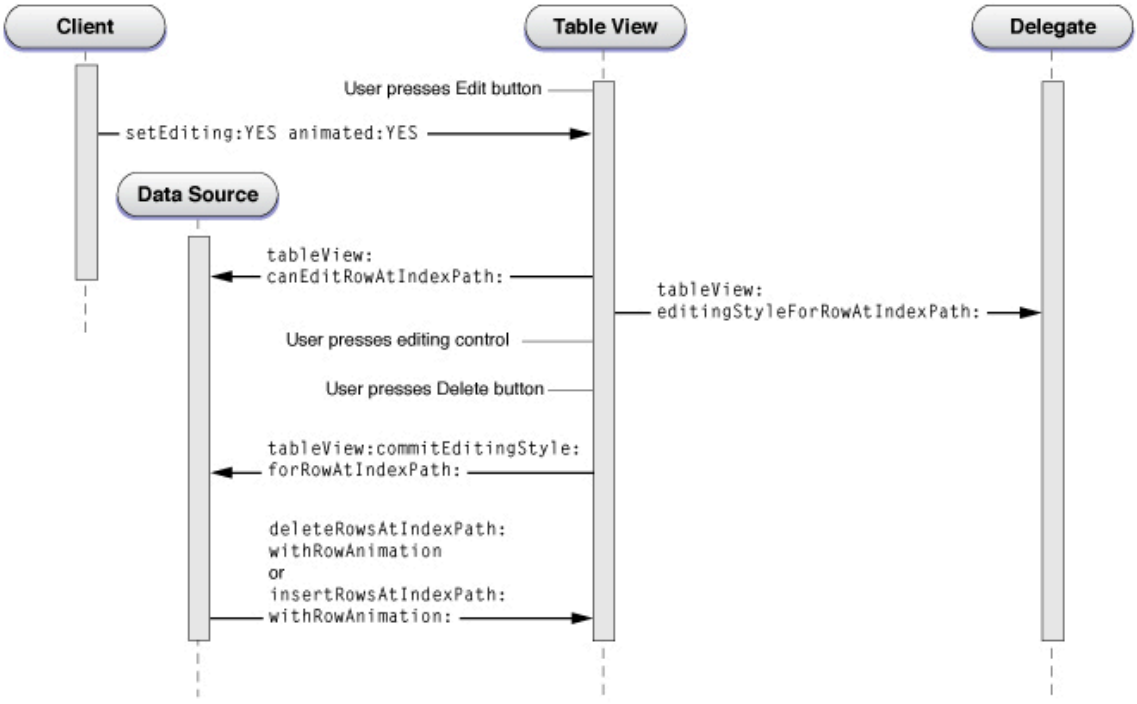

Figure 114 - Ajout ou Suppression de lignes [50]

[50] La documentation offcielle d'Apple fournit des explications détaillées du fonctionnement des UITableView

C'est dans cette méthode que l'on va ajouter ou supprimer des colonnes, suivant le paramètre editingStyle, comme on le voit dans le bout de code de la classe MenuViewController ci-dessous.

```
// Override to support editing the table view.
- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
   forRowAtIndexPath:(NSIndexPath *)indexPath
{
    //The user clicked the delete button
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        NSLog(@"Delete cell section %d row %d", indexPath.section, indexPath.row);

        NSObject *url = [[self.menuItems objectAtIndex:indexPath.section]  objectAtIndex: indexPath.row];
        if([url isMemberOfClass:[NSURL class]]){ //Delete the file
            if([self.internalFileManager deleteFileAt:(NSURL *)url]){
                //update array
                [[self.menuItems objectAtIndex:indexPath.section] removeObjectAtIndex:indexPath.row];

                //delete row
                [self.tableView beginUpdates];
                [self.tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:
            UITableViewRowAnimationAutomatic];
                [self.tableView endUpdates];
            }
        }
    }
}
```

Figure 115 - Suppression d'un élément du menu

8.6 Traitement de fichier au format libpcap

Cette section couvre le traitement des fichiers pcap au sens programmatique du problème. Le détail du format libpcap est expliqué dans le chapitre « Analyse » de ce rapport et ne sera pas réexpliqué dans ce chapitre.

La première option envisagée lors de la conception de ce projet a été d'utiliser un outil déjà existant. Libpcap (de TCPDUMP) par exemple, est une librairie C permettant le parsing de fichiers au format libpcap (la librairie possède le même nom que le format) ainsi que de nombreuses autres fonctionnalités, comme notamment du filtrage ou du sniffing directement sur une interface. Enormément de protocoles sont déjà implémentés dans libpcap et une bonne quantité de travail aurait pu être simplifié par l'utilisation de cette librairie. Le problème est qu'Apple interdit l'utilisation de cet outil dans toute application qui veut avoir une chance de finir sur l'Appstore.[51]

Pour cette raison, il est nécessaire de développer notre outil de traitement de fichiers pcap. C'est un bonne chose dans le sens ou il est possible de vraiment personnaliser le parseur afin d'être le plus efficace possible et de le concevoir spécialement pour cette application. Par contre, le grand désavantage est qu'il est nécessaire d'écrire le code interprétant chaque protocole. Ce travail est simplifié car libpcap est sous la

http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/tableview_iphone/ManageSelections/ManageSelections.html

[51] LGPL pose le même problème que lipbcap avec la stratégie d'Apple. http://loufranco.com/blog/lgpl-and-the-iphone. Les clauses d'acceptationt des applications sur l'Appstore sont disponibles à l'adresse suivante : https://developer.apple.com/appstore/guidelines.html.

licence BSD 3 clauses[52] et que le code peut être utilisé et modifié sous certaines conditions. Ainsi, les en-têtes des protocoles (fichier headers C) implémentés ont pu être la plupart du temps emprunté à libpcap.

8.6.1 Lecture synchrone et asynchrone

Un fichier de capture, une fois transféré sur l'appareil, peut alors être ouvert en le sélectionnant dans le menu de l'application. Plusieurs méthodes permettant d'ouvrir et de lire le contenu d'un fichier existent en Objective-C. Premièrement il faut choisir si l'on veut lire un fichier de manière synchrone ou asynchrone, les deux méthodes étant possible. Pour résumer, une lecture asynchrone du fichier signifie que les données peuvent être émises à n'importe quel moment. On peut ainsi se dire que pour lire des fichiers de captures, le plus simple et de le faire de manière synchrone.

Mais si l'on pense à l'évolution de l'application (voir chapitre perspectives), il est probable que celle-ci permette prochainement d'analyser un flux en cours de capture rendant une méthode synchrone inadaptée et un gros changement devrait être apporté à l'application plus tard. Par contre, si l'on utilise directement une lecture de fichier de type asynchrone, l'évolution future de l'application sera directement supportée et très peu de changement ne devront être apportés. C'est la lecture de type asynchrone qui a été choisie.

Maintenant plusieurs solutions asynchrones sont possibles. Ma décision s'est portée sur l'utilisation des flux grâce aux classes *NSStream* et *NSInputStream*. Les flux permettent la lecture de fichier grâce à une simple interface et peuvent typiquement gérer d'autres sources de données comme des sockets. C'est parfait pour l'évolutivité de l'application. [53]

Les objets de flux utilisent la boucle d'exécution du thread dans lequel ils sont enregistrés afin de planifier des opérations de lecture et d'écriture. Il est important de définir un delegate pour ce genre d'objets. Le flux d'entrée, NSInputStream, réveille la boucle d'exécution lorsque des données sont disponibles en notifiant le delegate. Sur l'extrait de code ci-dessous, on voit l'initialisation d'un objet NSInputStream en utilisant l'URL d'un fichier.

```
_fileStream = [NSInputStream inputStreamWithURL:url];
[_fileStream setDelegate:self];
[_fileStream scheduleInRunLoop:[NSRunLoop currentRunLoop] forMode:
 NSDefaultRunLoopMode];
```

Figure 116 - Mise en place d'un NSInputStream

[52] http://opensource.org/licenses/BSD-3-Clause
[53] http://developer.apple.com/library/ios/#documentation/FileManagement/Conceptual/FileSystemProgrammingGUide/TechniquesforReadingandWritingCustomFiles/TechniquesforReadingandWritingCustomFiles.html

8.6.2 Classe PcapParser

C'est la classe PcapParser du projet qui s'occupe de la lecture des fichiers en instanciant un objet de type NSInputStream grâce à l'URL transmise via la méthode d'initialisation de cette classe. Cette interface est une des interfaces les plus importantes de l'application car elle va permettre la lecture des fichiers ainsi que le parsing du contenu.

```
- (id) initWithURL:(NSURL *)url linkedTo:(PacketViewer *)UI;
```

La figure ci-dessous représente le diagramme de l'interface PcapParser mentionnée précédemment. La classe PcapParser va s'occuper de gérer le fichier dont on lui transmet l'URL grâce au constructeur

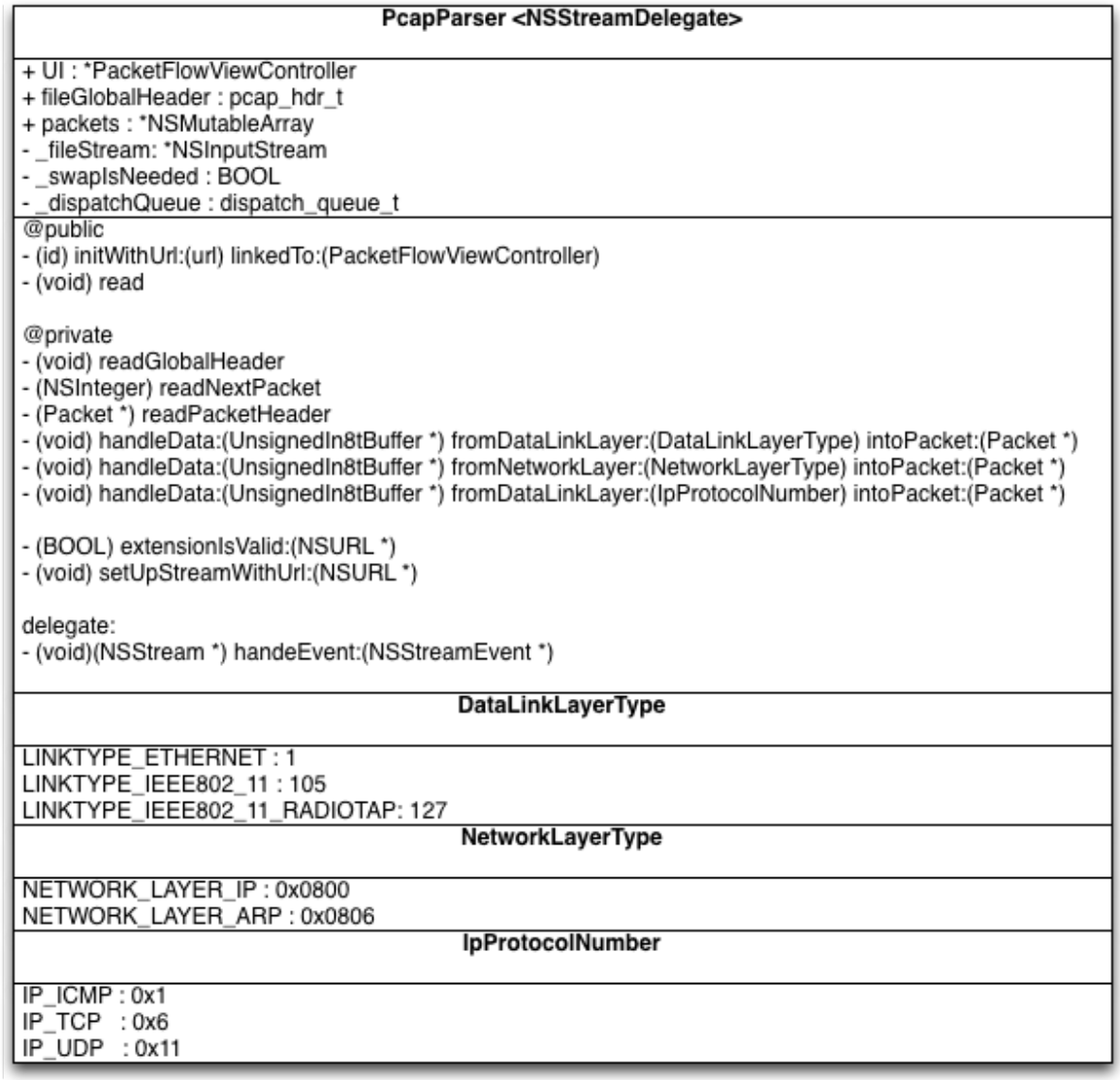

Figure 117 - Diagramme UML de PcapParser

PcapParser ne s'occupe pas seulement de lire le fichier mais c'est le pilier qui va créer les parseur du protocole approprié à chaque fois que c'est nécessaire. C'est pour cela que PcapParser contient plusieurs types énumérés avec des numéros de protocoles, les ethertypes, etc. Ces numéros sont passés aux méthodes *handleData* suivant la couche concernée. Plus d'informations sur le traitement des protocoles réseaux sont disponible dans la section « Traitement des protocoles réseaux » de ce chapitre.

Afin de pouvoir afficher le contenu du paquet, on va lier l'objet PcapParser à une instance de PacketViewer grâce à la propriété *UI*. C'est grâce à cette propriété que l'on pourra mettre à jour le packet viewer en y ajoutant les paquets lus à partir du fichier spécifié par *url*. Le constructeur va vérifier qu'il s'agît d'un fichier avec une extension pcap et va mettre en place le stream comme décrit ci-dessus.

Une fois l'objet PcapParser et le fichier prêt, il suffit d'envoyer le message *read* à PcapParser pour commencer le traitement du fichier. Read va simplement ouvrir le flux mis en place par le constructeur. Une fois ouvert, un flux NSInputStream va connaître jusqu'à six types d'évènements différents. Chaque fois qu'un événement se passe, la méthode *handleEvent* du delegate de ce flux est appelée.

Les évenements possibles sont les suivants :[54]

NSStreamEvent	Description
None	Aucun évènement n'est survenu.
OpenCompleted	Appelé lorsque le flux a été ouvert avec succès.
HasBytesAvailable	Le flux possède des octets à lire.
HasSpaceAvailable	Le flux possède de la place pour écrire des octets.
ErrorOccured	Une erreur est survenue.
EndEncountered	La fin du flux a été atteinte.

C'est dans handleEvent du delegate que l'on va traiter chacun des cas qui peuvent survenir. Le code de la méthode handleEvent de PcapParser est montré ci-dessous :

[54] Les évenements NSStreamEvent sont décrits à l'adresses suivante : http://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSStream_Class/Reference/Reference.html

```
- (void)stream:(NSStream *)stream handleEvent:(NSStreamEvent)eventCode {
    switch(eventCode)
    {
        case NSStreamEventOpenCompleted:
        {
            //We read the pcap file header
            [self readGlobalHeader];

            //We read all packets
            dispatch_async(_dispatchQueue,^{
                while ([self readNextPacket]);
            });
            break;
        }
        case NSStreamEventEndEncountered:
        {
            NSLog(@"End encountered !");
            dispatch_async(dispatch_get_main_queue(),^{
                [_UI changeInformationalStateTo:INFO_FILE_OPENED];
            });
            [_fileStream close];
            [_fileStream removeFromRunLoop:[NSRunLoop currentRunLoop]
                                   forMode:NSDefaultRunLoopMode];
            _fileStream = nil;
            break;
        }
        case NSStreamEventErrorOccurred:
        {
            NSError *theError = [stream streamError];
            NSLog(@"Error %i stream event occured. Domain : %@.", theError.code, theError.domain);
            [stream close];
            break;
        }
```

Figure 118 - handleEvent de PcapParser

Cette méthode va traiter particulièrement trois évènements. Lorsque le stream est correctement ouvert et qu'elle reçoit *NSStreamEventOpenCompleted*, la méthode va lire le fichier. Pour cela elle appelle *readGlobalHeader* qui va lire l'en-tête générale pcap du fichier. Ensuite on va lire tous les paquets présents dans le fichier. Plus d'informations concernant readNextPacket et readGlobalHeader seront données dans le chapitre « Traitement des protocoles réseaux ».

8.7 Architecture multi-threadée

Il y a un bout de code très intéressant dans le fragment de handleEvent montré dans la section précédente:

```
//We read all packets
dispatch_async(_dispatchQueue,^{
    while ([self readNextPacket]);
});
```

Figure 119 - General Central Dispatcher

Nous reviendrons sur le snippet de code de la figure ci-dessus, mais d'abord il faut introduire les possibilités d'exécuter du code en parallèle avec iOS et pourquoi le faire dans ce projet. Depuis que les fabricants de processeurs sont passés à des designs multi-cœurs, les développeurs ont la possibilité d'exécuter des tâches en parallèle permettant d'atteindre des performances de plus haut niveau. Le premier

iPad possédait déjà deux cœurs et le dernier sorti en date, l'iPad avec écran rétina, possède un chip A6X quadricœurs.[55]

Ce qu'il faut savoir concernant la programmation sous iOS est que le système prend automatiquement avantage de ces multi-cœurs. Le thread principal est dédié à l'affichage de l'interface utilisateur et au code écrit par le développeur.[56] Si l'on a beaucoup de calcul ou de traitement de données à faire, tout exécuter sur le thread principal va ralentir ou même bloquer l'interface utilisateur. On pourrait se dire que l'application développée dans le cadre de se projet ne nécessite pas de parallélisation étant donné que l'on va lire un fichier et que l'utilisateur doit attendre la fin du traitement du fichier pour interagir avec son contenu. Cependant si l'on pense à l'évolution future de l'application avec la lecture de paquets en temps réel, on ne veut pas bloquer l'interface utilisateur durant le sniffing. C'est pour cette raison qu'il est nécessaire de multi-threader l'application.

Une fois que le fichier est lié à l'objet PcapParser et que le stream est ouvert, on va paralléliser le code traitant les paquets afin de laisser le main thread, s'occupant de l'interface graphique, libre. Afin de mieux comprendre le fonctionnement de la classe PcapParser et de la parallélisation du traitement des paquets, la figure suivante est un diagramme de séquence simplifié de PcapParser. L'opération en cyan est faite hors du thread principal.

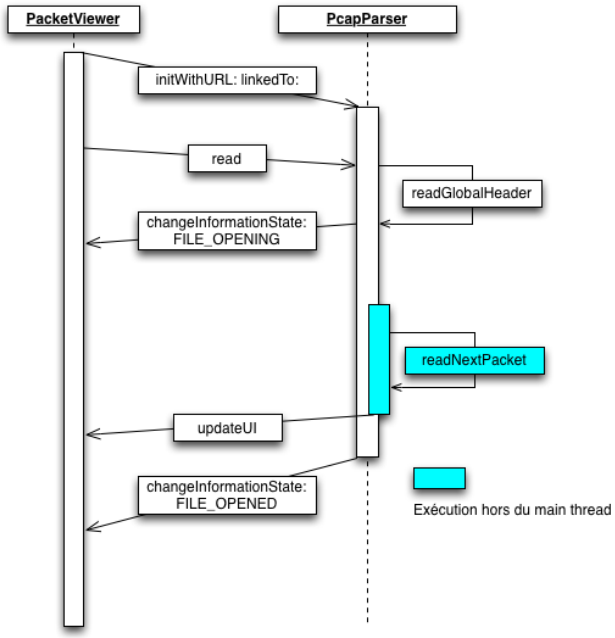

Figure 120 – PcapParser

[55] Caractéristiques techniques des iPad : http://www.apple.com/chfr/ipad/compare/
[56] Threading sous iOS : http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/Multithreading/AboutThreads/AboutThreads.html#

Bien que les apparences suggèrent que seul un tout petit bout du code est exécuté hors du thread principal, la méthode readNextPacket va prendre plus de 90% du temps d'exécution du code entier de l'application car elle s'occupe de lire et de traiter le moindre octet présent dans une capture pcap. Plus d'informations sur cette méthode sont disponible dans la section « Traitement des protocoles réseaux ».

La plupart du temps, on va utiliser des threads pour l'exécution de tâches en parallèle, mais avec iOS il existe plusieurs alternatives aux threads. Créer ses propre threads insère une certaine incertitude au code de l'application et peut devenir un problème. Les threads sont relativement bas niveau et il est nécessaire de comprendre totalement le fonctionnement de votre architecture afin de ne pas rencontrer des problèmes de synchronisation ou de timings.

La table ci-dessous résume les différentes alternatives aux threads sous iOS qui pourraient être utilisé dans le cadre de ce projet:

Technologie	Description
Cocoa threads	Implémentation des threads classique par Cocoa grâce à la classe NSThread.
POSIX threads	Les threads POSIX fournissent une interface basée en C pour la gestion des tâches. Ceci est le meilleur choix lors de code non Cocoa.
Asynchronous functions	Le système inclut des fonctions asynchrones qui gèrent la concurrence automatiquement. Ces API utilisent des processus ou des threads personnalisé pour exécuter des tâches et renvoient le résultat.
Separate processes	Plus lourd que des threads, créer un processus peut être utile dans le cas ou la tâche demande une large quantité de mémoire ou alors qu'elle a besoin des privilèges administrateur.
Grand Central Dispatch	GCD, Grand Central Dispatch, est une des meilleures alternatives aux threads qui nous laisse nous concentrer sur la tâche que l'on veut exécuter et non pas sur la gestion des threads.

Le fonctionnement du Grand Central Dispatch est le plus approprié pour ce projet. En effet, on ne souhaite que paralléliser le traitement de données éventuellement en grande quantité avec seulement un thread externe au thread principal. Il n'est pas nécessaire de pouvoir administrer les threads, c'est donc l'option idéale.

Avec GCD, on définit une tâche que l'on désire exécuter, la lecture du fichier dans ce cas-ci, et on l'ajoute à une file d'attente de travail qui va s'occuper de planifier l'exécution de la tâche dans un thread approprié. Les files de travail prennent en

compte le nombre de cœurs CPU à disposition et la charge actuelle afin d'exécuter le code de manière la plus efficace possible. Lors de la création de l'instance de PcapParser, une file de travail est crée et cette file va être appelée au moment de l'appel de la boucle lisant les paquet grâce à readNextPacket.

```
55 - (id) initWithURL:(NSURL *)url linkedTo:(PacketViewer *)UI
56 {
57     self = [super init];
58     if(self) {
59         if([self extensionIsValid:url])
60             [self setUpStreamWithUrl:url];
61         else
62             NSLog(@"File extension error - throw exception");
63
64         _dispatchQueue = dispatch_get_global_queue(0,0);
65         _UI = UI;
66         _packets = [NSMutableArray new];
67     }
68
69     return self;
70 }
```

Figure 121 - Création de la queue

On en revient au bout de code de handleEvent montré au tout début de ce chapitre (également disponible ci-dessous). Ce simple bloc va permettre la parallélisation de tout le traitement du contenu du fichier ouvert.

```
129             //We read all packets
130             dispatch_async(_dispatchQueue,^{
131                 while ([self readNextPacket]);
132             });
```

Figure 122 - General Central Dispatcher

Bien que très simple à mettre en place, cette parallélisation du traitement requiert tout de même plusieurs précautions et apporte un problème. Le problème vient du fait que, comme cité précédemment, la gestion de l'interface utilisateur est faite uniquement par le thread principal. Ainsi, il est impossible de mettre à jour l'interface à partir d'un thread secondaire, comme celui que l'on utilise. Il n'est donc pas faisable d'afficher les paquets dans le packet viewer, propriété UI de PcapParser rappelons le, à partir du thread effectuant le traitement des fichiers. Heureusement, il existe également une file de travail pour le main thread. Il suffit d'envoyer les tâches de mise à jour sur cette file et l'interface utilisateur pourra être mise à jour sans problème.

```
303     //if there was a known protocol added to the packet, we get the formatted reprensentation
304     //and update the UI
305     if ([[packet protocols] count] > 0){
306         [_packets addObject:packet];
307         NSMutableArray *uiformatted = [packet packetFormattedForTree];
308
309         [_UI addElementToTree:uiformatted withPacketIndexEqualTo:[_packets count] - 1];
310
311         dispatch_async(dispatch_get_main_queue(),^{
312             [_UI updateUITableView];
313         });
314
315         return packet.packetHeader.incl_len;
316     }
```

Figure 123 - Mise à jour de l'UI

Le code ci-dessus montre comment il est possible d'envoyer des tâches à la file de travail du thread principal. Cet extrait de code vient de la méthode readNextPacket.

Les précautions à prendre lors de parallélisation, mis à part le problème de gestion d'interface utilisateur, concernent les ressources partagées. Dès le moment ou il y a plusieurs thread et que ceux-ci ont accès aux même variables, il faut les protéger. Pour protéger des objets, on parle souvent de synchronisation. En Objective-C, les mécanismes de verrou et de sémaphores sont disponibles. Dans le cadre de l'application de ce projet, un objet en particulier a besoin d'être protégé. Lorsque le traitement d'un paquet et de toutes ses couches de protocoles et de donnée est terminé, le packet est ajouté à l'arbre du packet viewer. Comme vu précédemment, cet arbre est un tableau. D'un côté nous avons le thread parsant le fichier pcap qui va ajouter les nouveaux paquets à l'arbre et de l'autre le thread principal qui va parcourir l'arbre afin de pour l'afficher à l'utilisateur.

Avec deux threads accédant à cette variable, il est nécessaire de la protéger. Pour ce type de données partagées, une simple section critique protégée par un mutex est suffisante à limiter l'accès à un seul thread à la fois. En Objective-C, les objets de type NSLock[57] font office de verrou. Un thread va appeler la méthode *lock* sur une instance NSLock pour le verrouiller et le libérera une fois la tâche terminée avec *unlock*. Durant la période d'exécution ou l'objet NSLock est verrouillé, aucun autre thread ne pourra accéder à la section protégée.

```
779     [_treeItemsLock lock];
780     [_data addObject:item];
781     [_treeItemsLock unlock];
```

Figure 124 - Utilisation de NSLock

[57] https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSLock_Class/Reference/Reference.html

8.8 Traitement des protocoles réseaux

Lorsqu'un fichier est lié au flux et que celui-ci est ouvert, on a vu que l'événement NSStreamEventOpenCompleted est déclenché et traité par handleEvent de la classe PcapParser.

```
case NSStreamEventOpenCompleted:
{
    //We read the pcap file header
    [self readGlobalHeader];

    //We read all packets
    dispatch_async(_dispatchQueue,^{
        while ([self readNextPacket]);
    });
    break;
}
```

Figure 125 - Ouverture du flux réussie

Dès ce moment là, on peut commencer à lire les données contenues dans le fichier. La première méthode appelée est readGlobalHeader et va servir à traiter l'en-tête général ajouté par le format libpcap (voir chapitre « Format Libpcap ». Pour lire des données dans un flux NSInputStream il faut appeler la méthode read qui va enregistrer les bytes lus dans un buffer alloué au préalable.

```
180 - (void) readGlobalHeader
181 {
182     int sizeOfGlobalHeader = 24;
183
184     uint8_t buf[sizeOfGlobalHeader];
185     unsigned int len = [_fileStream read:buf maxLength:sizeOfGlobalHeader];
186     UnsignedInt8Buffer *data = [[UnsignedInt8Buffer alloc] initWithBuffer:buf ofLength:len];
187
```

Figure 126 - Lecture des octets

8.8.1 Gestion des octets avec UnsignedInt8Buffer

Afin de généraliser et simplifier la lecture d'octets dans des buffer, la classe UnsignedInt8Buffer a été créée. Cette classe est utilisée dans absolument tous les protocoles implémentés par l'application et est extrêmement utile grâce à sa gestion de l'ordre des octets. Lors du traitement de données lues, ce sera toujours un objet de type Unsignedint8Buffer qui sera donné.

Lors de l'initialisation d'un objet de cette classe, il faut lui passer en paramètres un tableau d'octets avec la longueur de ce tableau et également la manière de lire les octets. C'est là que cette classe s'avère extrêmement utile. Dans l'en-tête générale des fichiers pcap, un champ (magic word) va indiquer au parseur si l'application ayant écrit le fichier pcap fonctionne avec Little Endian ou Big Endian.

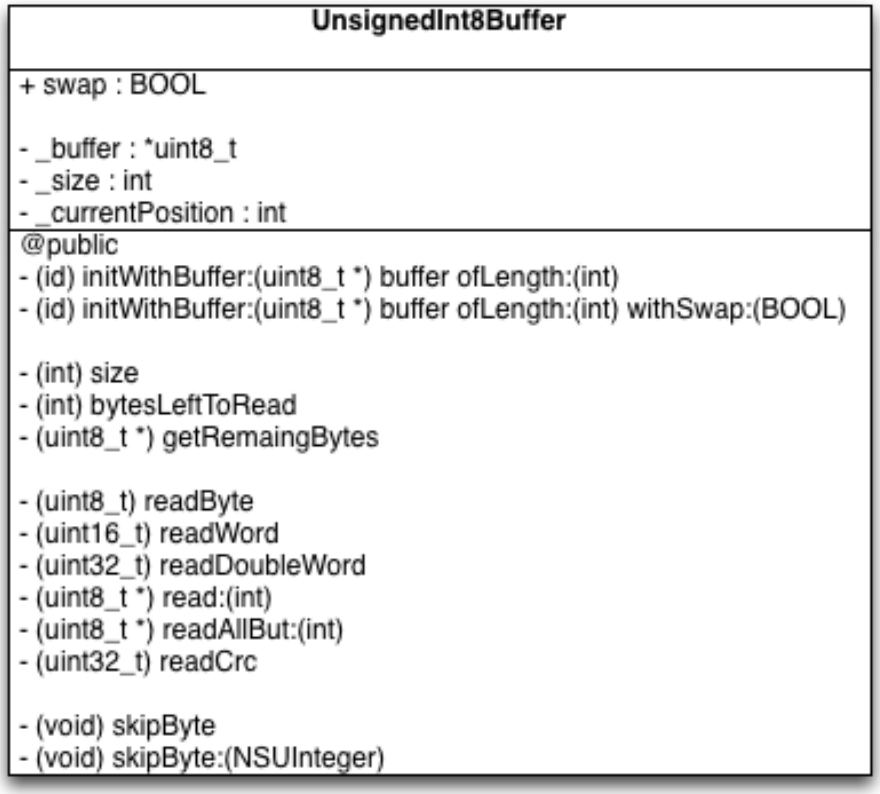

Figure 127 - Diagramme UML de UnsignedInt8Buffer

La classe UnsignedInt8Buffer va gérer automatiquement les deux types d'allocations grâce à l'attribut swap. Si swap est égal à YES alors chaque octet sera lu en Little Endian, sinon en Big Endian.

UnsignedInt8Buffer stocke les octets qu'on lui passe en paramètre dans la variable d'instance _buffer et permet de les lire grâce au nombreuses méthodes simplifiant l'accès aux données : readByte, readWord, readDoubleWord, et read. Un pointeur, _currentPosition, va être incrémenté à chaque lecture d'un octet afin de garder en mémoire ou en en est dans la lecture des données. Cette méthode permet aux parseurs de protocoles de simplement appeler séquentiellement les méthodes de lecture suivant le format du paquet du protocole en question.

Regardons le code lisant l'en-tête du protocole UDP afin d'observer la classe UnsignedInt8Buffer en action. Ce code provient de la classe UDPProtocol effectuant le traitement des datagrammes UDP.

```objc
- (int) readHeaderFromBuffer:(UnsignedInt8Buffer *)buf
{
    //Calculate the start of the datagram
    _nextOffsetStartPosition = [buf size] - [buf bytesLeftToRead];
    _datagramStart = _nextOffsetStartPosition;
    _cellColor = COLOR_UDP;

    //UDP doesn't use swapping, it's always Big Endian
    BOOL swap = [buf swap];
    [buf setSwap:NO];

    //we read the UDP header fields
    _header.uh_sport = [buf readWord];
    _header.uh_dport = [buf readWord];
    _header.uh_ulen = [buf readWord];
    _header.uh_sum = [buf readWord];

    [buf setSwap:swap];

    //return the destination port
    return _header.uh_dport;
}
```

Figure 128 - Lecture des champs de l'en-tête UDP

8.8.2 Lecture des paquets

Lorsque l'en-tête général de libpcap est lu, on va trouver les paquets capturés.

```objc
case NSStreamEventOpenCompleted:
{
    //We read the pcap file header
    [self readGlobalHeader];

    //We read all packets
    dispatch_async(_dispatchQueue,^{
        while ([self readNextPacket]);
    });
    break;
}
```

Figure 129 - Lecture des paquets

De la même manière que lors de la lecture de l'en-tête générale du fichier, on va allouer un nouveau UnsignedInt8Buffer et lire l'en-tête libpcap du paquet en cours de traitement grâce à la méthode readPacketHeader. L'information importante que l'en-tête va nous apporter est la taille totale du paquet. Grâce à ceci, on va allouer un dernier buffer qui va contenir la totalité des octets formant le paquet. Ce buffer va persister jusqu'à la fin du traitement du paquet puisqu'il sera passé en paramètre à chaque parseur de protocole détecté dans le paquet.

```objc
- (NSInteger) readNextPacket
{
    Packet *packet = [self readPacketHeader];
    [packet setPacketNumber:[_packets count]+1];

    //if inc_len = 0 it means we reached the end of the buffer, this is an empty packet
    if (packet.packetHeader.incl_len == 0) return 0;

    //we get the bytes
    uint8_t buf[packet.packetHeader.incl_len];
    unsigned int len = [_fileStream read:buf maxLength:packet.packetHeader.incl_len];
    UnsignedInt8Buffer *data = [[UnsignedInt8Buffer alloc] initWithBuffer:buf ofLength:len withSwap:_swapIsNeeded];
```

Figure 130 - Lecture de l'en-tête libpcap et des données d'un paquet

La méthode readPacketHeader va retourner un objet de type Packet. Cette classe est utilisée pour représenter les paquets présents dans les captures. Chaque objet de la classe Packet sera ajouté au tableau packets de PcapParser une fois traité. La

classe Packet représente plus précisément un paquet lu au format libpcap puisqu'elle contient une propriété packetHeader qui contient les informations de l'entête libpcap du paquet. Cet en-tête n'a rien à voir avec les en-têtes des protocoles présents dans le paquet mais est uniquement liée au format libpcap, comme expliqué dans le chapitre « Format Libpcap ».

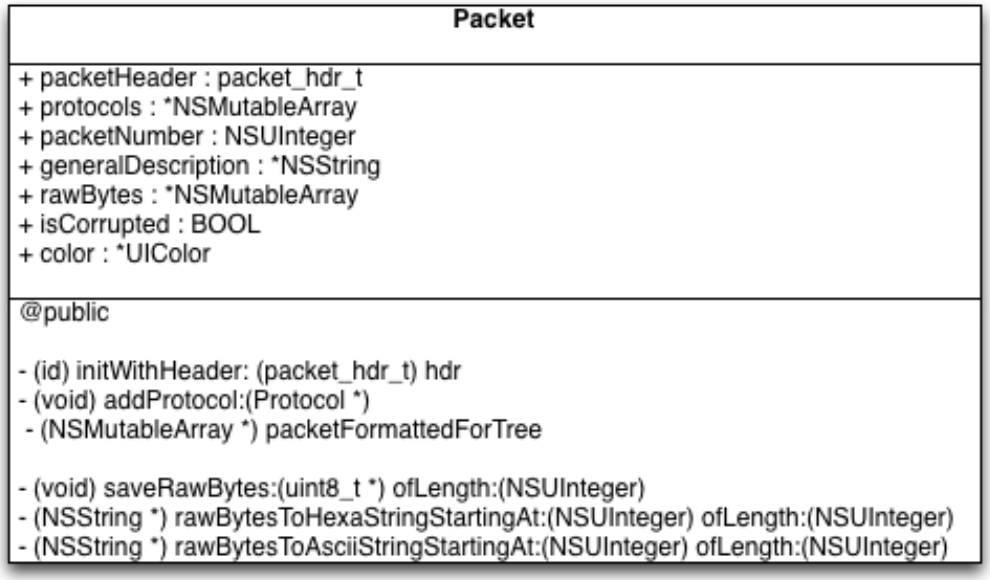

Figure 131 - Diagramme UML de Packet

Lorsque l'on veut afficher un paquet dans le packet viewer, c'est la méthode packetFormattedForTree qu'il faut appeler. Comme son nom l'indique, la méthode va retourner la représentation du paquet formatée pour l'arbre et prête à y être insérée.

```
104     if (!_isCorrupted){
105         for (id object in _protocols) {
106             id tmp = [object packetDetailsFormattedForTree];
```

Figure 132 - Ajout de tous les protocoles

Le point clé de cette fonction est l'appel de packetDetailsFormattedForTree pour chaque protocole présent dans le paquet. La propriété protocols contient tous les protocoles, sous la forme d'objets de type Protocol, présents dans le paquet. C'est là que la magie de la programmation orientée objets opère. Grâce au polymorphisme n'importe quel protocole, aussi différent des autres qu'il soit, pourra être inclut dans l'application sans avoir à modifier une seule ligne de code ailleurs. La section suivante explique plus en détails le fonctionnement de la classe Protocol.

8.8.3 Lecture des protocoles

Afin de rendre l'application modulable et évolutive, les concepts de programmation orientée objets ont été très utiles pour le traitement des protocoles. Chaque protocole implémenté doit dériver de la classe Protocol et ainsi implémenter le protocole UIPacketDetails (oui cela fait beaucoup de fois le mot protocole, mais c'est le terme Objective-C également !).

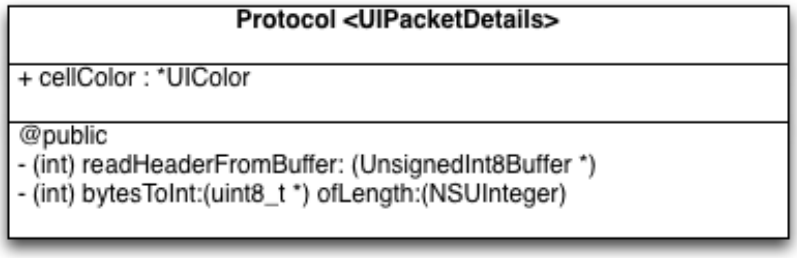

Figure 133 - Diagramme UML de la classe Protocol

La classe Protocol définit la méthode readHeaderFromBuffer qui va lire l'en-tête du protocole concerné. Cette méthode demande comme paramètre un buffer de type UnsingedInt8Buffer qui doit pointer sur le premier byte de l'en-tête du protocole. Cette méthode sera automatiquement appelée à chaque fois qu'un protocole est instancié et que l'on veut lire son en-tête et ses données. ReadHeaderFromBuffer doit retourner le numéro du protocole encapsulé dans son champ de données ou -1 dans le cas où il n'y a rien. Cette valeur de retour est utilisée pour les protocoles comme LLC qui possède un champ ethertype permettant de continuer le traitement du paquet de manière appropriée.

La propriété cellColor est utilisée afin de donner une couleur spéciale à certains paquets dans le packet viewer. Si aucune couleur n'est affectée à cette propriété, la couleur par défaut sera utilisée.

Le protocole UIPacketDetails, dans le sens Objective-C du terme, spécifie une méthode que toutes les classes l'implémentant devront implémenter : packetDetailsFormattedForTree. Cette méthode doit retourner un tableau construit au format standard pour être inséré dans l'arbre du packet viewer.

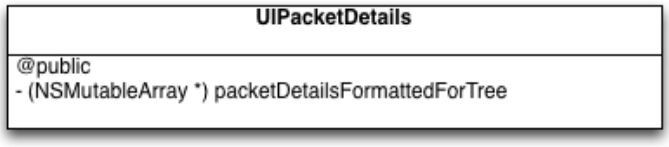

Figure 134 - Diagramme UML de UIPacketDetails

C'est cette méthode qui fait le lien entre les bytes lus dans un fichier pcap et l'interface utilisateur. Grâce à l'héritage et au polymorphisme de la programmation orientée objet, cette méthode pourra être appelée sur chaque protocole implémenté dans l'application sans aucun traitement particulier ou changement de code. Il suffit simplement qu'elle soit implémentée correctement pour retourner un tableau bien formatté pour l'arbre de PacketViewer. Regardons à nouveau l'implémentation du protocole UDP et sa méthode packetDetailsFormattedForTree afin d'avoir un exemple concret.

```
68  - (NSMutableArray *) packetDetailsFormattedForTree
69  {
70      NSMutableArray *desc = [NSMutableArray new];
71
72      //source port
73      [desc addObject:[ProtocolField fieldWithDescription:[ParsingTools format:UDP_SOURCE withValue:_header.uh_sport] startingAt:
            _nextOffsetStartPosition ofSize:2]];
74      _nextOffsetStartPosition += 2;
75
76      //dest port
77      [desc addObject:[ProtocolField fieldWithDescription:[ParsingTools format:UDP_DESTINATION withValue:_header.uh_dport] startingAt:
            _nextOffsetStartPosition ofSize:2]];
78      _nextOffsetStartPosition += 2;
79
80      //length
81      [desc addObject:[ProtocolField fieldWithDescription:[ParsingTools format:UDP_LENGTH withValue:_header.uh_ulen] startingAt:
            _nextOffsetStartPosition ofSize:2]];
82      _nextOffsetStartPosition += 2;
83
84      //checksum
85      [desc addObject:[ProtocolField fieldWithDescription:[ParsingTools formatHexa:UDP_CHECKSUM withValue:_header.uh_sum] startingAt:
            _nextOffsetStartPosition ofSize:2]];
86      _nextOffsetStartPosition += 2;
87
88      //description line (the header size is expressed in 32 bits, so we multiply by 4 to have the actual amount of bytes)
89      NSString *firstLineSummary = [NSString stringWithFormat:@"%@, Src port:%i, Dst port:%i", UDP_FIRST_LINE_DESCRIPTION, _header.uh_sport, _header
            .uh_dport];
90      [desc insertObject:[ProtocolField fieldWithDescription:firstLineSummary startingAt:_datagramStart ofSize:UDP_DATAGRAM_LEN] atIndex:0];
91      return desc;
92  }
```

Figure 135 – PacketDetailsFormattedForTree

On voit dans le code que la méthode va insérer dans un tableau un objet de type ProtocolField pour chaque champ qui sera affiché à l'écran. Le protocole UDP ne comporte que quatre champs et ces champs sont simple, ce qui rend cette méthode assez courte. Pour la norme 802.11 par exemple, nettemement plus complexe, la méthode packetDetailsFormattedForTree est largement plus longue.

Pour revenir au traitement des paquets dans la classe PcapParser, on avait que la méthode readPacketHeader retournait l'objet Packet dans lequel on va ajouter un objet Protocol pour chaque protocole lu. L'en-tête libpcap des paquets contient un champ qui va nous permettre de savoir quel protocole de la couche liaison de données a été utilisé lors de la capture des paquets : network. En se basant sur network, on va pouvoir définir quelle dérivation de la classe Protocol utiliser. Chaque protocole implémenté de chaque couche du modèle OSI possède une classe permettant d'interpréter ses octets. Pour la couche liaison de données, c'est la méthode suivante qui s'en occupe :

```
- (void) handleData:(UnsignedInt8Buffer *)data
  fromDataLinkLayer:(DataLinkLayerType)type intoPacket:(Packet *)p
```

Voilà en tant qu'exemple une partie du code de cette méthode, gérant le cas ou c'est radiotap qui est le protocole de liaison de données.

```
338 - (void) handleData:(UnsignedInt8Buffer *)data fromDataLinkLayer:(DataLinkLayerType)type intoPacket:(Packet *)p
339 {
340     int ethertype = -1;
341     switch (type) {
342         case LINKTYPE_IEEE802_11_RADIOTAP:
343         {
344             RadioTapProtocol *radio = [[RadioTapProtocol alloc] init];
345             [radio readHeaderFromBuffer:data];
346             [p addProtocol:radio];
347
348             if (data.bytesLeftToRead > 0){
349                 [self handleData:data fromDataLinkLayer:LINKTYPE_IEEE802_11 intoPacket:p];
350             } else
351                 NSLog(@"Error: Radiotap not followed by 802.11");
352             //radiotap must be followed by a ieee 802.11 header.
353             break;
354         }
```

Figure 136 - Snippet de handleDatafromDataLinkLayer

On retrouve, dans le bout de code ci-dessus, la méthode readHeaderFromBuffer héritée de Protocol. Une fois appelée, on ajoute le protocole à l'objet Packet p. Plus tard, lors de l'itération de tous les protocoles de p, la méthode packetDetailsFormattedForTree sera appelée sur l'instance de radiotap et un tableau formaté sera inséré dans l'arbre et affiché à l'utilisateur. Si l'on voulait ajouter un protocole de couche liaison de données, il suffirait d'ajouter un cas au switch-case de cette méthode et d'initialiser un objet dérivant de Protocol faisant le traitement approprié pour que ce protocole soit supporté !

La méthode handleDatafromDataLinkLayer va également récupérer l'ethertype afin de pouvoir parser le prochain protocole présent dans le paquet, qui proviendra de la couche réseau.

```
396     //if a network-layer protocol is following, we handle it.
397     if (ethertype > -1){
398         [self handleData:data fromNetworkLayer:ethertype intoPacket:p];
399     }
```

Figure 137 - Après la couche 2, on traite la 3

Cette fois ci c'est handleDatafromNetworkLayer et non plus handleDatafromDataLinkLayer qui va s'occuper de traiter les octets du buffer.

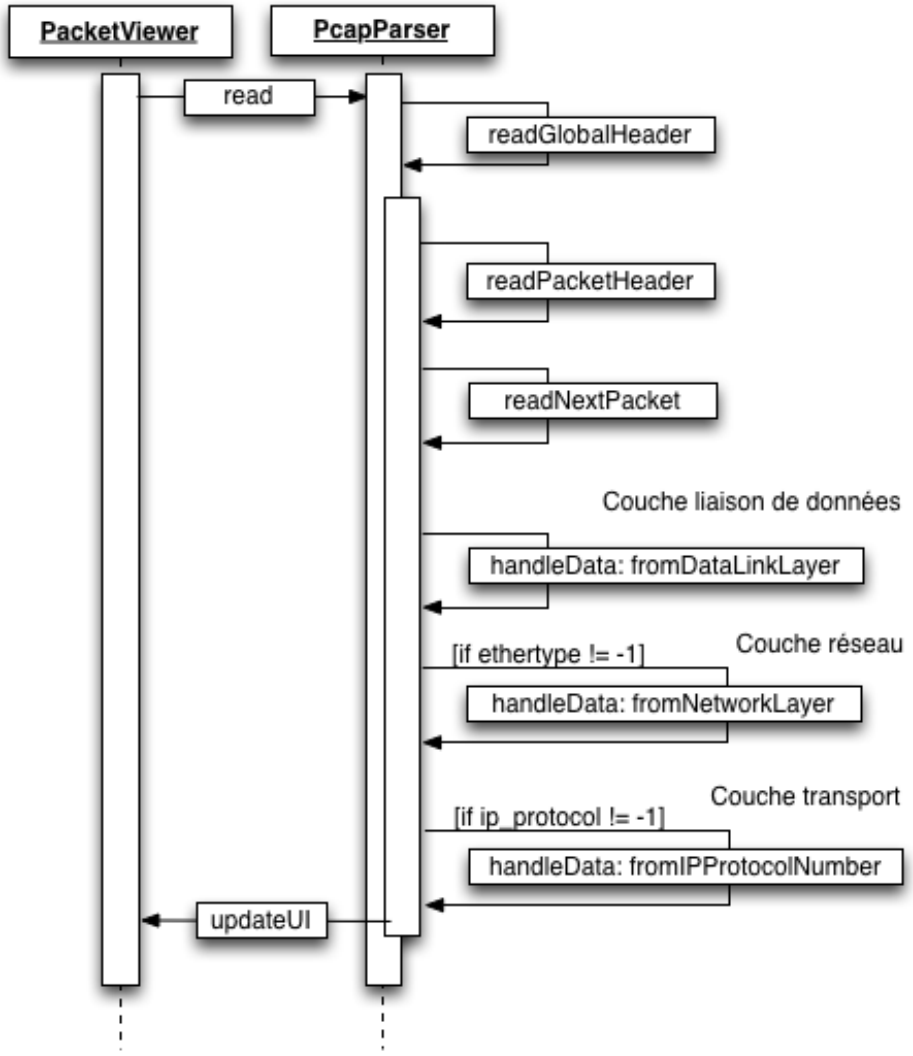

Figure 138 - Diagramme de séquence du traitement d'un fichier

Le principe de cette méthode est exactement le même que pour la méthode traitant la couche liaison de données ainsi que pour toutes les autres couches d'ailleurs. Ces méthodes sont appelées en cascade en respectant le modèle OSI.

On retrouve donc les méthodes suivantes :

- `(void) handleData:(UnsignedInt8Buffer *)data fromNetworkLayer:(NetworkLayerType)type intoPacket:(Packet *)p`
- `(void) handleData:(UnsignedInt8Buffer *)data fromIPProtocolNumber:(IpProtocolNumber)type intoPacket:(Packet *)p`

Lorsque l'on a fini de traiter toutes les couches, il est possible qu'il reste des données dans le paquet. En effet toutes ces en-têtes et ces protocoles ne servent essentiellement qu'à une chose : transporter des données. A la fin du traitement de

tous les protocoles, on vérifie s'il reste des données dans le buffer qui a circulé dans tous les parseurs et si c'est le cas, cela signifie qu'il reste des données. On va alors instancier un objet Data et stocker le reste des données dedans.

```
if ([data bytesLeftToRead] > 0){
    Data *d = [Data new];
    [d readHeaderFromBuffer:data];

    [packet addProtocol:d];
}
```

Figure 139 - Création d'un objet Data

La classe Data est une sous-classe de Protocol, tout comme n'importe quel parseur de protocole. Elle est implémentée de cette manière car tout comme un protocole, on va lire les données dans le buffer et on va avoir besoin d'une fonction qui va retourner un tableau pour le packet viewer. De plus, on veux pouvoir ajouter l'objet data dans le tableau de protocoles de chaque objet packet afin que l'appel des fonctions soit automatisé.

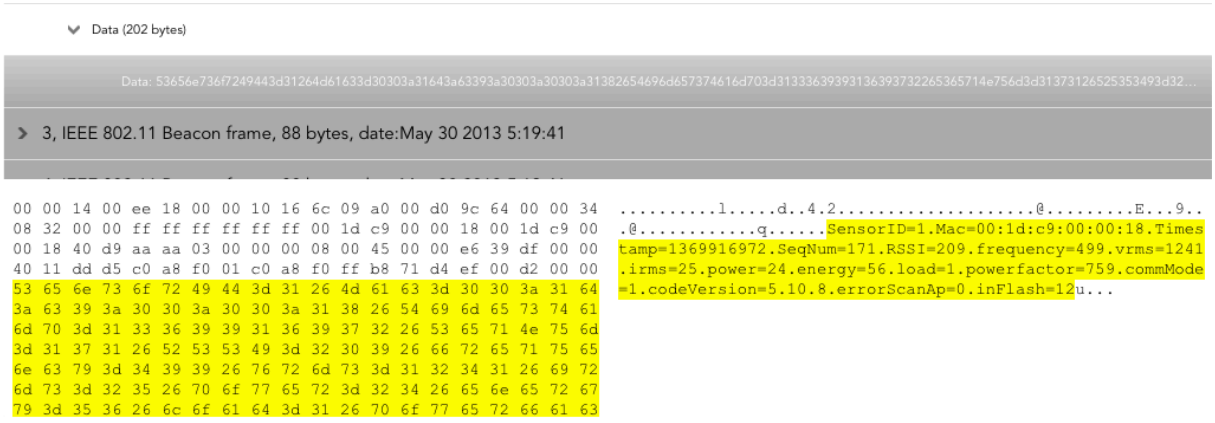

Figure 140 - Data d'un paquet

La figure ci-dessus montre comment est affiché l'objet Data dans le packet viewer. La représentation hexadécimale et ascii ne se fait pas seulement pour les objets data par contre, et cette fonctionnalité est expliqué au prochain chapitre. La partie en ascii montre bien que c'est des données parfaitement lisibles.

Une fois que le paquet est complètement traité, on va pouvoir appeler la méthode packetFormattedForTree de la classe Packet afin d'obtenir le tableau à insérer dans l'arbre du packet viewer. Il ne reste plus qu'à rafraîchir l'interface utilisateur pour voir notre paquet apparaître.

```
//if there was a known protocol added to the packet, we get the formatted reprensentation
//and update the UI
if ([[packet protocols] count] > 0){
    [_packets addObject:packet];
    NSMutableArray *uiformatted = [packet packetFormattedForTree];

    [_UI addElementToTree:uiformatted withPacketIndexEqualTo:[_packets count] - 1];

    dispatch_async(dispatch_get_main_queue(),^{
        [_UI updateUITableView];
    });

    return packet.packetHeader.incl_len;
}
else return 0;
```

Figure 141 – Code de l'affichage du paquet

8.9 Affichage brut des octets

Le troisième point le plus prioritaire du cahier des charges est de pouvoir afficher le contenu brut du paquet en hexadécimal. Cette fonctionnalité est utile car il est parfois plus facile de regarder le contenu brut d'un champ en hexadécimal ou en clair plutôt que de le chercher dans l'interface utilisateur. Cela est surtout valable pour des données car il n'est pas rare de trouver des informations en clair, surtout avec l'utilisation par exemple de http qui n'encrypte rien par défaut. On veut donc également ajouter une vue du contenu brut en ascii afin que ce soit plus facilement lisible par l'utilisateur que de l'hexadécimal.

```
00 00 14 00 ee 18 00 00 10 16 6c 09 a0 00 d0 9c 64 00 00 34  ..........l.....d..4
08 32 00 00 ff ff ff ff ff ff 00 1d c9 00 00 18 00 1d c9 00  .2..................
00 18 40 d9 aa aa 03 00 00 00 08 00 45 00 00 e6 39 df 00 00  ..@.........E...9...
40 11 dd d5 c0 a8 f0 01 c0 a8 f0 ff b8 71 d4 ef 00 d2 00 00  @..............q....
53 65 6e 73 6f 72 49 44 3d 31 26 4d 61 63 3d 30 30 3a 31 64  SensorID=1.Mac=00:1d
3a 63 39 3a 30 30 3a 30 30 3a 31 38 26 54 69 6d 65 73 74 61  :c9:00:00:18.Times
6d 70 3d 31 33 36 39 39 31 36 39 37 32 26 53 65 71 4e 75 6d  tamp=1369916972.SeqNum
3d 31 37 31 26 52 53 53 49 3d 32 30 39 26 66 72 65 71 75 65  =171.RSSI=209.freque
6e 63 79 3d 34 39 39 26 76 72 6d 73 3d 31 32 34 31 26 69 72  ncy=499.vrms=1241.ir
6d 73 3d 32 35 26 70 6f 77 65 72 3d 32 34 26 65 6e 65 72 67  ms=25.power=24.energ
79 3d 35 36 26 6c 6f 61 64 3d 31 26 70 6f 77 65 72 66 61 63  y=56.load=1.powerfac
```

Figure 142 - L'hexadécimal est plus difficile à lire

Pour implémenter cette fonctionnalité, la classe PcapParser va faire une copie de la totalité des bytes du paquet lorsqu'il va le lire sur le flux. On ne veut pas l'en-tête générale et l'en-tête du paquet ajouté par libpcap, alors on fais cette copie dans la fonction readNextPacket grâce à l'envoie du message saveRawByte de la classe Packet.

La classe PacketViewer permet l'espace pour la représentation hexadécimale et ascii des paquets grâce à deux propriétés de type UITextView placées au dessous de la table affichant les paquets : rawPacketHexaRepresentation et rawPacketAsciiRepresentation. Pour des questions de clarté, on ne va afficher les bytes d'un seul paquet à la fois dans ces textview. Afin d'afficher le contenu brut lorsque le PacketViewer reçoit le premier paquet ou que l'utilisateur clique sur une des ligne de la capture, deux méthodes de la classe Packet sont disponibles :

```
- (NSString *) rawBytesToHexaStringStartingAt:(NSUInteger)offset
ofLength:(NSUInteger)length;
```

```
182 - (NSString *) rawBytesToHexaStringStartingAt:(NSUInteger)offset ofLength:(NSUInteger)length {
183     NSMutableString *result = [NSMutableString new];
184     for (int i = offset; i < offset + length; i++) {
185         [result appendFormat:@"%02x ", [[_rawBytes objectAtIndex:i] intValue]];
186     }
187     return [NSString stringWithString:result];
188 }
```

Figure 143 - Code retournant la représentation hexadécimale du paquet

La méthode retournant la représentation hexadécimale du paquet ne comporte rien de particulier. On va simplement spécifier qu'on veut que chaque octet soit représenter par deux caractères, afin d'avoir également le 0 devant dans le cas d'un nombre plus petit que 16 et on va ajouter un espace entre chaque octet.

```
- (NSString *) rawBytesToAsciiStringStartingAt:(NSUInteger)offset
ofLength:(NSUInteger)length;
```

```
203 - (NSString *) rawBytesToAsciiStringStartingAt:(NSUInteger)offset ofLength:(NSUInteger)length {
204     NSMutableString *result = [NSMutableString new];
205     char formattedChar;
206
207     for (int i = offset; i < offset + length; i++) {
208         if ([[_rawBytes objectAtIndex:i] intValue] >= 48 && [[_rawBytes objectAtIndex:i] intValue] <= 126)
209             formattedChar = [[_rawBytes objectAtIndex:i] intValue];
210         else
211             formattedChar = '.';
212
213         [result appendFormat:@"%c", formattedChar];
214     }
215
216     return result;
217 }
```

Figure 144 - Code retournant la représentation ascii du paquet

Pour la représentation ascii, il est possible de rencontrer des problèmes si l'on ne filtre pas les caractères affichés. En effet, certain caractère non imprimable vont provoquer des décalages entre des octets et l'exactitude du paquet ne sera pas conservée. De plus, la plupart des caractères de la table ascii ne sont pas lisibles ou pas compréhensible. Pour cette raison, on va seulement garder un caractère s'il est entre l'intervalle de 48 à 126 dans la table ascii. Cet intervalle comporte toutes les lettres, chiffres et caractères utiles lors de la lecture d'un paquet. Tous les autres sont remplacés par un point. Cette méthode est très efficace et la lisibilité est nettement améliorée.

Ces deux méthodes sont donc utilisées par PacketViewer afin d'afficher les octets à l'utilisateur. Afin de rendre la fonctionnalité encore meilleure, lorsque l'utilisateur clique sur un champ, sur une couche, sur n'importe quel élément dans packet viewer, on veut que les octets concernés soient mis en évidence. Pour faire cela, il est nécessaire de savoir, pour chaque champ, à quel endroit exactement dans le paquet il commence et quelle taille il fait. La classe ProtocolField a été conçue à cet effet.

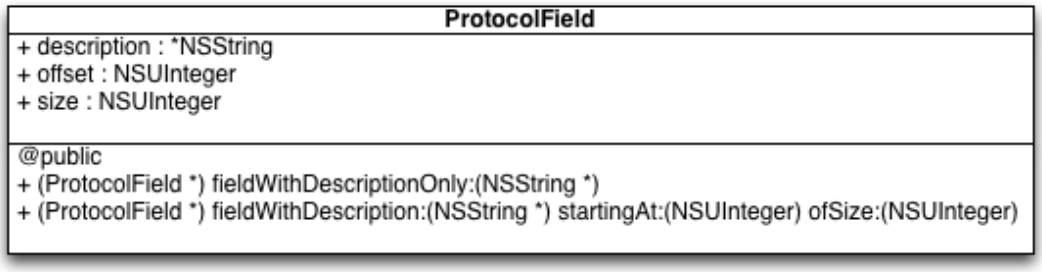

Figure 145 - Diagramme UML de ProtocolField

On a déjà parlé de la classe ProtocolField lors de l'explication de l'algorithme insérant des champs dans l'arbre. Chaque nœud individuel de l'arbre est en fait un objet de la classe ProtocolField. La propriété *description* est le texte qui sera affiché à l'écran et les propriétés offset et size vont être utilisé pour mettre en évidence les octets concernés dans la représentation hexadécimale et ascii du paquet. Offset est un indicateur d'ou commence le champ par rapport au premier octet du paquet.

Lorsque l'utilisateur clique sur un paquet, le contenu du paquet est affiché dans les deux UITextView en bas de l'écran. Si l'utilisateur clique sur un champ d'un paquet, ou une couche (IP, TCP, 802.11), la méthode didSelectrowAtIndexPath de PacketViewer va mettre en évidence les octets concernés dans les textview.

```
if (cell.treeItem.size != 0){ //Highlight needed if size is different that zero

    //Starts at offset times 2 (each octet is 2 chars) plus the spaces present before the start
    int startHighlightHex = cell.treeItem.offset * 2 + cell.treeItem.offset;
    //Each octet is 2 chars plus the spaces in between each two chars
    int lengthOfHighlightHex = cell.treeItem.size * 2 + (cell.treeItem.size - 1);

    //Each octet is one char and no spaces
    int startHighlightAscii = cell.treeItem.offset;
    int lengthOfHighlightAscii = cell.treeItem.size;
    //NSLog(@"ascii start : %i and length : %i", startHighlightAscii, lengthOfHighlightAscii);
    //NSLog(@"length of ascii : %i", _rawPacketAsciiRepresentation.text.length);
    [self highlightHexaRange:NSMakeRange(startHighlightHex, lengthOfHighlightHex)];
    [self highlightAsciiRange:NSMakeRange(startHighlightAscii, lengthOfHighlightAscii)];
}
```

Figure 146 - Mise en évidence des octets d'un champ

Etant donné que l'on a fait de la mise en page dans la représentation hexadécimale avec l'ajout d'espaces et que chaque octet est représenté par deux caractères, il faut effectuer un petit calcul afin de déterminer le début du champ ainsi que la taille exacte à mettre en évidence. Chaque octet compte deux chiffres hexa, et entre chaque deux chiffres se trouve un espace. Le début de la zone à surligner se calcule ainsi de la manière suivante:

$$début\ zone\ héxa = cellule.offset * 2 + cellule.offset$$

La multiplication permet de gérer le double chiffre par octet et on additionne offset afin de compenser tous les espaces présents dans les octets précédent ceux qui vont être surlignés. Pour la longueur à mettre en évidence, le calcul est similaire. On multiplie par deux afin de compenser le double chiffre et on additionne la longueur initiale pour les espaces. Il ne faut pas oublier de soustraire 1 à la longueur afin de ne pas prendre le dernier espace.

$$taille\ zone\ héxa = cellule.taille * 2 + (cellule.taille - 1)$$

La représentation pose moins de problème puisqu'un octet est un caractère et aucun espace n'est ajouté. Aucun traitement n'est nécessaire pour déterminer la zone à mettre en évidence.

8.10 Gestion des erreurs

Le monde des transmission est instable, c'est bien connu, et il n'est pas rare de tomber sur des paquets mal formés ou des erreurs des transmissions aléatoires, surtout lorsque l'on travaille avec des réseaux WLAN. De plus, il est difficile de supporter tous les protocoles existant et de comprendre tout ce qui est possible de trouver dans des paquets à l'heure actuelle, surtout dans le cadre d'un projet de bachelor. Il existe trop de protocoles. Pour ces raisons, il est nécessaire de rendre l'application robuste afin qu'elle ne crash pas dès qu'un paquet est malformé ou qu'un protocole n'est pas supporté.

Premièrement, afin de traiter uniquement les protocoles qui sont compris par l'application, chaque méthode handleData (traitant les couches) va avoir une clause « default » traitant les cas ou le numéro de protocole n'est pas connu.

```
default:
{
    NSLog(@"LINK-LAYER PROTOCOL NOT SUPPORTED : %i", type);
    break;
}
```

Figure 147 - Protocole inconnu

Ensuite, il est possible qu'un paquet soit mal formé. Dans ce cas, des champs seront peut-être manquants ou alors plus long ou plus court que prévu. Il est également possible qu'on s'attende à une certaine valeur dans un champ et qu'à cause d'une erreur de transmission la valeur ait été modifiée. Imaginons que le champ « ethertype» d'une trame LLC soit corrompu, au lieu de 0x0800 informant qu'IP suit, on ait 0x0801. Tous ces types d'erreurs sont totalement différents les uns des autres et tellement imprévisibles qu'on ne peut pas traiter une erreur à la fois.

Afin de supporter toutes les erreurs, il va falloir lever des exceptions spécifiques à des endroits précis dans le code et les récupérer. Le code suivant provient de PcapParser et s'occupe de récupérer toutes les exceptions survenues durant le

parsing d'un paquet. Si cela arrive, le paquet est marqué comme corrompu traitement de celui-ci est directement ajouté.

```
@try {
    //and we handle what's following
    [self handleData:data fromDataLinkLayer:_fileGlobalHeader.network intoPacket:packet];
    if ([data bytesLeftToRead] > 0){
        Data *d = [Data new];
        [d readHeaderFromBuffer:data];

        [packet addProtocol:d];
    }
}
@catch (NSException *exception) {
    NSLog(@"Corrupted packet %i. Will be added as corrupted", [packet packetNumber]);
    [packet setIsCorrupted:YES];
    [_packets addObject:packet];
    return -1;
}
```

Figure 148 - Récupération des exceptions de traitement

Le paquet est tout de même ajouté à la liste des paquets de la capture afin qu'il soit affiché dans le packet viewer qui va l'afficher en rouge en voyant qu'il a été marqué comme corrompu.

```
388     Packet *actualPacket = [[_parser packets] objectAtIndex:cell.treeItem.packetIndex];
389
390     //certain protocols have a different cell color
391     if (!cell.treeItem.parentSelectingItem){
392         [cell.treeItem setColorStyle:actualPacket.color];
393     }
394
395     //the packet is corrupted
396     if ([actualPacket isCorrupted] && !cell.treeItem.parentSelectingItem){
397         [cell.treeItem setCorrupted];
398         [cell.textLabel setText:[NSString stringWithFormat:@"%@ >> %@", PARSING_ERROR_MESSAGE, cell.textLabel.text]];
399     }
```

Figure 149 - Packet lu comme corrompu

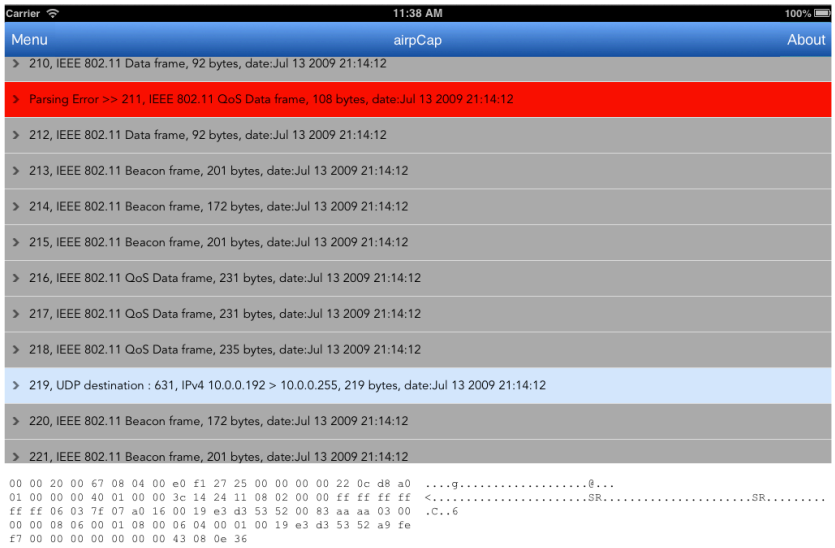

Figure 150 - Un paquet est corrompu

9 Perspectives

L'application possède une base robuste et évolutive permettant sans problème d'ajouter plus de protocoles afin d'augmenter son domaine de compétence. De nombreuses fonctionnalités pourraient être utiles à une telle application, à commencer par le filtrage des paquets ou l'émission de statistiques. Si l'on compare l'application avec wireshark et ses centaines de fonctionnalités, il y a énormément de possibilités et de perspectives pour continuer le développement de ce projet.

Actuellement, l'application n'en est qu'au point ou elle va pouvoir lire un fichier de capture enregistré par un autre appareil. Cela pose un problème logistique dans le sens où il faut prendre le temps de transférer cette capture faite par un autre appareil et la transférer sur l'iPad avec lequel on veut analyser les paquets. La perspective la plus intéressante présentée par Aginova pour directement capturer des paquets serait de développer un système embarqué comportant deux interfaces. La première interface capturerait les données sur un réseau WLAN 802.11 auquel elle est connectée. Elle transmettrait les paquets capturés à une deuxième interface qui elle serait pairée avec l'application tournant sur un iPad. Cette évolution possible est représentée sur la figure ci-dessous :

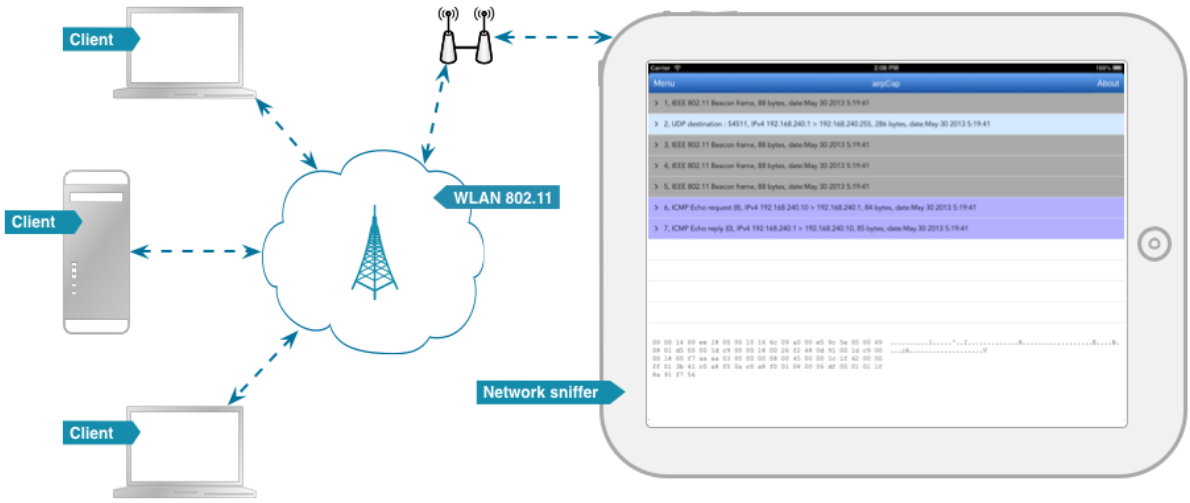

Figure 151 – Perspective d'un système embarqué à double interface

L'application a été développée en gardant en tête cette évolution possible et le système est prêt à être facilement adapté à cette architecture grâce à l'utilisation de flux pour la lecture du fichier. Ces flux peuvent également servir à lire un socket TCP ou UDP.

10 Conclusion

Au terme de ce travail, une application mobile permettant de lire des captures au format libpcap a été réalisée. Elle permet de lire tout fichier correctement formaté selon le standard libpcap et comprendra les protocoles IEEE 802.11, Radiotap, LLC, SNAP, IP, ICMP, TCP et UDP. Le contenu des fichiers de capture traité par l'application peut être inspecté jusqu'à un niveau microscopique. Chaque paquet est également visible dans sa représentation brute hexadécimale et ascii.

J'ai retiré beaucoup de choses de ce travail. Tout d'abord j'ai beaucoup appris au niveau technique puisque ce travail a requis l'apprentissage d'un tout nouveau langage de programmation pour moi, Objective-C, ainsi que du développement sur la plateforme iOS. En plus de cela, le traitement des protocoles réseaux a nécessité l'étude en profondeur de chacun de ces protocoles afin de savoir la signification du moindre octet que l'on pourrait trouver dans l'en-tête. De manière générale, le système final rassemble un nombre important de notions apprises au cours du projet. Je pense notamment à la norme 802.11 qui a pris énormément de temps à comprendre au vu de sa complexité et de son ampleur.

Le fait de devoir accomplir un projet de six mois consistant à développer une application de taille importante tout en apprenant une nouvelle technologie n'a pas toujours été facile. Le début du projet a particulièrement été difficile dans le sens ou il a fallut maîtriser Objective-C et iOS au plus vite afin de pouvoir commencer le réel développement de l'application. Cependant, je retire une grande expérience de ce projet. Le fait de devoir travailler de façon indépendante et de devoir trouver moi-même des solutions aux problèmes rencontrés m'a beaucoup apporté. Je me suis également enrichi du temps passé avec l'équipe d'Aginova, Karl et Thierry.

L'étape prioritaire qui devrait suivre ce projet est la mise en place d'un système de capture de paquets et de communication avec l'application afin d'avoir un système fonctionnel complètement mobile. Cette application avec un système permettant de capturer directement les paquets sans avoir besoin de transférer de fichier pourrait, à mon avis, avoir un grand succès.

San José, Californie, Etats-Unis, le 1er Août 2013 Mathieu Meylan

11 Liste des sources et des références

Toutes les sources utilisées durant ce projet, que ce soit pour la recherche d'information techniques pour le développement de l'application ou pour la rédaction de ce document, sont directement citées dans le texte. Ce chapitre résume par ordre alphabétique toutes les sources utilisées lors de ce projet.

[1] apple.com, comparaisons des modèles d'iPad
http://www.apple.com/chfr/ipad/compare/

[2] apple.com, informations techniques sur l'iPad
www.apple.com/iphone

[3] apple.com, informations techniques sur l'iPhone
www.apple.com/iphone

[4] ergodicthoughts.blogspot.com, explication des différences entre un mpdu et un ampdu
http://ergodicthoughts.blogspot.com/2012/02/difference-between-mpdu-msdu-ampdu-and.html

[5] github.com, librairie ECSlidingViewController
https://github.com/edgecase/ECSlidingViewController/blob/master/README.md

[6] github.com, librairie JWSlideMenu
https://github.com/mystcolor/JTRevealSidebarDemo

[7] gizmodo.com, statistiques concernant l'utilisation des smartphones au jourd d'aujourd'hui.
http://gizmodo.com/5882172/the-world-now-buys-more-smartphones-than-computers

[8] ieee.org, standard sur le WLAN 802.11
http://standards.ieee.org/about/get/802/802.11.html

[9] ieeee.org, standard sur 802.2 (LLC)
http://standards.ieee.org/about/get/802/802.2.html

[10] ietf.org RFC 791 sur IP
http://www.ietf.org/rfc/rfc791.txt

[11] ietf.org, RFC 1042 sur SNAP
http://www.ietf.org/rfc/rfc1042.txt

[12] ietf.org, RFC 768 sur UDP
http://www.ietf.org/rfc/rfc768.txt

[13] ietf.org, RFC 792 sur ICMP
http://www.ietf.org/rfc/rfc792.txt

[14] ietf.org, RFC 793 sur TCP
http://www.ietf.org/rfc/rfc0793.txt

[15] loufranco.com, explication de lgpl fonctionnant sur l'iPhone
http://loufranco.com/blog/lgpl-and-the-iphone.

[16] microsoft.com, explications sur le fonctionnement du wifi
http://technet.microsoft.com/en-us/library/cc757419(v=ws.10).aspx

[17] opensource.org, explication de la licence BSD 3 clauses
http://opensource.org/licenses/BSD-3-Clause

[18] packetlife.net, explications sur les flags TCP
http://packetlife.net/blog/2011/mar/2/tcp-flags-psh-and-urg/

[19] pcaptouch.net, site officiel de l'application Pcap Touch
http://pcaptouch.net/

[20] radiotap.org, structure des flags radiotap
http://www.radiotap.org/defined-fields

[21] radiotap.org, structure du protocole radiotap
http://www.radiotap.org/

[22] safaribooksonline.com, format des trames de contrôle action
http://my.safaribooksonline.com/book/certification/9781118075234/chapter-4-802dot11-management-frames/action_frames

[23] wi-fiplanet.com, format des trames de type data du protocole 802.11
http://www.wi-fiplanet.com/tutorials/article.php/3442991

[24] Site officiel des développeurs Apple
https://developer.apple.com.

[25] Site officiel des développeurs Apple, clauses d'acceptation d'une application
https://developer.apple.com/appstore/guidelines.html.

[26] Site officiel des développeurs Apple, comment créer et configurer une UITableView
http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/tableview_iphone/CreateConfigureTableView/CreateConfigureTableView.html

[27] Site officiel des développeurs Apple, documentation concernant les premiers pas en

tant que développeur Apple.
https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/GetToolsandInstall.html

[28] Site officiel des développeurs Apple, documentation concernant UIKit
http://developer.apple.com/library/ios/#documentation/uikit/reference/UIKit_Framework/_index.html

[29] Site officiel des développeurs Apple, documentation sur CoreData
http://developer.apple.com/library/mac/#documentation/cocoa/Conceptual/CoreData/cdProgrammingGuide.html

[30] Site officiel des développeurs Apple, documentation sur CoreGraphics
http://developer.apple.com/library/ios/#documentation/CoreGraphics/Reference/CoreGraphics_Framework/_index.html

[31] Site officiel des développeurs Apple, documentation sur Foundation
https://developer.apple.com/library/mac/#documentation/cocoa/reference/foundation/ObjC_classic/_index.html

[32] Site officiel des développeurs Apple, documentation sur les conventions de codage
https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CodingGuidelines/Articles/NamingBasics.html#//apple_ref/doc/uid/20001281-BBCHBFAH

[33] Site officiel des développeurs Apple, documentation sur QuartzCore
https://developer.apple.com/library/mac/#documentation/GraphicsImaging/Reference/QuartzCoreRefCollection/_index.html

[34] Site officiel des développeurs Apple, explication du multithreading
http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/Multithreading/AboutThreads/AboutThreads.html#

[35] Site officiel des développeurs Apple, gestion des séléctions
http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/tableview_iphone/ManageSelections/ManageSelections.html

[36] Site officiel des développeurs Apple, référence de la classe NSLock
https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSLock_Class/Reference/Reference.html

[37] Site officiel des développeurs Apple, référence de la classe NSStream
http://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSStream_Class/Reference/Reference.html

[38] Site officiel des développeurs Apple, référence de la classe NSURL

http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSURL_Class/Reference/Reference.html

[39] Site officiel des développeurs Apple, technique pour lire et écrire dans des fichiers http://developer.apple.com/library/ios/#documentation/FileManagement/Conceptual/FileSystemProgrammingGUide/TechniquesforReadingandWritingCustomFiles/TechniquesforReadingandWritingCustomFiles.html

[40] tcpdump.org, types de protocoles liaisons de données http://www.tcpdump.org/linktypes.html

[41] wiki.wireshark.org, informations concernant le format libpcap http://wiki.wireshark.org/Development/LibpcapFileFormat

[42] wikipedia.org, définition du protocole Logical Link Control (802.2) http://en.wikipedia.org/wiki/Logical_link_control

[43] wikipedia.org, informations concernant ICMP https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

[44] wikipedia.org, trames 802.11 de type control http://en.wikipedia.org/wiki/IEEE_802.11#Control_Frames

[45] wikipedia.org, trames 802.11 de type management http://en.wikipedia.org/wiki/IEEE_802.11#Management_Frames

[46] winpcap.org, informations concernant le format libpcap http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html.

[47] wireshark.org, informations concernant le logiciel wireshark http://www.wireshark.org/about.html

[48] 802.11 Wireless Networks the definitive guide, livre écrit par Matthew S. Gast, édition O'REILLY, 2ème édition

12 Liste des symboles et abréviations utilisées

[1] **A-MPDU** : un a-MPDU représente plusieurs MPDU agrégés sous le même en-tête.

[2] **A-MSDU** : MSDU directement opéré au niveau de la couche MAC, contrairement au MSDU standard transmis à la couche supérieur (généralement LLC, Logical Link Control). C'est des trames agrégées, c'est-à-dire qu'il s'agit d'une trame avec un seul en-tête contenant plusieurs trames destinées au même client.

[3] **AP** : Access Point, station wifi auquel centralisant un réseau wifi en mode infrastructure. D'autres stations vont s'associer avec un access point et l'utiliser comme intermédiaire pour toute communications.

[4] **ASCII** : American Standard Code for Information Interchange, jeu de caractères permettant l'échange de textes.

[5] **BSS** : Basic Service Set, élément du standard 802.11 fournissant le bloc de construction d'un WLAN 802.11. (mode infrastructure de 802.11)

[6] **IBSS** : Independant Basic Service Set, BSS indépendant. (mode ad-hoc de 802.11)

[7] **ICMP** : Internet Control Message Protocol, protocole de la suite IP permettant de véhiculer des messages de contrôle et d'erreurs.

[8] **IDE** : Integrated Development Environment, ensemble d'outil pour augmenter la productivié des développeurs de logiciels. Les IDE comportent généralement un éditeur de texte destiné à la programmation ainsi que des fonctions permettant de démarrer la compilation, l'édition des liens et l'exécution.

[9] **IEEE** : Institute of Electrical and Electronics Engineers, association professionnelle à but non lucratif ayant pour but de promouvoir la connaissance dans le domaine de l'ingénierie.

[10] **IEEE 802.11** : ensemble de norms concernant les réseaux sans-fils mis au point par l'IEEE.

[11] **iOS** : anciennement iPhone OS, iOS est le système d'exploitation mobile dérivé de MAC OS X développé par Apple destiné aux iPod nouvelle génération, iPad et iPhone.

[12] **IP**: Basé sur ARPA Internet Protocol, l'Internet Protocol actuel a été conçu pour permettre la communication de deux systèmes interconnectés. L'Internet Protocol permet la transmission de datagrammes d'une source à une destination identifiés par une adresse IP de longueur fixe.

[13] **LIBPCAP** : libpcap peut représenter le format standard utilisé pour formater une capture de paquets sur un réseau. Libpcap peut également signifier une librairie existante permettant la capture de paquets sur un réseau.

[14] **LLC** : Logical Link Control, sous-couche supérieure de la norme 802 permettant de fiabiliser le protocole MAC grâce à du contrôle d'erreur et de flux.

[15] **M File** : fichier corps d'une classe Objective-C.

[16] **MPDU** : PDU échangé entre des entités MAC.

[17] **MSDU** : MAC Service Data Unit, comprenant une en-tête et une en-queue de sécurité si la trame est protégée.

[18] **NIC** : Network Interface Controller, essentiellement une carte réseau.

[19] **OS X** : Operating System 10. Système d'exploitation construit sur une structure UNIX et distribué par Apple.

[20] **OSI** : Open System Interconnexion, standard de communication en réseau touchant tous les systèmes informatiques. Le modèle OSI spécifie un modèle en couches par lesquelles les informations sont encapsulées après des en-têtes permettant le transport de ces informations.

[21] **OUI** : Organizationnaly Unique Identifier, identifie une organisation de manière unique.

[22] **Packet Viewer** : Table permettant l'affichage des paquets dans l'application délivrée.

[23] **PDU** : Protocol Data Unit, ensemble des informations échangées entre couches du modèle OSI. Un PDU de couche 3 est un **paquet**, un PDU de couche 2 est une **trame**.

[24] **QoS** : Quality de service

[25] **RF** : Radio Frequency, taux d'oscillation entre 3 kHz et 300 GHz correspondant à la fréquences des ondes radio.

[26] **RX/TX** : Réception et Transmission

[27] **SDK** : Software Developement Kit, ensemble d'outils permettant aux développeurs de créer des applications de type défini (par exemple pour iOS ou Android).

[28] **SSID** : Service Set Identifier, nom d'un réseau sans fil selon la norme 802.11.

[29] **STA** : Station comprenant la norme 802.11. Cela peut être un access point, un téléphone, un laptop, etc.

[30] **TCP** : Transmission Control Protocol, protocole de transport fiable de la couche quatre du modèle OSI.

[31] **TCP/IP** : La suite TCP/IP est l'ensemble de protocole utilizes pour le transfert de données sur Internet.

[32] **TU :** Time Unit utilisé pour certains champs de la norme 802.11. Une unité de temps TU est égal à 1 seconde / 1024e-6.

[33] **UDP** : User Datagram Protocol, protocole de transport non fiable de la couche quatre du modèle OSI.

[34] **XIB** : User Interface Ressource File, fichier permettant de construire une vue de l'interface graphique d'un projet Xcode.

13 Liste des figures

14 Annexes

14.1 Importer le projet

Lors du rendu du projet, une archive nommée « PDB.MM code.zip » a été fournie. Cette archive contient le projet Xcode de l'application. Cette archive contient le dossier « PDB-Network Analyzer » qui une fois ouvert contient de multiples fichiers dont « PDB-Network Analyzer.xcodeproj » qui est le projet Xcode.

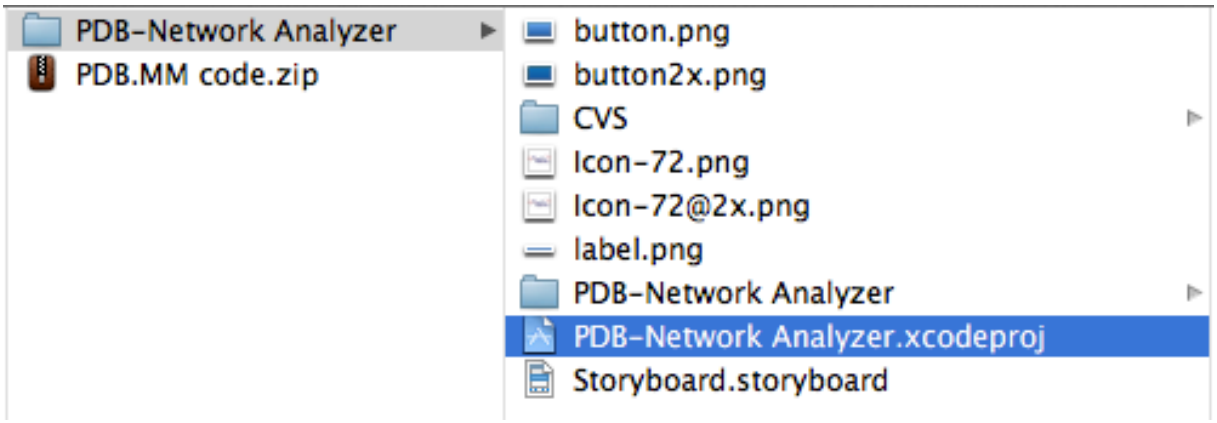

Figure 152 - Archive extraite

Il suffit d'ouvrir ce fichier en double cliquant dessus (il faut avoir Xcode d'installé et mis-à-jour évidemment). Xcode devrait afficher qu'il indexe les fichiers et après quelques minutes au plus, le message « Build Succeded » devrait s'afficher.

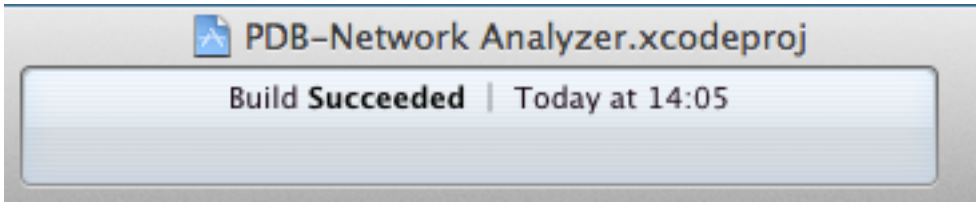

Figure 153 - Build succeeded

Si ce n'est pas le cas, le menu « Product > Clean » puis « Product > Build » devrait régler le problème. Une fois que ce message s'affiche, il suffit de sélectionner le simulateur iPad 6.1 et cliquer sur le bouton « Run ».

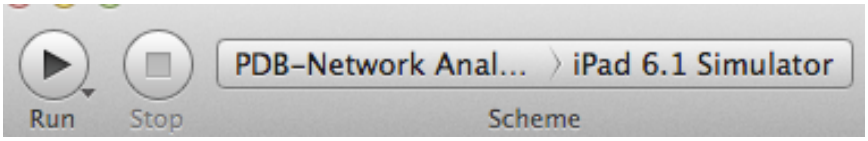

Figure 154 - Séléction de la cible

Il est également possible d'exécuter l'application sur un iPad réel mais c'est un peu plus compliqué que d'utiliser le simulateur. L'annexe suivante, « Licence de développement iOS » décrit le processus nécessaire.

Une fois l'application lancée et fonctionnant sur le simulateur, il faut lui ajouter des captures à lire. Le dossier « PCAP Exemples » de l'archive fournie contient des exemples de captures avec des protocoles supportés par l'application. Les fichiers pcap doivent être copiés dans le dossier de l'application du simulateur iPad.

~/Library/Application Support/iPhone Simulator/6.1/Applications/<Application_Home>/Documents

La variable <Application_Home> est automatiquement générée par Xcode. Si vous n'avez qu'un seul projet Xcode alors il n'y aura qu'un dossier possible. Si ce n'est pas le cas, il faudra ouvrir chaque dossier et regarder si le fichier « airpCap.app » est présent. Si c'est le cas, c'est le bon dossier et vous pouvez sauvegardez les captures dans le dossier « Documents ». Une fois copiées dans le dossier Documents, les captures sont disponibles dans l'application. Si l'application était déjà en marche lors de la copie des captures, il suffit de tirer le menu vers le bas pour le rafraîchir et voir les captures s'afficher.

14.2 Licence de développement iOS

Apple applique une politique de restriction importante concernant le développement sur sa plateforme. Tout d'abord il faut posséder un Mac avec OS X comme système d'exploitation. Cela est obligatoire car le développement s'effectue uniquement sur le logiciel Xcode. Bien qu'il soit théoriquement possible de développer pour iOS sur un autre IDE (Integrated Development Environment) qu'Xcode, cela est fortement déconseillé car il n'existe que deux implémentations de l'environnement pour Objective-C. Le premier est celui développé par l'équipe de la société NeXT, fondée par Steve Jobs après avoir été viré d'Apple dans les années 80. Ce compilateur est maintenant inclus dans Xcode. Le second est une implémentation libre, appelée GNUstep, qui est très instable.

Une fois que l'on a Xcode et le SDK iOS contenant toutes les API nécessaire à iOS, il est alors possible de commencer et créer des projets d'applications iOS. A ce moment-là, il est uniquement possible de tester ses applications sur le simulateur directement inclus dans Xcode. Si l'on veut pouvoir installer les applications directement sur du matériel ou si l'on veut les mettre en vente sur l'App store, il sera nécessaire d'obtenir une licence de développement Apple.

Figure 155 - Etapes d'enregistrement

https://developer.apple.com/ios/my/overview/index.action

La première étape nécessaire est de posséder un compte chez Apple. Une fois que l'on s'est enregistré, et que l'Apple ID est validé, il faut passer par un « team admin » possédant une licence afin de nous autoriser à développer et distribuer une application. Je suis passé par Mr. Joël Gonin, responsable de l'infrastructure IT de la HEIG-VD, qui m'a ajouté au groupe de l'école.

Toute l'administration du compte développeur se fait à l'adresse suivante :

https://developer.apple.com/ios/my/overview/index.action

Lorsque l'on a accepté l'invitation à rejoindre le groupe, il est obligatoire de faire une demande de certificat sur le Mac avec lequel on va développer. Cela se fait grâce à l'utilitaire « Trousseau d'accès » sur Mac OS X.

Figure 156 - Demande de certificat

https://developer.apple.com/ios/my/certificates/howto.action

Après que la demande de certificat ait été générée, il faut l'uploader sur le site et la faire valider par l'administrateur d'équipe. Le certificat est alors issu et disponible au téléchargement.

Figure 157 - Le certificat est issus

L'étape suivante consiste à ajouter l'appareil sur lequel on va développer. Pour ajouter un device, il suffit de fournir un nom qui le représente, le modèle précis et l'UDID. Dans le cas de ce projet de Bachelor, le développement se fait uniquement sur iPad. Ainsi, j'ai ajouté mon iPad personnel :

Nom	Modèle	UDID
MathieuMeylaniPad	iPad 4th generation	82e8695abaeb9817737ddd2880619b15d99bd7a7

Figure 158 – Ajout du device

https://developer.apple.com/ios/my/devices/howto.action

Pour finir l'étape de configuration du profile il faut encore effectuer un point : générer un AppID. L'AppID est une chaîne identifiant l'application de manière unique. Cela permettra par la suite de distribuer l'application.

Figure 159 – AppID

https://developer.apple.com/ios/my/bundles/howto.action

Il est nécessaire d'utiliser cet AppID en tant que « Bundle Identifier » du projet de développement de l'application si l'on veut être capable de l'installer sur le device ajouté précédemment. Si le Bundle Identifier n'est pas exactement le même que l'AppID, on obtiendra une erreur à la compilation.

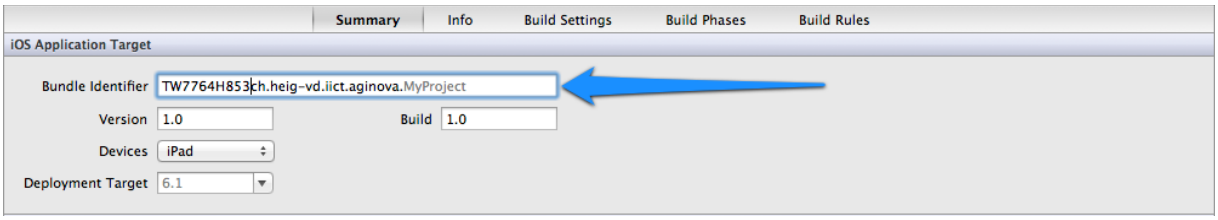

Figure 160 - Bundle Identifier d'un projet

L'AppID du projet a été créé. L'étoile que l'on voit ci-dessous est le un *wild card caracter* et sera remplacé par le nom de l'application

AppID

TW7764H853.ch.heig-vd.iict.aginova.*

Une fois que l'AppID est généré, il faut mettre en relation tout ce que l'on a fait jusqu'à maintenant : l'AppID, le certificat ainsi que le device. Cela est fait grâce au « Provisioning Profiles ». Le Provisioning Profile est une collection d'entités digitales qui lie de manière unique les développeurs et les appareils à une équipe de développement iOS autorisée.

Lorsqu'il est crée, il est accessible au téléchargement sur le site de développement d'Apple et doit être installé sur chaque appareil sur lesquels on veut exécuter le code de notre application. Le Provisioning Profile contient le certificat, l'identifiant unique de l'appareil ainsi que l'AppID.

Pour installer un Provisioning Profile, il faut connecter l'appareil au Mac. Ensuite, ouvrez Xcode et lancer l'*Organizer* à partir du menu *Window*. Dans la section *Devices* on trouve l'appareil que l'on vient de connecter à l'ordinateur. Il ne reste plus qu'à ajouter le *Provisioning Profile*, téléchargé précédemment, dans le menu *Provisioning Profiles* en le faisant simplement glisser ou en utilisant le bouton + *Add*.

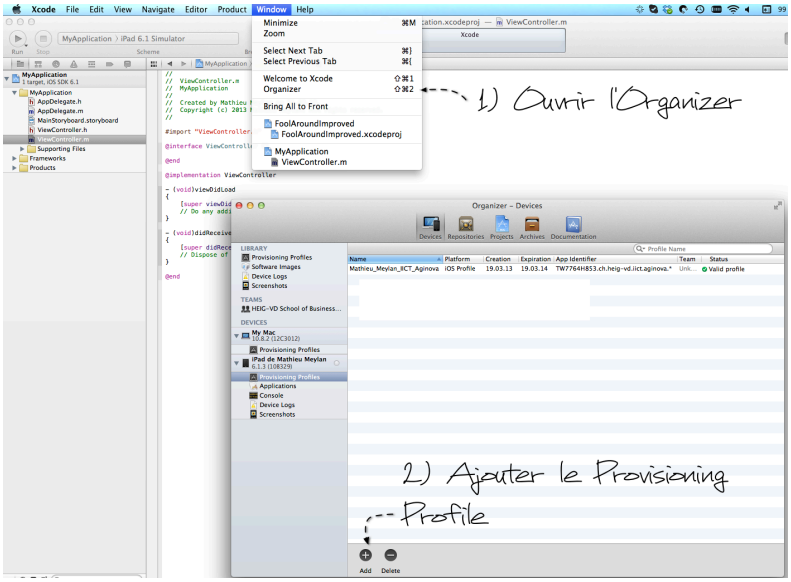

Figure 161 - Organizer

Lorsque le certificat et le profil sont installés, il est possible de lancer l'application sur l'appareil en simplement le sélectionnant dans la liste à côté du bouton *play*.

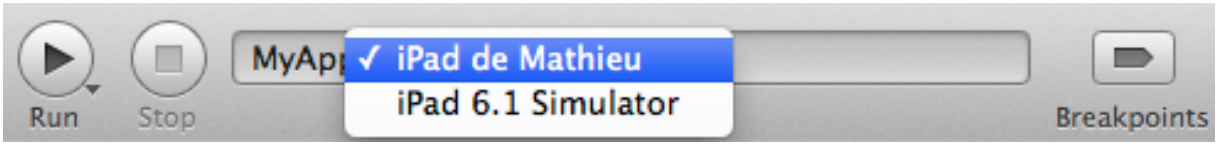

Figure 162 - Sélection de la destination

14.3 Normes et diagrammes

Des diagrammes UML sont présents dans ce rapport et la norme stricte UML n'étant pas adaptée à Objective-C, cette section est là pour expliquer les différents symboles que l'on trouve dans ces diagrammes afin de les comprendre totalement.

Tout d'abord les classes (ou interfaces) sont représentées par un rectangle et leur nom se trouve dans le premier sous-rectangle du haut. Lorsque le nom d'une classe est suivie d'un *<name>* cela signifie qu'elle implémente le protocole entre les symboles < et >.

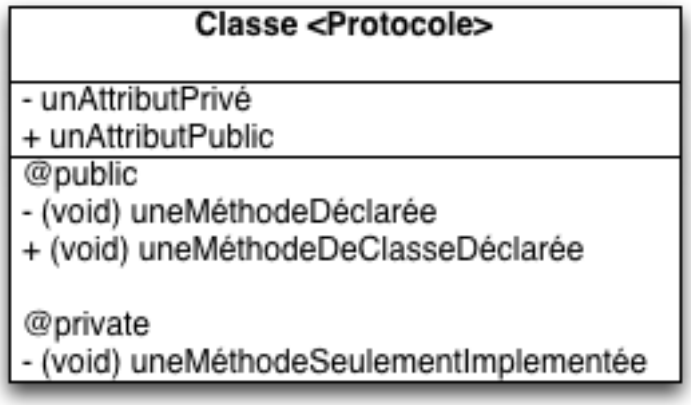

Figure 163 - Diagramme UML du rapport

On voit sur la figure ci-dessus que le deuxième rectangle d'une classe est réservé aux attributs. Les attributs d'une classe en Objective-C sont des variables instances ou des propriétés. Les propriétés sont publiques et on peut y accéder en dehors de la classe via des getters/setters. Les propriétés sont représentées avec un +. Les variables d'instances sont encapsulées dans les classes et on ne peut y accéder directement depuis l'extérieur. Ces variables sont représentées avec un -.

Ces mêmes signes n'ont pas la même signification pour les méthodes. Le + signifie que c'est une méthode de classe, qui peut donc être appelée directement sans avoir besoin d'instancier un objet d'une classe. Le - représente les méthodes classiques : méthode d'instances. Ces méthodes sont appelées sur un objet instance d'une classe. Les méthodes peuvent être privée ou publiques et se trouvent dans la section appropriée du rectangle (*@public* ou *@private*).

15 Journal de travail

Semaine 09 [25.02.2013] → Semaine 31 [02.08.2013]

Le journal de travail présenté ci-dessous comprend le travail effectué lors des jours de travail planifiés dans l'emploi du temps de l'école, c'est-à-dire deux jours par semaines lors de la première partie du projet et ensuite cinq jours par semaine lors des six dernières semaines.

Les heures de travail effectuées hors des heures planifiées par l'école sont difficiles à « quantifier » en termes de journées ou d'heures de travail étant donné qu'elles ont été effectuées plus ou moins sans arrêt lors de la durée du travail de bachelor. Par exemple, j'aurais pu passer une soirée à lire des articles sur le protocole wifi mais sans vraiment compter le temps que j'y ai passé. Pour cette raison, seules les journées planifiées dans l'emploi du temps sont notées dans le journal de travail avec la description du travail effectué lors de cette journée ainsi qu'hors des « heures de travail ».

Date	Travail effectué
Semaine 09 Jeudi 28.02.13	J'ai commencé à m'intéresser au sujet (bien que n'aillant pas encore d'informations dessus) en m'abonnant au cours « Developing Applications for iOS » de Stanford (Fall 2011) sur iTunes University. Cela me permet d'apprendre la programmation sur iOS.
Semaine 10 Jeudi 07.03.13	Aujourd'hui, j'ai commencé la planification initiale en mettant les « titres » des différentes tâches que j'aurais à effectuer en divisant déjà l'application en « gros modules » qui seront par la suite divisés en plus petites parties encore. J'ai également commencé l'analyse de l'apparence qu'aura l'application.
Semaine 11 Mardi 12.03.13	J'ai fait les « sketchs » de l'apparence de l'application en mode paysage et portrait (analyse). J'ai commencé à tester les « SplitViewController » iOS permettant de diviser l'interface de l'application en deux fragments. J'ai également fait la procédure nécessaire pour rejoindre la licence de l'école de développeur iOS afin de pouvoir

Date	Travail effectué
	développer directement sur le matériel.
Semaine 11 **Jeudi 14.03.13**	Finition de la planification et début de la documentation concernant l'enregistrement en tant que développeur auprès d'Apple. J'ai continué à apprendre la programmation Cocoa pour iOS ainsi que l'Objective C en me concentrant sur les « SplitViewController ». A faire la semaine prochaine : • Finir la planification • Finir l'annexe sur l'enregistrement en tant que développeur Apple.
Semaine 12 **Mardi 19.03.13**	Pendant toute la journée je me suis documenté sur la programmation Cocoa en Objective C en me focalisant sur l'iPad en particulier. Les tutoriaux du site http://www.appcoda.com/tutorials/ sont très bien expliqué et à jour pour les nouvelles version de Xcode (4) et d'iOS (5 et 6). Cela m'as permis d'être bien familiarisé avec le paradigme « delegate » très fréquent sous iOS. J'ai également fini l'annexe sur l'enregistrement en tant que développeur Apple dans la documentation. ✓ Finir l'annexe sur l'enregistrement en tant que développeur Apple.
Semaine 12 **Jeudi 21.03.13**	J'ai consacré toute la matinée de cette journée à finir la planification en rectifiant les titres des différents étapes du projet et en planifiant le temps consacré pour chaque point. ✓ Finir la planification

Date	Travail effectué
	Le reste de la journée, j'ai modifié le schéma (ou plutôt sketch) de l'apparence de l'application en mode portrait (après discussion avec Aginova, le mode paysage n'est pas une priorité). J'ai modifié le schéma d'après les discussions avec Aginova et appliqué les changements qu'ils désiraient. J'ai également pu compléter le schéma grâce à la première copie du cahier des charges que m'a fourni Karl. A faire la semaine prochaine : • Terminer la recherche sur la programmation Cocoa iOS et être à l'aise avec Xcode. • Commencer à aborder le problème de transfert de fichier sur l'application • Commencer le design du type « menu poussant la fenêtre (type youtube, facebook, ..) » comme conçu dans le schéma de l'apparence de l'application.
Semaine 13 **Mardi 26.03.13**	J'ai téléchargé et lu les « Directives de projet de bachelor 2013 » de M. Sanchez publiées sur le réseau et ai mis mon rapport à jour en conséquences (Ajout du nom en haut à droit des pages, ajout de toutes les sections demandées, etc.) J'ai commencé à coder le design « menu » poussant en choisissant la librairie ECSlidingViewController (https://github.com/edgecase/ECSlidingViewController). Mon choix s'est porté sur cette librairie car elle est compatible iOS5, elle est compatible iPad et est open source. ✓ Commencer le design du type « menu poussant la fenêtre (type Youtube, Facebook, ..) » comme conçu dans le schéma de l'apparence de l'application. J'ai pu déjà maîtriser la librairie et faire une application test avec ce menu. Il est ainsi prêt à être installé dans le design de l'application du projet. Bien que l'apprentissage de la programmation Cocoa ne sera jamais « finie » (dans le sens ou l'étudier à 100%

Date	Travail effectué
	n'est pas le but du projet), je me sens maintenant suffisamment à l'aise avec l'environnement Xcode et le langage Objective-C. J'ai maintenant terminé ce que j'appelle l'introduction à la programmation d'application pour iOS qui était jusqu'alors nouvelle pour moi. Le reste des connaissances nécessaires s'aciérera tout en continu durant l'avancée du projet. ✓ Terminer la recherche sur la programmation Cocoa iOS et être à l'aise avec Xcode.
Semaine 13 Jeudi 28.03.13 & Semaine 14 Jeudi 04.04.13	J'ai rédigé la documentation pour l'utilisation de la librairie permettant une architecture en couche avec un menu de navigation caché sous la page principale J'ai par la même occasion écrit la documentation de l'apparence de l'application. Après discussion avec Karl et Thierry d'Aginova, on a pris certaines décisions importantes concernant le projet : • Utilisation d'iOS 6 • La double orientation paysage / portrait n'est pas une priorité, il faut que l'application fonctionne. Malheureusement je ne suis pas arrivé au bout de toutes les tâches que j'aurais dû faire cette semaine. J'avais l'intention de commencer à aborder le transfert de fichier sous iOS mais à la place j'ai fini le menu de navigation caché et en ait rédigé la documentation. J'ai compensé en me concentrant sur le sujet de l'apparence de l'application en en faisant également la documentation déjà incluse dans le rapport. A faire la semaine prochaine : • Aborder le problème de transfert de fichier sur l'application • Commencer le traitement des fichiers pcap

Date	Travail effectué
Semaine 15 Mardi 09.04.13 & Jeudi 11.04.13	Le sujet principal de cette semaine est de développer le code permettant de transférer des fichiers sur l'application. J'ai codé le menu de l'application pour qu'il affiche tous les fichiers présents dans le dossier « Documents » propre à l'application, il est possible de recharger le menu en le tirant vers le bas afin de pouvoir vérifier en tout temps si un nouveau fichier a été ajouté. De plus, cette fonctionnalité permet également de transférer en temps réel des fichiers depuis iTunes sur l'application et de simplement tirer le menu vers le bas pour voir le nouveau fichier s'afficher, sans avoir besoin de relancer l'application ou de recharger une vue. J'ai également implémenté la possibilité de faire glisser le doigt sur la ligne d'un fichier afin de faire apparaître le bouton « delete » qui va supprimer le fichier et le retirer du menu sous pression. La gestion des fichiers est donc déjà terminée, il faut maintenant gérer la lecture et le parsing des fichiers pcap. Selon le planning, j'ai bien commencé le traitement des fichiers PCAP en lisant la documentation sur le format libpcap du site http://wiki.wireshark.org/Development/LibpcapFileFormat. ✓ Aborder le problème de transfert de fichier sur l'application ✓ Commencer le traitement des fichiers PCAP La semaine prochaine : • Continuer les recherches sur le format libpcap et le meilleur moyen de parser des tels fichiers sous iOS.

Date	Travail effectué
Semaine 16 Mardi 16.04.13	Malade
Semaine 16 Jeudi 18.04.13	J'ai cette semaine continué les recherches sur le libpcap et suis arrivé à des conclusions qui seront définitives d'après moi. La librairie libpcap de http://www.tcpdump.org/ est la librairie idéale pour la capture et le traitement de paquet au format libpcap. C'est une librairie C et ainsi compatible Objective-C. Malheureusement, à cause de la politique de sécurité d'Apple sur les appareils iOS, il n'est pas possible d'importer cette librairie et de l'utiliser dans une application sans que l'appareil ne soit jailbreaké. Il me faudra donc développer « from scratch » le parseur lisant les fichiers pcap et implémenter les différentes couches réseaux. Le développement du parseur est donc la prochaine étape et est, d'après moi, la plus importante. D'après les demandes d'Aginova, le parseur doit être évolutif et doit permettre par la suite le traitement de paquets en flux continu. De plus, toute l'application se base sur les paquets capturés, je dois donc créer une structure solide pour que l'application puisse évoluer de manière simple sans problèmes.
Semaine 17 Mardi 23 & Jeudi 25.04.2013	Cette semaine j'ai commencé le développement du parseur de fichier PCAP à partir de zéro après les constatations de la semaine dernière. Afin d'avoir un parseur évolutif et possible d'utilisation par la suite sur des sockets plutôt que des fichiers, j'ai fait des recherches sur le meilleur moyen de lire des fichiers. La solution que j'ai choisie est l'utilisation des classes NSInputStream et NSOutputStream (de NSStream). Ces classes permettent la lecture de fichier de manière asynchrone mais également la lecture sur des sockets de la même manière ! Si Aginova veut plus tard développer son interface matérielle sniffant et envoyant au format PCAP à l'application mobile, c'est la bonne méthode à adopter. ✓ Déterminer le meilleur moyen de parser les fichiers PCAP et commencer l'implémentation

Date	Travail effectué
	Grâce à ces classes je peux maintenant lire les fichiers PCAP. La prochaine étape est maintenant de les parser et d'en comprendre les informations extraites. Chaque fichier pcap possède un en-tête global possédant des informations importantes. J'ai implémenté le parseur de cet en-tête et l'application la lit maintenant correctement. L'information la plus importante lue est le champ « magic_number ». Ce champ contient la valeur 0xa1b2c3d4 ou 0xd4c3b2a1 suivant le programme qui l'a écrit. C'est en fait l'ordre des bytes qui est spécifié ici, Big Endian ou Little Endian. Afin de ne pas avoir de problème avec l'ordre des bytes, il sera plus simple d'implémenter ma propre classe « Buffer » qui lit les bytes dans l'ordre voulu. La semaine prochaine : • Automatiser la lecture de bytes suivant l'architecture du programme qui a écrit le fichier pcap (Little Endian ou Big Endian) • Lire les en-têtes des paquets des fichiers PCAP
Semaine 18 Mardi 30.04.2013	Je suis allé à l'ambassade des Etats-Unis pour faire mon visa.
Semaine 18 Jeudi 02.05.2013	Il faut maintenant automatiser la lecture des bytes des fichiers pcap et se focalisant sur le problème de « L'Endianess ». Pour parer à cela, j'ai développé la classe UnsignedInt8Buffer dans laquelle on peut encapsuler des bytes. La classe contient une propriété « swap » qui spécifie si l'on lit les bytes de manière Little ou Big Endian. Cette classe simplifie nettement la lecture de bytes d'un fichier PCAP car je n'ai maintenant plus besoin de m'occuper de la manière dont les bytes sont écrits, tout est géré ici.

Date	Travail effectué
	✓ Automatiser la lecture de bytes suivant l'architecture du programme qui a écrit le fichier pcap (Little Endian ou Big Endian) J'ai eu quelques difficultés ici à imaginer une manière de lire les bytes car chaque champ de chaque protocole sera de taille différente, on aura beaucoup d'octets représenté en uint8_t mais aussi des mots de uint16_t où double mots de uint32_t ou les mêmes types mais signés. Je ne savais pas si c'était le mieux d'utiliser les types standard C ou bien d'encapsuler dans des objets Objective-C. J'ai pris la décision d'utiliser les types standard C (uint8_t, uint16_t, uint32t, int8_t, etc.) car ils me semblent bien plus léger plutôt que d'encapsuler quelques bytes dans des gros objets de type NSData.
Semaine 19 Mardi 07.05 & Jeudi 09.05.2013	J'ai continué l'écriture du parseur de fichiers pcap. Maintenant que la lecture de l'en-tête globale, je me suis penché sur l'en-tête ajouté par le standard pcap avant chaque paquet. Le parseur comprend maintenant cette en-tête et va pouvoir tirer des informations utiles, telles que la taille du paquet suivant l'en-tête. ✓ Lire les en-têtes des paquets des fichiers PCAP Aginova a demandé dans le cahier des charges une focalisation sur les paquets wifi, j'ai donc commencé les recherches sur le format radiotap (protocole numéro 127 de couche 2 dans le standard libpcap). Radiotap est assez compliqué et m'a pris pas mal de temps à comprendre. Il utilise des bitmap sur plusieurs octets afin de spécifier les champs qui suivront (par exemple le canal, la vitesse de transmission, la puissance de l'antenne, le bruit, etc.). A faire la semaine prochaine : • Continuer le codage du parseur de paquets de type radiotap

Date	Travail effectué
Semaine 20 Mardi 14.05 & jeudi 16.05.2013	J'ai continué, cette semaine, le codage du parseur de paquets radiotap. Il est maintenant terminé et comprend quasiment tous les champs du protocole. Le problème est qu'il y a pas mal de bits de la bitmap spécifiant les champs apparaissant dans l'en-tête qui ne sont pas définis et qui sont simplement « prévu pour l'avenir » J'ai donc laissé le codage ouvert pour ces champs qui seront peut-être défini dans des versions de radiotap à venir. Ce qui m'a pris du temps cette semaine a été l'écriture de toutes les « descriptions » des champs de l'en-tête. Pour prendre un exemple, je lis par exemple la valeur « 0.5 » dans les octets réservés au champ « Rate ». Ce qui va prendre du temps c'est que l'on ne peut pas afficher à l'utilisateur un simple « 0.5 » sous le byte 18 et 19 (exemple tiré au hasard). Non, il faut lui écrire une chaîne lisible disant : « Rate : 2 Mb/s ». Pourquoi 2 ? Parce que le champ Rate dans le protocole radiotap est exprimé en unité de 500kbit/s. C'est un exemple montrant que le travail vient surtout à la transformation de valeures lues en données lisible par un utilisateur. Et le même traitement personnalisé doit être effectué pour quasiment tous les champs. J'ai donc dû non seulement recoder le protocole radiotap complet afin que l'application puisse le comprendre, mais également coder des chaînes pour que l'utilisateur lui puisse le comprendre. A faire la semaine prochaine : • Jusqu'à maintenant j'affichais les résultats du parseur dans la console. Maintenant que le premier protocole a été codé, il faut développer la table permettant de contenir et d'afficher sur la tablette toutes les informations à l'utilisateur.
Semaine 21 Mardi 21.05.2013	J'ai commencé mes recherches sur la possibilité des créer une table de type UITableViewController « expandable » (type de tableau extrêmement utilisé sous iOS et quasiment la seule possibilité d'afficher des informations en texte de manière ordrée). Le but est de pouvoir afficher tous les paquets sur une ligne avec quelques informations très importantes. On doit ensuite être capable d'étant la ligne en cliquant dessus et de voir

Date	Travail effectué
	ensuite chaque couche que le paquet « contient ». Chaque couche pourrait ensuite être cliquable pour pouvoir l'étendre et voir en détails chaque champ du protocole en question. Pour prendre un exemple concret, imaginons un paquet reçu à 23h30 : [paquet 1 – 23h30] <- click (1 ligne dans notre tableau) [paquet 1 – 23h30] (5 lignes dans notre tableau) [DNS] [IP] <- click va afficher la version, les adresses src et dst, etc. [Ethernet] [Global Info] La table pourrait ainsi être manipulable à souhait par l'utilisateur afin de voir les champs qui l'intéressent. J'ai trouvé une librairie permettant de faire ce genre de table et ait passé la plupart de ce jour à la tester. Malheureusement, cette librairie ne permet qu'un seul niveau d'imbrication et ce n'est pas suffisant pour cette application.
Semaine 21 **Jeudi 23.05.2013**	Je me suis basé sur le concept d'arbres afin de créer une table avec un niveau d'imbrications a priori sans limite afin de pouvoir y afficher de manière dynamique n'importe quel protocole. J'ai passé pas mal de temps et ait terminé une fonction récursive permettant l'insertion dans ma table (ou mon arbre) en lui passant un tableau. La fonction indente automatiquement d'un niveau a chaque fois qu'elle trouve un tableau dans le tableau et lie les parents et les enfants de l'arbre. Il me suffit donc maintenant de retourner un tableau correctement construit avec les informations sur un protocole à cette fonction afin de pouvoir les afficher de manière simple à l'écran de la tablette. ✓ Développer une table dynamique avec plusieurs niveaux d'indentation, de réduction et d'augmentation

Date	Travail effectué
	Cela fait plusieurs semaines que je ne fait que coder et il faut qu'à partir de la semaine prochaine j'écrive le rapport sur tout ce que j'ai fait jusqu'à maintenant afin de pouvoir respecter le rendu du rapport intermédiaire. J'aimerai arriver à afficher les paquets parsés radiotap dans la table de manière indentée et écrire le rapport intermédiaire jusqu'à là avant la fin de la partie du travail à temps partiel. Les semaines prochaines seront donc consacrées à la rédaction du rapport intermédiaire.
Semaine 22 Mardi 28.05.2013	J'ai passé la première journée de cette semaine à mettre la table dynamique ensemble avec le projet Xcode principale afin de pouvoir afficher les paquets dans la table. J'ai rencontré plusieurs problèmes avec le « delegate » de la table mais ai pu les régler assez rapidement. La table permet d'afficher les paquets avec en-tête radiotap (seul protocole implémenté jusqu'à maintenant). J'ai également résolu différents bugs concernant la lecture de tous les paquets d'une capture. Il y avait un bug ou l'application tentait de lire sur le stream même après que celui-ci soit terminé.
Semaine 22 Jeudi 30.05.2013	J'ai continué à résoudre divers bugs et problèmes de l'application et ait passé une bonne partie de la journée à réorganiser l'architecture des classes et des fichiers de l'application. Je n'étais pas satisfait de la façon dont les classes étaient organisées. J'ai donc réorganisé tout le système de parseur de fichiers pcap et la manière dont les paquets sont lus et stockés en divisant clairement les différentes couches. J'ai découvert un bug important lorsque l'on lit un grand fichier pcap contenant beaucoup de paquets (plus de 1000) et que la table prend plus de place que l'écran. Je n'ai pas pu le résoudre aujourd'hui et je devrais le faire avant de pouvoir continuer la rédaction du rapport qui était déjà planifiée pour cette semaine.

Date	Travail effectué
Semaine 23 Mardi 04.06.2013	J'ai pu résoudre le bug découvert la semaine prochaine en rechargeant la table entière au lieu de supprimer ou d'insérer des lignes. Le bug venait du fait que lorsqu'une ligne n'est plus visible à l'écran, le système la désalloue afin d'améliorer les performances. Devoir recharger toute la table à surtout un coup esthétique, on perd l'animation. J'en ai profité pour réorganisé un peu mon code afin de multi-threader le traitement des fichiers. Maintenant lorsque l'on lit un fichier pcap, le parsing se fait dans un thread en background et le thread principal (gérant l'interface graphique) ne fait que mettre à jour la table. Ce n'est pas encore fluide car je recharge toute la table, mais ce détail pourra être réglé plus tard. J'ai pu enfin continuer à écrire sérieusement le rapport dans l'approche du délai du rapport intermédiaire et j'ai écrit le chapitre concernant l'environnement de développement, la présentation de la plateforme et les librairies utilisées.
Semaine 23 Jeudi 06.06.2013	J'ai continué l'écriture du rapport intermédiaire et me suis penché sur le chapitre « Analyse » décrivant le format des fichiers pcap, les différents en-têtes présents dans ce fichier etc. La semaine prochaine il faut que je continue l'écriture du rapport en finissant le chapitre « Analyse » et je dois couvrir tous le code que j'ai développé jusqu'à maintenant dans le chapitre « Conception/Réalisation ».

Date	Travail effectué
Semaine 24 & 25	Rédaction du rapport intermédiaire.
Semaine 26 Lundi 24.06.2013	Je commence l'ajout de la fonctionnalité permettant de voir le contenu des paquets brut. A la base il était prévu de faire une fenêtre à part pour afficher le contenu représenté en hexadécimal et en ascii mais au vu de la place encore disponible sur l'écran de l'iPad après l'affichage de la liste des paquets, il est préférable de le diviser afin d'avoir directement une vue sur le contenu d'un paquet en cliquant dessus dans la liste. Je me suis occupé de diviser l'écran et d'ajouter les deux vues, hexa et ascii.
Semaine 27 Mardi 25.06.2013	Il est très facile d'afficher le contenu brut d'un paquet dans sa totalité, maintenant le but est de pouvoir cliquer sur n'importe quel champ d'un protocole montré à l'écran et que la partie brute du paquet représentant cette valeur soit mise en évidence (fond mis en jaune). J'ai fait des recherches sur le meilleur moyen d'implémenter cette fonctionnalité. La conclusion est qu'il faut connaître la taille de chaque champ ainsi que son emplacement concret dans la trame que l'on est en train de lire. Grâce à ces informations on peut localiser en mettre en évidence la partie concernée. Le problème est qu'il va falloir le faire de manière dynamique pour des protocoles comme radiotap ou 802.11 car la taille de l'en-tête varie.

Date	Travail effectué
Semaine 28 Mercredi 26.06.2013	Implémentation des recherches effectuées le jour d'avant concernant l'affichage brut des données ainsi que la mise en évidence d'un champ. L'ajout de la classe « ProtocolField » permet d'implémenter cette fonctionnalité car elle encapsule la taille ainsi que l'offset du premier bit par rapport au début du paquet pour chaque champ. Cette fonctionnalité était un point complet du cahier des charges et j'ai pu la réaliser plus vite que prévu je pourrais donc commencer à traiter la norme IEEE 802.11 dès demain. ✓ Affichage du contenu brut des paquets A faire : • Implémentation du parser de paquets wifi 802.11
Semaine 29 Jeudi 27.06.2013	J'ai commencé aujourd'hui à lire et comprendre le format des trames 802.11 qui est assez complexe et surtout très complet. D'après les bits « Type » et « Subtype » du champ « Frame Control » de l'en-tête, il est possible d'avoir plus de 40 trames 802.11 avec l'en-tête variant légèrement des autres. Le codage de se protocole va durer plus d'une semaine je pense. De plus l'analyseur s'oriente prioritairement sur le wifi et il est important que l'architecture du parseur lisant ces trames soit solide et adaptable à toutes les autres fonctionnalités qui viendront s'ajouter dessus (comme les filtres ou les endpoints statistiques).
Semaine 29 Vendredi 28.06.2013	Le développement du parseur de trames wifi est commencé et je me suis occupé surtout du traitement des champs « type » et « subtype » mentionné précédemment. La semaine prochaine sera quasiment entièrement consacrée à la programmation de ce parseur mais également à la rédaction du rapport sur les fonctionnalités ajoutées qui ne figurent pas encore dedans. • Implémentation du parser de paquets wifi 802.11 • Compléter le rapport

Date	Travail effectué
Semaine 30 Lundi 01.07.2013	Je continue l'implémentation de la norme 802.11. Le traitement de chaque possibilité du champ subtype pour chaque possibilité du champ type est terminé et tous les cas sont traités. Le parseur peut donc maintenant reconnaître de quelle genre de trame wifi il s'agit exactement et pourra traiter chaque trame différemment. L'affichage de la trame en question dans le « packet viewer » est également fait et on voit maintenant claire de quel type de trame il s'agit (Beacon, Probe, Ack, etc.) Je me suis également occupé de corriger deux bugs du packet viewer concernant le l'affichage de la flèche montrant si le menu est étendu ou pas.
Semaine 30 Mardi 02.07.2013	Aujourd'hui j'ai plutôt continué la documentation concernant les parties qui ont été effectuées dernièrement. De plus j'ai ajouté de nombreux commentaires dans le code ainsi que les en-têtes complets des fonctions et méthodes dans la plupart des classes. Je n'ai pas pu ajouter les commentaires dans toutes les classes encore, mais une bonne partie est faite.
Semaine 30 Mercredi 03.07.2013	J'ai continué le code ou je m'étais arrêté lundi. Il est maintenant question de coder le parseur pour chaque « sous-trame » de type et sous-type différent. Il existe de nombreuses trames différentes : Probes, Beacons, Ack, etc. (environ 30 différentes) et chacune ou presque doit être traitée différemment car elle possède d'autres attributs différents que les autres, ou alors dans un ordre différent ou même avec une signification qui n'est pas la même qu'avec les autres trames. J'ai donc commencé par coder une classe qui regroupe plusieurs attributs et méthodes qui se retrouveront dans chaque type de trame wifi afin de simplifier le code et le traitement. Toutes les classes représentants les trames wifi en hériteront donc afin de rassembler les points communs. Le premier type de trame que j'ai traité est la

Date	Travail effectué
	trame de contrôle ACK. La classe de cette trame est terminée et le parseur marche. Il va falloir maintenant continuer et toutes les implémenter.
Semaine 30 **Jeudi 04.07.2013** **Vendredi 05.07.2013**	Toutes les trames de type « management » se ressemblent plus ou moins et peuvent être regroupée en une seule classe qui va permettre le traitement des données. J'ai passé ces deux jours à rechercher le meilleur moyen de regrouper et de traiter les trames wifi 802.11 de type management. La documentation officielle du standard 802.11 est très dense et difficile à traiter (2'793 pages) et je me suis surtout approché du format des trames de type management pour l'instant. Les trames de type management sont les beacons, probes, associations, authentifications et autres. J'ai surtout passé ces deux jours à étudier et à comprendre le format de ces trames en profondeur. Chaque trame a des champs différents et ils doivent tous être traités. De plus, les trames de type management peuvent comporter des données fixe ou « taggée » de longueur indéterminée suivant l'en-tête. Je me suis également penché sur le format de ces données. Comme prévu, l'implémentation de la norme 802.11 prend beaucoup de temps, cela fait une semaine que j'ai commencé et je risque de devoir continuer environ encore une semaine. • Continuer l'implémentation de la norme 802.11 Je n'ai malheureusement pas eu le temps de continuer l'écriture du rapport car les recherches sur le wifi ont été intensives. Je ne pense pas continuer le rapport avant d'avoir terminé le traitement de la norme 802.11.

Date	Travail effectué
Semaine 31 Lundi 08.07.2013	J'ai commencé à implémenter le parseur s'occupant des trames wifi de type management. L'application comprend maintenant l'en-tête management et interprète tous les champs. Certaines trames wifi de type management ont également une certaine quantité de données qui vont suivre l'en-tête. Dans wireshark, on trouve cela sous le nom de « fixed parameters » et « tagged parameters » dans la section « Wireless LAN management frame » d'un paquet. Chaque sous-type de trame management (beacon ou probe par exemple) possède un certain nombre de paramètres fixes, qui sont obligatoire, et de paramètres « taggé » qui sont optionnels. Je me suis occupé aujourd'hui d'écrire une classe qui va lire ces données management, en traitant tous les champs obligatoires pour chaque type de trame bien précis et qui va dynamiquement lire les champs optionnels. Les champs fixes sont bien lus et traités pour chaque trame précise, il reste maintenant à comprendre les paramètres dits « taggés ».
Semaine 31 Mardi 09.07.2013	J'ai continué le traitement des données de trames management et l'application comprend maintenant dynamiquement les éléments « taggés » insérés dans une trame management. Le problème est maintenant qu'il y a des centaines d'éléments différents qui peuvent être insérés dans une trame. Pour chaque élément, les bytes représentent une donnée différente, ou alors possèdent une architecture différente. Le traitement de la représentation de tous ces éléments serait un travail de bachelor en soi même, j'ai donc simplement implémenté les principaux pour que l'application comprenne la représentation des paramètres utilisés le plus souvent. J'ai constaté un bug dans l'interface graphique qui fait crasher l'application et je vais devoir repousser la continuation du traitement du wifi afin de résoudre ce bug.

Date	Travail effectué
Semaine 31 Mercredi 10.07.2013	Je me suis penché aujourd'hui sur le bug que j'ai trouvé hier. L'application crash lorsque je clique sur un champ en particulier d'un certain paquet d'une capture pcap que j'utilise pour faire mes tests. J'ai constaté que ce bug n'apparaît que lorsque je clique sur ce paquet dans cette capture. Il a donc été plus facile de localiser l'origine du problème. Le menu que j'utilise pour afficher les paquets est construit en arbre. Toutes les lignes, quel que soit leur niveau, sont contenues dans le même tableau. Pas de tableau imbriqué. La différenciation entre les lignes, afin de pouvoir indenter, est faite au niveau du « path » de chaque ligne. Le bug venait en fait d'un chemin identique entre deux lignes qui se passaient uniquement dans ce fichier pcap. C'est un cas extrêmement rare et je suis content d'être tombé dessus. J'ai pu le corriger en rajoutant un ID unique à chaque ligne et ce n'est plus le chemin qui est utilisé pour les différencier (car il peut arriver d'avoir deux chemins identique) mais l'ID. La résolution de ce bug m'a pris une bonne partie de la journée, mais j'ai quand même pu passer quelques heures à continuer le traitement des trames wifi. J'ai pu m'occuper aujourd'hui des trames particulières suivantes : CFEnd et CFEndAck.
Semaine 31 Jeudi 11.07.2013	J'ai continué à traiter le protocole wifi aujourd'hui et j'ai pu m'occuper de toutes les trames de type control restantes, à savoir : PSPoll, RTS et CTS. J'ai également traité le cas des trames wifi de type Data (qui est le dernier type). Les trames de types data ont toute un format plus ou moins similaires, ce qui facilite le traitement. Il y a seulement quelques différences notables, mais il ne m'a pas pris longtemps pour les traiter. Toutes les trames possible wifi sont maintenant comprise par l'application. Comme je l'ai cité précédemment, il y a des cas ou les valeurs ne sont pas interprétées, bien qu'elles soient lues et affichées dans leur format brut. Cela est dû à la densité de la norme IEEE 802.11 et aux priorités du projet définies par Aginova.

Date	Travail effectué
Semaine 31 Vendredi 12.07.2013	Visite d'IBM avec la summer university.
Semaine 32 Lundi 15.07.2013	Les trames de type « Data » de la norme 802.11 sont généralement suivies d'une en-tête LLC (Logical Link Control) lorsque celle-ci comprennent des données (flag data du champ « frame-control »). J'utilise le terme « généralement » car je suis tombé sur des exemple ou ce n'était pas une en-tête LLC mais Ethernet II suivant l'en-tête 802.11. D'après mes recherches, ce genre de cas peut arriver mais est plutôt rare. J'ai donc passé la journée à programmer le parseur du protocole LLC. J'ai rencontré plusieurs bugs avec le parsing de LLC et je me suis rendu compte que les problèmes rencontrés peuvent venir de deux raisons : La présence de padding avant l'en-tête LLC ou bien l'absence de CRC à la fin de la trame. D'après les recherches effectuées, la présence de padding et du CRC peut-être connu dans le champ « flag » de l'en-tête radiotap de la trame. J'ai donc passé du temps à adapter l'application pour qu'elle prenne en compte le padding et le CRC.

Date	Travail effectué
Semaine 32 Mardi 16.07.2013	Visite de Plug-and-Play et d'Intel.
Semaine 32 Mercredi 17.07.2013	L'application marche très bien pour les protocoles connus ainsi que les paquets bien formés de protocoles connus. Lorsque un protocole n'est pas implémenté et donc pas compris par l'application, le comportement observé est variable et quelque peu imprévisible. Même observation pour des trames de protocoles connus qui ont connus des erreurs de transmission. J'ai consacré la journée à gérer ce genre de cas. Pour cela j'ai recherché les points stratégiques de l'application ou il fallait pouvoir récupérer et traiter ces différentes erreurs. L'application est maintenant plus robuste et va gérer les protocoles inconnus et les trames malformées en affichant les données lu du paquet et en mettant la cellule en rouge.
Semaine 32 Jeudi 18.07.2013	Après discussion avec Karl d'Aginova, on a décidé de mettre la priorité sur l'implémentation de plus de protocoles pour les quelques semaines restantes. Il faut que je m'occupe des parseurs des protocoles IP, UDP, TCP et ICMP afin que l'on puisse avoir une application qui peut lire des captures du protocole de bas niveau jusqu'à dans les couches les plus hautes. J'ai commencé à implémenter le parseur du protocole IP aujourd'hui et il marche bien pour les vingt premier bytes de l'en-tête. Il me reste à traiter les options d'IP ainsi que de récupérer le contenu du champ « protocol » afin de pouvoir traiter correctement les données suivant l'en-tête IP.

Date	Travail effectué
	A faire la semaine prochaine : • IP • UDP • TCP • ICMP • Compléter le rapport sur l'avancement
Semaine 33 **Vendredi 19.07.2013**	Visite de Google.
Semaine 34 **Lundi 22.07.2013**	J'ai terminé le traitement du protocole IP. Le protocole est maintenant complètement supporté, avec les options et le protocole de couche supérieur est connu et le traitement pourra être effectué par la suite. ✓ IP J'ai également eu le temps d'implémenter complètement le protocole UDP aujourd'hui. Il est simple et ne possède pas beaucoup de champs, cela n'a pas pris beaucoup de temps. ✓ UDP

Date	Travail effectué
Semaine 34 Mardi 23.07.2013	J'ai remarqué un bug dans le protocole Radiotap pour certaine de mes nouvelles captures que j'utilise pour faire les tests. Après les recherches, j'ai pu localiser le bug, il arrivait lorsque le champ « flag » était égal à zéro. J'ai corrigé le bug. Je me suis également occupé des données restantes à la fin d'un traitement d'un paquet. En effet, jusqu'à maintenant, lorsqu'il restait des données au delà de toutes les en-têtes parsés, ou qu'un protocole n'était pas supporté, les bytes étaient simplement laissés dans le buffer et pas traités. Ces bytes sont maintenant affichés dans une cellule « Data » tout comme dans wireshark. Le support avec l'affichage des octets bruts en hexadécimal et en ascii est assuré, lorsque l'on clique sur la cellule « Data », les données brut concernées seront bien mise en évidence.
Semaine 35 Mercredi 24.07.2013	J'ai passé la journée à implémenter le protocole ICMP. L'application comprend maintenant l'en-tête ICMP sans le champ Information. J'ai mis à jour le rapport en expliquant les différents protocoles implémentés dans l'application en y expliquant les champs et comment les interpréter ainsi qu'une petite description sur chaque protocole. ✓ ICMP ✓ Mettre à jour le rapport sur les protocoles
Semaine 36 Jeudi 25.07.2013	La vue des paquets brut en ascii est buggée. Il y a des caractères spéciaux, des caractères qui ne s'affichent pas correctement et l'application ne peut pas toujours mettre en évidence les bytes concernés. J'ai trouvé le problème venant de caractères invisibles. J'ai décidé de changer la façon dont la vue des bytes en ascii fonctionne tout en réglant le problème. Il est inutile d'essayer de transformer chaque caractère trouvé en son correspondant en ascii à travers la table ascii. Je n'affiche plus que les caractères disons « normaux » de la table ascii afin d'avoir une vue claire sur les données transmises. Pour tous les autres caractères, c'est un point qui est affiché.

Date	Travail effectué
Semaine 36 Vendredi 26.07.2013	J'ai implémenté le protocole TCP et il est maintenant complètement compris par l'application. ✓ TCP Il y a énormément de rapport qu'il reste encore à écrire concernant la partie « Conception » et il va falloir consacrer les jours de la dernière semaine prochaine à faire cela. Vu qu'il reste 4 jours avant le rendu et qu'il reste beaucoup de rapport à écrire et à corriger par rapport au rapport intermédiaire, je vais passer la dernière semaine à faire cela. Je ne toucherai au code de l'application que pour éventuellement corriger des bugs trouvés lors des tests. • Terminer le rapport • Faire plus de tests de l'application
Semaine 37 Lundi 29.07.2013	J'ai passé la journée à compléter le rapport et surtout à retoucher la partie conception écrite pour le rapport intermédiaire afin de la mettre à jour avec les changements qu'il y a eu dans le code lors de l'implémentation. J'ai fait tout une section montrant les différences entre ce qui avait été prévu lors de la conception initiale et ce qui a changé lors de l'implémentation en expliquant pourquoi ces changements avaient besoin d'être appliqués.

Date	Travail effectué
Semaine 37 Mardi 30.07.2013	Ecriture du rapport la plupart de la journée et correction d'un bug lorsqu'un protocole de couche liaison de données n'était pas supporté, l'interface graphique se comportait de manière étrange.
Semaine 37 Mercredi 31.07.2013	Fin de l'écriture du rapport et tests de l'application afin de trouver des éventuels bugs restant afin de les corriger si le temps le permet ou sinon de les référencer. Un bug d'affichage dans la barre de statut apparaît lorsqu'une capture possède une grande quantité de paquets et met du temps à être parsée. Le bug a été corrigé.
Semaine 37 Jeudi 01.08.2013	Rendu du rapport et du code. Fin du travail de bachelor.