République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur
et de la recherche scientifique
Université de Saida - Dr. MOULAY Tahar
Faculté : Technologie
Département : Informatique

Memoire de Master
Option: Réseaux informatique et systèmes répartis

Thème

Création d'un réseau ADHOC à base smartphone

Présenté par :
- BAKHTAOUI Rachid
- KHELIFATI Allaa Eddine

encadré par
Dr. K.MEKKAOUI

année universitaire 2019-2020

Remerciement

Nous tenons à remercier notre encadreur Mr K. MEKKAOUI pour son disponibilité et ses conseils, son orientation et surtout son aide. Il nous a toujours guidé dans la bonne direction dans notre travail.
Ainsi nous remercions tous les enseignants du département de l'informatique qui nous ont donnée beaucoup au cour de notre cursus de formation et aussi tous le personnel du département.

Table des matières

Liste des figures

Introduction générale

Les réseaux composent la structure de base du septième continent qui se forme sous nos yeux. Par l'immense séisme qu'il engendre en ce début de siècle, la planète entre dans une ère nouvelle. Ce nouveau continent est celui de la communication. Constitué de réseaux et se parcourant à la vitesse de la lumière, il représente une révolution analogue à celle de l'apparition de l'écriture ou de la grande révolution industrielle.

Ces réseaux, qui innervent aujourd'hui complètement la planète, se constituent grâce à la fibre optique, aux ondes hertziennes et à divers équipements qui permettent d'atteindre de hauts débits. Internet constitue pour le moment la principale architecture de ces communications.

Les réseaux mobiles ad hoc ou MANET (Mobile Ad hoc NETworks) ont fait l'objet de nombreux travaux de recherche au cours de la dernière décennie. Ces travaux ont dans un premier temps été menés essentiellement à des fins d'applications militaires, mais depuis quelques années des applications civiles sont également envisagées. La communication en mode ad hoc peut en effet se justifier dès lors que le recours à un réseau d'infrastructure s'avère soit techniquement difficile, soit économiquement peu rentable (e.g. secouristes intervenant à la suite d'une catastrophe naturelle, équipes de scientifiques travaillant en terrain désertique, systèmes de communication inter-véhicules, réseaux de capteurs, etc.). Les premiers travaux de recherche menés dans ce domaine ont eu pour objectif principal de définir des méthodes pour assurer l'acheminement de messages —ou, plus classiquement, de paquets IP — de bout en bout dans un MANET.[1]

Le but de notre travail est la réalisation d'une application permettant la création d ' un reseaux adhoc basé sur les smartphones .

Organisation du mémoire

Chapitre I : Les réseaux AD-HOC et Internet des objets (IOT)

dans ce chapitre nous étudions les réseaux AD-HOC et l'internet des objets

Chapitre II : La programmation réseaux

dans ce chapitre nous abordons les differents fases de la programmation réseaux.

Chapitre III : La programmation mobile

dans ce chapitre nous abordons le sujet de la programmation sur des platformes mobiles

et enfin le Chapitre IV : création d'un réseau adhoc à base smartphone qui décrit en screenshot les differentes phases de notre application.

Chapitre I: Les réseaux AD-HOC et Internet des objets (IOT)

I.1 Introduction

Dans ce chapitre on va définir les réseaux adhoc et donner leurs types.

En présisant l'étude sur les réseaux mobile adhoc, le routage et quelques protocoles proactifs, réactifs et hybrides.

Aussi on va essayer d'éclairer le concept de l'internet des objets qui le sujet d'aujourd'hui.

I.2 Définition d'un réseau ad-hoc

Un réseau ad-hoc est une collection d'hôtes équipés par des antennes qui peuvent communiquer entre eux sans aucune administration centralisée, en utilisant une technologie de communication sans fil comme WiFi, Bluetooth, etc. À l'opposé des réseaux filaires où uniquement certains nœuds dits "routeurs" sont responsables de l'acheminement des données, dans un réseau ad-hoc tous les nœuds sont à la fois routeurs et terminaux. Le choix des nœuds qui vont assurer une session de communication dans un réseau ad-hoc se fait dynamiquement selon la connectivité du réseau, d'où l'appellation "ad-hoc".[2]
Dans un réseau ad-hoc, un nœud peut communiquer directement (mode point-à-point) avec n'importe quel nœud s'il est situé dans sa zone de transmission, tandis que la communication avec un nœud situé en dehors de sa zone de transmission s'effectue via plusieurs nœuds intermédiaires (mode multi-sauts). Formellement[3], un réseau ad-hoc peut être représenté par un graphe non-orienté G = (V,E) où V désigne l'ensemble des nœuds et E ⊆ V 2 dénote l'ensemble des arcs correspondants auxcommunications directes possibles. Soit i et j deux nœuds de V, l'arc (i, j) existe, si et seulement si, i peut envoyer

directement un message à j on dit alors que j est voisin de i. Les couples appartenant à E dépendent de la position des nœuds et de leur portée de communication. Si on retient l'hypothèse que tous les nœuds ont une portée R identique, et si d(i,j) désigne la distance entre les nœuds i et j, alors l'ensemble E peut-être défini comme suit :

$$E = \{ (i , j) \in V\ 2\ d (i , j) \leq R \}$$

I.3 Types des réseaux ad-hoc

Il existe trois types de réseaux ad-hoc, à savoir : les réseaux mobiles ad-hoc (MANETs), les réseaux de capteurs (WSNs) et les réseaux maillés (WMNs).

I.3.1 Les réseaux mobiles ad-hoc (MANETs)

Un MANET (Mobile Ad-hoc NETwork) est un réseau ad-hoc dont les nœuds (par exemple : ordinateurs portables, téléphones mobiles, etc.) sont souvent caractérisés par une constante mobilité. Les MANETs peuvent être déployés pour assurer la communication dans des environnements hostiles. Par exemple, dans une situation de guerre et grâce à un MANET les groupes de soldats peuvent communiquer avec d'autres groupes, avec des tanks, des hélicoptères ou des avions pour échanger leurs localisations ou pour émettre /recevoir des ordres militaires. En outre, les MANETs peuvent être utilisés comme infrastructure de communication alternative dans des situations où les infrastructures de communication conventionnelles sont détruites à cause d'une catastrophe naturelle (par exemple : tremblements de terre, ouragans, etc.), ou encore à cause des attaques ennemies, pour mieux coordonner les opérations de secours. Les MANETs sont aussi convenables quand il y a besoin en une communication transitoire comme dans le cas des conférences.[4]

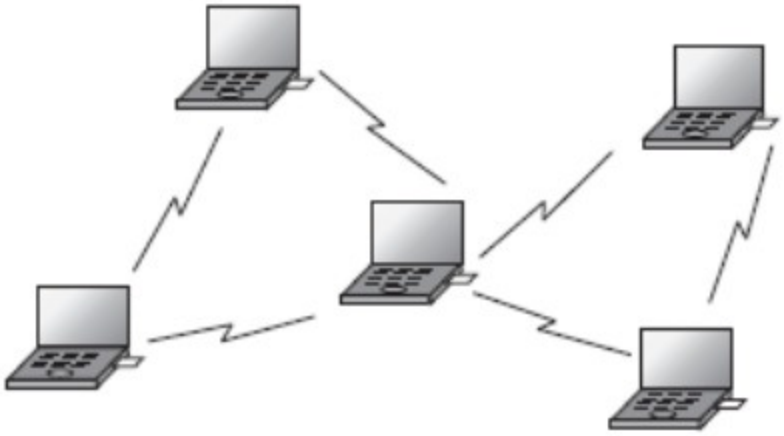

Figure 1: Un réseau mobile ad hoc

Une autre application des MANETs est bien les VANETs (Vehicular Ad-hoc NETWorks).[2] Un VANET assure la communication entre les véhicules (Inter-Vehicle Communication) aussi bien qu'entre les véhicule et les équipements de la route par l'intermédiaire de la communication d'équipement-à-Véhicule (Roadside-to-Vehicle Communication). L'objectif de l'introduction des VANETs est de rendre les routes plus sûres et plus efficaces, en fournissant les informations pertinentes aux conducteurs.[5]

I.3.2 Les réseaux de capteurs (WSNs)

Les WSNs (Wireless Sensor Networks) sont des réseaux ad-hoc constitués de nœuds capteurs intelligents fonctionnant grâce à des batteries, et ils sont dotés de capacités de traitement et de stockage réduites. En effet, les nœuds capteurs sont capables d'accomplir trois tâches complémentaires : le relevé d'une grandeur physique ou environnementale (par exemple : température, pression,

pollution, etc.), le traitement éventuel de cette information et enfin le routage. Le réseau peut comporter un grand nombre de nœuds (des milliers) généralement statiques et déployés aléatoirement (par exemple par largage depuis un hélicoptère) dans des environnements pouvant être dangereux. En plus des nœuds capteurs, un WSN comprend des stations de base riches en énergie (nœuds puits) caractérisées par une capacité de traitement et de stockage plus importante. Ces dernières agissent comme des passerelles entre les nœuds capteurs et l'utilisateur final.

On distingue une variété d'applications pour les réseaux de capteurs, on cite entre autres: [6]

• Applications militaires : les réseaux de capteurs peuvent être utilisés à la surveillance des activités des forces ennemies, à l'analyse du terrain et à la détection d'agents chimiques ou de radiations avant d'y envoyer des troupes.

• Applications à la sécurité : un réseau de capteurs peut constituer un système d'alarme distribué qui servira à détecter les intrusions sur un large secteur.

• Applications environnementales : des thermo-capteurs dispersés sur une forêt peuvent signaler un éventuel début d'incendie ; ce qui permettra une meilleure efficacité pour la lutte contre les feux de forêt. Sur les sites industriels, les centrales nucléaires ou dans les sites pétroliers, des capteurs peuvent être déployés pour détecter des fuites de produits toxiques (gaz, produits chimiques, éléments radioactifs, pétrole, etc.) et alerter les utilisateurs dans un délai suffisamment court pour permettre une intervention efficace.[7]

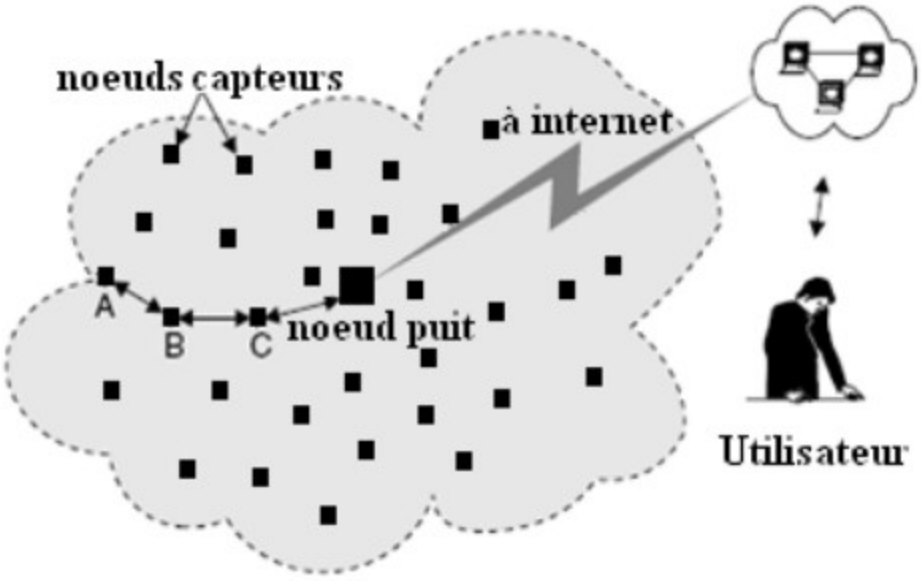

Figure 2: Paradigme de routage dans un réseau de capteurs.

I.3.3 Les réseaux maillés (WMNs)

Les réseaux ad-hoc peuvent être utilisés comme moyen flexible et économique pour étendre l'accès à Internet. En effet, une nouvelle classe de réseaux ad-hoc découle de cette vision : les WMNs (Wireless Mesh Networks). Un WMN est constitué d'un ensemble de meshrouteurs statiques généralement équipés par plusieurs interfaces de communication sans fil et placés sur les toits des bâtiments. Les mesh-routeurs connectés à internet sont appelés "passerelles Internet". Un nœud client se connecte au mesh-routeur le plus proche et exploite l'infrastructure ad-hoc sans fil pour avoir accès à internet. Les clients conventionnels équipés par des cartes Ethernet peuvent directement communiquer avec les mesh-routeurs via des liens Ethernet. Les clients opérants avec la même technologie sans fil que les mesh-routeurs peuvent directement communiquer avec ces derniers. Dans le cas où différentes technologies sans fil

sont utilisées, les clients doivent d'abord communiquer avec des stations de base qui sont connectées par des liens Ethernet aux mesh-routeurs.[8]

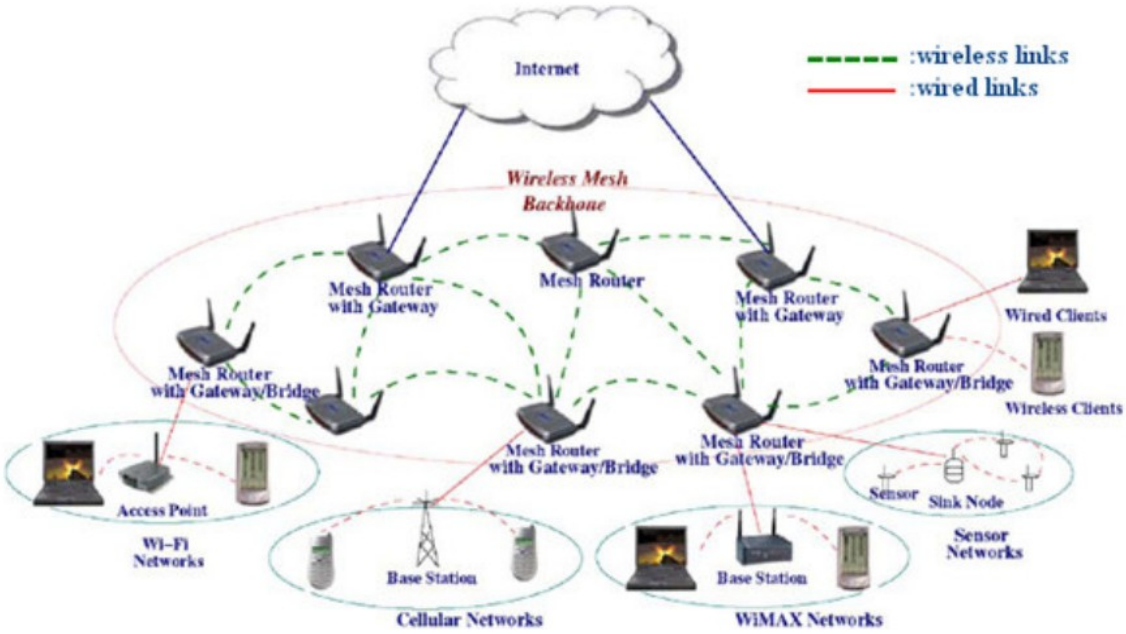

Figure 3: Architecture générale d'un réseau maillé

I.3.4 Différences entre WSNs, WMNs et MANETs

Bien que les MANETs et les WSNs présentent plusieurs caractéristiques communes, mais ils se diffèrent en plusieurs aspects:

• La caractéristique principale des nœuds constituants un MANET est la mobilité, tandis que les nœuds capteurs dans un WSN sont statiques.

• Dans les MANETs la communication peut s'effectuer entre des nœuds quelconques du réseau, tandis que dans un WSN la communication est toujours initiée vers ou à partir des nœuds puits ;

de plus les communications capteur-à-capteur sont rares, mais les transmissions multicast et broadcast sont communes.

• Les MANETs sont caractérisés par une plus faible densité par rapport aux WSNs.

• Dans un MANET tous les nœuds sont égaux, de ce fait la panne de n'importe quel nœud a la même importance, tandis qu'un WSN est plus sensible à la panne des nœuds puits qu'à celle des capteurs. Les réseaux WMNs sont conceptuellement similaires aux MANETs dans le sens où la communication entre les mesh-routeurs s'effectue en mode multi-sauts. Cependant, ils ont les particularités suivantes :

• Les mesh-routeurs dans un WMNs sont statiques.

• La consommation d'énergie dans les WMNs n'est plus un problème, car les routeurs sont directement alimentés en électricité.[7]

I.4 Caractéristiques des MANETs

La perception d'un MANET comme étant équivalent à un réseau filaire conventionnel dont les câbles sont remplacés par des antennes est un malentendu commun.

Les MANETs ont des caractéristiques uniques qui nécessitent des solutions particulières: [9]

• Communications multi-sauts : dans un MANET, les nœuds qui ne peuvent directement atteindre les nœuds destinations auront besoin de relayer leurs données via d'autres nœuds. Ainsi, chaque nœud doit être capable d'accomplir la tâche de routage.

• Topologie dynamique : la mobilité, l'apparition et la disparition des nœuds, la présence d'obstacles (arbres, bâtiments, etc.), les conditions environnementales (pluie, neige, etc.) et les interférences des ondes, sont tous des facteurs qui affectent la qualité de

propagation des ondes émises et se manifestent comme des changements de topologie.

• Bande passante limitée : le médium de communication sans fil a une capacité plus réduite que celui filaire. De plus, le débit effectif de la communication sans fil (avec prise en compte des effets du bruit, d'affaiblissement, des collisions, etc.) est souvent inférieur au débit maximal théorique. Une conséquence directe de la capacité relativement faible du medium sans fil, est bien la congestion facile du réseau.

• Terminaux équipés par des batteries : les entités constituant les MANETs sont alimentées en énergie par des batteries dont la durée de vie est limitée.

• Sécurité limitée : la communication sans fil entre les nœuds est assurée par l'échange d'ondes électromagnétiques qui se propagent dans l'air. Ces ondes peuvent être facilement capturées, surveillées et modifiées ce qui compromet la sécurité dans les MANETs. Par exemple, le trafic peut être facilement désorienté de sa destination réelle. De plus, les attaques de type déni de service sont faciles à implémenter.

I.5 Le routage dans les réseaux ad hoc

Le routage est une méthode d'acheminement des informations à la bonne destination à travers un réseau donné. Le rôle de routage est de déterminer un acheminement optimal des paquets à travers le réseau selon un certain critère de performance.

La zone de couverture radio est limitée, par conséquent les informations dans les réseaux Ad hoc peuvent exigées plusieurs sauts pour être transportées. Alors, le routage devient un mécanisme indispensable pour supporter la transmission radio multi-sauts. Les nœuds des réseaux Ad hoc changent fréquemment de position d'une

façon aléatoire et imprévisible, ce qui aboutit à des interruptions et des ruptures brusques des liaisons.

Pourtant, les changements rapides de la topologie dans les réseaux Ad hoc demandent des protocoles de routage spéciaux qui s'adaptent facilement. En fait, des protocoles de routage pour les réseaux Ad hoc ont été développés dans le cadre du groupe de recherche MANET de IETF (Internet Engineering Task Force).

Pour évaluer les performances d'un protocole de routage, nous avons besoin des mesures qualitatives et quantitatives simultanément. Ces métriques doivent être indépendantes de tous les protocoles de routage existants. Les propriétés qualitatives souhaitables sont : le traitement distribué, la liberté de bouclage, le traitement basé sur la demande, le traitement proactif dans certains contextes, la sécurité, le traitement des périodes de "sommeil" et le support des liaisons unidirectionnelles. Cependant, les unités quantitatives requises sont : le flux et le délai de données de bout en bout, le temps d'acquisition d'itinéraires, le pourcentage de réception dans le mauvais ordre, l'efficacité. De plus, il faut considérer le contexte du réseau dans lequel les performances du protocole sont mesurées. Les paramètres essentiels sont : la taille du réseau, la connectivité du réseau, le taux de changement de topologie, la capacité des liaisons, le taux de liaisons unidirectionnelles, le type de trafic, la mobilité, et le ratio et la fréquence des périodes de sommeil des nœuds. En outre, il apparaît important que toute conception de protocole de routage doit étudier les problèmes suivants :

- Minimiser la charge du réseau ;
- Offrir un support pour pouvoir effectuer des communications multi-sauts fiables ;
- Assurer un routage optimal ;
- Offrir une bonne qualité concernant le délai.

Selon la façon de la création et de la maintenance des routes lors de l'acheminement des données, ces protocoles sont divisés en deux catégories :

- Les protocoles proactifs ;
- Les protocoles réactifs.

I.5.1 Protocoles de routage proactifs

Les protocoles de routage proactifs sont basés sur la même philosophie des protocoles de routage utilisés dans les réseaux câblés traditionnels tels que les protocoles d'état de lien (Link State) et ceux du vecteur de distance (Distance Vector). Ce type de protocoles exige une mise à jour périodique des tables de routage. Exemples de ces protocoles: DSDV (Destination Sequenced Distance Vector Routing), OLSR (Optimized Link State Routing), CGSR (Clusterhead Gateway Switch Routing) et WRP (Wireless Routing Protocol).

I.5.1.1 Le protocole de routage DSDV

Le protocole DSDV ou (Destination Sequenced Distance Vector), développé en 1994 par C. Perkins, est basé sur l'algorithme distribué de Bellman-Ford en rajoutant quelques améliorations. Chaque station mobile maintient une table de routage qui contient toutes les destinations possibles, le nombre de sauts nécessaire pour atteindre la destination et le numéro de séquences qui correspond à un nœud destination. La mise à jour de la table de routage dépend des deux paramètres qui sont la période de transmission et les événements.

La mise à jour du paquet contient le nouveau numéro de séquence incrémenté du nœud émetteur ainsi que l'adresse de la destination, le nombre de sauts et le numéro de séquence tels qu'ils ont été écrits par la destination pour chaque nouvelle route. Le DSDV élimine les deux problèmes de boucle de routage "routing loop", et celui du

"counting to infinity". Le DSDV éprouve quelques désavantages : il demande une mise à jour régulière de ses tables de routage, ce qui réduit l'efficacité de la largeur de bande. Toutefois, dans ce protocole, le nœud mobile doit attendre jusqu'à ce qu'il reçoive la prochaine mise à jour initiée par la destination, afin de mettre à jour la table de routage. De plus, il n'est pas approprié au très grand réseau c'est à dire qu'il n'est pas mis à l'échelle. Ainsi, il n'est pas adapté pour les réseaux fortement dynamiques.

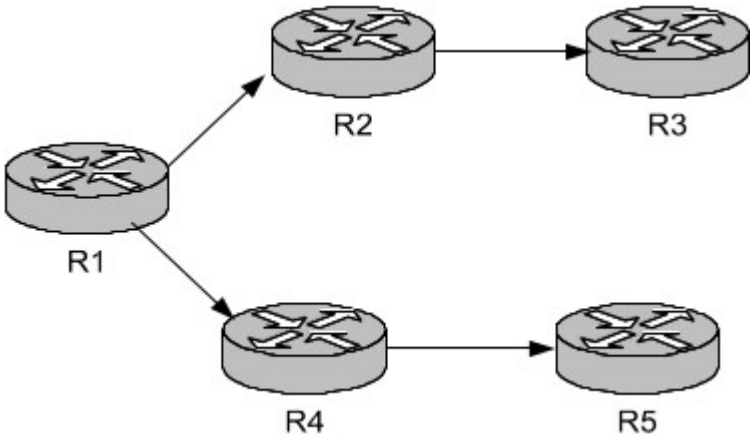

Figure 4: Exemple des noeuds de routage Ad hoc.

La table de routage du nœud R1

Destination	Nombre de sauts	Prochain nœud	Numéro de séquence
R1	0	R1	1
R2	1	R2	4
R3	2	R2	5
R4	1	R4	6
R5	1	R5	3

I.5.1.2 Le protocole OLSR

Le protocole OLSR (Optimized Link State Routing), développé pour les réseaux MANETs. Il est basé sur la méthode "état de lien" et permet d'éch anger des informations sur la topologie du réseau avec les autres nœuds. OLSR utilise le principe du relais multipoints MPR (MultiPoint Relays). Tous les nœuds du réseau envoient des messages "HELLO" pour déterminer la nature des liens qui les relient et découvrir l'ensemble du réseau. Ensuite, ces messages « HELLO » transmettent l'état et le type de lien entre l'expéditeur et chaque nœud voisin puis ils spécifient le MPR choisi par l'expéditeur.
Ces nœuds particuliers MPR expédient des messages de diffusion pendant le processus d'inondation et produisent les messages d'état de lien.

I.5.2 Protocoles de routage réactifs

Les protocoles de routage réactifs créent et maintiennent les routes selon les besoins. Lorsqu'une route est demandée, une procédure de découverte globale est lancée par la source afin de trouver le meilleur chemin. Exemples des protocoles réactifs : Dynamic Source Routing (DSR), Ad hoc On-Demand Distance Vector Routing (AODV), Temporally-Ordered Routing Algorithm (TORA), Associativity-Based Routing (ABR) et Signal Stability Routing (SSR). Dans ce qui suit, nous écrivons en détails les deux protocoles DSR et AODV.

I.5.2.1 Le protocole de routage DSR

Le protocole "Routage à Source Dynamique" (DSR : Dynamic Source Routing protocol), est basé sur la technique de routage par la source. La source des données détermine la séquence complète des nœuds intermédiaires par lesquels les informations vont transiter. Quand un nœud veut envoyer des données, il diffuse un paquet

requête « route request » qui contient un champ permettant d'enregistrer tous les nœuds qu'il va visiter jusqu'à l'atteinte de la destination. En cas de découverte d'une route, la source reçoit un paquet réponse de la route « route reply » qui contient la séquence des nœuds traversés. Ensuite, la source insère la séquence de nœuds de la route reconnue dans l'entête de tous les paquets qu'il désire transmettre. Dans ce cas, les nœuds intermédiaires jouent un rôle de simple relayeur d'information. À la réception d'un paquet, chaque nœud supprime son adresse de la séquence de nœuds contenue dans l'entête, puis l'achemine au nœud suivant dans la séquence.

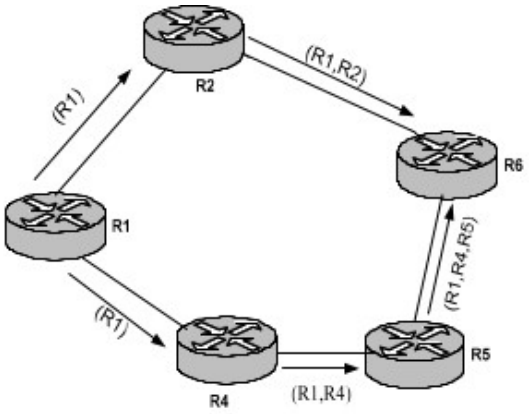

Figure 6: La détermination d'une route selon DSR

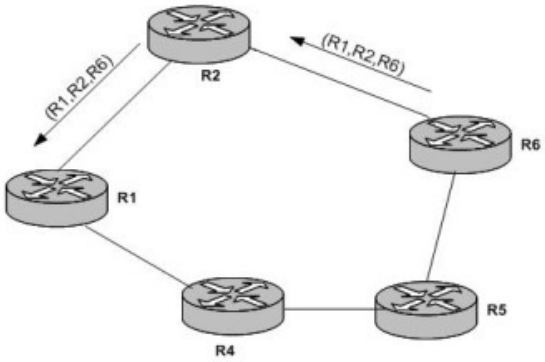

Figure 5: Le renvoi du chemin calculé

.

Le protocole DSR exécute une procédure de maintenance de routes afin d'assurer la validité des chemins utilisés. Un message erreur de route « route error » est envoyé à l'émetteur original du paquet, lors de la détection d'un problème majeur. Ce message contient l'adresse du nœud qui a détecté l'erreur et celle du nœud suivant dans le chemin. Lorsque le nœud source reçoit ce message d'erreurs, le nœud concerné par l'erreur est supprimé du chemin sauvegardé, et tous les chemins contenant ce nœud sont coupés à ce point là.

Ensuite, l'émetteur initie une nouvelle opération de découverte de routes vers la destination.
Les paquets de données contiennent toutes les décisions de routage ce qui résulte que les nœuds intermédiaires n'aient pas besoin de maintenir les informations de mise à jour pour envoyer les paquets de données. Dans ce protocole, il n'y a pas de boucle de routage, parce que la route entre la source et la destination est une partie des paquets de données envoyés.

I.5.2.2 Le protocole de routage AODV

Le protocole AODV (Ad-hoc On Demand Vector Distance), représente une amélioration de l'algorithme DSDV. Il est prévu pour être utilisé par les réseaux Ad hoc mobiles. L'amélioration par rapport au DSDV réside dans le fait que, le protocole AODV permet de mettre à jour la table de routage d'un nœud sans que celui-ci ait à communiquer avec tous ses voisins ce qui diminue considérablement le nombre de paquets diffusés dans le réseau. L'AODV utilise le principe des numéros de séquence pour maintenir la consistance des informations de routage. Comme dans le DSR, l'AODV utilise le principe d'inondation pour trouver une route vers une certaine destination en envoyant un paquet RREQ « route request ». Cependant, contrairement au DSDV, chaque nœud recevant ce paquet prépare une entrée dans sa table de routage afin de pouvoir rediriger plus tard les paquets qu'ils recevront.
Le paquet RREQ contient dans le champ « numéro de séquence destination » le dernier numéro de séquence associé à la destination. Ce numéro est recopié de la table de routage. Si ce numéro n'est pas connu, la valeur nulle ne sera prise par défaut. Afin de maintenir des routes consistantes, une transmission périodique du message "HELLO" est effectuée. Un lien est considéré défaillant, si trois messages "HELLO" ne sont pas reçus consécutivement à partir

d'un nœud voisin. Le protocole AODV ne présente pas de boucle de routage et évite le problème "counting to infinity" de Bellman-Ford, ce qui offre une convergence rapide quand la topologie du réseau Ad hoc varie.

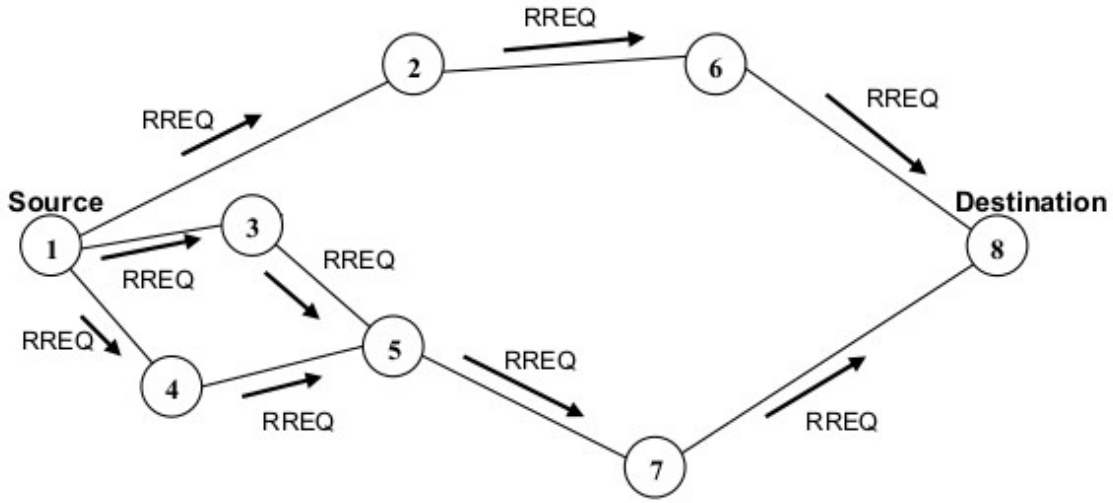

Figure 7: La propagation du paquet RREQ (requête de route)

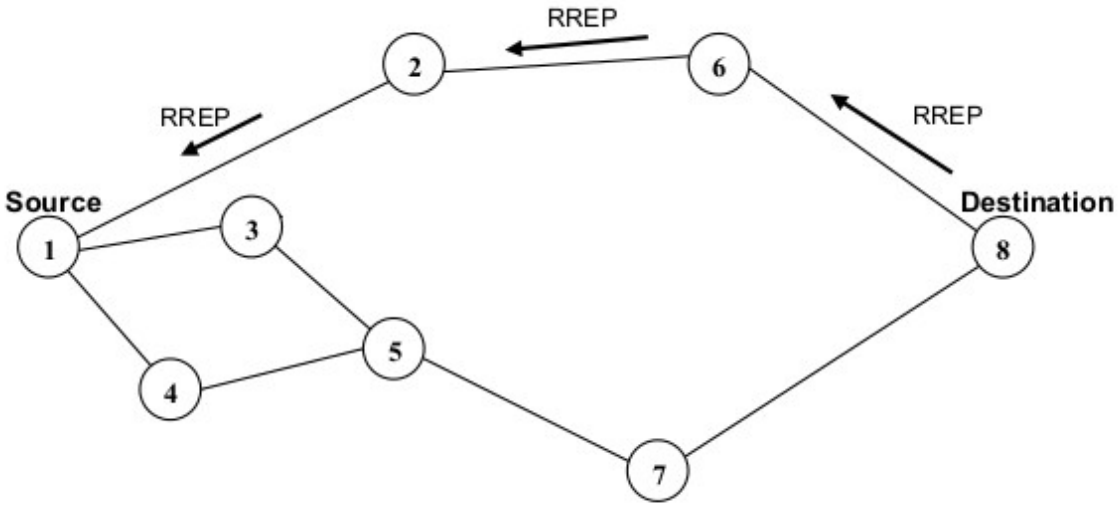

Figure 8: Le chemin pris par le paquet RREP (requête de réponse)

I.5.3 Protocoles réactifs Vs Protocoles proactifs

Les protocoles proactifs possèdent de bonnes aptitudes en termes de temps de réponse, car quand un nœud souhaite communiquer avec un autre nœud, il trouve immédiatement dans sa table de routage la route adéquate pour atteindre ce nœud. Cependant, un surcoût significatif est engendré par les échanges périodiques de messages de contrôle. Les protocoles réactifs résolvent plus ou moins le défaut des protocoles proactifs tout en augmentant le temps de réponse. Le surcoût engendré en utilisant l'approche réactive est moindre dans certaines configurations réseau. Néanmoins, il peut demeurer élevé dans d'autres. De manière générale, les protocoles réactifs et proactifs présentent des performances différentes selon les caractéristiques du réseau. Dans le cas d'un réseau dense par exemple et/ou lorsque l'échange des données est intense, le choix d'un protocole proactif s'avère plus judicieux, car un protocole réactif entraîne la diffusion excessive de messages de demande de routes qui risquent de saturer le réseau. En revanche, dans une configuration opposée, à savoir un réseau à densité faible et/ou à faible trafic, choisir un protocole réactif est plus intéressant, car il ne surchargera pas inutilement le réseau par des flux continuels d'informations de routage entre les nœuds.

I.5.4 Protocoles hybrides

Afin de tirer profit des avantages des protocoles réactifs et proactifs, des protocoles qui mêlent généralement les deux modes (fonctionnant dans l'un ou dans l'autre mode) suivant des conditions prédéfinies, ont été proposés.

Dans ce type de protocoles, on utilise une reconnaissance locale de la topologie réseau jusqu'à un nombre relativement petit de sauts, par un échange périodique de trames de contrôle, autrement dit par une technique proactive. Les routes vers des nœuds plus lointains

sont obtenues par un schéma réactif, c'est-à-dire par l'utilisation de paquets de demande de routes qui sont dans la majorité des cas diffusés dans tout le réseau.

Parmi les protocoles hybrides ZRP (Zone Routing Protocol). ZRP a été introduit en 1997 par Haas et Pearlman [Haa97, HP01]. ZRP définit pour chaque nœud une zone de routage (zone radius), qui inclut tous les nœuds dont la distance minimale (en nombre de sauts) à ce nœud est d. Les nœuds qui sont exactement à distance d sont appelés nœuds périphériques. Pour trouver une route vers des nœuds situés à une distance supérieure à d, ZRP utilise un système réactif, qui envoie une requête à tous les nœuds périphériques. Pour cela, ZRP met en œuvre deux types de fonctionnement : IARP (IntrAzone Routing Protocol) et IERP (IntErzone Routing Protocol).

Basé sur un protocole à vecteur de distance, IARP permet, en utilisant une technique proactive, de trouver toutes les routes jusqu'à une distance d. IERP quant à lui, permet d'établir les routes vers les nœuds à plus de d sauts d'une façon réactive.

I.6 Internet des Objets (IOT)

I.6.1 Définitions

a. Kevin Ahston , le cofondateur de l'Auto-ID Center du MIT a employé le terme « Internet Of ThingsT(Internet des Objets) » en 1999. IdO a été prononcé dans le cadre d'une présentation pour l'entreprise Procter & Gamble (P&G)). Ce terme convoque, le monde d'objets, d'appareeils et de capteurs qu ui sont interconnectés par internet.[10]

b. Le CERP-IdO « Cluster des projets européens de recherche sur l'Internet des objets» définit l'Internet des objets comme : «une infrastructure dynamique d'un réseau global. Ce réseau global des capacités d'auto-configuration basée sur des standards et des

protocoles de communication interopérables. Dans ce réseau, les objets physiques et virtuels ont des identités, des attributs physiques, des personnal lités virtuelles et des interfaces intelligentes, et ils sont intégrés au réseau d'une façon transparente»

c. Définition de l'UIT (Union Internationale des Télécommunications) : « L'IoT est l'infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution». [11]

d. «Un réseau de réseaux qui permet, via des dispositifs d'identification électronique d'entités physiques ou virtuelles, dites objets connectés et des systèmes de communication appropriés, sans fil notamment, de communiquer directement et sans ambiguïté, y compris au travers de l'Internet, avec ces objets connectés et ainsi de pouvoir récupérer, stocker, transférer et traiter sans discontinuité les données s'y rattachant».[12]

La diversité des définitions proposées pour l'IoT, et leur longueur très variable, montrent bien qu'il s'agit d'une notion complexe, évolutive et susceptible de recevoir différentes formes d'instanciation. Ces instanciations sont souvent « décorées » du vocable de « smart » : smart citizen, smart home, smart building, smart grid, smart factory, smart city, smart territory... Si le domaine d'application se trouve ainsi précisé, le flou demeure sur le contour et le contenu du système que l'on entend désigner.

L'Internet des objets apparait comme un concept fonctionnel très général qui associe trois composantes fondamentales : les objets, les réseaux de communication, les données.

I.6.2 Les composants de l'IOT

I.6.2.1 Les objets connectés

L'IoT se donne comme objectif de réaliser la connexion entre objets de toute nature, qu'ils soient matériels ou immatériels. Cisco a proposé l'appellation « Internet of Everything (IoE) » afin de souligner l'universalité du concept. Les objets matériels, ce sont les « choses » de la vie : tous les objets de la vie courante mais aussi les équipements du monde industriel ou professionnel. Les objet simmatériels, ce sont les constructions de l'esprit et tout particulièrement les logiciels. A cette dichotomie, il faut ajouter les humains qui associent capacités matérielles et logicielles. En fait, tous les objets du monde de l'IoT dispose d'une capacité logicielle, d'une forme d'intelligence, plus oumoins développée, qui leur permet de percevoir et saisir des informations, de les transmettre et éventuellement les traiter et, en retour, de comprendre des instructions et d'agir en conséquence.

La distinction pertinente est donc plutôt à faire entre « personnes » et « machines ». Les communications « personne à personne » sont celles rendues possibles par l'Internet, tel que nous le connaissons depuis 40 ans et plus. Les communications « machine à machine » sont celles du MtoM (ou M2M), qui sont parfois assimilées à l'IoT mais qui en constituent en fait un sousensemble. Ce sous-ensemble est extrêmement important du point de vue de la valeur ajoutée qui lui est associée. Ce sont en effet les communications M2M qui sont, pour une large part, à l'origine des progrès de productivité que l'on peut espérer grâce au développement « d'usines du futur » dotées de machines « intelligentes », plus efficaces, plus adaptatives, plus robustes, permettant de diminuer les coûts de production et de simplifier les chaînes d'approvisionnement.

On notera cependant que le M2M serait insuffisant s'il n'était pas associé à une communication entre les machines et les personnes car c'est in fine la conjonction de l'intelligence créatrice de l'Homme avec celle programmée de la machine qui permet de faire face aux situations les plus variées et de prendre, le plus rapidement possible, les bonnes décisions.

Les objets de l'IoT ont une personnalité propre : ils sont capables de saisir et de recevoir de l'information et de les échanger avec les autres objets de l'IoT. Mais il faut pour cela qu'ils puissent être identifiés de façon certaine. Le problème n'est pas trivial compte tenu du nombre considérable d'identifiants à gérer potentiellement. De sa solution dépend la possibilité pour les correspondants de s'identifier mutuellement de façon certaine, sans risque d'être abusés par des cyberattaques. L'identification est indispensable sur le plan local, elle l'est aussi sur un plan beaucoup plus large si l'on veut gérer la mobilité et offrir des fonctions d'itinérance (roaming). Enfin, pour les opérateurs de réseau, elle est nécessaire pour procéder, s'il y a lieu, à la facturation des services de communication.

Aujourd'hui, plusieurs systèmes d'identification coexistent :

• ceux du monde des télécoms et des réseaux cellulaires en particulier : numéros de téléphones mobiles, identifiants de carte SIM ;

• ceux du monde de l'Internet : adresses IP (v4 et v6), adresses MAC, adresses EUI 64 ;

• ceux du monde des RFID (étiquettes radiofréquences) et du NFC (Near Field Communication) : plusieurs systèmes d'identifiants coexistent aujourd'hui, notamment ceux promus par l'organisation EPCglobal.

Par ailleurs beaucoup de réseaux locaux utilisent des systèmes d'adressage locaux qui n'ont de signification que dans le périmètre couvert par ces réseaux. [13].

I.6.2.2 Les réseaux de communication

Les réseaux de communication entre objets constituent la deuxième composante essentielle de l'IoT. Si l'on admet comme objectif plausible le chiffre de 50 milliards d'objets connectés, on peut considérer que 10 % de ces objets seront des personnes et 90 % des machines avec, pour ces dernières, des perspectives de développement bien plus considérables que pour les humains. L'Internet a résolu le problème de la connectivité entre humains mais la connectivité entre objets pose des problèmes qui sont, au moins, de deux ordres de grandeur plus complexes. L'appellation « Internet des objets » laisse à penser que c'est l'Internet qui s'imposera et que l'IoT sera un gigantesque réseau regroupant autour de l'Internet des milliards d'objets. Certaines illustrations, telles que celle de la figure I.7, incitent à croire qu'il en ira ainsi. En fait, tel n'est pas du tout le cas aujourd'hui. La plupart des objets n'ont pas besoin de disposer d'une connexion directe à l'Internet et il serait absolument inutile de déverser dans le nuage (le cloud) des milliards de milliards de données qui n'ont qu'un intérêt local ou éphémère.

Figure 9: Une image d'Epinal. Les objets de l'IoT @Pixabay

Les réseaux « élémentaires » sont de plus en plus des réseaux locaux sans fil construits autour de divers systèmes de communication tels que Bluetooth, ZigBee, les Wi-Fi ou les LPWAN et qui communiquent avec l'Internet au travers d'un routeur de bordure qui peut assurer également la conversion de protocole pour pouvoir acheminer les données vers les serveurs connectés à Internet.

Cette structure fédérative peut être illustrée par la figure I.8 où sont représentés des réseaux locaux et un réseau longue distance fédérés par l'Internet.

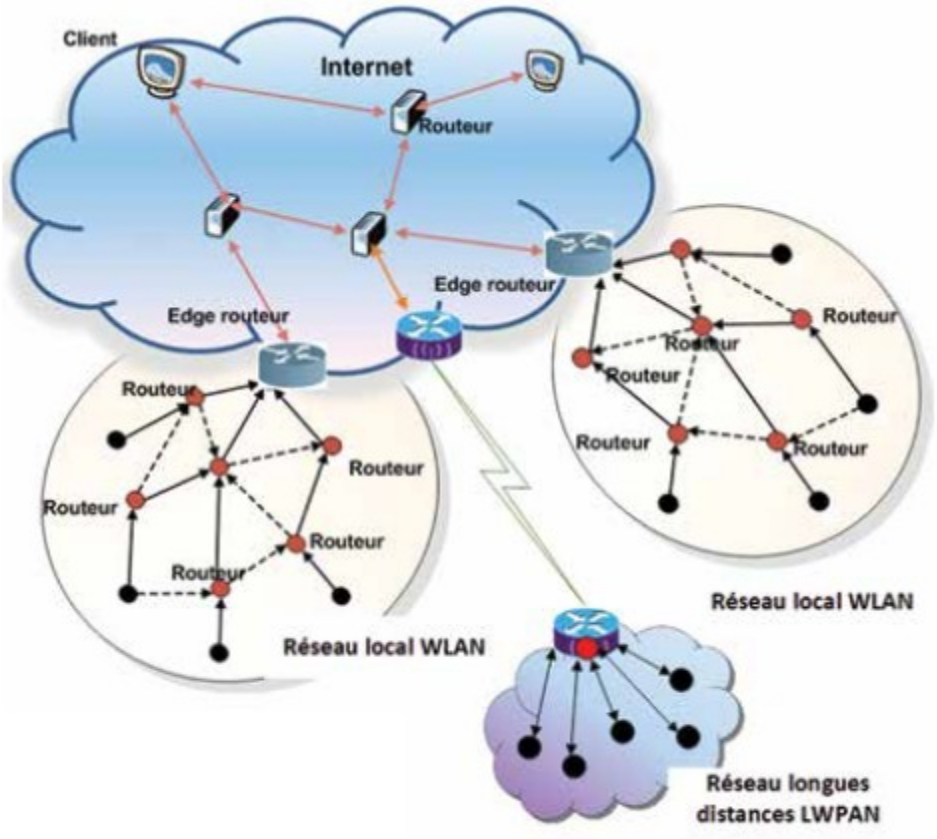

Figure 10: L'Internet des objets vu comme un réseau de réseaux

I.6.2.3 Les données

Troisième volet du triptyque de l'IoT, les données sont en fait au cœur du concept d'Internet des objets. Plus de données, plus facilement : l'accroissement considérable du volume de données accessibles est le premier élément nouveau apporté par l'IoT en matière de data. Le développement de capteurs communicants, miniaturisés et bon marché, permet d'instrumenter de façon beaucoup plus fine tous les procédés – au sens conceptuel du terme : qu'il s'agisse d'uneinstallation industrielle ou agricole, d'un bâtiment, de l'environnement d'un malade, d'un véhicule, etc. – et de

collecter ainsi, au travers de réseaux performants à très haut débit, des masses considérables de données.

Mais collecter les données n'est pas un but en soi. Les données n'ont d'intérêt que si elles sont utilisées pour accroître l'efficacité des procédés contrôlés. Traditionnellement, l'utilisateur était en prise directe avec le procédé et réagissait en fonction des données qu'il recevait. Un individu retire sa main s'il vient malencontreusement la placer sur une plaque chauffante ou remonte la température de réglage de son thermostat s'il constate, en rentrant chez lui, que la température de confort n'est pas atteinte. Au fil des années, l'algorithmie s'est enrichie et l'opérateur a pu disposer de méthodes d'analyse, de simulation et d'optimisation de plus en plus sophistiquées avant de prendre ses décisions sous forme d'envoi de consignes appropriées. L'IoT n'ôte rien à ces possibilités de contrôle direct des processus. Mais il ouvre de nouveaux horizons grâce aux capacités de traitement et de stockage hébergées dans le nuage qu'offre l'infonuagique (ou cloud computing).

En effet, après avoir été collectées par les réseaux d'extrémité appropriés, les données deviennent disponibles au niveau d'un routeur de bordure capable de les envoyer dans le monde de l'Internet. Réceptionnées par des serveurs appropriés, elles vont être triées et validées avant d'être stockées dans de gigantesques centres de données (data centers) capables d'héberger des exaoctets de données (1 Eo = 10 18 octets soit l'équivalent de 10 milliards de dictionnaires Larousse en cinq volumes). Ces données sont alors mises à la disposition des utilisateurs ayant souscrit aux services du centre de données afin de leur permettre de les traiter et de les utiliser à leur convenance. L'utilisateur pourra soit faire appel à des services de traitement proposés de façon standardisée par la plateforme Internet – c'est ce qu'on appelle les analytics, soit les

rapatrier vers ses propres serveurs pour y effectuer les traitements de son choix.

Ce passage par le nuage offre plusieurs avantages. Il permet en effet de :

• disposer de capacités de traitement et de stockage considérables qu'un utilisateur aurait des difficultés à acquérir et à maintenir de façon isolée ;

• sécuriser les données et y donner accès à partir de postes clients situés n'importe où dans le monde ;

• accéder à des méthodes d'exploitation statistique relevant du traitement des données massives (Big Data) afin de tirer parti, en particulier, des signaux faibles enfouis dans un volume considérable de données ;

• croiser les flux de données entre eux et prendre en compte l'ensemble des paramètres pouvant influer sur la décision. Dans le domaine agricole, on pourra par exemple croiser entre elles des informations sur l'état des sols, la maturité des cultures, la météo, les cours sur les marchés, etc. Dans le domaine de la santé à domicile, on croisera les informations venant du patient avec l'expertise des médecins et les données disponibles dans des bases de données, etc.

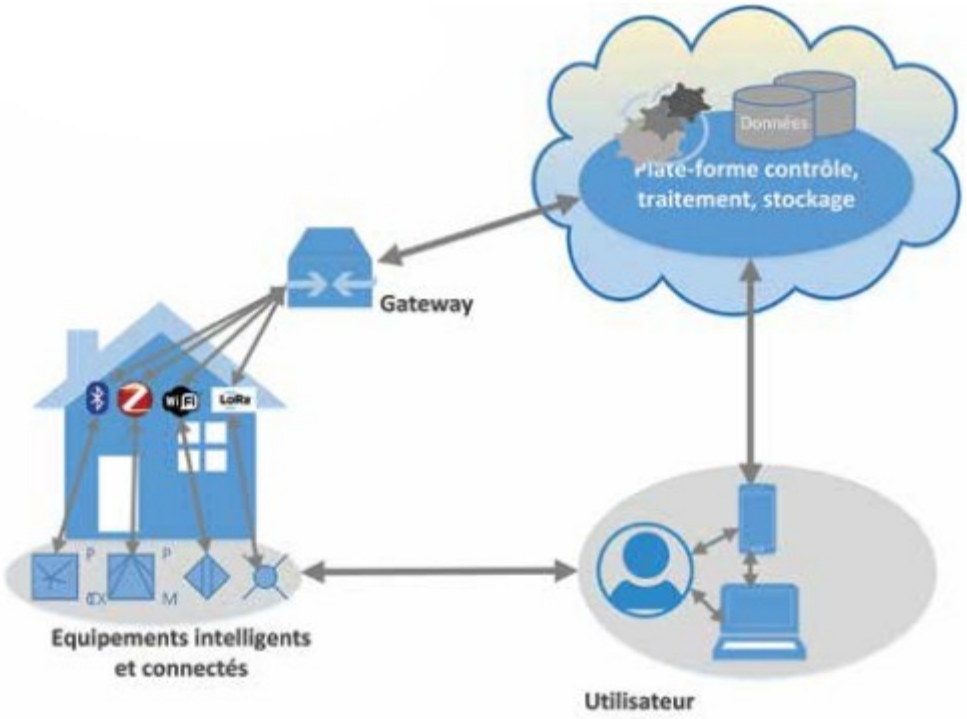

Figure 11: Schéma du fonctionnement d'une maison intelligente (smart home)

Ce schéma est transposable, moyennant adaptation, aux autres domaines d'application de l'IoT.

Bien évidemment se trouve posée la question, déjà évoquée à propos de la connectivité et des réseaux, de l'opportunité d'envoyer sur l'Internet des données massives qui n'ont qu'un intérêt limité. Pour éviter un gaspillage de ressources au niveau du nuage et aussi pour offrir une meilleure réactivité dans le traitement de ces données, est apparu le concept d'informatique de brouillard (fog computing) développé par Cisco et repris aujourd'hui par Microsoft sous le vocable d'informatique à la marge (edge computing).

Cette approche consiste à appliquer le principe de subsidiarité dans le traitement et le stockage des données et à réaliser ces opérations de façon distribuée, là où des ressources (passerelles intelligentes ou

nœuds terminaux) sont disponibles, au plus près des sources de données et/ou de leurs utilisateurs.
Ce principe de calcul décentralisé, aussi dénommé mesh computing, peer-to-peer computing, self-healing computing, autonomic computing, grid computing est repris par tous les grands offreurs d'infrastructures dans le domaine de l'IoT.

I.6.3 Les technologies de l'IoT

L'IoT n'est pas une technologie mais un concept qui s'appuie sur un ensemble de technologies qui se sont développées au cours des dernières décennies et qui, à différents niveaux, concourent à rendre possible l'IoT. Il est vain de chercher à lister toutes ces technologies car l'IoT irrigue l'ensemble de l'économie et par conséquent toutes les technologies que l'homme a développées peuvent trouver dans le concept d'IoT une justification nouvelle. A la base de toutes ces technologies, on trouve bien entendu les technologies matérielles et logicielles, toujours en progrès, qui permettent de doter, à des coûts et avec un encombrement de plus en plus réduits, les objets des capacités de traitement, de mémoire et de communication nécessaires à l'IoT. De façon plus spécifique, les points suivants doivent être soulignés :

I.6.3.1 Les capteurs

la technologie des capteurs continue à faire des progrès considérables en termes de précision, de fiabilité, d'encombrement, de capacité de traitement et de communication. Ces progrès associent l'utilisation de phénomènes physiques très diversifiés aux possibilités offertes par la microélectronique. Les microsystèmes électromécaniques sont des systèmes miniaturisés présentant des dimensions micrométriques, comprenant un ou plusieurs éléments mécaniques, utilisant l'électricité comme source d'énergie, en vue de réaliser une fonction de capteur ou d'actionneur. Ces dispositifs

jouent un rôle fondamental et on les retrouve à présent dans un très grand nombre de secteurs : automobile, aéronautique, médecine, biologie, électronique.

I.6.3.2 Les communications

les techniques de communication sont au cœur de l'IoT puisque ce sont elles qui permettent de construire les réseaux nécessaires à son fonctionnement : réseaux d'extrémité sur lesquels viennent se connecter les objets, réseaux d'infrastructure permettant de fédérer ces réseaux d'extrémité, réseaux d'amenée permettant de convoyer les données vers le monde de l'Internet. Parmi ces technologies:

• les radiocommunications qui jouent un rôle primordial, la plus grande attention doit surtout être portée à la 5ème génération de radiocommunications pour les mobiles qui, grâce à la mise en œuvre de technologies réellement « disruptives » (MIMO massif, ondes millimétriques, etc.) pourrait devenir la technologie de base en matière de communications dans le domaine de l'IoT.

• les technologies Li-Fi associées à l'éclairage par LED semblent ouvrir des domaines d'application intéressants.

• les technologies satellitaires, encore onéreuses aujourd'hui, mais qui peuvent assez rapidement, grâce aux galaxies de satellites en cours de constitution, offrir des possibilités immenses pour l'interconnexion des objets, sur terre, dans l'air ou dans l'espace.

I.6.3.3 Le traitement des données

les technologies nouvelles de traitement des données sont un autre moteur de l'IoT. Nous rappellerons simplement les technologies de base qui permettent de miniaturiser les capacités de traitement et les mémoires et celles, technologies optiques notamment, qui permettent de construire les grands centres de données hébergés dans le nuage et de proposer les services d'infonuagique sous leurs

différentes formes. Plus spéci fiques à l'IoT, sont les technologies de traitement des données massives autour des fameux cinq V : Volume, Velocity, Veracity, Variability et Variety. Ces techniques du Big Data, d'inspiration statistique, débouchent sur des méthodes de recherche de tendances lourdes à partir de signaux faibles, d'analyse de risques, de détection de fragilités et de maintenance préventive.

Nous appelons également l'attention sur les technologies de Digital Twins, ou jumeaux numériques, qui permettent d'associer à un objet réel sa réplique numérique qui regroupe et modélise l'ensemble de ses caractéristiques, permettant d'effectuer sur lui toute sorte de tests, de simulations et de comparaisons. Ce concept est évidemment associé aux techniques de conception assistée mais il connaît aujourd'hui un prolongement essentiel avec les technologies d'impression 3D qui permettent de passer du jumeau numérique à son instanciation physique (figure12).

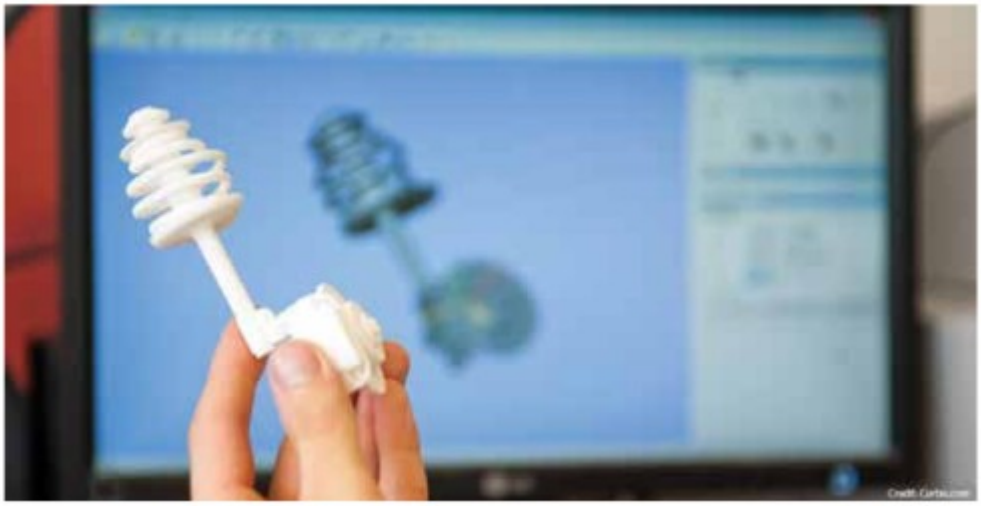

Figure 12: Illustration du concept de jumeaux numériques – @Corbis.com

I.6.4 Les standards de l'IoT

L'un des fondements de l'IoT repose sur l'aptitude des objets à communiquer entre eux, quelle que soit leur nature, leur origine et leur localisation. Interopérabilité et ouverture sont deux mots clés de l'Internet des objets. Certains pensent que le recours au protocole IP est suffisant pour atteindre cet objectif. La question est en fait infiniment plus complexe et pose d'énormes problèmes de standardisation, des principales organisations (ETSI, IEEE, ISO/IEC, IETF) ont émis de très nombreux standards interférant de près ou de loin avec l'IoT. Selon l'IEC, il y en aurait aujourd'hui plus de 400. Et pourtant le problème de la standardisation de l'IoT est loin d'être réglé. Il n'existe pas d'appréhension globale du concept de l'IoT conduisant à des déclinaisons sectorielles cohérentes avec une vision d'ensemble. Il existe de nombreuses approches spécifiques traitant d'une partie du problème dans un cadre donné mais la réconciliation du puzzle reste à faire.
L'ISO y travaille mais nous sommes aujourd'hui encore loin du compte. Il faudrait pour cela travailler concurremment sur plusieurs chantiers :

• **L'identification unique des objets :** nous avons évoqué précédemment cette question essentielle qui est un prérequis incontournable pour l'organisation de l'interopérabilité et de la cybersécurité. L'adressage IP, la numérotation mobile, l'identification RFID et NFC, l'identification ISO sont autant de briques qu'il faudrait consolider dans un ensemble cohérent ;

• **Les couches basses de communication :** c'est le domaine où les initiatives, sans qu'il soit possible aujourd'hui de savoir quelles solutions fini ront par s'imposer. Au niveau le plus bas, celui de la couche physique, la standardisation des fréquences indispensables à l'IoT est loin d'être achevée, exception faite de la bande libre des 2,4

GHz mais qui est aujourd'hui trop fortement sollicitée. L'élargissement au niveau euro péen de la petite bande des 868 MHz, entreprise en France par l'ARCEP, est une initiative qu'il faut faire aboutir ;

• **La couche IP et les protocoles de niveau supérieur :** la question des protocoles Internet, est tout aussi importante que celle des niveaux inférieurs. Les protocoles Internet sont nombreux et se concurrencent. Sans doute faut-il attendre que le marché joue son rôle et, de la même façon qu'il a su reconnaître l'universalité du protocole HTTP pour les communications entre personnes sur le Web, qu'il détermine les protocoles les plus adaptés au transfert des données de l'IoT, CoAP et MQTT étant aujourd'hui deux candidats bien placés. Cependant, il faudra aller plus loin et définir des profils adaptés à chacune des classes d'équipements afin, par exemple, de pouvoir assurer l'interopérabilité fonctionnelle des objets, sans devenir tributaire d'un fournisseur particulier.

• **La gestion des données :** l'administration des données issues de l'IoT pose aujourd'hui de grands défis. Il faut en effet intégrer les données de nature différente, structurées ou non structurées, issues de sources disparates, afin qu'elles soient analysées et utilisées de concert avec des données déjà existantes. A défaut, les objets connectés ne resteront que des gadgets livrant à l'utilisateur des données éphémères dont il fera un usage pour le moins limité.[14]

L'une des caractéristiques des données de l'IoT est de se présenter souvent sous forme de séries temporelles mais il n'est pas sûr que ceci confère à la donnée IoT un caractère très distinctif des données traditionnellement manipulées. Il est clair par contre que la valeur de l'IoT réside pour une large part dans l'exploitation de données issues de différentes origines. Pour en extraire cette valeur, il faut les agréger entre elles et avec les données préexistantes, et y ajouter

une dimension d'analyse voire d'apprentissage. Cette fusion des données en ensembles cohérents sera facilitée si des standards suffisamment précis sont élaborés et acceptés.

• **La gestion des systèmes :** les systèmes pouvant se réclamer de l'IoT sont des systèmes complexes qui impliquent le déploiement de fonctionnalités allant bien au-delà de la gestion des communications : identification et accueil des nouveaux équipements, procédures d'accueil ou de configuration de ces nouveaux objets (provisioning), management des réseaux, gestion de la sécurité.

• **La cybersécurité :** la cybersécurité représente un défi majeur pour le développement de l'IoT. Des normes existent ou sont en cours de finalisation afin de fixer les exigences à satisfaire par les systèmes d'information, y compris les systèmes de contrôle de procédé (séries ISO/IEC 27000 et IEC 62443 notamment). Mais ces normes ne sont pas spécifiquement conçues pour répondre aux préoccupations de l'IoT. Une extension et une adaptation seraient nécessaires. Par ailleurs il faudrait définir de façon normative, ce qu'on appelle objet "Secure by Design". A une époque où les citoyens sont particulièrement sensibilisés aux questions de sécurité, il serait hautement souhaitable que des mécanismes de certification, reposant sur des référentiels normatifs, puissent leur apporter l'assurance que les objets qu'ils acquièrent ou manipulent, répondent aux exigences essentielles en matière de cybersécurité. De telles exigences ont été définies au niveau national pour certaines catégories d'équipements (compteurs communicants, contrôleurs d'automatisme), il serait souhaitable que cette approche soit étendue à des gammes beaucoup plus larges d'équipements, afin de mieux garantir leur sécurité de fonctionnement et d'éviter qu'ils ne soient pris en otage dans des armées de zombies (botnets)

comme ce fut le cas dans les attaques en DDoS du deuxième semestre 2016.

I.7 Conclusion

Nous avons consacré la premiere partie de ce chapitre à l'étude des réseaux mobile adhoc (MANET), le routage des paquets suivant et les protocole proactif : DSDV, OLSR et réactif : DSR, AODV.

Dans la deuxieme partie , le concept de l'internet des objets (iot) en partant des définitions, aux differents composants à savoir les objets connectés, réseaux de communication et les données, les technologies utilisées : Les capteurs, communications et les traitements des données ,et en dernier les standarts de l'iot.

Chapitre II : La programmation réseaux

II.1 Introduction

Le transport des données d'une extrémité à une autre d'un réseau nécessite un support physique ou hertzien de communication. Pour que les données arrivent correctement au destinataire, avec la qualité de service, ou QoS (Quality of Service), exigée, il faut en outre une architecture logicielle chargée du contrôle des paquets dans le réseau. Les deux grandes architectures suivantes se disputent actuellement le marché mondial des réseaux :l'architecture OSI (Open Systems Interconnection), ou interconnexion de systèmes ouverts, provenant de la normalisation de l'ISO (International Standardization Organization) ; l'architecture TCP/IP utilisée dans le réseau Internet ; l'architecture introduite par l'UIT (Union internationale des télécommunications) pour l'environnement ATM (Asynchronous Transfer Mode).[1]

II.2 l'architecture OSI (Open Systems Interconnection)

Le modèle de référence OSI comporte sept niveaux, ou couches, plus un médium physique. Le médium physique, que l'on appelle parfois couche 0, correspond au support physique de communication chargé d'acheminer les éléments binaires d'un point à un autre jusqu'au récepteur final. Ce médium physique peut prendre diverses formes, allant du câble métallique aux signaux hertziens, en passant par la fibre optique et l'infrarouge.

Les concepts architecturaux utilisés pour décrire le modèle de référence à sept couches proposé par l'ISO sont décrits dans la norme ISO 7498-1. [1]

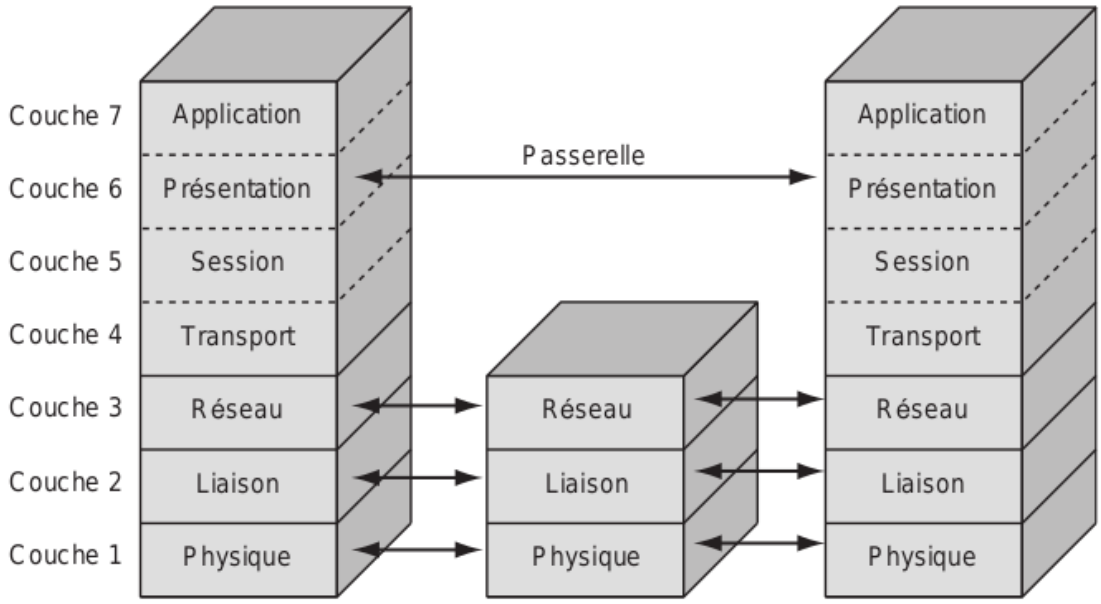

Figure 13: L'architecture OSI

II.2.1 La couche 1 (niveau physique)

Le niveau physique correspond aux règles et procédures à mettre en œuvre pour acheminer les éléments binaires sur le médium physique. On trouve dans le niveau physique les équipements réseau qui traitent l'élément binaire, comme les modems, concentrateurs, ponts, hubs, etc.

II.2.2 La couche 2 (niveau trame)

La trame est l'entité transportée sur les lignes physiques. Elle contient un certain nombre d'octets transportés simultanément. Le rôle du niveau trame consiste à envoyer un ensemble d'éléments binaires sur une ligne physique de telle façon qu'ils puissent être récupérés correctement par le récepteur.

II.2.3 La couche 3 (niveau paquet)

La couche 3, ou niveau paquet, peut aussi être appelée couche réseau, puisque l'échange de paquets de bout en bout donne naissance à un réseau. Le niveau paquet doit permettre d'acheminer correctement les paquets d'information jusqu'à l'utilisateur final. Pour aller de l'émetteur au récepteur, il faut passer par des nœuds de transfert intermédiaires ou par des passerelles, qui interconnectent deux ou plusieurs réseaux. Le niveau paquet nécessite trois fonctionnalités principales : le contrôle de flux, le routage et l'adressage

II.2.4 La couche 4 (niveau message)

Le niveau message prend en charge le transport du message de l'utilisateur d'une extrémité à une autre du réseau. Ce niveau est aussi appelé couche transport pour bien indiquer qu'il s'agit de transporter les données de l'utilisateur. Il représente le quatrième niveau de l'architecture, d'où son autre nom de couche 4.

II.2.5 La couche 5 (niveau session)

Le rôle du niveau session est de fournir aux entités de présentation les moyens nécessaires à l'organisation et à la synchronisation de leur dialogue. À cet effet, la couche 5 fournit les services permettant l'établissement d'une connexion, son maintien et sa libération, ainsi que ceux permettant de contrôler les interactions entre les entités de présentation.

II.2.6 La couche 6 (niveau présentation)

Le niveau présentation se charge de la syntaxe des informations que les entités d'application se communiquent. Deux aspects complémentaires sont définis dans la norme :

• La représentation des données transférées entre entités d'application.

• La représentation de la structure de données à laquelle les entités se réfèrent au cours de leur communication et la représentation de l'ensemble des actions effectuées sur cette structure de données.

En d'autres termes, la couche présentation s'intéresse à la syntaxe tandis que la couche application se charge de la sémantique. La couche présentation joue un rôle important dans un environnement hétérogène. C'est un intermédiaire indispensable pour une compréhension commune de la syntaxe des documents transportés sur le réseau. Les différentes machines connectées n'ayant pas la même syntaxe pour exprimer les applications qui s'y effectuent, si on les interconnecte directement, les données de l'une ne peuvent généralement pas être comprises de l'autre. La couche 6 procure un langage syntaxique commun à l'ensemble des utilisateurs connectés.

II.2.7 La couche 7 (niveau application)

Le niveau application est le dernier du modèle de référence. Il fournit aux processus applicatifs le moyen d'accéder à l'environnement réseau. Ces processus échangent leurs informations par l'intermédiaire des entités d'application.

Le niveau application contient toutes les fonctions impliquant des communications entre systèmes, en particulier si elles ne sont pas réalisées par les niveaux inférieurs. Il s'occupe essentiellement de la sémantique, contrairement à la couche présentation, qui prend en charge la syntaxe.[1]

II.3 L'architecture TCP/IP

Dans les années 70, le département de la Défense américain, ou DOD (Department Of Defense), décide, devant le foisonnement de machines utilisant des protocoles de communication différents et incompatibles, de définir sa propre architecture.

Cette architecture, dite TCP/IP, est à la source du réseau Internet. Elle est aussi adoptée par de nombreux réseaux privés, appelés intranets. Les deux principaux protocoles définis dans cette architecture sont les suivants :[1]

• IP (Internet Protocol), de niveau réseau, qui assure un service sans connexion.

• TCP (Transmission Control Protocol), de niveau transport, qui fournit un service fiable avec connexion.

TCP/IP définit une architecture en couches qui inclut également, sans qu'elle soit définie explicitement, une interface d'accès au réseau. En effet, de nombreux sous-réseaux distincts peuvent être pris en compte dans l'architecture TCP/IP, de type aussi bien local qu'étendu. Cette architecture est illustrée à la figure II.2. Il faut noter dans cette figure l'apparition d'un autre protocole de niveau message (couche

4), UDP (User Datagram Protocol). Ce protocole utilise un mode sans connexion, qui permet d'envoyer des messages sans l'autorisation du destinataire.

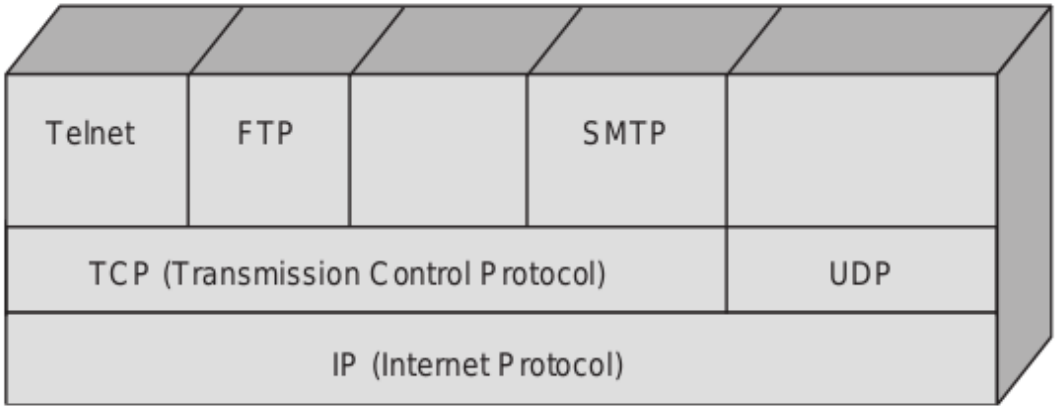

Figure 14: Architecture TCP/IP

Cette architecture a pour socle le protocole IP, qui correspond au niveau paquet (couche 3) de l'architecture du modèle de référence. En réalité, il ne correspond que partiellement à ce niveau. Le protocole IP a été conçu comme protocole d'interconnexion, définissant un bloc de données d'un format bien défini et contenant une adresse, mais sans autre fonctionnalité. Son rôle était de transporter ce bloc de données dans un paquet selon n'importe quelle autre technique de transfert de paquets. Cela vaut pour la première génération du protocole IP, appelée IPv4, qui est encore massivement utilisée aujourd'hui. La deuxième version du protocole IP, Ipv6, joue réellement un rôle de niveau paquet, avec de nouvelles fonctionnalités permettant de transporter les paquets d'une extrémité du réseau à une autre avec une certaine sécurité.
Les paquets IP sont indépendants les uns des autres et sont routés individuellement dans le réseau par le biais de routeurs. La qualité

de service proposée par le protocole IP est très faible, sans détection de paquets perdus ni de possibilité de reprise sur erreur.
Le protocole TCP regroupe les fonctionnalités de niveau message (couche 4) du modèle de référence. C'est un protocole assez complexe, qui comporte de nombreuses options permettant de résoudre tous les problèmes de perte de paquet dans les niveaux inférieurs. En particulier, un fragment perdu peut être récupéré par retransmission sur le flot d'octets. Le protocole TCP est en mode avec connexion, contrairement à UDP. Ce dernier protocole UDP se positionne aussi au niveau transport mais dans un mode sans connexion et n'offre pratiquement aucune fonctionnalité. Il ne peut prendre en compte que des applications qui demandent peu de service de la part de la couche transport.
Les protocoles situés au-dessus de TCP et d'UDP sont de type applicatif et proviennent en grande partie du monde UNIX.
Toute la puissance de cette architecture repose sur la souplesse de sa mise en œuvre au-dessus de n'importe quel réseau existant. Soi, par exemple, X et Y, respectivement un réseau local et un réseau étendu à commutation de cellules ou de paquets. Le protocole IP est implémenté sur toutes les machines connectées à ces deux réseaux. Pour qu'il soit possible de passer d'un réseau à l'autre, un routeur, dont le rôle est de décapsuler le paquet arrivant du réseau X et de récupérer le paquet IP, est mis en place. Après traitement, essentiellement du routage, le paquet IP est encapsulé dans le paquet du réseau Y. Le rôle du routeur est, comme son nom l'indique, de router le paquet vers la bonne destination.[1]

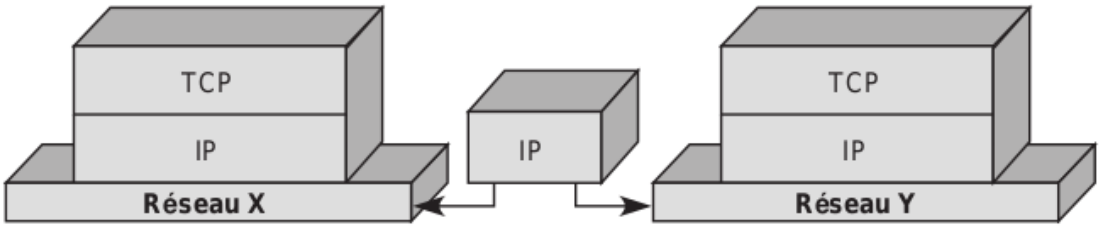

Figure 15: Architecture d'interconnexion du réseau Internet

II.4 La sémantique d'association

La sémantique d'association propose deux types de dialogue entre les entités communicantes : le mode avec connexion (connection oriented) et le mode sans connexion (connectionless oriented). [1]

II.4.1 Le mode avec connexion

La norme de base ISO 7498 définit explicitement la mise en place d'une connexion pour les communications entre des entités de même niveau. Elle indique qu'une entité de niveau N ne peut émettre de bloc d'information qu'après avoir demandé à l'homologue avec lequel elle souhaite communiquer la permission de le faire.

Pour mettre en place une connexion, le protocole de niveau N émet donc un bloc d'information contenant une demande de connexion de niveau N. Le récepteur a le choix d'accepter ou de refuser la connexion par l'émission d'un bloc de données indiquant sa décision. Dans certains cas, la demande de connexion peut être arrêtée par le gestionnaire du service, qui peut refuser de propager la demande de connexion jusqu'au récepteur, par exemple par manque de ressources internes. Une demande d'ouverture de circuit virtuel de niveau 3, qui n'est rien d'autre qu'une connexion réseau, peut ainsi être stoppée dans un nœud intermédiaire si la mémoire est insuffisante ou si la capacité d'émission est dépassée.[1]

La mise en place du mode avec connexion, permettant la communication entre entités homologues, se déroule en trois phases distinctes :

- Établissement de la connexion.
- Transfert des données de l'utilisateur d'une entité à l'autre.
- Libération de la connexion.

L'avantage du mode avec connexion est évident pour la sécurisation du transport de l'information. Puisque les émetteurs et les récepteurs se mettent d'accord, l'ensemble de l'activité du réseau est facilement contrôlable, tout au moins au niveau des nœuds extrémité. Au moment de l'ouverture d'une connexion, des paramètres peuvent de surcroît être passés entre l'émetteur et le récepteur pour équilibrer la transmission dans des limites admissibles par les deux extrémités. On parle en ce cas de négociation de la qualité de service, ou QoS (Quality of Service), laquelle s'effectue au moment de l'ouverture de la connexion. Pendant toute la durée de vie de la connexion, des paramètres peuvent être échangés entre les participants à la communication.

Le mode avec connexion présente cependant plusieurs difficultés, engendrées notamment par la lourdeur de la mise en place d'une connexion. Même pour n'envoyer que quelques octets, il faut mettre en place la connexion et discuter des valeurs des paramètres de service et, le cas échéant, de la qualité de service. S'il faut ouvrir une connexion à chaque niveau de l'architecture OSI, le temps d'émission de quelques octets est considérablement plus long que dans le mode sans connexion.

L'accès à des applications multipoint est par ailleurs délicat dans ce mode, puisqu'il faut ouvrir autant de connexions que de points à atteindre. Si, par exemple, on veut diffuser un fichier vers 1 000 utilisateurs distants, il est nécessaire d'ouvrir 1 000 connexions, c'est-

à-dire d'émettre 1 000 demandes de connexion, et ce à tous les niveaux de l'architecture.[1]

II.4.2 Le mode sans connexion

Dans le mode sans connexion, les blocs de données sont émis sans qu'il soit nécessaire de s'assurer au préalable que l'entité distante est présente. L'existence d'une connexion à l'un quelconque des niveaux de l'architecture est cependant nécessaire pour s'assurer que le service rendu n'est pas complètement inutile. Pour mettre en place une telle connexion, il faut utiliser les services des couches inférieures, ce qui implique nécessairement leur activité.

La principale difficulté d'une communication en mode sans connexion réside dans le contrôle de la communication, puisqu'il n'y a pas de négociation entre l'émetteur et le récepteur. Une station peut ainsi recevoir des données venant simultanément d'un grand nombre de stations émettrices, alors que, dans le mode avec connexion, la station réceptrice n'accepterait pas d'ouvrir autant de connexions. En raison de la difficulté à contrôler la communication, le gestionnaire du réseau doit souvent prendre plus de précautions dans une communication sans connexion que dans le mode avec connexion.

Le mode sans connexion est intéressant pour le transport de messages courts, tandis que celui avec connexion est plus adapté aux messages longs, à condition que les temps de mise en place et de libération des connexions soient négligeables par rapport à la durée de la communication. Comme expliqué précédemment, le mode avec connexion est privilégié dans la norme de base.

Si une connexion est réalisée à un niveau N, les niveaux supérieurs peuvent utiliser un mode sans connexion. Parmi les nombreuses applications qui peuvent utiliser le mode sans connexion, citons la messagerie électronique dans sa définition la plus large. La

messagerie est le moyen d'émettre de l'information vers un utilisateur lointain dont on ne sait s'il est présent ou non. Lorsque le client n'est pas actif, il est remplacé par une boîte aux lettres. La connexion de session s'effectue avec la machine qui gère cette boîte aux lettres. Quantité d'autres applications fonctionnent dans le mode sans connexion, notamment les suivantes :

• Transfert de fichiers. Il suffit de s'assurer que le représentant de l'utilisateur final est capable de mémoriser l'ensemble des données contenues dans le fichier.

• Conférence répartie. Différents clients mettent en commun des informations dans une boîte aux lettres spécialisée, accessible à l'ensemble des éléments du groupe. Cette application se satisfait très bien du mode messagerie.

• Accès à une base de données distribuée. Un utilisateur à la recherche d'informations d'un type non complètement spécifié émet sa demande en messagerie et obtient une réponse quelques heures plus tard.

• Transactionnel. Par essence, cette application fonctionne en mode avec connexion, mais elle peut aussi, dans le cas où le temps réel n'est pas nécessaire, se contenter d'un temps de réponse de quelques secondes au lieu d'une fraction de seconde. L'utilisation d'un mode sans connexion n'est alors pas contre-indiquée.[1]

II.5 Les Socket

II.5.1 Définition

La notion de sockets a été introduite dans les distributions de Berkeley (un fameux système de type UNIX, dont beaucoup de distributions actuelles utilisent des morceaux de code), c'est la raison pour laquelle on parle parfois de sockets BSD (Berkeley Software Distribution).[15]

le terme « socket » signifie douille, prise électrique femelle.

Il s'agit d'une API(Application Program Interface) qui permet la communication et échange de données entre dans la même machine ou à travers un réseau TCP/IP.

Un mécanisme universel de bas niveau, utilisable depuis tout langage (exemple : C, Java).

La communication par socket est souvent comparée aux communications humaines. On distingue ainsi deux modes de communication:

• Le mode connecté (comparable à une communication téléphonique), utilisant le protocole TCP. Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que la socket de destination n'est pas nécessaire à chaque envoi de données.

• Le mode non connecté (analogue à une communication par courrier), utilisant le protocole UDP. Ce mode nécessite l'adresse de destination à chaque envoi, et aucun accusé de réception n'est donné.

Les extrémités de communication sont identifiées (dans TCP/IP) par trois informations : le protocole de transport utilisé (TCP ou UDP), une adresse IP, un numéro de port (entier sur 16 bits, de 0 à 65535)

II.5.2 Les Sockets TCP

II.5.2.1 Le modèle client/serveur

Le protocole TCP offre un service en mode connecté et fiable. Les données sont délivrées dans l'ordre de leur émission. La procédure d'établissement de connexion est dissymétrique. Un processus, appelé serveur, attends des demandes de connexion qu'un processus, appelé client, lui envoie. Une fois l'étape

d'établissement de connexion effectuée le fonctionnement redeviens symétrique. Le schémas suivants présentent les algorithmes de fonctionnement des clients et serveurs. Il est à noter que côté serveur on utilise deux sockets : l'un, appelé socket d'écoute, reçoit les demandes de connexion et l'autre, appelé socket de service, sert pour la communication. En effet, un serveur peut être connecté simultanément avec plusieurs clients et dans ce cas on utilisera autant de sockets de service que de clients. **[16]**

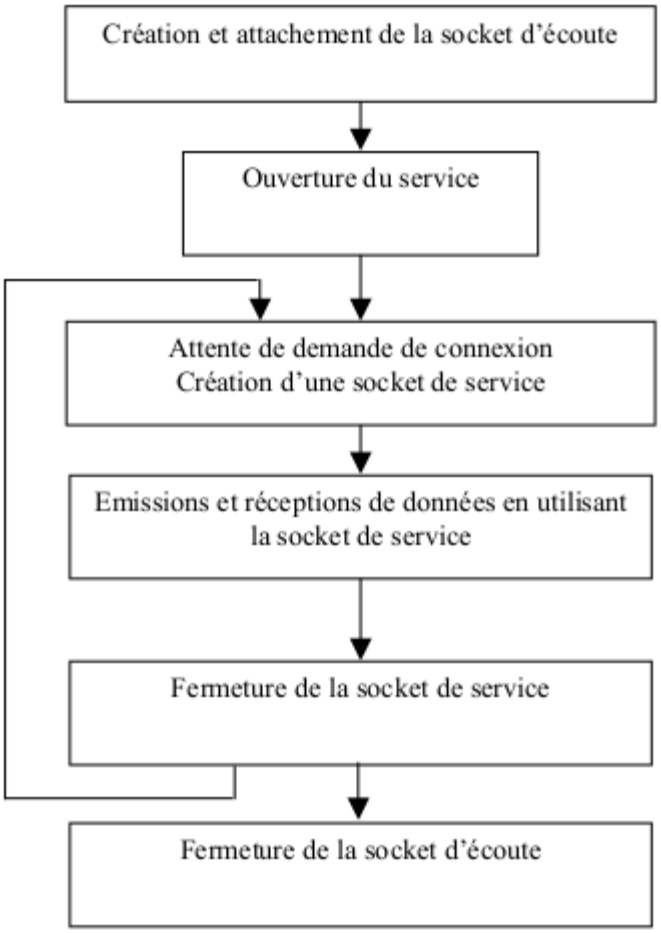

Figure 16: Fonctionnement du serveur

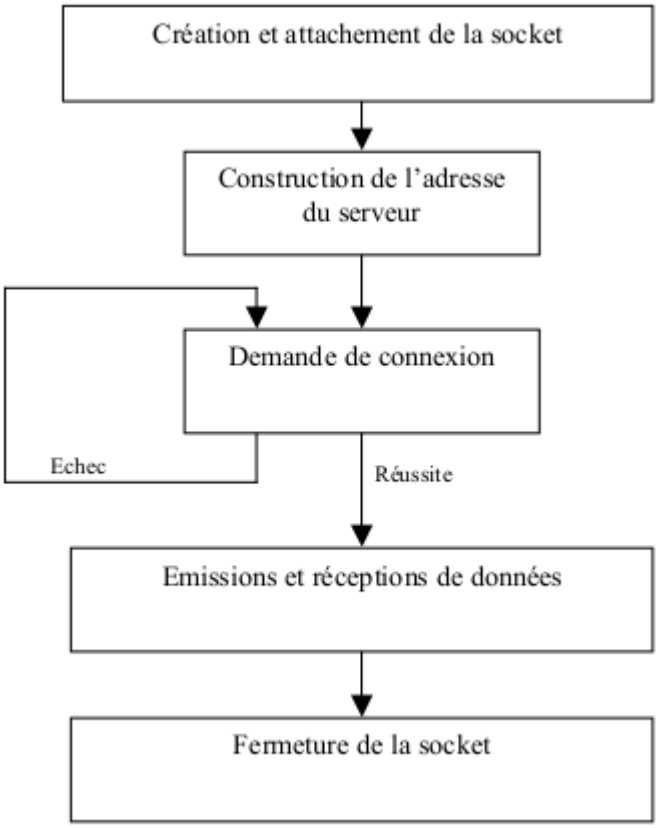

Figure 17: Fonctionnement du client

II.5.2.1 La classe socket

La classe Socket représente en Java les sockets utilisés côtés client ou les sockets de service.**[16]**

a. Constructeurs :

public Socket (String hote, int port) throws UnknownHostException, IOException

Ce constructeur crée un socket TCP et tente de se connecter sur le port indiqué de l'hôte visé. Le premier paramètre de ce constructeur représente le nom de la machine serveur. Si l'hôte est inconnu ou que le serveur de noms de domaine est inopérant, le constructeur

générera une UnknownHostException. Les autres causes d'échec, qui déclenchent l'envoi d'une IOException sont multiples : machine cible refusant la connexion sur le port précisé ou sur tous les ports, problème lié à la connexion Internet, erreur de routage des paquets...
ex : Socket leSocket = new Socket("www.socket.dz", 80);

public Socket (InetAddress addresse, int port) throws IOException

Ce constructeur fonctionne comme le premier, mais prends comme premier paramètre une instance de la classe InetAddress. Ce constructeur provoque une IOException lorsque la tentative de connexion échoue mais il ne renvoie pas d'UnknownHostException puisque cette information est connue lors de la création de l'objet InetAddress.

public Socket(String hote, int port, InetAddress addresseLocale, int portLocal) throws IOException

Comme les deux précédents, ce constructeur crée un socket et tente de se connecter sur le port indiqué de l'hôte visé. Le numéro du port et le nom de la machine sont fournis dans les deux premiers paramètres et la connexion s'établit à partir de l'interface réseau physique (choix d'une carte réseau sur un système possédant plusieurs accès réseau) ou virtuelle (système multi-adresse) et du port local indiqués dans les deux derniers paramètres.
Si portLocal vaut 0, la méthode (comme les deux premières d'ailleurs) choisit un port disponible (parfois appelé port anonyme) compris entre 1024 et 65 535. Comme la première méthode cette méthode peut générer une IOException en cas de problème de connexion et lorsque le nom d'hôte donné n'est pas correct (il s'agit dans ce cas d'une UnknownHostException).

protected Socket()

protected Socket(SocketImpl impl) throws SocketException

Ces deux derniers constructeurs sont protégés. Ils créent un socket sans le connecter. On utilise ces constructeurs pour implanter un socket original qui permettra par exemple de chiffrer les transactions ou d'interagir avec un Proxy.

b. Méthodes informatives :

public InetAddress getInetAddress ()

public int getPort ()

Ces méthodes renvoient l'adresse Internet et le port distants auquel le socket est connecté.

public InetAddress getLocalAddress ()

public int getLocalPort ()

Ces méthodes renvoient l'adresse Internet et le port locaux que le socket utilise.

c. Communication avec un socket :

public InputStream getInputStream () throws IOException

Cette méthode renvoie un flux d'entrées brutes grâce auquel un programme peut lire des informations à partir d'un socket. Il est d'usage de lier cet InputStream à un autre flux offrant d'avantage de fonctionnalités (un DataInputStream par exemple) avant d'acquérir les entrées.

Ex:

```
DataInputStream Entree = new DataInputStream(leSocket.getInputStream());
```

public OutputStream getOutputStream () throws IOException

Cette méthode renvoie un flux de sortie brutes grâce auquel un programme peut écrire des informations sur un socket. Il est d'usage de lier cet OutputStream à un autre flux offrant d'avantage de fonctionnalités (un DataOutputStream par exemple) avant d'émettre des données.

Ex:

```
DataOutputStream Sortie = new DataOutputStream(leSocket.getOutputStream());
```

d. Fermeture d'un socket

public void close() throws IOException

Bien que Java ferme tous les sockets ouverts lorsqu'un programme se termine , il est fortement conseillé de fermer explicitement les sockets dont on n'a plus besoin à l'aide de la méthode close.

Une fois un socket fermé on peut toujours utiliser les méthodes informatives, par contre toute tentative de lecture ou écriture sur les « input/output streams » provoque une IOException.

II.5.2.2 La classe ServerSocket

Cette classe permet de créer des sockets qui attendent des connections sur un port spécifié et lors d'une connexion retournent un Socket qui permet de communiquer avec l'appelant.

a. Constructeurs :

public ServerSocket (int port) throws IOException

Ce constructeur crée un socket serveur qui attendra les connexions sur le port spécifié. Lorsque l'entier port vaut 0, le port est sélectionné par le système. Ces ports anonymes sont peu utilisés car le client doit connaître à l'avance le numéro du port de destination. Il faut donc un mécanisme, comme le portmapper des RPC, qui permet d'obtenir ce numéro de port à partir d'une autre information.

L'echec de la création se solde par une IOException (ou à partir de Java 1.1 une BindException qui hérite de IOException) qui traduit soit l'indisponibilité du port choisi soit, sous UNIX, un problème de droits (sous UNIX les ports inférieurs à 1024 ne sont disponibles que pour le super utilisateur).

public ServerSocket (int port, int tailleFile) throws IOException

Ce constructeur permet en outre de préciser la longueur de la file d'attente des requêtes de connexion. Si une demande de connexion

arrive et que la file est pleine la connexion sera refusée sinon elle sera stockée dans la file pour être traitée ultérieurement. La longueur de la file d'attente par défaut (pour le premier constructeur) est 50.

Lorsque la valeur donnée est supérieure à la limite fixée par le système, la taille maximale est affectée à la file.

public ServerSocket (int port, int tailleFile, InetAddress adresseLocale) throws IOException

Ce constructeur permet en outre de préciser l'adresse Internet locale sur laquelle attendre des connexions. Ainsi on pourra choisir l'une des interfaces réseau de la machine locale si elle en possède plusieurs. Si adresseLocale est à null le socket attendra des connexions sur toutes les adresses locales (ce qui est aussi le cas quand on utilise l'un des deux autres constructeurs).

b. Accepter et clore une connexion :

public Socket accept () throws IOException

Cette méthode bloque l'exécution du programme serveur dans l'attente d'une demande de connexion d'un client. Elle renvoie un objet Socket une fois la connexion établie.

Si vous ne voulez pas bloquer l'exécution du programme il suffit de placer l'appel à accept dans un thread spécifique.

public void close () throws IOException

Cette méthode ferme le socket serveur en libérant le port. Les sockets serveurs sont eux aussi fermés automatiquement par le système à la fermeture de l'application.

II.5.2.3 Exemple de programme client

```
import java.io.*;
import java.net.*;
public class ClientEcho extends Object {
public static void main (String args[]) {
String  reponse;
Socket  leSocket;
PrintStream fluxSortieSocket;
BufferedReader fluxEntreeSocket;
try {
// creation d'une socket et connexion à la machine marine sur le port numéro 7
leSocket = new Socket("marine.edu.ups-tlse.fr", 7);
System.out.println("Connecté sur : "+leSocket);
// création d'un fluxSortieSocket flux de type PrintStream lié au flux de sortie de la
//socket
fluxSortieSocket = new PrintStream(leSocket.getOutputStream());
// creation d'un fluxEntreeSocket flux de type BufferedReader lié au flux d'entrée de
//la socket
fluxEntreeSocket = new BufferedReader(new
InputStreamReader(leSocket.getInputStream()));
// envoi de données vers le serveur
fluxSortieSocket.println("Bonjour le monde!");
// attente puis réception de données envoyées par le serveur
reponse = fluxEntreeSocket.readLine();
System.out.println("Reponse du serveur : " + reponse);
leSocket.close();
} // try
catch (UnknownHostException ex)
{
System.err.println("Machine inconnue : "+ex);
ex.printStackTrace();
}
catch (IOException ex)
{
System.err.println("Erreur : "+ex);
ex.printStackTrace();
}
} // main
} // class
```

II.5.2.3 Exemple de programme serveur

```
import java.io.*;
import java.net.*;
public class ServeurEcho extends Object {
```

```
public static void main (String args[]) {
ServerSocket socketEcoute;
Socket socketService;
InputStream entreeSocket;
OutputStream
sortieSocket;
try {
// création du socket d'écoute (port numéro 7)
socketEcoute = new ServerSocket(7);
while (true) {
// attente d'une demande de connexion
socketService = socketEcoute.accept();
System.out.println("Nouvelle connexion : " + socketService);
// récupération des flux d'entrée/sortie de la socket de service
entreeSocket = socketService.getInputStream();
sortieSocket = socketService.getOutputStream();
try {
int b = 0;
while (b != -1) {
b = entreeSocket.read();
sortieSocket.write(b);
} // while
System.out.println("Fin de connexion");
} // try
catch (IOException ex)
{
// fin de connexion
System.out.println("Fin de connexion : "+ex);
ex.printStackTrace();
}
socketService.close();
} // while (true)
} // try
catch (Exception ex)
{
// erreur de connexion
System.err.println("Une erreur est survenue : "+ex);
ex.printStackTrace();
}
} // main
} // class
```

II.5.3 Les Sockets UDP

Le protocole UDP offre un service non connecté et non fiabilisé. Les données peuvent être perdues, dupliquées ou délivrées dans un ordre différent de leur ordre d'émission.
Le fonctionnement est ici symétrique. Chaque processus crée un socket et l'utilise pour envoyer ou attendre et recevoir des données.
En Java on utilise la classe DatagramPacket pour représenter les datagrammes UDP qui sont échangés entre les machines.**[16]**

II.5.3.1 La classe DatagramPacket

a. Constructeurs :

public DatagramPacket(byte[] tampon, int longueur)

Ce constructeur crée un objet utilisé pour recevoir les données contenues dans des datagrammes UDP reçus par la machine. Le paramètre tampon doit correspondre à un tableau d'octets (type byte en java) correctement créé (attention si le tableau n'est pas assez grand les données en excès seront détruites). Le paramètre longueur indique la longueur du tableau.
Ex: byte[] tampon = new byte[1024];
DatagramPacket datagramme = new DatagramPacket(tampon, tampon.length);

public DatagramPacket(byte[] tampon, int longueur, InetAddress adresse, int port)

Ce constructeur crée un objet utilisé pour envoyer un datagramme UDP. Le paramètre tampon doit correspondre aux données à envoyer. Le paramètre longueur doit indiquer la longueur des données à envoyer.
Le paramètre adresse et le paramètre port doivent indiquer respectivement l'adresse IP et le port de destination.

Il est à noter qu'il existe d'autres constructeurs qui permette d'indiquer un offset dans un tableau de grande dimension. Ils doivent être utilisés pour envoyer un tableau dont la taille dépasse la taille maximale des datagrammes IP.

Ex: String message = "Bonjour le monde!";
byte[] tampon = message.getBytes();
InetAddress adresse = InetAddress.getByName("marine.edu.ups-tlse.fr");
DatagramPacket datagramme = new DatagramPacket(tampon, tampon.length, adresse, 7);

b. Accesseurs :

public byte[] getData()

public void setData(byte[] buf)

Ces méthodes permettent de récupérer et de changer le tableau d'octets utilisé soit pour recevoir soit pour envoyer un datagramme UDP.

public void setLength(int length)

public int getLength()

Ces méthodes permettent de récupérer et de changer la longueur du tableau d'octets utilisé soit pour recevoir soit pour envoyer un datagramme UDP.

Public InetAddress getAddress()

public void setAddress(InetAddress iaddr)

public int getPort()

public void setPort(int iport)

Ces méthodes permettent de récupérer et de changer les adresses IP et numéros de ports. Si l'objet est utilisé pour envoyer des datagrammes UDP il s'agira de l'adresse et du port de destination sinon il s'agira de l'adresse et du port de l'émetteur du datagramme UDP.

II.5.3.2 La classe DatagramSocket

a. Constructeurs :

public DatagramSocket() throws SocketException

Ce constructeur crée un socket UDP qui permet d'envoyer et recevoir des datagrammes UDP. Le port qu'utilise ce socket est choisi par le système d'exploitation. De même l'adresse IP qu'utilise ce socket est choisie automatiquement par le système.

public DatagramSocket(int port) throws SocketException

Ce constructeur crée un socket UDP qui permet d'envoyer et recevoir des datagrammes UDP. Le port qu'utilise ce socket est indiqué par le paramètre port. L'adresse IP qu'utilise ce socket est choisie automatiquement par le système.

public DatagramSocket(int port, InetAddress adresse) throws SocketException

Ce constructeur crée un socket UDP qui permet d'envoyer et recevoir des datagrammes UDP. Le port qu'utilise ce socket est indiqué par le paramètre port. L'adresse IP qu'utilise ce socket est indiqué par le paramètre adresse.

Ces trois constructeurs peuvent générer des exceptions de type SocketException si il y a un problème de configuration réseau, si le port choisi est déjà utilisé ou si l'adresse choisie n'est pas l'une des adresses de la machine locale.

b. Emission et Réception de Datagrammes :

public void send(DatagramPacket p) throws IOException

Envoie un datagramme UDP qui contiendra toutes les informations referencées par l'objet p.

public void receive(DatagramPacket p) throws IOException

Attends un datagramme UDP et lors de sa réception stocke ses informations dans l'objet p.

Des exceptions de type IOException peuvent être générées si un problème d'entrée/sortie survient.

c. Méthodes informatives :

public InetAddress getLocalAddress ()

public int getLocalPort ()

Ces méthodes renvoient l'adresse Internet et le port locaux que le socket utilise.

d. Fermeture du socket :

public void close() throws IOException

Bien que Java ferme tous les sockets ouverts lorsqu'un programme se termine ou bien lors d'un « garbage collect », il est fortement conseillé de fermer explicitement les sockets dont on n'a plus besoin à l'aide de la méthode close.

Une fois un socket fermé on peut toujours utiliser les méthodes informatives, par contre toute tentative d'émission ou de réception de datagrammes UDP provoque une IOException.

II.5.3.3 La classe MulticastSocket

Cette classe permet d'utiliser le multicasting IP pour envoyer des datagrammes UDP à un ensemble de machines repéré grâce à une addresse multicast (classe D dans IP version 4 : de 224.0.0.1 à 239.255.255.255) .

a. Constructeurs :

Les constructeurs de cette classe sont identiques à ceux de la classe DatagramSocket.

b. Abonnement/résiliation

Pour pouvoir recevoir des datagrammes UDP envoyés grâce au multicasting IP il faut s'abonner à une adresse multicast (de classe D pour IP version 4). De même lorsqu'on ne souhaite plus recevoir des datagrammes UDP envoyés à une adresse multicast on doit indiquer la résiliation de l'abonnement.

public void joinGroup(InetAddress adresseMulticast) throws IOException

Cette méthode permet de s'abonner à l'adresse multicast donnée.
Note : un même socket peut s'abonner à plusieurs adresses multicast simultanément. D'autre part il n'est pas nécessaire de s'abonner à une adresse multicast si on veut juste envoyer des datagrammes à cette adresse. Une IOException est générée si l'adresse n'est pas une adresse multicast ou si il y a un problème de configuration réseau.

public void leaveGroup(InetAddress adresseMulticast) throws IOException

Cette méthode permet de résilier son abonnement à l'adresse multicast donnée.
Une IOException est générée si l'adresse n'est pas une adresse multicast, si la socket n'était pas abonnée à cette
adresse ou si il y a un problème de configuration réseau.
esse ou si il y a un problème de configuration réseau.

c. Choix du TTL :

Dans les datagrammes IP se trouve un champ spécifique appelé TTL (Time To Live – durée de vie) qui est normalement initialisé à 255 et décrémenté par chaque routeur que le datagramme traverse. Lorsque ce champ atteint la valeur 0 le datagramme est détruit et une erreur ICMP est envoyé à l'émetteur du datagramme. Cela permet d'éviter que des datagrammes tournent infiniment à l'intérieur d'un réseau IP.
Le multicasting IP utilise ce champ pour limiter la portée de la diffusion. Par exemple avec un TTL de 1 la diffusion d'un datagramme est limitée au réseau local.

public int getTimeToLive() throws IOException
public void setTimeToLive(int ttl) throws IOException

Ces méthodes permettent la consultation et la modification du champ TTL qui sera écrit dans les datagrammes envoyés par ce socket.

II.5.3.4 Exemple de programme qui envoie un datagramme

```
import java.net.*;
import java.io.*;
class EnvoiDatagramme
{
public static void main(String argv[])
throws SocketException, IOException,
UnknownHostException
{
String message = "Bonjour le monde!";
byte[] tampon = message.getBytes();
InetAddress adresse = null;
DatagramSocket socket;
// recupère l'adresse IP de la machine marine
adresse = InetAddress.getByName("marine.edu.ups-tlse.fr");
// crée l'objet qui stockera les données du datagramme à envoyer
DatagramPacket envoi=
new DatagramPacket(tampon,tampon.length,adresse,50000);
// crée un socket UDP (le port est choisi par le système)
socket=new DatagramSocket();
// envoie le datagramme UDP
socket.send(envoi);
6.5.
Exemple de programme qui reçoit un datagramme
import java.net.*;
import java.io.*;
class ReceptionDatagramme
{
public static void main(String argv[])
throws SocketException, IOException
{
byte[] tampon = new byte[1000];
```

```
String texte;
// crée un socket UDP qui attends des datagrammes sur le port 50000
DatagramSocket
socket =new DatagramSocket(50000);
// crée un objet pour stocker les données du datagramme attendu
DatagramPacket
reception = new DatagramPacket(tampon, tampon.length);
// attends puis récupère les données du datagramme
socket.receive(reception);
// récupère la chaîne de caractère reçue
// Note: reception.getLength() contient le nombre d'octets reçus
texte=new String(tampon, 0, reception.getLength());
System.out.println("Reception de la machine "+
reception.getAddress().getHostName()+
" sur le port "
+reception.getPort()+" :\n"+
texte );
}
}
```

II.5.3.5 Exemple de programme qui envoie et reçoit des datagrammes avec le multicasting IP

```
import java.net.*;
import java.io.*;
class MulticastingIP
{
public static void main(String argv[])
throws SocketException, IOException
{
String msg = "Bonjour le monde!";
// on traveillera avec l'adresse multicast 228.5.6.7
InetAddress groupe = InetAddress.getByName("228.5.6.7");
// crée le socket utilisé pour émettre et recevoir les datagrammes
// il utilisera le port 50000
MulticastSocket s = new MulticastSocket(50000);
// s'abonne à l'adresse IP multicast
```

```
s.joinGroup(groupe);
// crée l'objet qui stocke les données du datagramme à envoyer
DatagramPacket envoi = new DatagramPacket(msg.getBytes(), msg.length(),
groupe, 50000);
// envoie le datagramme a tout le monde
s.send(envoi);
while (true) {
byte[] tampon = new byte[1024];
DatagramPacket reception = new DatagramPacket(tampon, tampon.length);
// attends les réponses
s.receive(reception);
String texte=new String(tampon, 0, reception.getLength());
System.out.println("Reception de la machine "+
reception.getAddress().getHostName()+
" sur le port "
+reception.getPort()+" :\n"+
texte );
}
// si la boucle n'était pas infinie on pourrais écrire:
// s.leaveGroup(groupe);
}
}
```

II.6 Conclusion

Dans ce chapitre nous avons vu les deux fameuses architectures OSI et TCP/IP , le dialogue client/serveur dans le mode connecté(TCP) et le mode non connecté(UDP) à travers les sockets. En dernier la programmation en java coté client/serveur et les différentes classes utilisées .

Chapitre III : La programmation mobile

III.1 Introduction

Dans ce chapitre on va étudier comment coder des application sur des platforme mobile en décrivant les différents type d'une application mobile, ces avantages et incovenients.

L'utilisation d' Android pour developer des applications mobile

III.2 Application mobile

Une application mobile est un logiciel applicatif développé pour être installé sur un appareil électronique mobile, comme un Smartphone, une tablette ou un baladeur numérique.
Une application mobile peut être soit installée directement sur l'appareil dès sa fabrication en usine soit téléchargée depuis un magasin d'applications dit « application store » telle que Google Play, l'App Store ou encore le Windows Phone Store. Une partie des applications disponibles sont gratuites tandis que d'autres sont payantes.
Il existe plusieurs systèmes d'exploitation mobiles (OS) dont les plus répandus sont les suivants :

- iOS (Apple) utilisé sur iPhone et iPad,
- Android (Google) qui anime un grand nombre de smartphones tels que Samsung, HTC, LG, Motorola...
- Blackberry OS,
- Windows Phone (Microsoft),
- Symbian (Nokia),
- Bada (Samsung).

III.3 Types d'application mobile

Techniquement, il y a trois types d'application mobile que tout utilisateur peut rencontrer:

a. Application native: Il s'agit d'application conçue pour une grande partie de systèmes d'exploitation fiables par les smartphones en se référant à un langage particulier à chacun d'eux.

Ce mode d'application est accessible seulement sur les plateformes d'applications suivent ses particularités et ses formules.

Le développement de l'application native nécessite le recours à la mémoire du smartphone sans omettre les options reliées au système d'exploitation en question. De cette façon, le résultat se résume dans l'aboutissement à des applications mobiles avec des fonctions plus professionnelles, développées et performantes au même niveau que les applications en HTML5/CSS3 et les applications hybrides.

Le souci des applications natives est que les utilisateurs doivent avoir un système d'exploitation mobile donné pour qu'ils peuvent les utiliser. Pour assurer un usage plus exponentiel de ces applications mobiles, il faut penser à lancer la même application_mobile compatible à tout système d'exploitation mobile.

b. Application web: Toute application conçue avec HTML et CSS de plus opérationnelle sur navigateur internet pour un smartphone est appelée application web. Peu importe la marque de votre smartphone, vous pouvez accéder à l'application web par le biais de son navigateur et donc vous n'avez pas besoin de la télécharger. Vu qu'elle ne tient pas en compte les divergences persistantes entre les systèmes d'exploitation et les marques de smartphone, l'application web manque d'ergonomie et de plus elle ne se sert pas de la mémoire du smartphone ce qui la place en infériorité par rapport à l'application native.

c. Application hybride: Il s'agit d'une application mobile qui fusionne entre les caractéristiques de web application (développement en HTML 5) et celles de l'application native. De cette manière, l'application mobilesera accessible sur toutes les plateformes d'application. Ce type d'application mobile minimise les charges et la durée de son développement même si cela sera au détriment de perfectionnement et de la qualité qui caractérise l'application native. Notons que les applications hybrides sont accessibles exclusivement sur iPhone et Android.

III.4 Avantages et incovenients d'application mobile

III.4.1 Avantages

a. Une parfaite ergonomie est assurée pour les applications mobiles en comparaison aux sites mobiles cela encourage les utilisateurs à demeurer fidèles aux applis. En effet le developpement d'application mobile tient compte la taille du smartphone, le temps de chargement et autres paramètres.

b. Les applications mobiles favorisent l'intégration des options de téléphone et ainsi, l'expérience utilisateur devient plus développée.

c. Facile à trouver sur les stores par rapport aux sites mobiles, les applications mobiles ont connu ainsi un usage plus répandu auprès des jeunes surtout qu'elles notifient sur les événements en cours.

III.4.1 Inconvenients

a. La soumission aux normes et règles éditées par les sociétés des plateformes mobiles à savoir Apple, Google, Windows et autres.

b. Un investissement lourd pour le développement d'une application mobile adaptée à chaque système d'exploitation mobile contrairement au cout qu'exige le développement d'un site mobile.

c. Lors de toute mise à jour d'application mobile, le mobinaute se trouve dans l'obligation de la faire à travers le store alors que le site mobile se met à jour d'une manière automatique.[17]

III.5 Android

Android était développé par la startup Android Inc. en 2003, puis racheté par Google en 2005. Pour pouvoir réalisé un système complet, ouvert et gratuit dans le monde du mobile, une coalition de 35 entreprise évoluant dans l'univers du mobile, dont Google, a été créé. Ce rassemblement se nomme l'Open Handset Alliance (OHA) et compose aujourd'hui de 80 membres. Cette alliance a pour but de développer un système open source « c'est-à-dire dont les sources sont disponible librement sur internet » pour l'exploitation sur mobile, Android. Android est à l'heure actuelle le système d'exploitation pour smartphones et tablettes le plus utilisé. Les terminaux visés par Android inclus les téléphones portables, Netbook/Smartbook, tablettes multimédia, automobile, GPS, Réfrigérateur, etc.

III.5.1 Architecture du système Android

La figure suivante illustre les composants principaux du système d'exploitation Android.

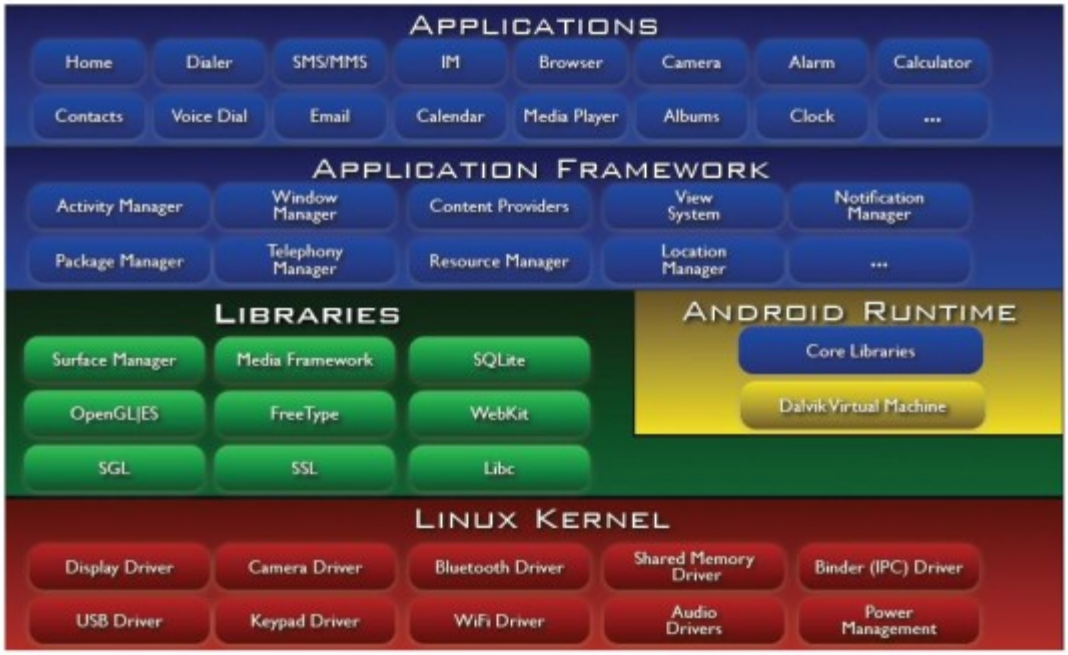

Figure 18: Les composants principaux de système d'exploitation Android

a. Linux Kernel : Android est basé sur un kernel linux 2.6. Alors, il y a plusieurs avantages comme la grande mémoire, la gestion du processus, le modèle de sécurité et le soutien de bibliothèque partagé.

b. Libraries : les librairies C/C++ utilisées par un certain nombre de composants du système Android.

c. l'Android Runtime : Cette couche contient les librairies cœurs du Framework ainsi que la machine virtuelle exécutant les applications.

d. Le framework : le cadre de travail permettant au développeur de créer des applications.

c. Les applications : Ce sont les applications qui marchent sous la plateforme Android comme le réveil, la calculatrice, le calendrier, la caméra, les contacts, etc. Toutes les applications sont développées en Java.

III.5.2 Cycle de vie d'une activité (Activity Lifecycle)

Pour développer d'une application sur Android, on doit comprendre le cycle de vie d'une activité.

- **L'état Active/courant (Running) :** l'activité se trouve au premier plan, et reçoit les interactions utilisateurs. Si l'appareil a besoin des ressources, l'activité se trouvant en bas de la back est tuée.
- **L'état Paused (il est en pause) :** L'activité est visible mais l'utilisateur ne peut pas interagir avec. La seule différence avec l'état précédent est la non-réception des événements utilisateurs.
- **L'état Stopped :** L'activité n'est pas visible mais toujours en cours d'exécution. Toutes les informations relatives à son exécution sont conservées en mémoire.
- **L'état Dead :** l'activité est tuée, elle n'est plus en cours d'exécution, et disparait de la back stack.
- **Back stack :** toutes les activités sont stockées dans une liste que l'on appelle généralement back stack.Il existe trois boucles principales:

Lors de la création d'activité, la méthode onCreate est appelée .Cet appel est suivi par la méthode onStart afin de signifier le lancement effective de l'application. Puis la méthode onResume est appelée afin d'exécuter tous les traitements nécessaires au fonctionnement de l'activité. Ces traitement devront être arrêtés lors d'appel à la méthode onPause est relancés si besoin lors d'un futur appel à la méthode onResume. **[18]**

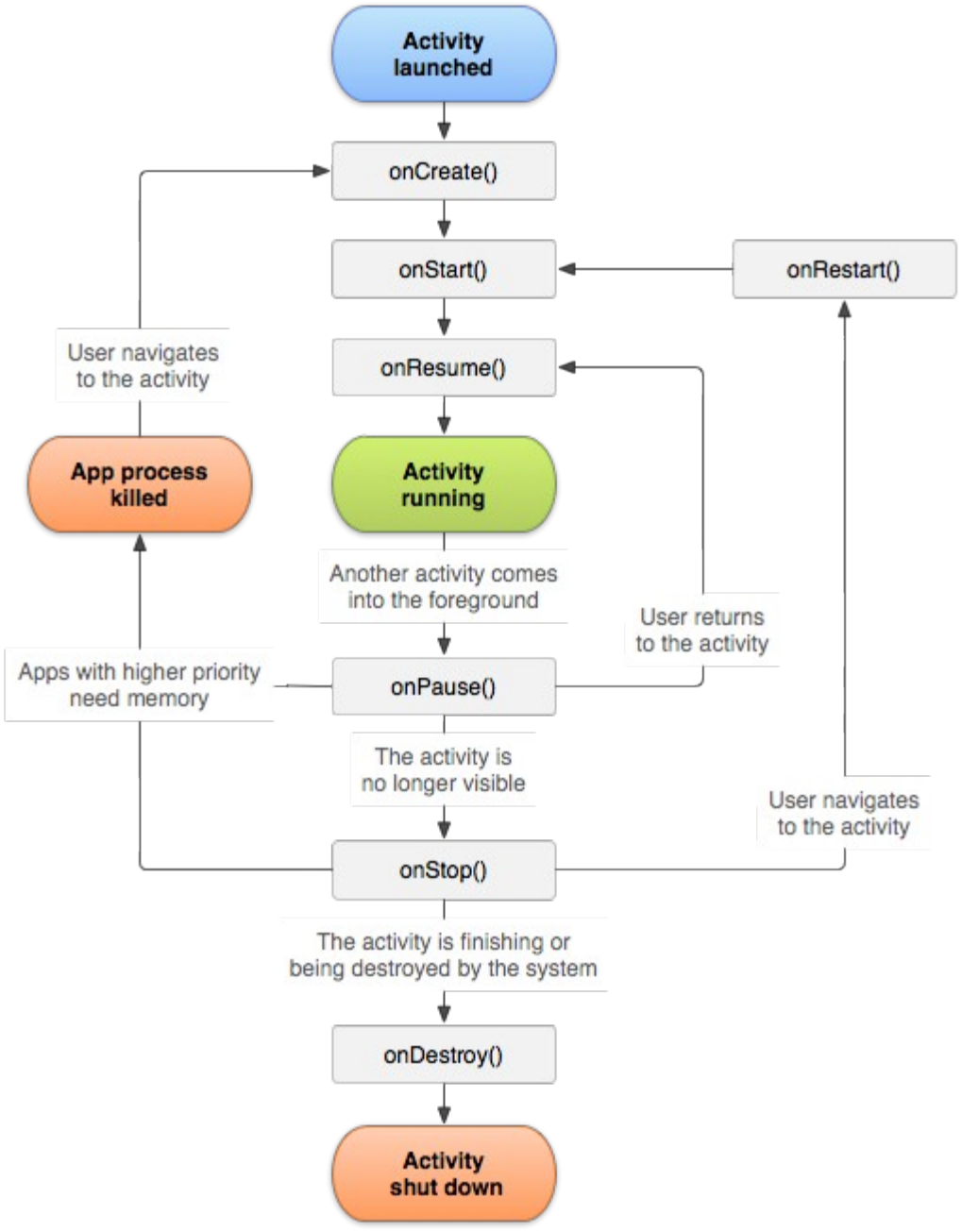

Figure 19: Cycle de vie d'une activité

III.6 Conclusion

Nous avons vue dans ce chapitre les applications mobiles leurs types et définir les différents composants et le cycle de vie d'une application Android.

Chapitre IV: Création réseau adhoc à base smartphone

IV.1 Introduction

Dans ce chapitre on commencera avec l'objectif de ce thème et l'aspect théorique utilisé et l'implémentation pratique puis les outils utilisés et l'Environnement pour développer ce projet et cette application mobile. Dans la fin nous avons démontré en pratique notre application de partage fichiers et de chat.

IV.2 Objectifs

Ce projet comprend le développement d'une application de connexion avec un réseau ad hoc, l'échange de messages (chat) et le partage de fichiers , basé sur la plateforme Android studio. Ce type de réseau permet aux utilisateurs d'échanger des messages (chat) et partager des fichiers sur leurs propres réseaux Ad Hoc, il aide donc les participants d'une session de se connecter entre eux localement sans l'utilisation d'un réseau avec architecture ou l'utilisation d'un routeur(point d'acces).

IV.3 Environnement d'implémentation

On a opté pour la plateforme Android studio pour développer cette application qui est basée sur java et XML .

IV.3.1 Android studio

Android Studio est l'environnement de développement intégré (IDE) officiel pour le développement d'applications Android, basé sur IntelliJ IDEA. En plus du puissant éditeur de code et des outils de développement d'IntelliJ, Android Studio offre encore plus de fonctionnalités qui améliorent votre productivité lors de la création d'applications Android, telles que :

- Un système de construction flexible basé sur Gradle

- Un émulateur rapide et riche en fonctionnalités
- Un environnement unifié où vous pouvez développer pour tous les appareils Android
- Appliquer les modifications au code push et aux modifications de ressources à votre application en cours d'exécution sans redémarrer votre application
- Modèles de code et intégration GitHub pour vous aider à créer des fonctionnalités d'application communes et à importer un exemple de code
- Outils et cadres de test étendus
- Outils Lint pour détecter les performances, la convivialité, la compatibilité des versions et d'autres problèmes
- Prise en charge de C ++ et NDK
- Prise en charge intégrée de Google Cloud Platform, facilitant l'intégration de Google Cloud Messaging et d'App Engine [19]

IV.3.2 java

Java est défini comme un langage orienté objet similaire au C++, mais simplifié pour éliminer les fonctionnalités du langage qui provoquent des erreurs de programmation courantes. Les fichiers de code source (fichiers avec une extension .java) sont compilés dans un format appelé bytecode (fichiers avec une extension .class), qui peuvent ensuite être exécutés par un interpréteur Java. Le code Java compilé peut s'exécuter sur la plupart des ordinateurs car les interpréteurs Java et les environnements d'exécution, appelés machines virtuelles Java (VM), existent pour la plupart des systèmes d'exploitation, y compris UNIX, le système d'exploitation Macintosh et Windows. Le bytecode peut également être converti directement en

instructions en langage machine par un compilateur juste à temps (JIT). En 2007, la plupart des technologies Java ont été publiées sous la licence publique générale GNU.[20]

IV.3.2 XML

Qu'est-ce que XML?

Le langage XML (Extensible Markup Language) est un format texte simple pour représenter des informations structurées: documents, données, configuration, livres, transactions, factures, et bien plus encore. Il a été dérivé d'un ancien format standard appelé SGML (ISO 8879), afin d'être plus adapté à une utilisation Web.

À quoi sert XML?

XML est l'un des formats les plus utilisés pour partager des informations structurées aujourd'hui: entre programmes, entre personnes, entre ordinateurs et personnes, à la fois localement et à travers les réseaux.

IV.4 Architecture de notre application

Dans ce qui suit, nous présenterons l'architecture et le fonctionnement de notre application de chat et partage fichier entre les peers (les appareils). Un diagramme représentant l'architecture de notre application OurLocalChat est présente dans la (Figure 20). On a utilisé le wifi direct api dans le développement de cette application.

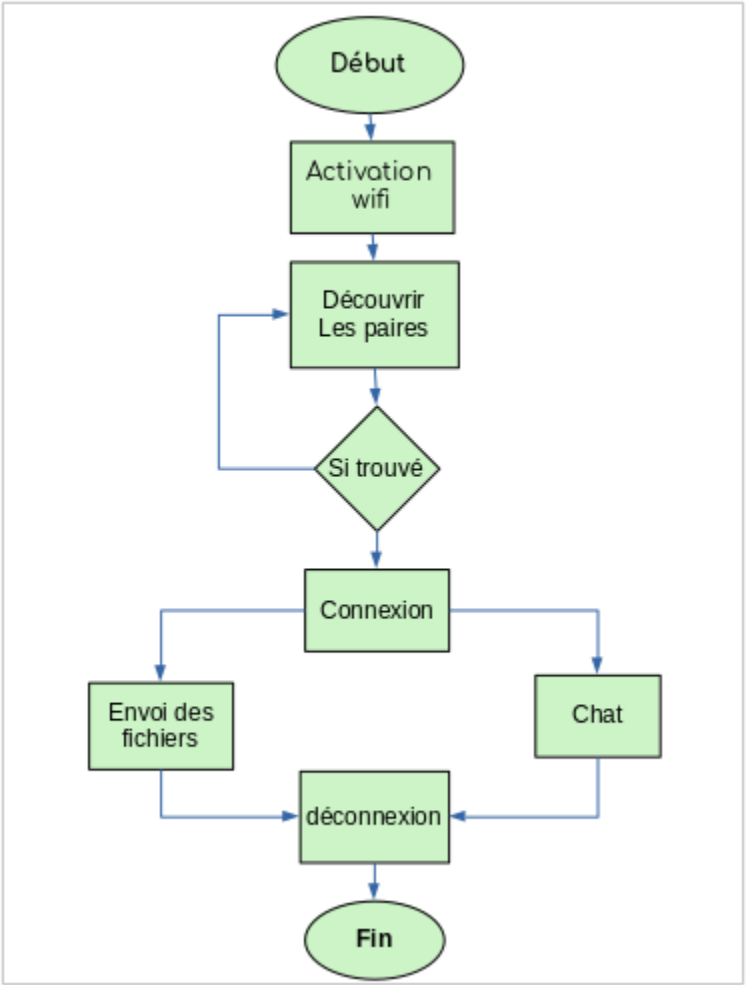

Figure 20: diagramme de fonctionnement de notre application

IV.5 Les principaux classes et méthodes de notre application

IV.5.1 wifi direct API

La WifiP2pManagerclasse fournit des méthodes pour vous permettre d'interagir avec le matériel Wi-Fi de votre appareil pour faire des choses comme découvrir et vous connecter à des pairs. Les actions suivantes sont disponibles :[21]

IV.5.1.1 Méthodes Wi-Fi P2P

Méthode La description

- **initialize() :** Enregistre l'application avec le framework Wi-Fi. Cela doit être appelé avant d'appeler toute autre méthode Wi-Fi P2P.
- **connect() :** Démarre une connexion peer-to-peer avec un périphérique avec la configuration spécifiée.
- **cancelConnect() :** Annule toute négociation de groupe peer-to-peer en cours.
- **requestConnectInfo() :** Demande les informations de connexion d'un appareil.
- **createGroup() :** Crée un groupe peer-to-peer avec le périphérique actuel comme propriétaire du groupe.
- **removeGroup() :** Supprime le groupe peer-to-peer actuel.
- **requestGroupInfo() :** Demande des informations de groupe d'égal à égal.
- **discoverPeers() :** Lance la découverte de pairs.
- **requestPeers() :** Demande la liste actuelle des pairs découverts.

WifiP2pManager Les méthodes vous permettent de passer un auditeur, afin que le framework Wi-Fi P2P puisse notifier votre activité de l'état d'un appel. Les interfaces d'écouteur disponibles et les

WifiP2pManager appels de méthode correspondants qui utilisent les écouteurs sont décrits dans le tableau suivant:

Écouteurs P2P Wi-Fi

Interface d'écoute Actions associées WifiP2pManager.ActionListener connect(), cancelConnect(), createGroup(), removeGroup() Et discoverPeers()

WifiP2pManager.ChannelListener initialize()

WifiP2pManager.ConnectionInfoListener requestConnectInfo()

WifiP2pManager.GroupInfoListener requestGroupInfo()

WifiP2pManager.PeerListListener requestPeers()

Les API Wi-Fi P2P définissent les intentions qui sont diffusées lorsque certains événements Wi-Fi P2P se produisent, par exemple lorsqu'un nouveau pair est découvert ou lorsque l'état Wi-Fi d'un appareil change.

IV.5.1.2 Notre utilisation de l'API

```
WifiP2pManager.PeerListListener peerListListener = new
WifiP2pManager.PeerListListener() {
    @Override
    public void onPeersAvailable(WifiP2pDeviceList peerList) {
        if(!peerList.getDeviceList().equals(peers))
        {
            peers.clear();
            peers.addAll(peerList.getDeviceList());

            deviceNameArray = new
String[peerList.getDeviceList().size()];
            deviceArray = new
WifiP2pDevice[peerList.getDeviceList().size()];
            int index = 0;

            for(WifiP2pDevice device : peerList.getDeviceList())
            {
                deviceNameArray[index] = device.deviceName;
                deviceArray[index] = device;
                index++;
            }
```

```
            ArrayAdapter<String> adapter = new
ArrayAdapter<String>(getApplicationContext(),android.R.layout.simple_li
st_item_1,deviceNameArray);
            listView.setAdapter(adapter);
        }

        if(peers.size()==0)
        {
            Toast.makeText(getApplicationContext(),"No Device
Found",Toast.LENGTH_SHORT).show();
            return;
        }
    }
};

WifiP2pManager.ConnectionInfoListener connectionInfoListener = new
WifiP2pManager.ConnectionInfoListener() {
    @Override
    public void onConnectionInfoAvailable(WifiP2pInfo wifiP2pInfo) {
        final InetAddress groupOwnerAddress =
wifiP2pInfo.groupOwnerAddress;

        if(wifiP2pInfo.groupFormed && wifiP2pInfo.isGroupOwner)
        {
            connectionStatus.setText("Host");
            serverClass = new ServerClass();
            serverClass.start();
        }else if(wifiP2pInfo.groupFormed)
        {
            connectionStatus.setText("Client");
            clientClass = new ClientClass(groupOwnerAddress);
            clientClass.start();
        }
    }
};
```

IV.5.1.3 La classe « WifiDirectBroadcastReceiver »

```
public WifiDirectBroadcastReceiver(WifiP2pManager mManager,
WifiP2pManager.Channel mChannel, MainActivity mActivity)
    {
        this.mManager = mManager;
        this.mChannel = mChannel;
        this.mActivity = mActivity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action))
```

```
{
            int state =
intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE,-1);

            if(state== WifiP2pManager.WIFI_P2P_STATE_ENABLED){
                Toast.makeText(context,"Wifi is ON",
Toast.LENGTH_SHORT).show();
            }else {
                Toast.makeText(context,"Wifi is OFF",
Toast.LENGTH_SHORT).show();
            }
        }else
if(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)){
            //do something
            if(mManager!=null)
            {

mManager.requestPeers(mChannel,mActivity.peerListListener);
            }
        }else
if(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)){
            //do something
            if (mManager==null)
            {
                return;
            }

            NetworkInfo networkInfo =
intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);

            if (networkInfo.isConnected())
            {

mManager.requestConnectionInfo(mChannel,mActivity.connectionInfoListener
);
            }else {
                mActivity.connectionStatus.setText("Device
Disconnected");
                mActivity.connectBtn.setVisibility(View.INVISIBLE);

mActivity.getSupportActionBar().setTitle("OurLocalChat");
            }
        }else
if(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)){
            //do something
        }

    }
}
```

IV.5.2 Utilisation des sockets

IV.5.2.1 La classe «server »

```
public class ServerClass extends Thread {

    Socket socket;
    ServerSocket serverSocket;
    int port;

    public ServerClass(int port) {
        this.port = port;
    }

    @Override
    public void run() {
        try {


            serverSocket = new ServerSocket(port);
            Looper.prepare();
            socket = serverSocket.accept();
            Log.d(TAG, "Connection established from server");
            sendReceive = new SendReceive(socket);
            sendReceive.start();
        } catch (IOException e) {
            e.printStackTrace();
            Log.d(TAG, "ERROR: " + e);
        } catch (Exception e) {
            Log.d(TAG, "ERROR: " + e);
        }
    }

    public void closeSocket()
    {
        if (socket != null) {
            if (socket.isConnected()) {
                try
                {
                    serverSocket.close();

                }
                catch (IOException e)
                {
                    e.printStackTrace();
                    Log.d(TAG, "ERROR/n"+e);
                }
            }



        }
```

```
    }

}
```

IV.5.2.2 La classe « client »

```
public class ClientClass extends Thread {
    Socket socket;
    String hostAdd;
    int port;

    public ClientClass(InetAddress hostAddress, int port) {
        this.port = port;
        this.hostAdd = hostAddress.getHostAddress();
    }

    @Override
    public void run() {
        try {
            socket = new Socket(hostAdd,port);

            sendReceive = new SendReceive(socket);
            sendReceive.start();
            showToast("Connected to other device. You can now exchange
messages.");

            Log.d(TAG, "Client is connected to server");

        } catch (IOException e) {
            e.printStackTrace();
            Log.d(TAG, "Can't connect to server. Check the IP address
and Port number and try again: " + e);
        } catch (Exception e) {
            Log.d(TAG, "ERROR: " + e);
        }
    }

    public void closeSocket()
    {
        if (socket != null) {
            if (socket.isConnected()) {
                try
                {
                    socket.close();

                }
                catch (IOException e)
                {
                    e.printStackTrace();
                    Log.d(TAG, "ERROR/n"+e);
                }
            }
        }
```

```
    }
}
```

IV.5.2.3 La classe «SendReceive »

```
private class SendReceive extends Thread {
    private Socket socket;
    private InputStream inputStream;
    private OutputStream outputStream;

    public SendReceive(Socket skt) {
        socket = skt;
        try {
            inputStream = socket.getInputStream();
            outputStream = socket.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        byte[] buffer = new byte[1024];
        int bytes;

        while (socket != null) {
            try {
                bytes = inputStream.read(buffer);
                if (bytes > 0) {
                    handler.obtainMessage(MESSAGE_READ, bytes, -1,
buffer).sendToTarget();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    // writing
    public void write(String msg) {
        new Thread(() -> {
            try {
                //outputStream.write(msg.getBytes());
                outputStream.write(msg.getBytes());
                addMessage(Color.parseColor("#FCE4EC"), msg);
                runOnUiThread(() ->
                        messageEditText.setText("")

                );
            } catch (IOException e) {
                Log.d(TAG, "Can't send message: " + e);
            } catch (Exception e) {
                Log.d(TAG, "Error: " + e);
```

```
            }
        }).start();

    }

    public void closeSocket()
    {
        if (socket != null) {
            if (socket.isConnected()) {
                try
                {
                    socket.close();
                    inputStream.close();
                    outputStream.flush();
                    outputStream.close();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                    Log.d(TAG, "ERROR/n"+e);
                }
            }
        }


    }
}
```

IV.5.3 Cryptage

Pour crypter l'échange des message on a adopté une simple méthode qui la méthode «caesar cipher encryption» parceque la sécurité n'est pas le but de notre application.

```
public static String caesarCipherEncryption(String plainText, int shift)
{
    if( shift > 26 ) shift = shift % 26;
    else if( shift < 0 ) shift = (shift % 26) + 26;

    String cipherText = "";
    int length = plainText.length();
    for(int i = 0; i< length; i++){
        char ch = plainText.charAt(i);
        if(Character.isLetter(ch)){
            if(Character.isLowerCase(ch)){
                char c = (char)(ch + shift);
                if(c > 'z'){
                    cipherText += (char)(ch - (26-shift));
                }
                else{
```

```
                    cipherText += c;
                }
            }
            else if(Character.isUpperCase(ch)){
                char c = (char)(ch + shift);
                if(c > 'Z'){
                    cipherText += (char)(ch - (26-shift));
                }
                else{
                    cipherText += c;
                }
            }
        }
        else{
            cipherText += ch;
        }
    }

    return cipherText;
}

// decreption on the other end
public static String caesarCipherDecryption(String plainText, int shift)
{
    if( shift > 26 ) shift = shift % 26;
    else if( shift < 0 ) shift = (shift % 26) + 26;

    String cipherText = "";
    int length = plainText.length();
    for(int i = 0; i< length; i++){
        char ch = plainText.charAt(i);
        if(Character.isLetter(ch)){
            if(Character.isLowerCase(ch)){
                char c = (char)(ch - shift);
                if(c < 'a'){
                    cipherText += (char)(ch + (26-shift));
                }
                else{
                    cipherText += c;
                }
            }
            else if(Character.isUpperCase(ch)){
                char c = (char)(ch - shift);
                if(c < 'A'){
                    cipherText += (char)(ch + (26-shift));
                }
                else{
                    cipherText += c;
                }
            }
        }
        else{
            cipherText += ch;
```

```
        }
    }

    return cipherText;
}
```

IV.6 Description des interfaces de notre application

Dans le test on va utilisé deux appareils mobiles :

- Mobile 01 : Samsung A 50 Android 10.
- Mobile 02 : Huawei P Smart Android9.

Premièrement après avoir lancé l'application une demande d'autorisation de localisation et d'accès au fichier interne (Figure 21), (Figure 22), Ensuite l'interface index sera affichée donc l'utilisateur doit activer le switch wifi pour démarrer le service (Figure 23)

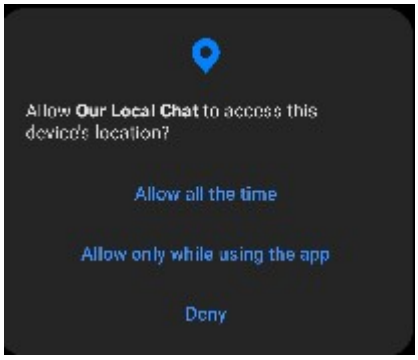

Figure 22: autorisation de localisation

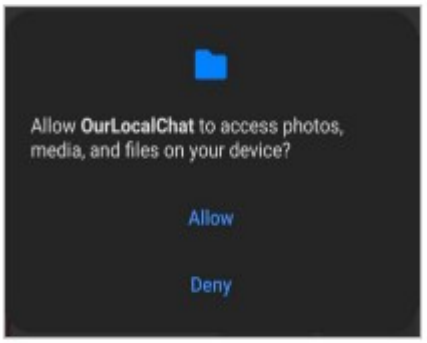

Figure 21: autorisation d'accès au fichier

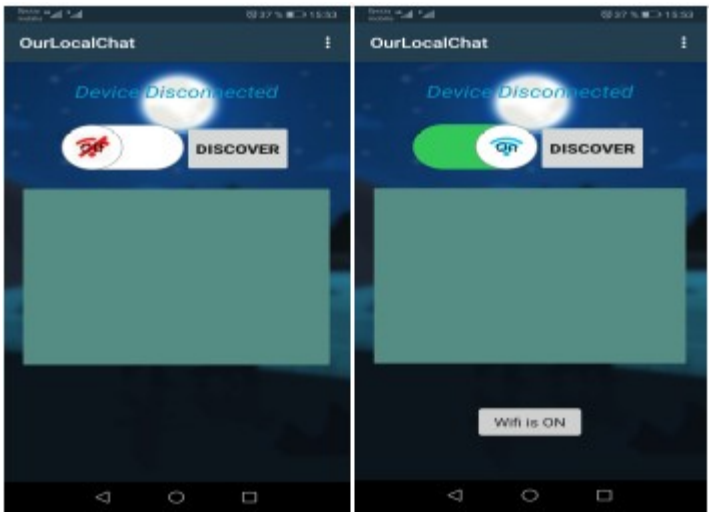

Figure 23: activation de wifi

Puis "discover boutton pour découvrir les appareil disponible On doit utiliser 2 appareil (figures 24 et 25)

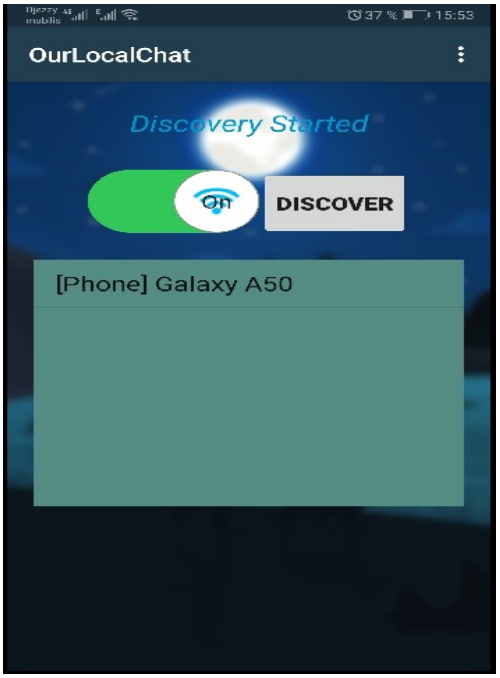

Figure 24: Mobile 02. découvrir des pairs disponibles

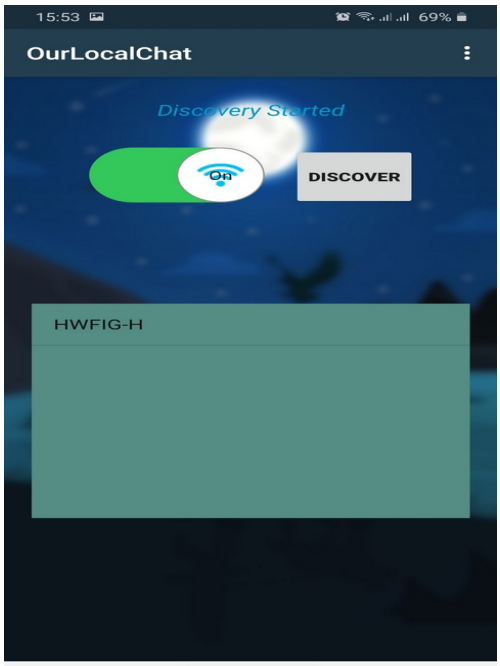

Figure 25: Mobile 01. découvrir des pairs disponibles

Le mobile 02 sélectionne le mobile 01 dans la liste des paires pour établir la conexion (figure 26)

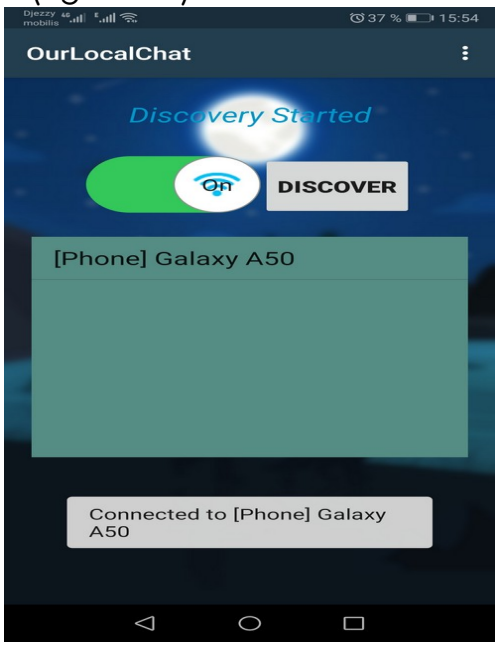

Figure 26: connexion établie

Il va afficher qui est le host (serveur) et qui est le client et on appuie sur le boutton connect pour démmarer la communication (figures 27 et 28)

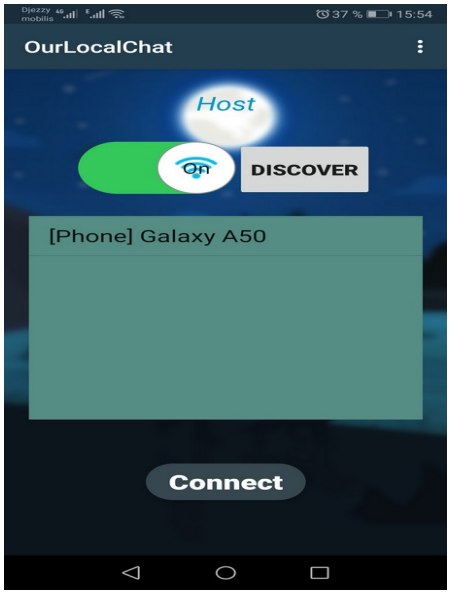

Figure 27: Mobile 02 devenu le serveur(host)

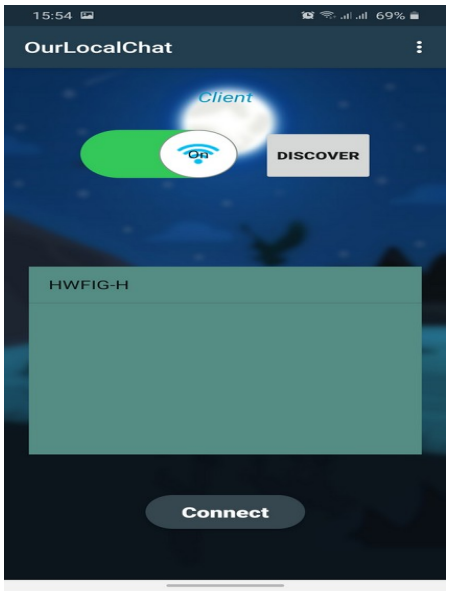

Figure 28: Mobile 01 devenu le client(client)

commencer la communication entre les deux mobiles (figures 29 et 30).

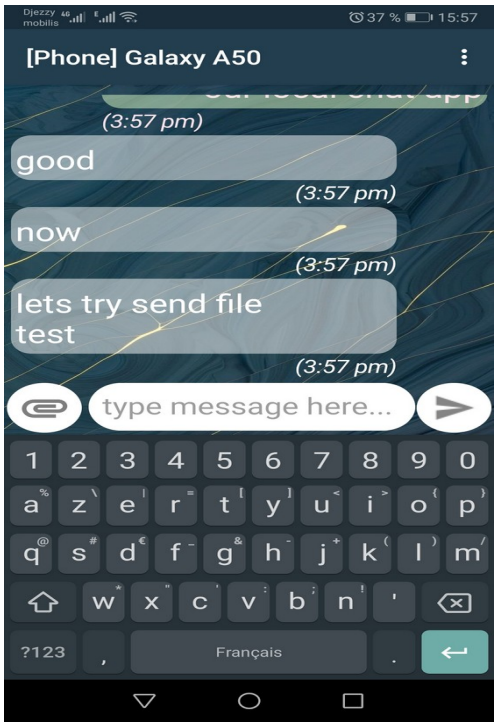

Figure 29: Mobile 02 .Le chat

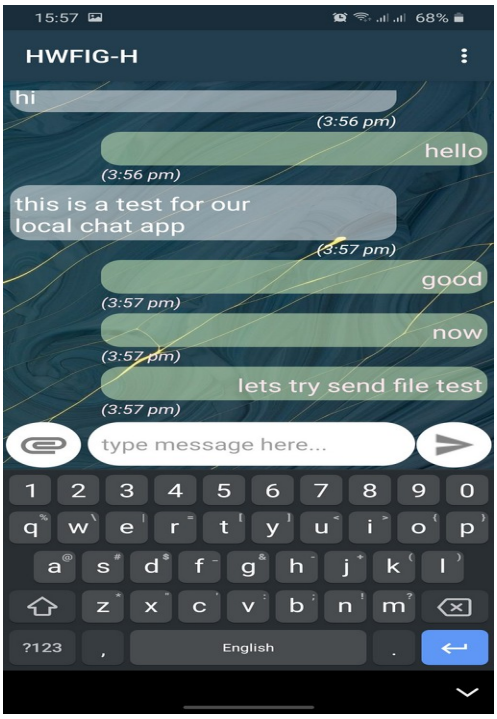

Figure 30: Mobile 01 .Le chat

Pour envoyer des fichiers on click sur l'icone de jointure puis parcourir et choisir le fichier concerné de l'expéditeur (mobile 01) vers le destinataire (mobile 02) . (figures 31, 32, 33 et 34)

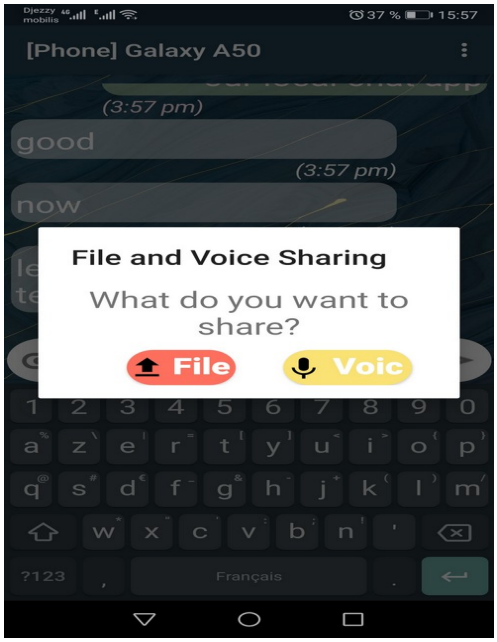

Figure 31: click le boutton file

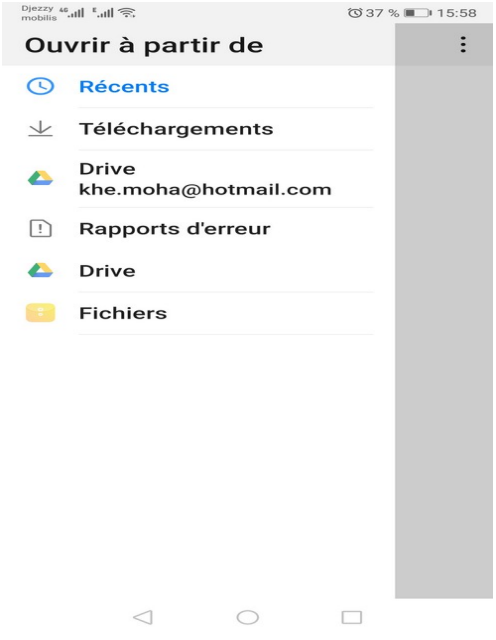

Figure 32: Parcourir les fichiers du mobile 02

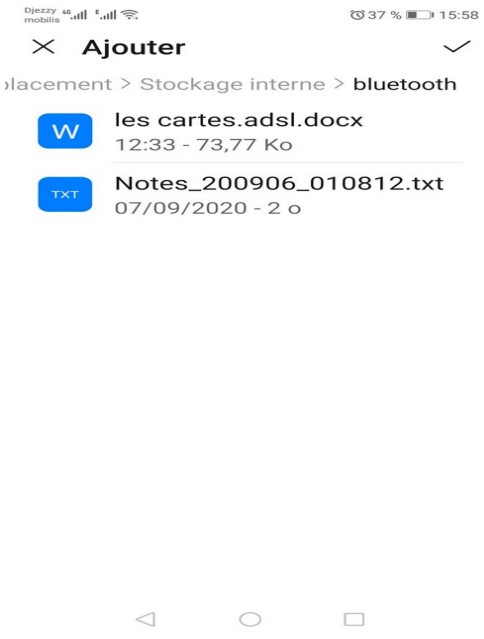

Figure 33: Choisir le fichier

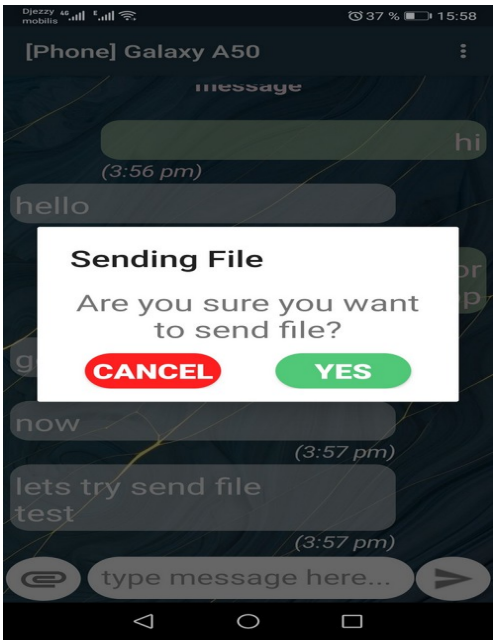

Figure 34: Confirmation de l'envoi

L'envoi et la reception du fichier de mobile 02 au mobile 01 (figures 35 et 36).

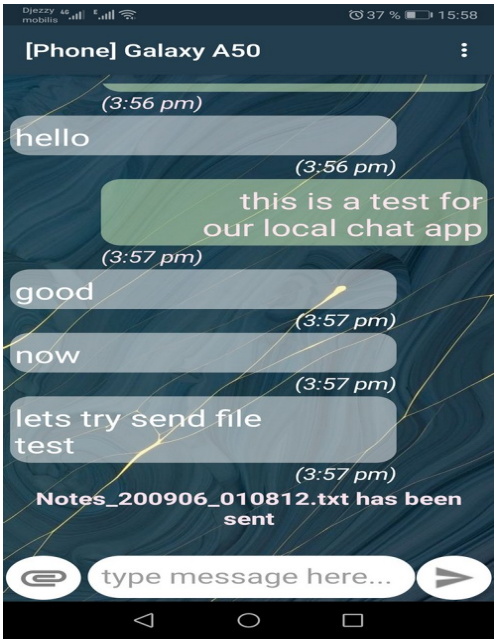

Figure 35: Mobile 02. message de l'envoi

Figure 36: Mobile 01. message de la reception

Visualisation du fichier recevé par le mobile 01 (figure 37 et 38)

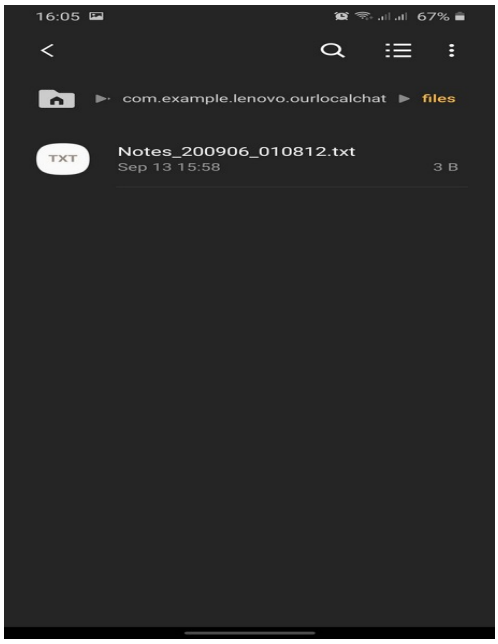

Figure 38: Mobile 01. L'emplacement du fichier recevé

Figure 37: Mobile 01. visualisation du fichier recevé

enregistrement du chat sous forme d'un fichier texte(.txt)(figure 39, 40,41, 42).

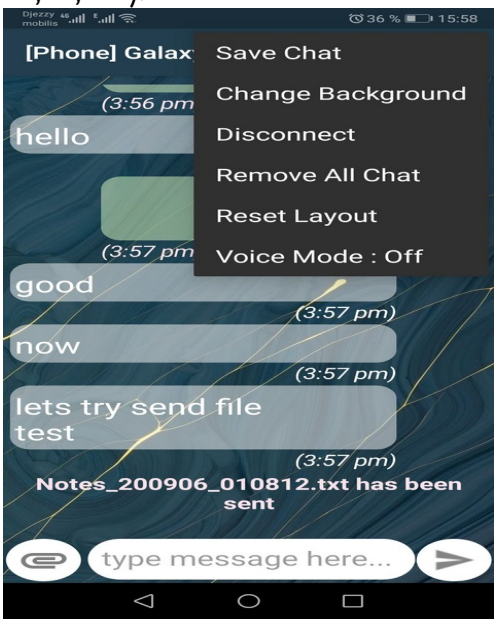

Figure 39: Choisir option « Save Chat »

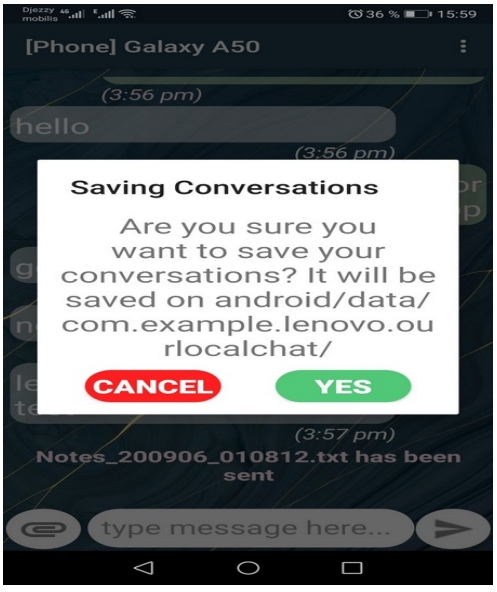

Figure 40: Confirmation de la sauvegarde

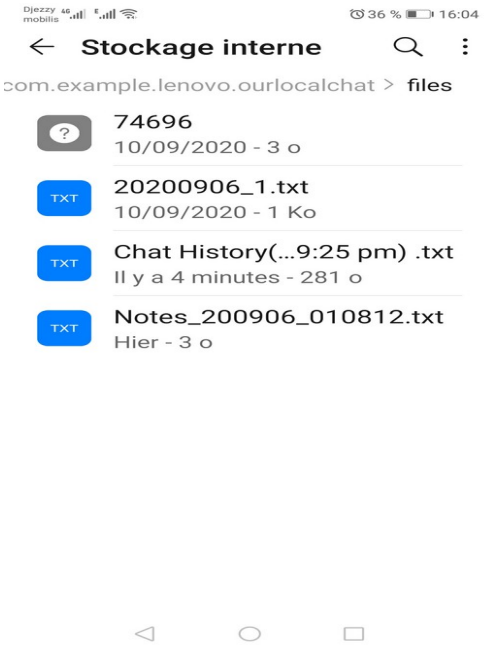

Figure 41: La création du fichier « hat history.txt »

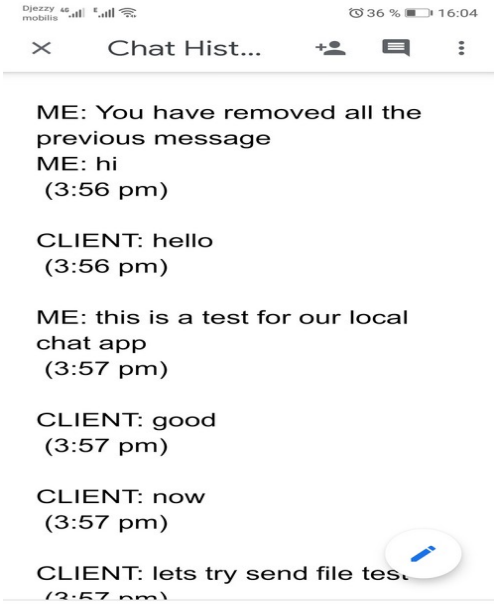

Figure 42: Visualisation du fichier créer « Chat history »

Option de changement de l'arrière plan du Chat (figures 43 et 44)

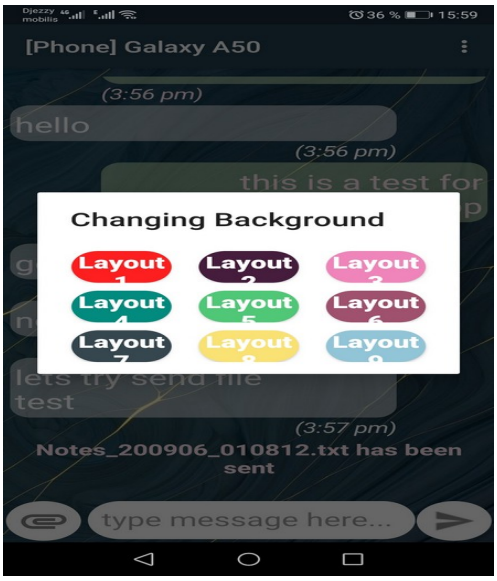

Figure 44: Option de choix

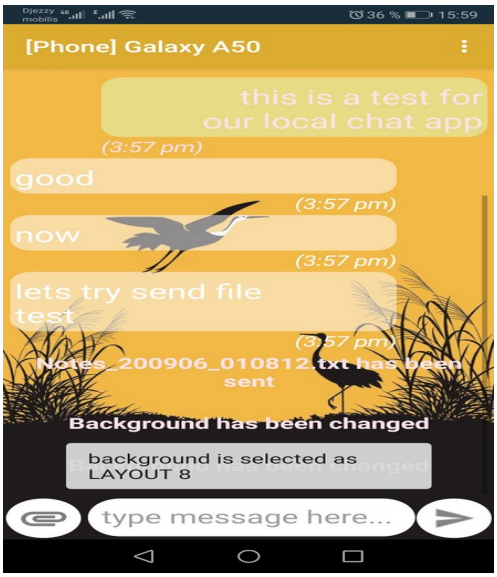

Figure 43: Changement de l'arrière plan effectué

Options sur l'effacement de l'historique du Chat (figure 45, 46, 47, 48, 49 et 50)

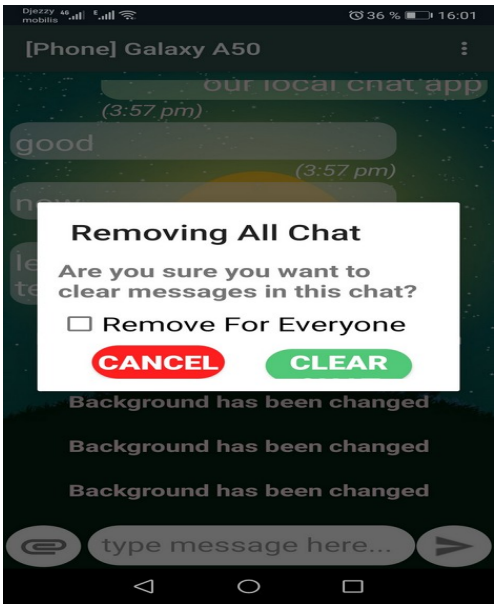

Figure 46: Mobile 02. message de suppression sans option « Remove For Everyone »

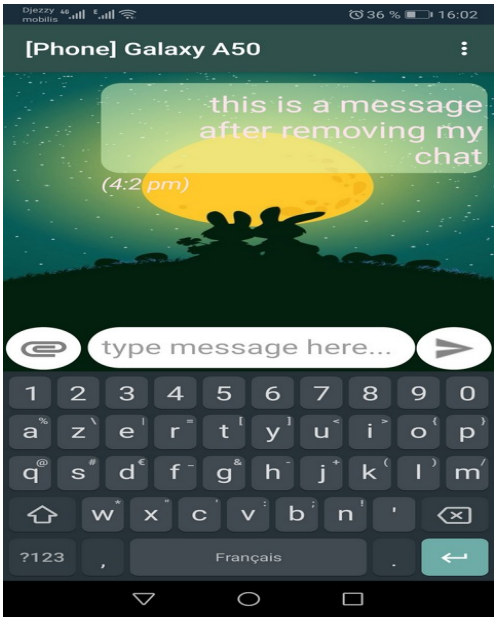

Figure 45: Mobile 02. resultat de suppression sans option

Figure 48: Mobile 01. resultat de suppression sans option

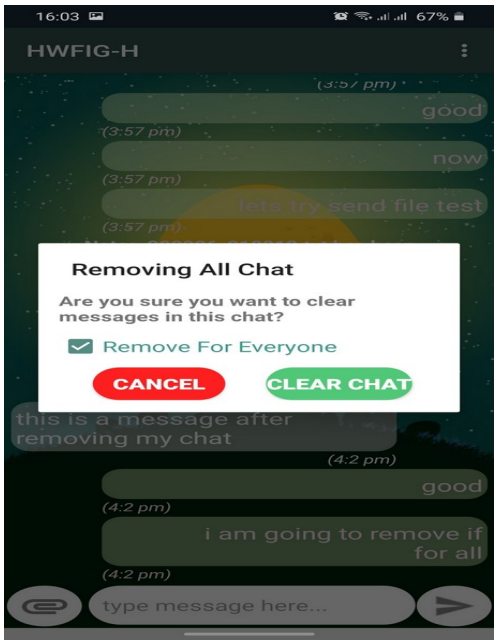

Figure 47: Mobile 01. message de suppression sans option « Remove For Everyone »

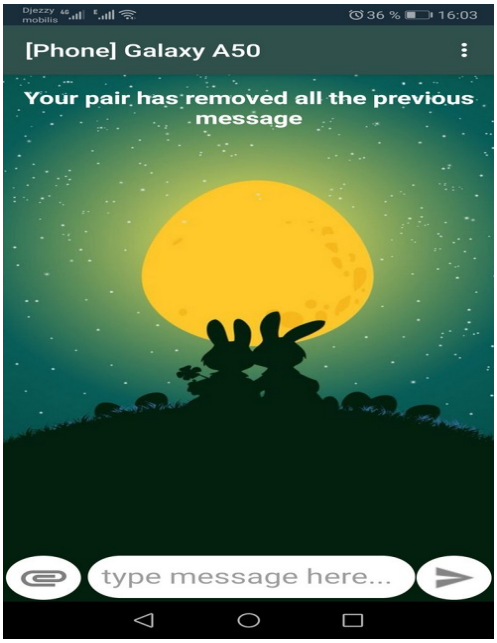

Figure 49: Mobile 02. resultat de suppression avec option "Remove For Everyone"

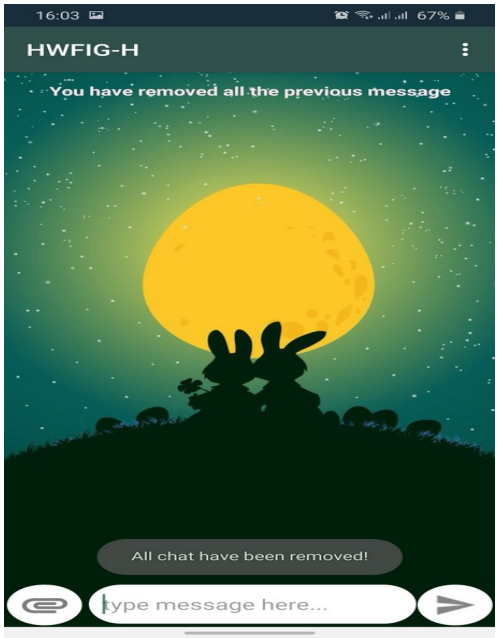

Figure 50: Mobile 01. resultat de suppression avec option "Remove For Everyone"

IV.7 Conclusion

Dans ce chapitre on a décrit la conception de notre application et comment elle a été réalisée.

Nous avons fait quelques captures d'écran de l'interface de cette application telque recherches des peers ,connexion ,chat et partage de fichier, L'application est basée sur java, XML et elle a été développé sur la plateforme Android studio.

Nous avons préparé une vidéo qui illustre notre application « our local chat » à savoir : chat et ses options et le partage de fichiers.(lien de l'application : https://youtu.be/6OUto6TpCyw)

Conclusion générale

Ce travail nous a permis d'avoir une idée plus claire et plus approfondie sur les réseaux de communication et plus précisément sur les réseaux ad-hoc , leurs protocoles de routage, et l'échange d'informations entre client-serveur à travers les sockets.

Ce type de réseau permet aux utilisateurs d'échanger des messages (chat) et partager des fichiers sur leurs propres réseaux Ad Hoc, il aide donc les participants d'une session de se connecter entre eux localement sans l'utilisation d'un réseau avec architecture ou l'utilisation d'un routeur(point d'acces).

Pour conclure, le développement d'une application android sur une platforme mobile qui est trop demandée par les utilisateurs.

Notre perspective est de travailler dans d'autres projets sur le nouveau concept de l'internet des objets qui décrit la communication entre les objets qui est la technologie d'aujourd'hui.

Bibliographie

1: G. P UJOLLE, «Les réseaux» 5 édition ÉDITIONS EYROLLES, www.editions-eyrolles.com ,
2: Chandra, D.M. Dobkin, A. Bensky, R.Olexa, D.A. Lide, F. Dowla, "Wireless Networking"; UK, Elsevier Inc, ISBN: 978-0-7506-8582-5, 2008.,
3: T.-W. Chen, M. Gerla, "Global state routing: a new routing scheme for ad-hoc wireless networks"; Proc. of the IEEE ICC, 1998.,
4: B. Tavli, W. Heinzelman, "Mobile Ad Hoc Networks: Energy-Efficient Real-Time Data Communications"; Netherlands, Springer, ISBN-13 978-1-4020-4633-9, 2006.,
5: R. Morris, J. Jannotti, F. Kaashoek, J. Li, D. Decouto; " CarNet : A scalable ad hoc wireless network system "; Proc. 9th workshop on ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System, Denmark , September 2000.,
6: I.F . Akyildiz, W. Su, Y. Sankarasubramaniam, E. l. Cayirci, "A survey on sensor networks"; IEEE Communications Magazine, Vol. 40, No. 8, pp. 102-116 , Août 2002.,
7: S.L. Wu, Y.C. Tseng, "Wireless Ad Hoc Networking: Personal-Area, Local-Area, and the Sensory-Area Networks"; USA, Auerbach publications, ISBN: 0-8493-9254-3, 2007,
8: I.F. Akyildiz, X. Wang, W. Wang, " Wireless mesh networks: a survey " ; Computer Networks, Vol.47 , pp. 445–487, 2005,
9: M. Hülsmann, K. Windt, Eds, "Understanding Autonomous Cooperation and Control in Logistics - The Impact on Management, Information and Communication and Material Flow", Springer, ISBN: 978-3-540-47449-4, 2007.,
10: RFID Journal, « That “Internet of Things” Thing - », consulté é le 2 janvier 2017,http://www.rfidjourn nal.com/articl les/view?49866,
11: ITU-T Recommendations, – Overview of Internet of things (2012) http:// handle.itu.int/11.1002/1000/11559-en?locatt=format:pdf&auth,

12: Pierre-Jean Benghozi, Sylvain Bureau, Françoise Massit-Folea, L'Internet des objets. Quels enjeux pour les Européens ? Rapport de la chaire Orange "Innovation and régulation", Ecole polytechnique et TELECOM Paris Tech. 2008. https://halshs.archives-ouvertes.fr/hal-00405070/document,
13: ISO/IEC 29161:2016, Information technology -- Data structure -- Unique identification for the Internet of Things (2016) https://www.iso.org/standard/45240.html,
14: , The Internet of Useless things – http://www.Internetofuselessthings.io/,
15: , Cours « communications en réseaux - LES SOCKETS -» module « Déploiement des Services et Interopérabilité » 1ère année master 2018 – 2019,
16: , Cour « Programmation Réseau en Java » http://www.eteks.com/coursjava/,
17: , https://fr.wikipedia.org/wiki/Application_mobile,
18: N.BENBOURAHLA, "Android les fondamentaux fe developpement d'application java",2012 .,
19: , https://developer.android.com/guide/topics/connectivity/wifip2p?fbclid=IwAR09hOvpbqd06r6GpkhV7XzpxeN6b5iPhoUkO1M5WpNdZfgvo3VTtGplYv0#api,
20 :https://www.java.com/en/download/faq/whatis_java.xml
21 :https://developer.android.com/guide/topics/connectivity/wifip2p?fbclid=IwAR09hOvpbqd06r6GpkhV7XzpxeN6b5iPhoUkO1M5WpNdZfgvo3VTtGplYv0#api