REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI TIZI-OUZOU
FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

Mémoire

De fin d'étude

En vue de l'obtention du diplôme
Master 2 en informatique

Option : réseaux, mobilité et systèmes embarqués

Thème

integration de l'application 'vanet-highway' dans le simulateur reseau NS3

Proposé et dirigé par :

M^me^ Aoudjit Rachida

réalisé par :

M^me^ Cherfaoui Nadia née Touati

Promotion : 2014

- *Je remercie mon mari qui m'a toujours poussé vers le savoir « tu es mon exemple »*
- *Je remercie ma promotrice pour sa modestie, sa gentillesse et son soutien. Bien que notre insertion dans le système LMD fût tardive, madame Aoudjit nous a soutenu, rassuré et encouragé. « Merci madame ».*
- *Je remercie les membres du jury.*
- *Je remercie, de ma part et de la part de beaucoup d'autres étudiants en informatique, les premiers qui ont lancé la culture open source sans laquelle l'informatique n'aurait pas encore fait un pas. Aussi je remercie les sites pirates (logiciels informatiques, documentation gratuite, vidéo de formation en programmation).*

Liste des figures :

Liste des tableaux :

Introduction générale:

Vehicular ad-hoc network (VANET) est une sous-classe des réseaux Manet. Dans les VANET, la communication se fait d'un véhicule à un autre ou d'un véhicule à une infrastructure.

Les VANET ont leurs propres caractéristiques à savoir une topologie fortement dynamique, des déconnections très fréquente, une forte mobilité, espace de mouvement fixé par l'architecture des routes, etc.

L'étude des réseaux VANET (comparer des algorithmes de routages, modifier les algorithmes existant …) requière des outils de simulation puissants et efficaces.

NS3 propose plusieurs modèles de mobilité, mais aucun n'est approprié pour la mobilité dans les VANET. La mobilité d'un nœud dans NS3 dépend uniquement du nœud lui-même alors que dans la réalité, elle dépend de tout son environnement (des conditions de la route, des messages reçus par les nœuds voisins, des feux rouges, des voitures de police qui contrôle le trafic, de la connexion wifi …).

C'est pour cela que la simulation des VANET est séparée en deux. La simulation du trafic et la simulation du réseau. Des simulateurs du trafic routier comme CORSIM (Halati et al. 1997), SUMO (Krajzewicz et al. 2006), VISSIM (PTV America 2010), et VanetMobiSim (Fiore et al. 2006) ont été utilisé pour générer des traces du mouvement des véhicules qui seront injectés ensuite dans un simulateur réseau comme NS2 (Breslau et al. 2000), ns-3 (Henderson et al. 2006), QualNet (Scalable Network Technologies 2010), OPNET (2010) ou GloMoSim (Zeng et al. 1998) pour mesurer les performances du réseau .

Pour faciliter l'interaction entre les simulateurs du trafic et les simulateurs réseau, des outils comme TraNS (Piorkows-ki et al. 2008) et MOVE (Karnadi et al. 2007) on été utilisés.

Pour la simulation d'un réseau VANET la première méthode propose d'utiliser un simulateur de trafic tel que SUMO. Cette méthode est relativement simple puisque SUMO dispose d'une interface graphique qui permet de décrire la carte routière et le trafic routier sans se perdre dans des lignes de programme, puis injecter le fichier résultant dans NS2 et enfin dans NS3. Mais cette méthode ne nous permet pas d'intervenir en temps réel sur la simulation du trafic.

Dans la réalité, un message de prudence dans le réseau provoque un changement de comportement des conducteurs. Si l'utilisateur VANET veut provoquer un événement pendant la simulation il doit refaire complètement la simulation de trafic. En général, l'utilisateur du réseau dans la première méthode néglige soit la simulation du trafic soit la simulation du réseau. Sans oublier que cette méthode est très longue et requière la connaissance d'au moins deux simulateurs.

La deuxième méthode est celle qui nous intéresse, elle est toujours en mode expérimental. Cependant elle est très intéressante puisque elle ne fait pas appel à d'autres simulateurs pour générer le trafic routier. Elle permet à l'utilisateur du réseau d'intervenir en temps réel pendant la simulation et le résultat est un réseau réaliste.

L'objet de notre travail consiste en l'intégration d'une application spéciale vannet 'vanet-highway' dans NS3 et la simulation d'un réseau VANET en utilisant cette application.

Notre mémoire est organisé en quatre chapitres :

Dans **Le premier chapitre** nous donnerons un état de l'art des réseaux sans fil et les différents concepts liés à ce type de réseaux. Nous aborderons aussi les réseaux mobiles, les réseaux mobiles Ad Hoc (MANET), les réseaux Ad Hoc véhiculaires (VANET) et leurs caractéristiques.

Dans **Le deuxième chapitre** nous traiterons le routage dans les réseaux mobiles Ad hoc. Nous nous intéresserons à la problématique du routage et les contraintes liées aux réseaux Ad Hoc. Nous décrierons également les principaux protocoles et leurs classifications.

Le troisième chapitre est consacré à l'étude de l'environnement de simulation, NS2 et NS3.

Le quatrième chapitre est consacré à la réalisation et à la simulation qui consiste en l'intégration de l'application 'vanet-highway' dans NS3 et la simulation d'un réseau VANET avec 'vanet-highway'.

I. Les réseaux sans fil:

I.1. Introduction :

La technologie de la communication sans fil a évolué d'une manière spectaculaire. Dans notre quotidien, cette technologie est devenue nécessaire et même vitale (dans le domaine du secourisme par exemple). La civilisation de l'homme moderne est basée sur cette technologie sans fil et on ne peut même pas imaginer vivre sans elle (imaginer la vie sans le téléphone portable). La mobilité dans les réseaux informatique ouvre des portes nouvelles pour l'homme et la société et ses applications ont touchées tous les domaines.

Un tel succès est dû principalement à la vulgarisation des équipements mobiles offrant ainsi plus de souplesse, plus de rapidité et moins de frais.

Les réseaux mobiles sans fil peuvent être classés en deux catégories : les réseaux avec infrastructure qui nécessitent généralement l'installation des stations de base et les réseaux sans infrastructure ou Ad Hoc caractérisés par leur facilité d'installation. Les réseaux sans fil sont utilisés dans plusieurs applications à savoir la téléphonie, les applications militaires, les applications commerciales, les applications embarquées et la sécurité routière.

Dans ce chapitre nous allons présenter les réseaux sans fil, leurs catégories, leurs caractéristiques ainsi que leurs contraintes. Les réseaux mobiles, leurs classifications, leurs caractéristiques et les problèmes liés à la mobilité seront également décrits. Nous aborderons aussi les réseaux mobiles Ad Hoc (MANET) et nous en citerons quelques applications pour enfin parler des réseaux qui nous intéressent, les réseaux de véhicule (VANET).

I.2. Définition : R[6]

Un réseau sans fil (Wireless network) est un réseau dans lequel les machines participantes peuvent communiquer sans liaison filaire.

Les réseaux sans fil sont basés sur des liaisons utilisant des ondes radio (ou infrarouge) à la place des câbles habituels (coaxial, paire-torsadée ou fibre optique).

Dans ce type de réseau, les utilisateurs ont la possibilité de se déplacer dans un certain périmètre de couverture géographique sans perdre le signal.

I.3 Techniques de communication sans fil :R[5]

Dans tout système de transmission de données, le support de transmission est défini comme le chemin physique entre l'émetteur et le récepteur. Ce support, aussi appelé média ou medium peut être guidé (câble en cuivre, fibre optique,…) ou non guidé (deux médias peuvent être utilisés : les liaisons infrarouges et les ondes radios).

- **Liaisons infrarouges**

Les liaisons infrarouges sont utilisées dans les communications courtes, elles sont simples et peu coûteuses. Elles conviennent aux réseaux à faible portée. Les émetteurs et récepteurs à infrarouge sont capables de fournir des débits élevés à des coûts relativement faibles. Les bandes passantes disponibles sont très larges, les liaisons infrarouges pénètrent à travers le verre, mais pas à travers les murs ou tout autre obstacle opaque, ce qui implique que les communications se font dans la même pièce, ce qui augmente la sécurité.

- **Liaisons radios**

Le principe est d'émettre des ondes électromagnétiques qui constituent la porteuse du signal à transmettre. Ces ondes sont donc propagées dans toutes les directions et peuvent être captées par plusieurs antennes. Le medium radio est découpé en bandes de fréquences divisées en canaux. Il est caractérisé par sa moindre fiabilité par rapport au medium filaire, c'est un medium partagé, c'est-à-dire si deux nœuds géographiquement proches tentent d'émettre une trame simultanément, il y a interférence entre les deux émissions, ces interférences entrainent des collisions entre les trames, ce qui oblige les émetteurs à émettre une autre fois. Ce problème de collisions est résolu grâce à des techniques d'accès au medium utilisées dans les réseaux Ad Hoc :

- TDMA : (Time Division Multiple Access): Solution à accès multiple à répartition dans le temps.

- CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) : Technique permettant d'éviter les collisions entre les utilisateurs d'un réseau, en s'assurant que chacun parle à son tour et pas tous en même temps.

I.4. Catégories des réseaux sans fil :R [4] R[6]

Nous trouvons généralement quatre catégories de réseaux sans fil suivant la zone de couverture et la distance entre les nœuds.

- **WPAN Wireless Personal Area Network** (réseau personnel sans fil)

Réseau domestique d'une faible portée (quelques dizaines de mètres). L'objectif principal de ce type de réseau est de relier des périphériques à une unité centrale câblée ou de connecter deux ou trois machines très peu éloignées. Les technologies utilisées sont : Bluetooth, ZigBee, HomeRF et IrDA.

- **WLAN Wireless Local Area Network** (réseau local sans fil)

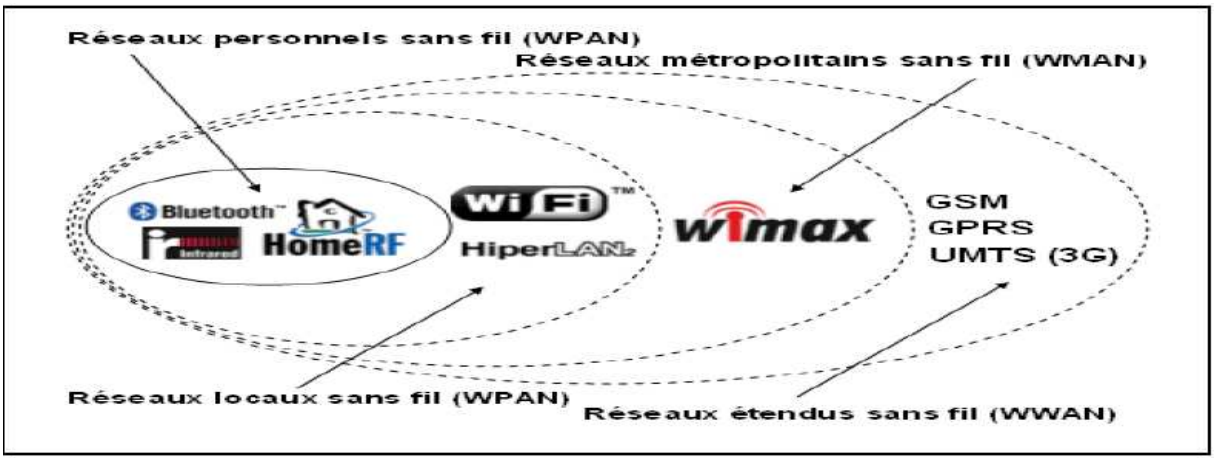
Cette catégorie comprend les réseaux sans fil offrant une zone de couverture correspondant à un réseau local d'entreprise, soit quelques centaines de mètres. Le but de ce réseau est d'interconnecter différentes machines qui sont situées dans un périmètre géographiquement restreint, semblable aux réseaux fixes de type LAN (Local Area Network). Les technologies utilisées sont : Wifi, HiperLan1 et HiperLan2.

- **WMAN Wireless Metropolitan Area Network** (réseau métropolitain sans fil)

Réseau basé sur la norme IEEE 802.16 : sont inclus dans cette catégorie les réseaux offrant une couverture comparable à un campus ou un quartier d'une ville. Le but de ce réseau est d'interconnecter divers réseaux fixes ou réseaux sans fil se trouvant sur un même campus ou dans un même quartier. La technologie utilisée est : Wimax .

- **WWAN Wireless Wide Area Network** (réseau étendu sans fil)

Connu également sous le nom de réseau cellulaire mobile, cette famille regroupe les réseaux étendus sur une zone de couverture de plusieurs kilomètres. Les technologies principales utilisées sont :
GSM (Global **S**ystem for **M**obile communication.), GPRS(**G**enera**l** **P**acket **R**adio **S**ervice.) et UMTS : **U**niversel **M**obile **T**élécommunication **S**ystem.



*-**La figure ch1.1** : la classification des réseaux sans fil.-*

Le tableau suivant résume les différentes technologies utilisées dans chaque catégorie et leurs caractéristiques.

Catégorie	**WPAN**	**WLAN**	**WMAN**	**WWAN**
Standard	IEEE 802.15	IEEE 802.11	IEEE 802.16	IEEE 802.20
Technologie	Bluetooth HomeRF Zigbee IR(infrarouge)	Wifi Hyperlan1 Hyperlan2	Wimax	GSM GPRS UMTS
Couverture	Quelque dizaines de mètres	Une centaine de mètres	Quelques dizaines de kilomètres	Une centaine de kilomètres
Débit	< 1Mbps	2 à 54 Mbps	Jusqu'à 70 Mbps	10 à 385 Kbps
Applications	Point à point Equipement à équipement	Réseau d'entreprise	Fixe, accès au dernier kilomètre	GSM PDA

*-**Tableauch1.1** : Classification des réseaux sans fil.-*

I.5. Avantages et contraintes de la communication sans fil[4][6]

Nous citerons dans la section suivante les avantages liés à l'utilisation du médium radio dans les réseaux sans fil. Les contraintes liées à cet environnement seront également présentés.

Avantage :

- **La mobilité**

La mise en place d'un réseau sans fil entre les éléments portables permet d'éviter les câbles de connexion au réseau et les câbles d'alimentation, afin de permettre le mouvement libre des utilisateurs avec leurs terminaux portables.

- **Faibles coûts**

Contrairement au réseau filaire où le câblage représente un coût supplémentaire, le réseau sans fil s'affranchit de ce coût. Néanmoins, les protocoles de routage et de configuration doivent être repensés pour permettre la bonne gestion et l'acheminement des données dans le réseau.

Contraintes:

- **Dégradation de la qualité du signal**

Cette contrainte est causée par l'affaiblissement de la puissance du signal avec la distance et les conditions atmosphériques. De plus, le bruit dû à d'autres signaux parasites cause une altération du signal. D'autres paramètres tels que l'absorption atmosphérique du signal par la vapeur d'eau et l'oxygène, la propagation multi trajet causée par les obstacles entre l'émetteur et le récepteur, font que le signal se dégrade davantage.

- **Sécurité**

La confidentialité des données circulantes sur les réseaux sans fil doit être assurée, car les transmissions radioélectriques sont sensibles aux interférences et sujettes à l'écoute par un utilisateur mal intentionné. Cet utilisateur peut se placer dans le périmètre des équipements du réseau afin de récupérer les informations qui lui permettront d'avoir accès au réseau. Ceci représente le plus grand problème des réseaux sans fil.

- **Débit**

Le simple fait d'avoir un trop grand nombre d'utilisateurs dans un réseau sans fil peut entraîner une diminution importante de débit. Cette diminution de débit peut même conduire à une perte de connectivité ce qui est très contraignant.

II. Les réseaux mobiles : R [1][2][3][6]

II.1. Définition de la mobilité :

Le terme mobilité est la capacité ou la facilité d'un objet ou d'une personne à se déplacer par rapport à un lieu, à une position ou à un ensemble d'objets de même nature.

Dans le domaine de réseaux, la mobilité se traduit par la possibilité que certaines entités peuvent passer d'une cellule à une autre sans perdre la liaison.

II.2. Définition d'un réseau mobile :

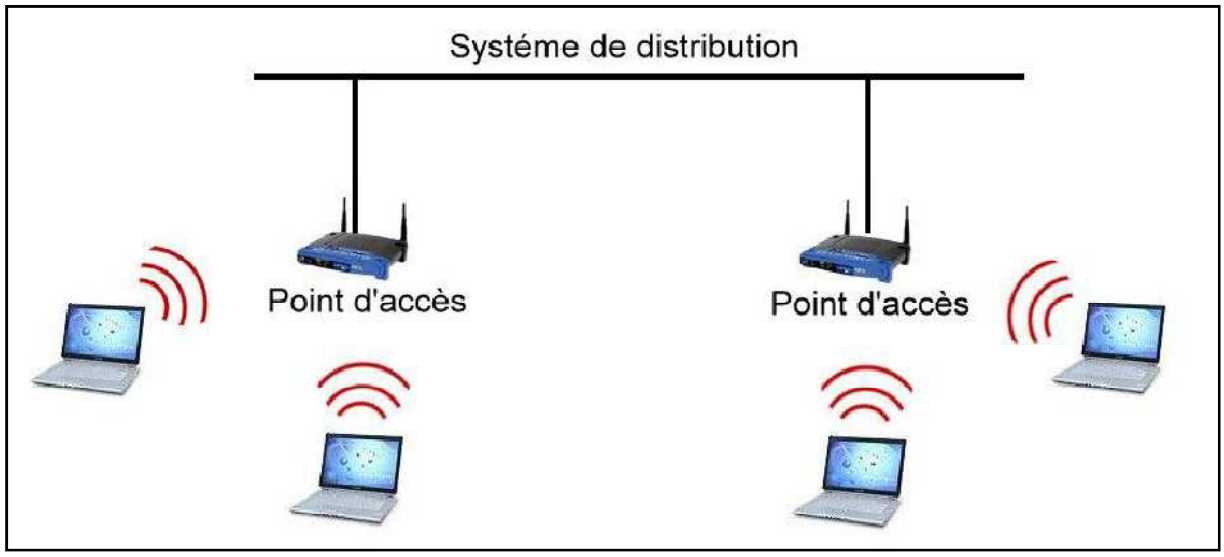
Un réseau mobile est un système composé de nœuds reliés les uns aux autres par des liaisons de communication sans fil. Ces nœuds sont libres de se déplacer sans perte de leurs connexions au réseau. Un réseau mobile peut contenir des sites fixes pour permettre l'accès à d'autres types de réseaux (filaires).

II.3. Classification des réseaux mobiles selon l'infrastructure :

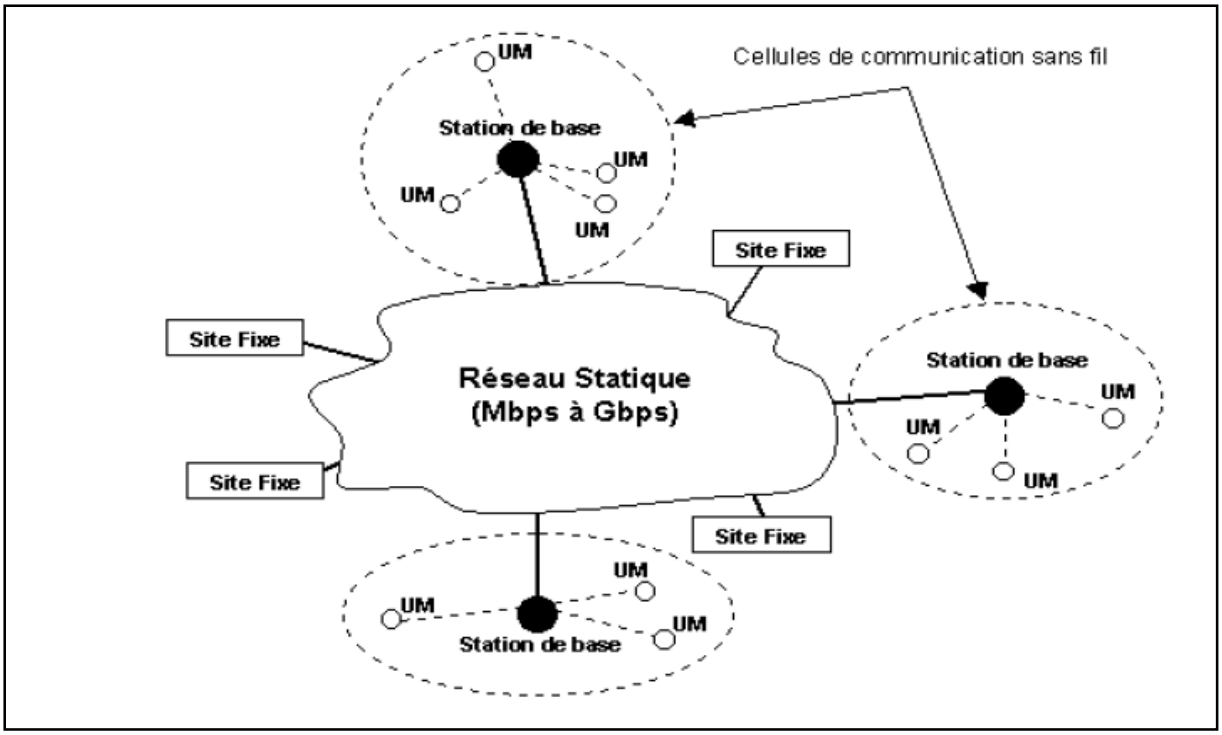
Les réseaux mobiles peuvent être divisés en deux classes à savoir : les réseaux mobiles basés sur une infrastructure et les réseaux mobiles sans infrastructure.

a. Les réseaux mobiles avec infrastructure (cellulaires)

Dans ce mode, chaque nœud (appelé aussi unité mobile) se connecte à un point d'accès (appelé aussi site fixe ou station de base SB) via une liaison sans fil. L'ensemble formé par le point d'accès et les unités mobiles situées dans sa zone de couverture est appelé ensemble de services de base (en anglais *Basic Service Set*, noté BSS) et constitue une cellule. La figure ci-dessous illustre un exemple d'un réseau mobile avec infrastructure.



- ***Figure ch1.2****: Mode avec infrastructure-*

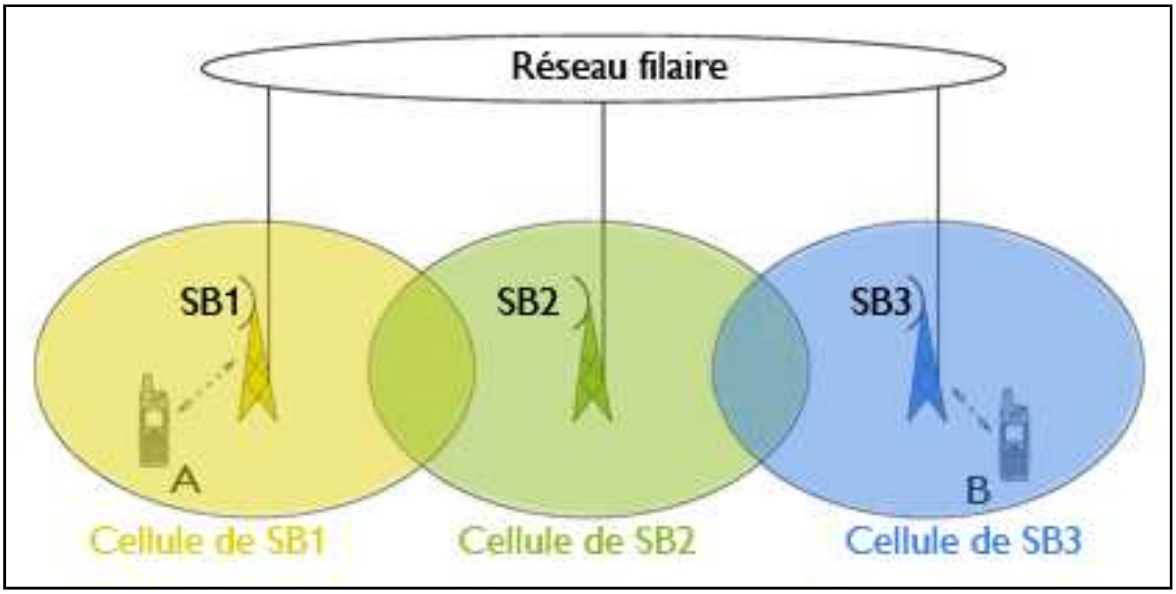


*-**Figure ch1.3** : **exemple 2** : modèle de réseaux mobile avec infrastructure-*

Il est possible de relier plusieurs points d'accès entre eux par une liaison appelée système de distribution (notée DS pour *Distribution System*) afin de constituer un ensemble de services étendus (*Extended Service Set* ou ESS). Le système de distribution peut être un réseau filaire, généralement fiable et à haut débit, ou bien un réseau sans fil.

- **Réseaux cellulaires et stations de base**

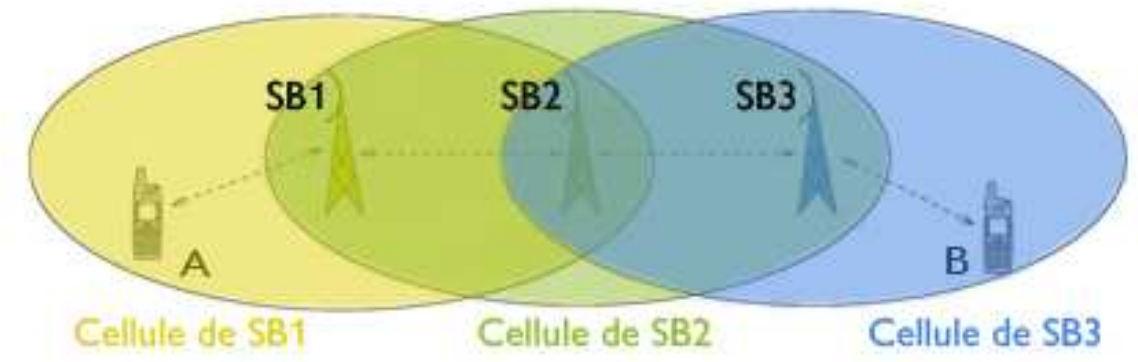
Les nœuds mobiles se connectent à des stations de bases « points d'accès » qui gèrent l'ensemble des communications dans une même zone géographique (cellule). Ce mode peut permettre un accès à un réseau filaire. Les unités mobiles peuvent se déplacer au sein de la cellule et garder une liaison directe avec le point d'accès. La puissance des antennes équipant ces points d'accès étant plus importants que celle des nœuds mobiles, le réseau peut couvrir une zone très importante.



*-**Figure ch1.4:** Réseau à stations de base.-*

- **Réseaux radio maillés : « Suppression définitive des fils »**

Pour utiliser un réseau sans fil à l'échelle d'une ville à moindre coût, on peut utiliser un réseau radio maillé, appelé aussi « Mesh Network ». Ce dernier consiste en un ensemble de stations de base « SB » qui couvrent une zone visée. La différence avec les réseaux à stations de bases est que dans ce type de réseau, les stations de base communiquent entre elles par liaison radio. Ainsi, les mobiles peuvent communiquer entre eux à travers ces stations de base. Dans un réseau radio maillé, chaque mobile est rattaché à la station de base la plus proche, avec laquelle il communique exclusivement.
La figure ci-dessous illustre un exemple de communication entre des nœuds d'un tel réseau. Lorsque le mobile 'A' souhaite communiquer avec le mobile 'B', le mobile 'A' transmet son message à la station de base 'SB1' qui le transmet via son interface radio à la station 'SB2', qui à son tour retransmet le message à la station 'SB3', qui finalement envoie ce dernier au destinataire 'B'.



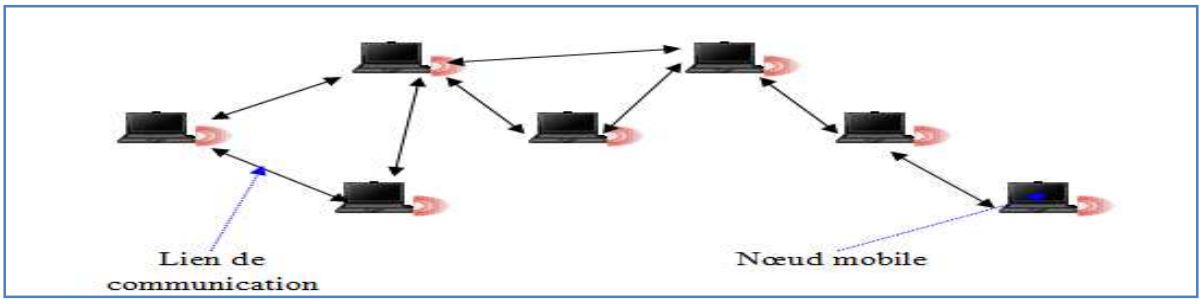
*-**Figure ch1.5:** Réseau radio maillé.-*

Si deux mobiles souhaitent communiquer, il faut que les stations de base auxquelles ils sont rattachés puissent communiquer. Idéalement, chaque mobile peut communiquer avec

n'importe quel autre mobile du réseau radio maillé. Ceci est réalisé, si et seulement si, le graphe des stations de base du réseau est connexe, c'est-à-dire qu'il existe toujours une route utilisant éventuellement des stations de base intermédiaires, reliant deux stations de base quelconques du réseau.

b. Réseau mobile sans infrastructure

Dans le mode sans infrastructure, la notion de site fixe ou point d'accès n'existe pas. Toutes les stations du réseau se connectent les unes aux autres afin de construire un réseau point à point (P2P pour *peer to peer*). Ainsi, chaque machine joue en même temps le rôle de client et le rôle de point d'accès.
Dans un environnement ad hoc, lorsqu'une donnée est envoyée d'un nœud source à un nœud destination, il se peut qu'elle transite par plusieurs nœuds intermédiaires avant d'arriver à destination. C'est ce qui s'appelle le *multi hop* (saut multiple ou multi-saut).



*- **Figure ch1.6**: réseau sans infrastructure (ad hoc)-*

II.4. Problèmes liés à la mobilité :

- **Nommage et adressage des nœuds mobiles**

Dans les réseaux fixes, l'adresse d'un nœud est souvent confondue avec son nom. Par contre, nous ne pouvons pas nommer un nœud mobile par son adresse car il change de cellules et donc utilise plusieurs points d'accès, d'où le changement fréquent d'adresses.

- **Routage**

La mobilité des nœuds et le changement fréquent de la topologie nécessitent des protocoles de routage qui doivent réagir rapidement à ces changements, afin de permettre le bon acheminement des paquets à leurs destinations.

- **Récupération des adresses**

La déconnexion (accidentelle ou volontaire) d'un nœud mobile nécessite l'adaptation des protocoles de récupération des données.
Il est aussi important de supprimer les données résiduelles stockées pour éviter la saturation de l'espace mémoire.

- **Diffusion de données**

Dans le cas d'une émission multidestinataires (diffusion sélective) d'un message, ce dernier peut être reçu par des stations différentes à des moments différents. Donc, il est possible qu'un message diffusé ne soit pas reçu par une unité mobile destinataire, car son entrée en cellule a eu lieu après la diffusion du message ou sa sortie a eu lieu avant la diffusion du message. Une unité mobile peut recevoir plusieurs copies du même message à travers des stations de base différentes.

- **Dépendance de la localisation**

Dans les réseaux mobiles à stations de base, un nœud mobile peut avoir besoin des services des réseaux filaires (Internet par exemple), il est donc préférable de faire appel à la station de base la plus proche de sa localisation.

III. Les réseaux Ad hoc :

III.1. Définition 1 : R[6]

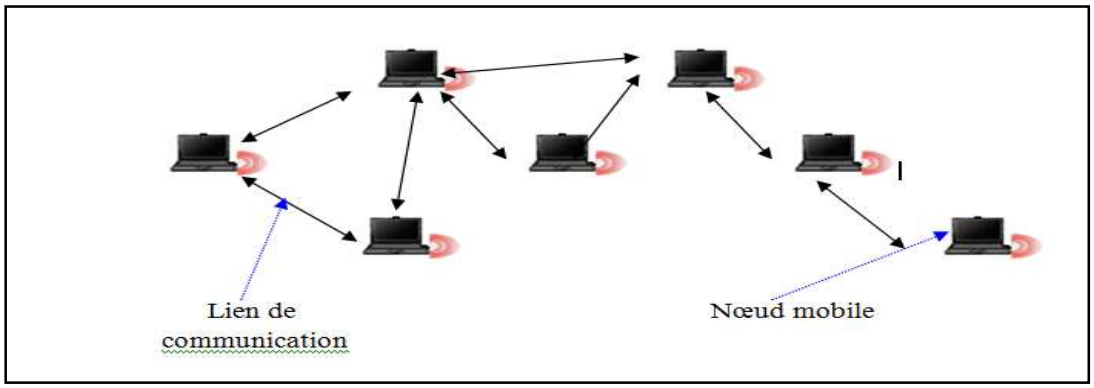
Les réseaux ad hoc (en latin : « qui va vers ce vers quoi il doit aller », c'est-à-dire « formé dans un but précis »), sont des réseaux sans fil spontanés créés à la demande pour répondre à un besoin spécifique, capables de s'organiser sans infrastructure définie préalablement.

III.2. Définition 2 [21]

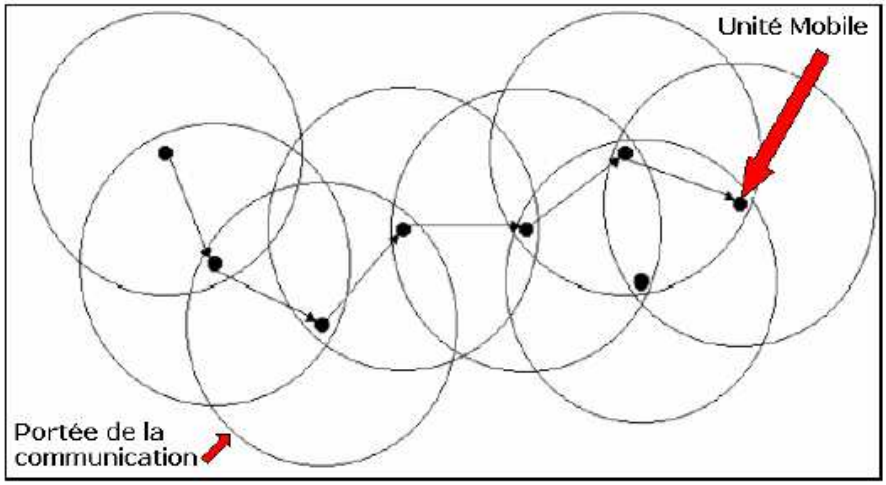
Un réseau mobile ad hoc, appelé généralement MANET (Mobile Ad hoc NETwork), est un réseau sans fil qui consiste en une grande population, relativement dense, d'unités mobiles qui se déplacent dans un territoire quelconque et dont le seul moyen de communication est l'utilisation des interfaces sans fil généralement le medium radio, sans l'aide d'une infrastructure préexistante ou administration centralisée. Ces unités mobiles jouent à la fois le rôle de terminaux et de routeurs pour permettre le passage de l'information entre elles.

Il permet donc à deux nœuds qui sont chacun à portée des ondes l'un de l'autre (condition appropriée de propagation d'ondes radio) de rentrer en communication directement.

Un réseau ad hoc doit être facilement déployé, les nœuds peuvent joindre ou quitter le réseau de manière totalement dynamique sans informer le réseau, et si possible sans effet de bord sur les communications des autres membres.



*-**figure ch1.7** : exemple d'un réseau ad hoc-*



*- **figure ch1.8** : représentation des portés de la communication des nœuds dans un réseau ad hoc-*

III.3. Historique et évolution des réseaux Ad Hoc :R[7]

Historiquement, les réseaux mobiles ad hoc ont été principalement utilisés afin d'améliorer les communications dans le domaine militaire. L'armée visait alors le déploiement rapide de systèmes de communication dans des zones difficiles, telles que champs de bataille ou lieux de catastrophes naturelles (mise en place d'un hôpital de campagne, par exemple). Les premières recherches sur les réseaux Ad hoc ont démarré dans les années 60 au sein d'un organisme de l'armée américaine **DARPA** (**D**efence **A**dvanced **R**esearch **P**roject **A**gency).

Les premières applications dans les réseaux ad hoc sont apparues avec le projet PRNet (Packet Radio Network) de DARPA en 1972, et avaient pour objectif de trouver une architecture de réseau radio facile à déployer pour l'échange de données tactiques.

Différents algorithmes ont été développés dans le cadre de ce projet, mais ils étaient basés sur des réseaux ne comportant que quelques nœuds.

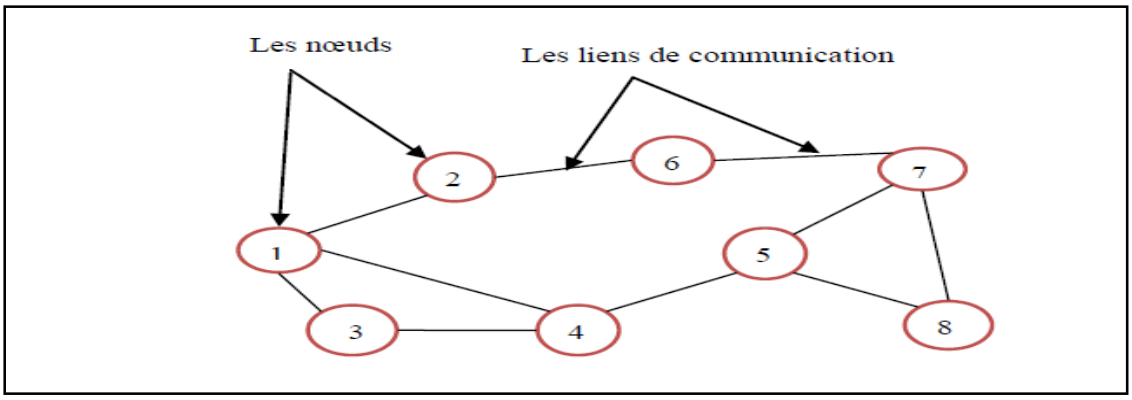
Pour pallier les principaux problèmes du projet PRNet, le projet SURAN (Survivable Radio Networks) a été lancé en 1983, son principal objectif était d'étendre les algorithmes développés par PRNet pour des réseaux plus larges.

Un autre projet IT (Tactical Internet) a été développé en 1997, qui est l'une des implémentations des réseaux mobiles ad hoc multi sauts.
En 1999 ELB ACTD (Extending the Littoral Battle-space Advanced Concept Technology Demonstration) a été développé et dont l'objectif est de démontrer la faisabilité de concepts militaires pour les communications des bateaux en mer aux soldats sur la terre par l'intermédiaire d'un relais aérien.
Aujourd'hui les recherches continuent avec encore plus de ferveur et d'enthousiasme suscitant beaucoup d'intérêt dans le domaine des réseaux sans fil.

III.4. Modélisation des réseaux mobiles Ad Hoc :

Un réseau mobile Ad Hoc peut être modélisé par un graphe Gt = (Vt, Et) où :
Vt : représente l'ensemble des nœuds (les unités ou les hôtes mobiles) du réseau.
Et : modélise l'ensemble des connexions entre ces nœuds.
Si e = (u, v) appartient à Et, cela veut dire que les nœuds u et v peuvent communiquer directement à l'instant t.

Le réseau mobile Ad Hoc est modélisé par un graphe dont les nœuds représentent les stations mobiles du réseau et une connexion entre deux nœuds qui signifie qu'à cet instant, les deux nœuds peuvent communiquer directement (sans passer par d'autres nœuds pour acheminer le paquet transmis).



***-Figure ch1.9** : Modélisation d'un réseau Ad Hoc-*

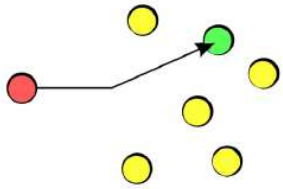
III.5. Modes de communication dans les réseaux mobile Ad Hoc :R [8]

Les échanges de données dans les réseaux mobiles utilisent les modes de communication suivants :

- **Le mode Unicast**

Le terme unicast définit une connexion réseau point à point. On entend par unicast le fait de communiquer entre deux ordinateurs identifiés chacun par une adresse réseau unique.
Les paquets de données sont routés sur le réseau suivant l'adresse du destinataire ; seul le destinataire intercepte et décode le paquet qui lui est adressé.

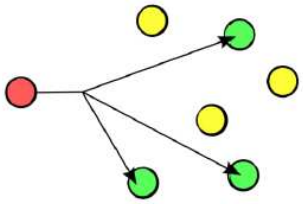
-Le routage Unicast :



- **Le mode Multicast (multipoint)**

On entend par multicast, le fait de communiquer simultanément avec un groupe d'ordinateurs identifié par une adresse spécifique (adresse de groupe). Son avantage par rapport au mode classique unicast devient évident quand on veut diffuser de la vidéo. Les paquets de données sont routés sur le réseau selon l'adresse des destinataires encapsulée dans la trame transmise. Seuls les destinataires interceptent et décodent les paquets qui leurs sont adressés.

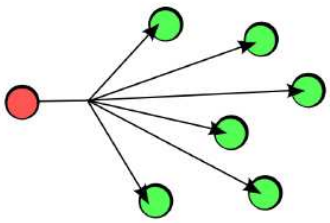
-Le routage multicast :



- **Le mode Broadcast (la diffusion)**

Le broadcast est un terme anglais définissant une diffusion de données depuis une source unique à un ensemble de récepteurs. Contrairement à une communication Point à Point, il est possible d'adresser des paquets de données à un ensemble de machines d'un même réseau uniquement par des adresses spécifiques qui seront interceptées par toutes les machines du réseau ou sous réseau**.**

-Le routage broadcast :

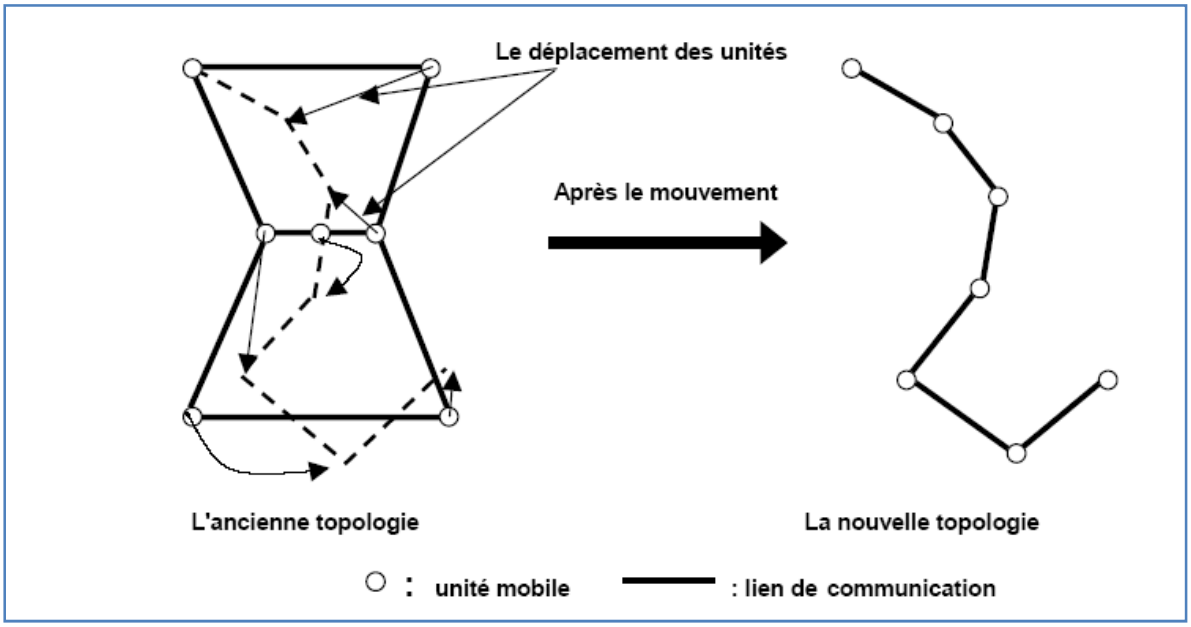


III.6. Caractéristiques des réseaux Ad Hoc :

Les réseaux sans fil Ad Hoc se caractérisent principalement par :

- **Topologies dynamiques**

Les nœuds sont libres de se déplacer arbitrairement. Ainsi, la topologie du réseau peut changer aléatoirement et rapidement et peut être constituée à la fois de liaisons unidirectionnelles et bidirectionnelles. Cela fait que la déconnexion des unités mobiles est très fréquente.



*-**figure ch1.10** : changement de topologie dans les réseaux ad hoc-*

- **Une architecture robuste :**

Il est peut probable de voir le réseau s'effondrer, il faudrait qu'un grand nombre d'unités qui la compose, cessent de fonctionner. Si une unité vient à disparaître, cela se répercute par un changement de route sans pour autant gêner les autres éléments.

- **Hétérogénéité des nœuds :**

Un nœud mobile peut être équipé d'une ou de plusieurs interfaces radio ayant des capacités de transmission variées et opérant dans des plages de fréquences différentes. Cette hétérogénéité de capacités peut engendrer des liens asymétriques dans le réseau. De plus les nœuds peuvent avoir des différences en terme de capacité de traitement (CPU, mémoire), de logiciel, de taille (petit, grand) et de mobilité (lent, rapide). Dans ce cas, une adaptation dynamique de protocoles s'avère nécessaire pour supporter de telles situations.

- **La contrainte d'énergie :**

Les équipements mobiles disposent de batteries limitées, et dans certains cas très limitées tel que les PDA, et par conséquent d'une durée de traitement réduite. Sachant qu'une partie de

l'énergie est déjà consommer par la fonctionnalité de routage (qu'elle soit active ou passive, l'unité reçoit des informations qu'elle transmet en consommant des ressources d'énergie), cela limite les services et applications supporter par chaque nœuds.

- **Les interférences :**

Elles peuvent être liées à des transmissions simultanées ou à des stations cachées. D'autres interférences sont provoquées par des équipements comme le cas des fours micro-onde avec le WIFI qui émettent dans la même bande de fréquences (2,4GHz).

- **L'atténuation radio :**

Comme tout réseau sans fil la portée est limitée et les obstacles rencontrés atténuent fortement le signale.

- **La sécurité :**

Un réseau sans fil est potentiellement utilisable par toute personne qui se trouve dans la zone de couverture radio de la station émettrice. Sans mesure de sécurité, il facile d'écouter le trafique, de rejouer les transmissions, de manipuler les en-têtes des paquets ou de saturer le signale. Donc une protection pour réduire les risques d'attaques, est souhaitable.

- **Une bande passante limitée**

Une des caractéristiques primordiales des réseaux basés sur la communication sans fil est l'utilisation d'un médium de communication partagé. Ce partage fait que la bande passante réservée à un hôte soit modeste.

- **Absence d'infrastructure**

Les réseaux ad hoc se distinguent par l'absence d'infrastructure préexistante et de tout genre d'administration centralisée. Chaque nœud mobile est responsable d'établir et de maintenir la connectivité du réseau et agit en tant que routeur pour relayer des communications ou gérer ses propres données. La gestion du réseau est ainsi distribuée.

- **Possibilité d'extension à un réseau filaire**

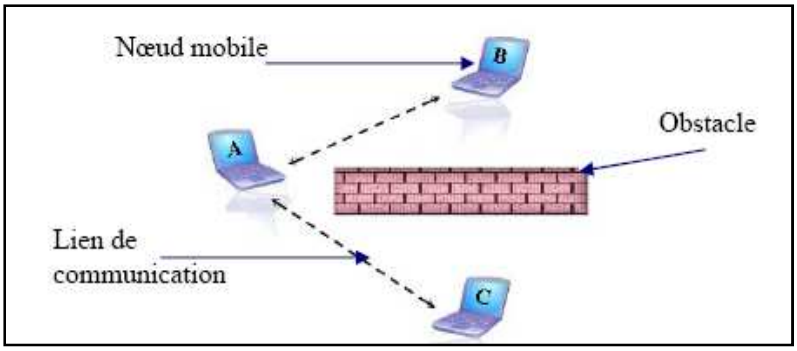
Pour étendre un réseau filaire par un réseau sans fil, les réseaux ad hoc sont la solution la plus flexible et la plus économique car elle permet d'augmenter la capacité globale du réseau avec un coût réduit. En effet, l'extension par un réseau cellulaire aurait été plus coûteuse à cause du coût élevé des BTS.

- **Taille du réseau Ad Hoc**

Elle est souvent de petite ou moyenne taille (une centaine de nœuds). Cependant quelques applications des réseaux Ad Hoc nécessitent une utilisation allant jusqu'à des dizaines de milliers de nœuds.

- **Nœuds cachés**

Ce phénomène est très particulier à l'environnement sans fil. Un exemple de ce phénomène est illustré par la figure suivante. Les nœuds B et C ne s'entendent pas, à cause d'un obstacle qui empêche la propagation des ondes. Les mécanismes d'accès au canal vont permettre alors à ces nœuds de commencer leurs émissions simultanément, ce qui provoque des collisions au niveau du nœud A.



*-**figure ch1.11** : nœud caché-*

III.7. Domaines d'applications des réseaux ad hoc :R[9] [6]

Les réseaux mobiles ad hoc ont une très large palette d'utilisations. En effet, ils sont robustes, peu coûteux et s'adaptent aussi bien aux milieux urbains, qu'aux milieux ruraux.
Parmi les applications nous citons :

- **Les applications militaires**

Un réseau mobile ad hoc est la solution idéale pour maintenir la liaison entre des chars d'assauts, des avions de chasse ou même entre des soldats et leur supérieurs au cours des exercices militaires ou dans un champ de bataille.

- **Opérations de secours**

Lors des catastrophes naturelles (Incendies, inondations, tremblement de terre…etc.), les réseaux mobiles Ad Hoc peuvent résoudre le problème de communication là où une installation filaire ne peut être réalisée qu'après de très longs délais d'attente.

- **Applications commerciales**

Utilisé pour un paiement électronique distant ou pour l'accès mobile à l'Internet ainsi que dans le service de guide en fonction de la position de l'utilisateur.

- **Les sites de patrimoine**

Dans les sites archéologiques, musées, châteaux ou tout autre lieu semblable et dans la plupart des cas où il est interdit d'installer des réseaux filaires vu le risque de nuisance sur de tels environnements.

- **Utilisation à des fins éducatives**

Les réseaux Ad Hoc facilitent l'échange et le partage d'informations entre les participants d'une conférence ou d'une séance de cours.

- **Les réseaux véhiculaires (Vehicular Ad hoc Network)**

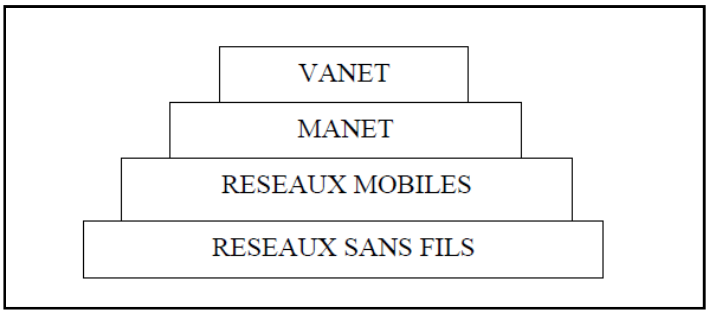
Grâce à cette application, aujourd'hui, on peu éviter des accidents de route, faciliter la circulation routière et améliorer le confort des passagers. C'est cette application qui nous intéresse.

IV. Réseaux Véhiculaires Ad hoc

IV.1. définition d'un réseau VANET [10][6][11]

Un réseau VANET est une particularité des réseaux MANET où les nœuds mobiles sont des véhicules (intelligents) équipés de calculateurs, de cartes réseau et de capteurs. Comme tout autre réseau Ad hoc, les véhicules peuvent communiquer entre eux (pour échanger les informations sur le trafic par exemple) ou avec des stations de base placées tout au long des routes (pour demander des informations ou accéder à internet…).
Les réseaux véhiculaires regroupent deux grandes classes d'applications, à savoir les applications qui permettent de bâtir un système de transport intelligent ITS (Intelligent transport System) et celles liées au confort.

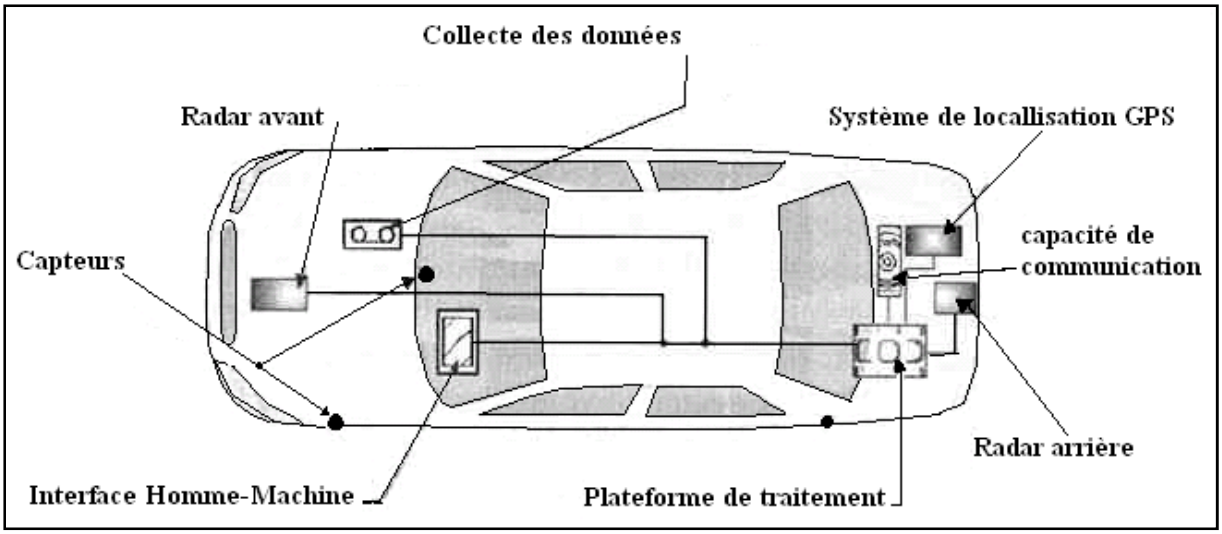
La figure ci-dessous représente la hiérarchie des réseaux sans fil où elle schématise l'inclusion des réseaux véhiculaires Ad Hoc VANET dans les réseaux mobile Ad Hoc MANET, les MANET dans les réseaux Mobiles ainsi que les réseaux mobiles dans les réseaux sans fil.



*-**figure ch1.12** : hiérarchie des réseaux sans fil-*

IV.2. Le Nœud du réseau VANET [10]

Un nœud d'un réseau VANET est un véhicule équipé de terminaux tels que les calculateurs, les interfaces réseaux ainsi que des capteurs capables de collecter les informations et de les traiter. On parle de la notion de « véhicule intelligent ». La figure ci-dessous modélise un véhicule intelligent.



*-**Figure ch1.13** : Véhicule intelligent R[6]-*

IV.3. Technologies utilisées dans la communication véhiculaire[6][10] :

Les réseaux véhiculaires par analogie à ce qui existe dans les réseaux sans fil peuvent être déployés suivant trois catégories :

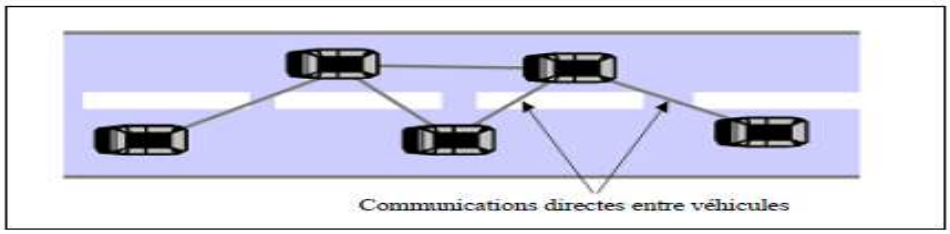
- **Communication de véhicule à véhicule :(v2v)**

Dans cette catégorie, un réseau de véhicule est vu comme un cas particulier du réseau MANET (Mobile Ad Hoc Network) où les contraintes d'énergie, de mémoire et de capacité sont relaxées et où le modèle de mobilité n'est pas aléatoire mais prévisible avec une très grande mobilité. Cette architecture peut être utilisée dans le scénario de diffusion d'alertes (freinage d'urgence, collision, ralentissement…) ou pour la conduite coopérative. Aucune infrastructure n'est utilisée, aucune installation n'est nécessaire sur les routes et tous les véhicules sont équipés pour communiquer directement entre eux n'importe où, que se soit sur les autoroutes, des routes de montagnes ou des routes urbaines, ce qui donne une communication moins coûteuse et plus flexible. Cette approche souffre de certains inconvénients dont nous citons :

a. Les délais de communication qui sont élevés, étant donné que la communication se fait en utilisant le multi sauts.

b. Les déconnexions fréquentes dues au fait que les véhicules sont mobiles.

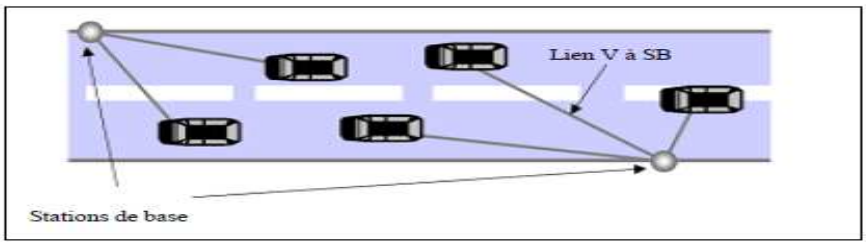
c. La sécurité réseau est très limitée



*-**Figure ch1.14** : communication v2v-*

- **Communication de véhicule avec utilisation d'infrastructures :**

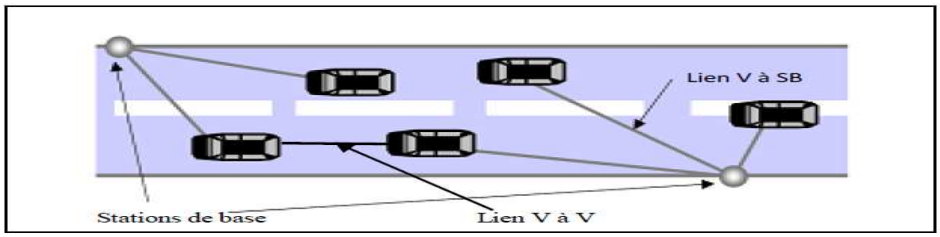
Dans cette catégorie, on ne se concentre pas seulement sur de simples systèmes de communications inter véhicules mais aussi sur ceux qui utilisent des stations de bases ou points d'infrastructure RSU (Road Side Units, dénomination proposée par le consortium C2C-CC). Cette approche repose sur le modèle client/serveur où les véhicules sont les clients et les stations installées le long de la route sont les serveurs. Ces serveurs sont connectés entre eux via une interface filaire ou sans fil. Toute communication doit passer par eux. Ils peuvent aussi offrir aux utilisateurs plusieurs services concernant le trafic, l'accès à internet, l'échange de données de voiture-à-domicile et même la communication de voiture-à-garage pour le diagnostic distant.
L'inconvénient majeur de cette approche est que l'installation des stations le long des routes est une tâche coûteuse et prend beaucoup de temps, sans oublier les coûts relatifs à la maintenance des stations.



*-**Figure ch1.15** : Communication véhicule à station de base.-*

- **Communication Hybride**

La combinaison des communications v2v avec les communications de véhicules avec utilisation d'infrastructures, permet d'obtenir une communication hybride très intéressante. En effet, les portées des infrastructures (stations de bases) étant limitées, l'utilisation des véhicules comme relais permet d'étendre cette distance. Dans un but économique et afin d'éviter la multiplication des stations de bases à chaque coin de rue, l'utilisation des sauts par véhicules intermédiaires prend toute son importance.



*-**Figure ch1.16** : communication hybride.-*

IV.4. Caractéristiques des réseaux VANET :[6][10]

Les réseaux véhiculaires ont des caractéristiques spécifiques qui les distinguent des réseaux Ad Hoc, à savoir :

- **La Collecte d'informations et la perception de l'environnement proche**

La collecte d'informations se fait en utilisant différents capteurs de toutes catégories (caméras, capteurs de pollution, capteurs de pluies, capteurs de l'état de la route et de voiture, etc…) qui permettent au conducteur à bord de son véhicule de disposer d'un certain nombre d'informations et d'une meilleure visibilité pour pouvoir réagir d'une manière adéquate aux changements de son environnement proche.

- **Capacité de traitement, d'énergie et de communication**

Contrairement au contexte des réseaux Ad Hoc où la contrainte d'énergie, à titre d'exemple, représente une des problématiques traitées, les éléments du réseau VANET n'ont pas de limite en terme d'énergie et disposent d'une grande capacité de traitement et peuvent avoir plusieurs interfaces de communication (WIFI, Bluetooth et autres). Grâce aux **N**ouvelles **T**echnologies de l'**I**nformation et de la **C**ommunication (NTIC) le conducteur peut prendre une décision à l'aide des traitements et des interprétations des informations collectées.

- **Environnement de déplacement et modèle de mobilité**

Les environnements pris en compte par les réseaux Ad Hoc sont souvent limités à des espaces ouverts ou indoor. Les déplacements des véhicules quant à eux sont liés aux structures des routes (intersections, panneaux de signalisation, etc…) et aux stations de base routières (infrastructures) que se soit dans les autoroutes ou au sein d'une zone métropolitaine. Les contraintes imposées par ce type d'environnement affectent considérablement le modèle de mobilité et la qualité des transmissions radio à prendre en compte dans les protocoles de routage. En outre la mobilité est un facteur lié directement au conducteur du véhicule.

- **Forte mobilité, topologie du réseau et connectivité**

A la différence des réseaux Ad Hoc, les réseaux VANET sont caractérisés par la forte mobilité des nœuds (véhicules), liée à la vitesse des voitures qui est très importante dans les autoroutes. Par conséquent, un nœud peut rejoindre ou quitter le réseau en un temps très court ce qui rend les changements de topologie très fréquent. De plus, des problèmes peuvent

apparaitre quand le système IVC (Inter Vehicle Communication) n'est pas équipé dans la majorité des véhicules.

- **Type de l'information transportée et diffusée**

Un des objectifs des réseaux VANET étant la sécurité routière, les types de communications s'axeront sur les diffusions de messages d'une source vers plusieurs destinataires. Néanmoins, les véhicules sont concernés par la diffusion d'informations en fonction de leurs positions géographiques et leurs degrés d'implication dans l'évènement déclenché. Dans de telles situations, les communications sont principalement unidirectionnelles.

IV.5. Applications des réseaux VANET [6][10] :

Les principales applications des réseaux VANET peuvent être classées en trois catégories.

- **Application dans la prévention et la sécurité routière**

La sécurité routière est devenue une priorité dans la plupart des pays développés, cette priorité est motivée par le nombre croissant d'accidents sur les routes associé à un parc de véhicules de plus en plus important. Les VANET permettent de prévenir les collisions et les travaux sur les routes, de détecter les obstacles (fixes ou mobiles) et de distribuer les informations météorologiques par envoi de messages d'alerte. A titre d'exemple, alerter un conducteur en cas d'accidents permet d'avertir les véhicules qui se dirigent vers le lieu de l'accident que les conditions de circulations se trouvent modifiées et qu'il est nécessaire de redoubler de vigilance. Les messages d'alertes et de sécurité doivent être de taille réduite pour être transmis le plus rapidement possible et doivent être émis à des périodes régulières.

- **Application pour l'optimisation du trafic et aide dans la conduite**

Le trafic automobile peut être grandement amélioré grâce à la collecte et au partage de données collectées par les véhicules, ce qui devient un support technique pour les conducteurs. Une voiture peut, par exemple, être avertie en cas d'un ralentissement anormal (bouchon, embouteillage, éboulement de rochers ou travaux).

- **Applications au confort du conducteur et des passagers**

Les réseaux véhiculaires peuvent aussi améliorer le confort des conducteurs et des passagers. Ce confort est illustré par l'accès à internet, la messagerie, le chat inter –véhicule, etc. Les passagers dans la voiture peuvent jouer en réseaux, télécharger des fichiers MP3, envoyer des cartes à des amis, etc.

IV.6. Travaux dans le domaine des VANET [10][6]

Les propriétés des réseaux véhiculaires offrent des challenges importants. Les VANET s'ouvrent à plusieurs domaines de recherche dont nous citons les plus importants:

- **Sécurité**

La sécurité est un défi majeur ayant un grand impact sur le futur déploiement des réseaux véhiculaires ainsi que sur leurs applications. En raison de la sensibilité des domaines d'utilisation des VANET, une intrusion d'un véhicule malveillant aurait des conséquences graves sur l'ensemble des véhicules interconnectés. C'est pour cette raison que beaucoup de travaux de recherche ont été réalisés pour développer un mécanisme de sécurité instituant les relations de confiance entre les nœuds communicant et garantissant le contrôle d'accès aux services.

- **L'accès au canal**

Les réseaux véhiculaires utilisent des communications radio. Par conséquent, il est important de concevoir des solutions spécifiques aux réseaux VANET qui permettent d'apporter de la qualité de service et de gérer les priorités en résolvant les problèmes d'interférences radio, les problèmes de propagation à multi-trajets des ondes ainsi que les irrégularités électromagnétiques.

- **Localisation des véhicules**

Si l'un des véhicules du réseau doit être localisé (dans le cas d'un accident par exemple), les autres doivent être informés de sa position. Le problème est que tous les véhicules ne sont pas équipés d'un système de repérage par satellite (GPS). Pour cette raison, un mécanisme de localisation sans utilisation de GPS est nécessaire.

- **Problèmes de congestion**

L'un des problèmes des VANET est que chaque véhicule communique avec tous ceux qui sont dans sa zone de couverture. Ceci entraîne une dégradation de la qualité de service (QoS) avec l'augmentation du nombre de véhicules. Ce problème a fait l'objet de plusieurs études.

- **Mobilité dans la simulation des réseaux**

Dans la simulation des VANET, le facteur mobilité a longtemps été négligé. On ne considérait pas la différence de mouvements entre les nœuds des VANET et des MANET, ce qui pouvait altérer les résultats de la simulation. Pour cette raison, de plus en plus d'équipes de recherche s'intéressent à l'étude de la mobilité dans les VANET.
Avec un bon simulateur, plus le modèle de mobilité est réaliste, plus les résultats de la simulation sont proches de la réalité. D'où l'impact direct des modèles de mobilité sur la réussite d'une simulation.

- **Routage**

Le routage dans les réseaux VANET est un problème très difficile à gérer et un axe de recherche pour beaucoup de chercheurs. Pour que les véhicules puissent communiquer entre eux, un protocole de routage doit être défini. En effet quand les terminaux ne sont pas à une

portée de transmission radio directe, le routage est exigé pour établir la communication entre les véhicules.

IV.7. Conclusion

Dans ce chapitre nous avons présenté les réseaux sans fil, les réseaux mobiles Ad Hoc ainsi que les réseaux véhiculaires Ad Hoc VANET qui ne sont qu'une particularité des réseaux MANET. Nous avons décrit également leurs caractéristiques, leurs applications et leurs contraintes.

Dans le prochain chapitre nous traiterons le routage dans les réseaux mobiles Ad hoc. Nous nous intéresserons à la problématique du routage et les contraintes liées aux réseaux Ad Hoc. Nous décrierons également les principaux protocoles et leurs classifications.

I. notions de base sur le routage et les protocoles de routage

I.1. Introduction :

Une des caractéristiques des réseaux Ad Hoc est l'absence d'infrastructure fixe préexistante, ce qui fait que ces réseaux doivent assurer automatiquement leur propre organisation pour pouvoir acheminer les données entre les unités mobiles. L'acheminement des données requière l'utilisation de protocoles de communication ou de routages spécifiques pour assurer une stratégie qui garantit à n'importe quel moment, la connectivité du réseau, malgré l'absence des stations de base. Chaque nœud ou unité mobile joue alors le rôle de station et de routeur. La stratégie du routage doit prendre en considération les changements de topologie, la forte mobilité, la capacité limitée des liaisons radio.
Dans cette partie du chapitre nous allons étudier le concept de routage dans les réseaux Ad Hoc, les problèmes liés à la conception des protocoles de routage, la classification des protocoles selon différentes manières et plusieurs critères, à savoir :

- L'architecture (uniforme ou non uniforme).
- L'approche de routage (protocole proactif, réactif ou hybride).
- L'algorithme utilisé : algorithmes dynamiques (vecteur de distance ou état de liens) ou algorithmes à la demande (source ou apprentissage en arrière).

Nous décrivons également quelques protocoles ainsi que leurs fonctionnements.

I.2. Quelques définitions : R [12][13][14][15]

a. Un protocole :

Un protocole est un ensemble de règles de communication respectées par tous les systèmes interconnectés afin de permettre la liaison entre systèmes émetteurs et systèmes récepteurs.
Les objectifs d'un protocole sont:

- L'information doit arriver à destination le plus rapidement possible.
- L'information doit arriver à destination sans erreurs.
- L'expéditeur doit être informé, éventuellement, de la bonne réception par un acquittement.
- Il ne doit pas y avoir de conflit en cas de requêtes simultanées.
- La transparence du réseau pour l'utilisateur

b. Le routage :

Le routage est une méthode d'acheminement des informations à la bonne destination à travers un réseau de connexion donné.
Le routage est, dans un réseau Ad Hoc, la brique technologique fondamentale permettant d'assurer la connectivité du réseau. Dans un tel réseau, les nœuds radio ne sont pas nécessairement à portée directe. Un paquet peut donc devoir être relayé par des nœuds intermédiaires pour atteindre sa destination finale.
Le routage dans les réseaux ad-hoc est assez délicat étant donnée la nature changeante de la topologie de ce type de réseaux. De nombreux protocoles et algorithmes ont été proposés et leurs performances ont été analysées dans différentes situations.

c. Le protocole de routage : C'est un protocole qui permet d'acheminer un paquet envoyé par une source à une destination en respectant certains critères.

I.3. Problème de routage dans les réseaux mobiles Ad Hoc [6][13] :

Si certaines propriétés des réseaux MANET offrent des atouts qui n'existent pas dans les réseaux filaires, il n'empêche que des problèmes en découlent, voici quelque un :

- La capacité limitée des liens radio : un protocole de routage doit être efficace malgré la bande passante limitée, il doit minimiser le trafic du contrôle nécessaire à l'établissement et à la maintenance des routes afin de réduire la charge sur le réseau.

- L'inexistence d'infrastructure ou d'administration fixe de communications, en particulier les routeurs, ce qui attribue la tâche de routage aux nœuds eux-mêmes du réseau et impose un fonctionnement distribué.

- La limitation de la portée de transmission d'un nœud et les problèmes inhérents aux transmissions sur le canal radio (interférence, masquage, atténuation de parcours, etc.).

- Les changements fréquents, brusques et imprévisibles de l'état des liens et des nœuds doivent être pris en compte par le protocole de routage.

- Les ressources matérielles restreintes des unités mobiles excluent les algorithmes exigeant en capacité de traitement et de mémoire.

- La contrainte énergétique due à la capacité limitée des batteries doit être prise en compte par le protocole de routage.
- Environnement de déplacement : on distingue les environnements suivants :

  - autoroute : environnement ouvert caractérisé par une grande vitesse de déplacement (avec des limites : vitesse min et vitesse max), avec des dépassements de véhicules et une densité de nœuds qui est fonction de l'heure de la journée, du jour de la semaine et du la période de l'année ;

- ville : vitesse modérée avec une probabilité d'intersection plus grande. Il existe des endroits d'arrêt aux feux, des densités de voitures plus ou moins grandes et l'existence de routes plus fréquentées que d'autres (routes principales, endroit commercial ou touristique par exemple);
- campagne : caractérisée par des vitesses moins importantes avec une densité de voitures plus faible.

Vu ces difficultés, il s'avère nécessaire de mettre en œuvre des protocoles de routage qui s'adaptent aux exigences de ces réseaux.

I.4. objectifs lors de l'élaboration d'un protocole de routage MANET :

Les critères suivants sont adoptés comme objectifs lors de l'élaboration d'un protocole de routage MANET :

- **Eviter les boucles de routage** : Une boucle de routage se produit lorsqu'un nœud d'une route choisit pour saut suivant un autre nœud qui l'a précédé dans cette même route. Les boucles de routages sont néfastes pour les réseaux MANET d'une part parce qu'ils ne convergent pas, d'autre part parce qu'elles gaspillent de la bande passante et des ressources (en terme de mémoire et de traitement). Ces boucles doivent, par conséquent, être évitées.

- **Pouvoir maintenir une topologie dynamique** : Les réseaux MANET se caractérisent par un changement fréquent de la topologie, en particulier à cause de la technique multi saut. Si une route est établie vers une destination, il est très probable qu'un lien qui compose cette route soit coupé si l'un des nœuds qui le relie se déplace en dehors de la portée de l'autre. Les protocoles de routage MANET doivent s'adapter rapidement aux changements de topologies, en proposant des routes de longueurs acceptables, même en cas de forte mobilité des terminaux afin de garantir la viabilité des routes vers les destinations, même si des liens de ces routes sont coupés.

- **Réduire la taille des entêtes :** Les messages de contrôle consomment de la bande passante, des ressources mémoires et de la puissance de la batterie.
  De ce fait, les protocoles de routage doivent réduire au maximum la taille des données de contrôle et de signalisation afin de minimiser le délai de transmission et les pertes de données pour allouer la grande partie des ressources aux données utiles.

- **Réduire les traitements sur les entêtes** : Les algorithmes des protocoles de routage dans les réseaux MANET doivent avoir une complexité réduite afin de réduire le nombre de cycles pour effectuer leurs traitements et ainsi diminuer la consommation des ressources.
- **Offrir le routage multi-saut** : Un protocole de routage se doit d'offrir un mécanisme de découverte de routes multi-sauts entre une source et une destination si celles-ci ne s'entendent pas.

I.5. Classification des protocoles de routage dans les réseaux mobiles Ad Hoc [6]:

Les protocoles de routage destinés aux réseaux mobiles Ad Hoc peuvent être classés de différentes manières, selon plusieurs critères. Ils peuvent être classés selon leur architecture (uniforme ou non uniforme), leur approche de routage (protocole proactif, réactif ou hybride) ainsi que par leur type d'algorithmes dynamiques (vecteur de distance ou état de liens) ou algorithmes à la demande (source ou apprentissage en arrière).

I.5. 1. Classification selon l'architecture :

Ce critère divise les protocoles de routage en deux classes

- **Les protocoles uniformes :** Les protocoles de routage uniformes « à plat » **c**onsidèrent que tous les nœuds sont égaux, dans le même niveau hiérarchique et possèdent, ainsi, les mêmes rôles et fonctions. Par conséquent, aucune hiérarchie n'est définie entre les nœuds du réseau, chaque nœud envoie et reçoit des messages de contrôle de routage. La décision d'un nœud de router des paquets dépendra de sa position.

- **Les protocoles non uniformes :**

Les protocoles de routage non uniformes « hiérarchiques » tentent de limiter la complexité du routage en réduisant le nombre de nœuds qui contribuent à la détermination des routes, ils fonctionnent en attribuant aux nœuds des rôles qui varient de l'un à l'autre. Une structure hiérarchique entre les nœuds est définie selon leurs fonctions. Certains nœuds sont élus pour accomplir des tâches bien particulières qui conduisent à une vision en plusieurs niveaux de la topologie du réseau afin de faciliter l'équilibrage de la charge et de mieux la gérer (surtout dans les réseaux mobiles Ad Hoc de grande taille), ce qui conduit à une meilleure qualité de service.

I.5. 2. Classification selon l'approche de routage [13][14] :

Dans cette classification basée sur le mécanisme d'établissement de la route, nous distinguons trois classes :

- **Les protocoles proactifs**

Le principe de base des protocoles proactifs est de calculer les routes à l'avance, en continuant ainsi de maintenir à jour les tables de routage de tel sorte que lorsqu'un nœud désire envoyer un paquet à un autre nœud, une route soit immédiatement connue.
Les nœuds dans les réseaux mobiles Ad Hoc peuvent apparaitre et disparaitre de manière aléatoire d'où la nécessité de mise en place d'un système d'échange continuel des paquets de

contrôle. Les tables de routage sont donc maintenues grâce à ces paquets. Cette manière de procéder permet aux nœuds de construire de façon distribuée la topologie du réseau, lorsqu'un nœud reçoit un paquet de contrôle, il met à jour ses tables de routage. Ainsi, de nouvelles routes seront construites sur la base des informations topologiques transportées par les trames de contrôle. Ce processus est déclenché à chaque changement de topologie pour reconstruire à nouveau les routes.
Un des avantages de ces protocoles est la disponibilité immédiate de la route lors du besoin. Cependant, la bande passante diminue le trafic généré par l'échange de paquets de contrôle.

- **Les protocoles réactifs (routage à la demande):**

Le principe des protocoles réactifs également appelés protocoles de routage à la demande (*on-demand routing protocols*) est de lancer le processus de recherche de routes uniquement en cas de besoin (à la demande). Cela permet d'économiser de la bande passante et de l'énergie. Lorsqu'un paquet doit être envoyé d'un nœud source vers un nœud cible, le protocole de routage va rechercher un chemin jusqu'à la destination, une fois ce chemin trouvé, il est inscrit dans la table de routage et peut être utilisé tant que la destination est joignable ou jusqu'au moment où la route devient inutile. En général, cette recherche se fait par inondation (un paquet de recherche de route est transmis de proche en proche dans tout ou une partie du réseau). Dans ce cas, la bande passante est plus large, cependant, du fait que l'on ne dispose pas immédiatement de la route vers la destination. Le délai d'établissement de la route est plus important en comparaison avec les protocoles proactifs.

- **Les protocoles hybrides :**

Les protocoles hybrides combinent les deux approches précédentes. Ils utilisent une technique proactive dans un petit périmètre autour de la source où le nombre de sauts est assez petit (par exemple trois à quatre sauts) et réactive pour les nœuds plus éloignés.
L'avantage des protocoles hybrides est le fait qu'ils s'adaptent mieux aux réseaux de grandes tailles. Cependant, ce type de protocole cumule les inconvénients des protocoles proactifs et ceux des protocoles réactifs, tels que l'échange de paquets de contrôle réguliers et l'inondation de l'ensemble du réseau pour chercher une route vers un nœud éloigné.
Une comparaison entre les deux classes proactive et réactive est présentée dans le tableau suivant :

Routage proactif		Routage réactif	
Avantages	**inconvénients**	**Avantages**	**inconvénients**
La topologie du réseau est connue de tous les mobiles. Les routes sont disponibles immédiatement.	Il faut diffuser régulièrement des informations sur les changements de topologie du réseau.	Les mobiles ne conservent pratiquement aucune information sur la topologie globale du réseau : seules les informations sur les routes actives sont stockées.	
Les protocoles proactifs disposent en permanence d'une route pour chaque destination dans le réseau.	Un volume de signalisations important.	Les protocoles réactifs génèrent à priori un volume plus faible de signalisations.	Les protocoles réactifs engendrent un délai lors de la construction (ou de la reconstruction) des routes et produisent plus difficilement des routes optimales.

*-**Tableau ch2.1 :** Comparaison entre protocoles proactifs et protocoles réactifs-.*

I.5. 3. Classification selon le type d'algorithme utilisé :

Une autre classification basée sur le type d'algorithme utilisé est possible pour les protocoles de routage Ad Hoc. Il existe deux grandes familles d'algorithmes de routage dynamique (vecteur de distance ou état de liens) et deux grandes familles d'algorithmes de routage à la demande (source ou apprentissage en arrière).

- **Les protocoles de routage à vecteur de distance (Distance Vector Protocols)**

Ces protocoles sont basés sur l'algorithme de Bellman-Ford. Leur principe est basé sur l'échange entre les nœuds voisins d'informations de distance des destinations connues. Autrement dit, chaque nœud envoie à ses voisins la liste des destinations joignables et le coût (distance) associé au chemin le plus court menant vers cette destination.
A chaque réception d'un paquet contenant les informations topologiques, le nœud en question met à jour sa liste de destination par le coût minimum.
Les protocoles à vecteur de distance sont simples à programmer et facile à implémenter. En revanche, ils ont un mauvais comportement dans les réseaux dynamiques tel que les boucles de routage et le comptage à l'infini connu aussi sous le nom de problème de Bellman-Ford.

- **Les protocoles de routage à état des liens (Link State Protocols) [16]**

Ces protocoles utilisent un algorithme plus efficace en calcul du plus court chemin que celui des protocoles de routage à vecteur de distance, qui est l'algorithme de Dijkstra. Ils sont basés sur le l'état des liens (topologie) du réseau, l'ensemble de ces informations permet aux nœuds de dessiner une vue globale sur le réseau. Une table de routage est maintenue dans chaque nœud. Elle est construite à partir des informations échangées sur l'état des liens du réseau. A partir de cette vue globale du réseau, il est facile de trouver des routes alternatives lorsqu'un lien est rompu. Ainsi une route est immédiatement disponible à la demande. Il est même possible d'utiliser simultanément plusieurs routes vers une même destination, augmentant ainsi la répartition de la charge et la tolérance aux pannes dans le réseau. En contre partie, si le réseau est étendu, la quantité d'informations sur l'état de tous les liens du réseau au niveau de chaque nœud nécessite un espace de stockage considérable.

- **Le protocole de routage source**

Chaque nœud doit posséder la topologie et les caractéristiques du réseau en entier. Les données doivent être parfaitement à jour. Ce routage convient aux réseaux de tailles moyennes (pour éviter la surcharge de la mémoire) à hauts débits (le calcul de routes est effectué une seule fois). Pour cela, le temps de calcul des routes ne doit pas être trop grand.
Dans cet algorithme, afin d'émettre un paquet de données à un nœud, l'émetteur spécifie dans l'entête du paquet à envoyer l'adresse de chaque nœud à travers lequel le paquet va passer pour atteindre la destination (route source). Par la suite, l'émetteur transmet le paquet via son interface, au premier nœud spécifié dans la route source. Un nœud qui reçoit le paquet, et qui est différent de la destination, supprime son adresse de l'entête du paquet reçu et le transmet au nœud suivant identifié dans la route source. Ce processus se répète jusqu'à ce que le paquet atteigne sa destination finale. Enfin, le paquet est délivré à la couche réseau du dernier hôte.

- **Le protocole de routage par apprentissage en arrière**

Le chemin établi entre les nœuds est un chemin bidirectionnel simultané *(full duplex).* La source gardera trace du chemin tant qu'il restera en cours d'utilisation. Ce type de routage nécessite moins de mémoire que le routage source. Par conséquent, il est plus adapté pour des réseaux de plus grandes tailles.
Afin de transmettre un paquet à l'aide de cette méthode, le nœud émetteur inonde le réseau avec sa requête. Ainsi chaque nœud intermédiaire, dit le transit, indique le chemin au nœud source lors de la réception de la requête. On dit qu'il apprend le chemin au nœud source, tout en sauvegardant la route dans la table transmise. Enfin, lorsque la requête arrive au nœud destinataire, et suivant le même chemin, ce dernier transmet sa réponse sous forme de requête.

I.6. protocoles de routage :

I. 6. 1. Le protocole OLSR (Optimized Link State Protocol) [6][14]

OLSR pour Optimized Link State Protocol est un protocole de routage proactif développé dans le cadre du projet Hypercom de l'Institut National de la Recherche en Informatique et Automatique (INRIA) de France et proposé en tant que RFC (Request For Comment) expérimentale à l'IETF (Internet Engineering Task Force). Il est considéré comme une optimisation du protocole à état des liens filaires pour les réseaux mobiles Ad Hoc. Il a pour objectif de fournir des routes de plus court chemin vers une destination en termes de nombre de sauts en utilisant l'algorithme de Dijksra. Son innovation réside dans sa façon d'économiser les ressources radio lors des diffusions, ceci est réalisé grâce à l'utilisation de la technique des relais multipoints (MPR : Multi-Point Relaying), donc le principe est que chaque nœud construit un sous ensemble appelé MPR, parmi ses voisins, qui permet d'atteindre tous ses voisins à deux sauts, les nœuds de cet ensemble servent à acheminer et retransmettre les messages qu'ils reçoivent. Les voisins d'un nœud qui ne sont pas MPR, lisent et traitent les paquets mais ne les retransmettent pas.
Le processus de construction des routes dans OLSR passe par les étapes suivantes :

- Découverte du voisinage
- Sélection des relais multipoints
- Annonce des MPR et diffusion des voisinages
- Calcul des tables de routage

I.7. Quelque protocoles de routage pour les réseaux VANET

Différentes solutions pour le routage dans les réseaux VANET ont été proposées, nous distinguons deux classes de protocoles de routage: les protocoles basés sur la topologie qui sont divisés en protocoles proactifs, réactifs et hybrides et les protocoles basés sur la localisation qui utilisent la position physique des nœuds mobiles pour configurer le routage. Parmi les protocoles de routage VANET on cite :

I.7.1. Le protocole GSR (Geographic Source Routing) [13] :

GSR est un protocole de routage géographique qui combine le routage basé sur la localisation avec le routage basé sur la topologie des routes pour construire une connaissance adaptée à l'environnement urbain. Le principe de GSR est que le véhicule source désirant envoyer des données vers un véhicule cible, calcule le chemin de routage le plus court à partir des informations géographiques d'une carte routière et en utilisant les algorithmes de recherche du plus court chemin, par exemple Djikstra. A partir du chemin du routage calculé, le véhicule source sélectionne ensuite une séquence d'intersections par lesquelles le paquet de données doit transiter afin d'atteindre le véhicule destinataire. Cette séquence d'intersections

est constituée d'un ensemble de points géographiques fixe de passage du paquet de données. Pour envoyer les messages d'une intersection à une autre, l'approche Gloutonne est utilisée.

I.7.2. Le protocole A-STAR (Anchor-based Street and Traffic Aware Routing) [6]

A-STAR est un protocole de routage basé sur la localisation (position) pour un environnement de communication véhiculaire métropolitain. Il utilise particulièrement les informations sur les itinéraires d'autobus de ville pour identifier une route d'ancre (anchor route) avec une connectivité élevée pour l'acheminement des paquets. A-STAR est similaire au protocole GSR du fait que les deux adoptent une approche de routage basée sur l'ancrage (anchor based) qui tient compte des caractéristiques des rues. Cependant, contrairement à GSR, il calcule les anchor paths en fonction du trafic (trafics de bus, véhicules, etc.). Un point est associé à chaque rue en fonction de sa capacité (grande ou petite rue qui est desservie par un nombre de bus différent). Les informations de routes fournies par les bus donnent une idée sur la charge du réseau véhiculaire dans chaque rue. Ce qui donne une image de la ville a des moments différents.

I.7.3. Le protocole UMB (Urban Multi hop Broadcast Protocol)[6]

C'est un protocole efficace de la norme 802.11, basé sur l'algorithme de diffusion multi saut pour les réseaux inter véhiculaires avec support d'infrastructure dans le but de réduire les collisions et d'utiliser efficacement la bande passante. Contrairement aux protocoles de diffusion par inondation, UMB confie les opérations d'envoi et de reconnaissance des paquets aux nœuds les plus éloignés sans connaître à priori des informations sur la topologie du réseau.
UMB est décomposé en deux phases : la première appelée diffusion directionnelle, où le véhicule source sélectionne un nœud dans la direction de diffusion pour faire un reliage de données sans aucune information sur la topologie. La deuxième diffusion aux intersections pour disséminer les paquets dans toutes les directions, pour cela UMB utilise des répéteurs installés dans les intersections pour l'envoi des paquets vers tous les segments. On suppose que chaque véhicule est équipé par un récepteur GPS (Global Position System) et une carte routière électronique. Le principal avantage du protocole UMB est la fiabilité de diffusion multi-saut dans les canaux urbains.

I.7.4. VADD (*Vehicle-Assisted Data Delivery*) R[17] :

VADD est un protocole de routage qui prend en considération le contexte des réseaux de véhicules et exploite le mouvement prévisible des vehicules pour décider de retransmettre ou non le message. Il utilise particulièrement les informations sur le trafic routier au niveau d'une route pour estimer le délai mis par un paquet pour parcourir un tel segment. Par conséquent, les paquets seront acheminés le long d'un chemin ayant le plus faible délai de bout en bout.

I.7.5. Le protocole de routage MORA (*MOvement-based Routing Algorithm*) :

MORA exploite la position et la direction de mouvement de véhicules pour adapter les décisions de retransmission au contexte des vehicules et faire face ainsi a la forte mobilité des nœuds et au changement assez fréquent de la topologie.

I.7.6. Le protocole MURU (*A MUlti-hop Routing protocol for Urban vehicular ad hoc networks*) R[17] :

MURU est un autre protocole de routage basé sur le mouvement et adapté aux environnements urbains. Les auteurs utilisent une métrique appelée degré de déconnexion attendu (*Expected Disconnexion Degree*) pour évaluer la qualité du chemin. Cette métrique est calculée à partir des informations sur la prédiction de la vitesse et la trajectoire de chaque véhicule.

I.7.7. Le protocole HOP R[17] (C*onditional Transmissions*) :

HOP est un protocole de routage adapté à une communication one-to-many. Il utilise une diffusion basée sur les transmissions conditionnelles. La solution proposée ne nécessite pas de connaissance a priori du voisinage, ni des récepteurs. Elle permet la transmission de messages dans une zone géographique donnée en avant ou en arrière de l'émetteur.

I.8. Conclusion :

Dans ce chapitre nous avons traité le routage dans les réseaux mobiles Ad hoc en s'intéressant à la problématique du routage et les contraintes liées aux réseaux Ad Hoc. Nous avons décrit également les principaux protocoles et leurs classifications.

Le chapitre suivant est consacré à l'étude de l'environnement de simulation, NS2 et NS3.

Chapitre III : Environnement de simulation pour les réseaux Vanet

Première partie

I. Le simulateur NS.2 :

I.1. Introduction :

Plusieurs simulateurs ont été conçus pour étudier les performances des réseaux, tels que Network Simulator 2, NS3, OPNET, OMNET++, etc. Notre choix s'est porté sur NS, Ce choix se justifie par la large diffusion de ce simulateur auprès de la communauté scientifique et par sa souplesse puisque le code source peut facilement être modifié selon le besoin. En effet NS est un logiciel open source extensible, écrit en C++, avec une documentation disponible, gratuite et très détaillée ; il suffit de comprendre la langue anglaise et on a accès à tous les détails croustillants sur ce logiciel qui est une preuve des capacités de la programmation orientée objet.

I.2. Étude du simulateur NS2 (network simulateur) :

I.2.1 . Introduction :

NS est un outil logiciel de simulation de réseaux informatiques, il est disponible sur internet et son utilisation est gratuite.

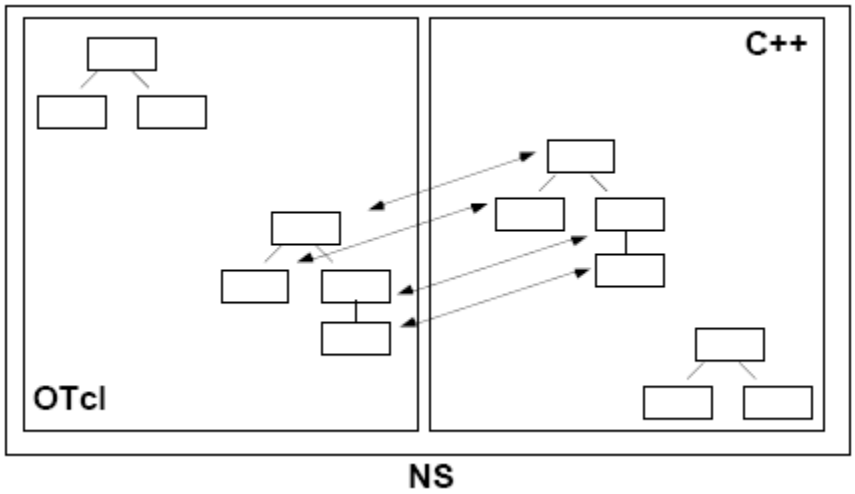
C'est un simulateur à événements discrets orienté objet. Il est écrit en C++ avec une interface textuelle (ou shell) qui utilise le langage OTcl (Object Tool Command Langage). L'OTcl est une extension objet du langage de commande Tcl. Le langage C++ sert à décrire le fonctionnement interne des composants de la simulation. Pour reprendre la terminologie objet, il sert à définir les classes. Quant au langage OTcl, il fournit un moyen flexible et puissant de contrôle de la simulation comme le déclenchement d'événements, la configuration du réseau, la collecte de statistiques, etc.

L'application NS se compose de deux éléments fonctionnels : **un interpréteur** et **un moteur de simulation**. Au moyen de l'interpréteur l'utilisateur est capable de créer le modèle de simulation ce qui revient à assembler les différents composants nécessaires à l'étude. Les composants du modèle de simulation sont appelés objets ou encore instances de classe. Le moteur de simulation effectue les calculs applicables au modèle préalablement construit par l'utilisateur via l'interpréteur. NS bénéficie de toutes les possibilités qu'offrent les techniques objets comme l'héritage, le polymorphisme, la surcharge, etc. L'héritage permet d'élaborer des arborescences de classes. Le modèle de simulation est construit à partir d'une arborescence de classes qui se dédouble :

- l'une définie en OTcl et dite arborescence interprétée. Elle est utilisée par l'interpréteur et est visible par l'utilisateur.

- l'autre définie en C++ que l'on nommera compilée. Elle forme l'arborescence utilisée par le moteur de simulation (que l'on appellera par la suite simulateur). C'est l'ombre de l'arborescence interprétée.

Les deux arborescences sont très proches l'une de l'autre. Du point de vue de l'utilisateur, il y a une correspondance univoque entre une classe d'une arborescence et une classe de l'autre arborescence. La figure ci-dessous montre la dualité des classes qui peut exister dans NS.



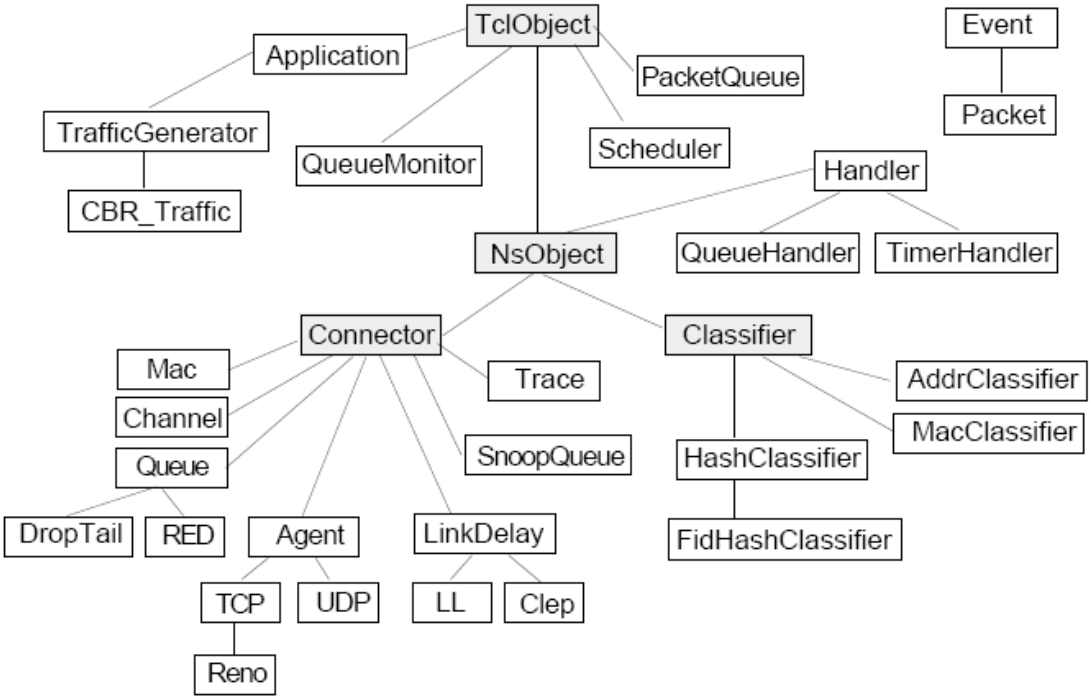
- ***Figure CH3.1:*** *Dualité des classes OTcl et C++-*

Le principe général de la création des objets du modèle de simulation est le suivant : l'utilisateur crée un nouvel objet via l'interpréteur OTcl, le nouvel objet interprété est cloné en un objet compilé correspondant dans le simulateur.

En réalité toutes les classes ne sont pas dans les deux arborescences de NS. On peut avoir des classes qui ne sont que dans l'interpréteur : elles servent à faire par exemple des assemblages (ou agrégations de classes) pour faciliter la manipulation. On peut avoir des classes qui sont purement dans le simulateur : elles ne sont pas visibles par l'utilisateur et servent au fonctionnement interne d'un composant comme par exemple certaines structures de données.

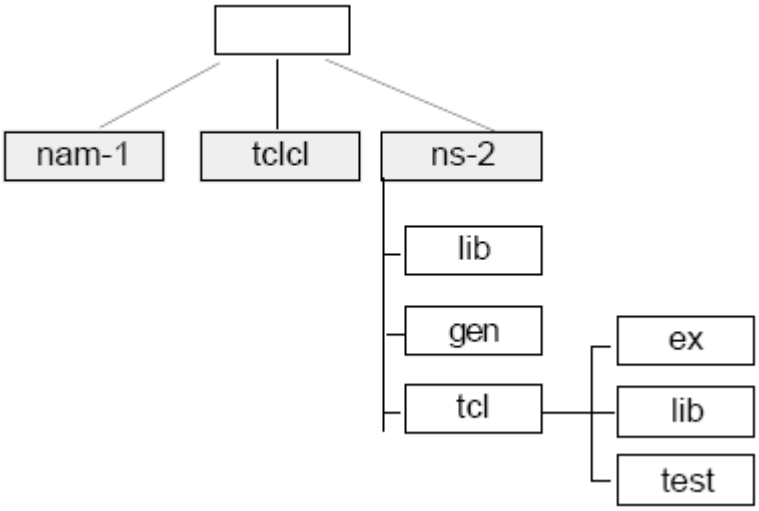
I.2.2. Arborescence des classes compilées

La figure ci-dessous représente l'arborescence de classes utilisée par le simulateur. Le nom des classes correspond à celui utilisé dans le code source.



- ***Figure ch3.2 :*** *Arborescence de dérivation des classes C++ du simulateur-*

I.2.3. Arborescence des fichiers



-Figure ch3.3 *Arborescence des fichiers de la distribution NS.-*

La distribution de NS comprend principalement 3 répertoires:

- ***ns-2 :*** l'application NS. Ce répertoire contient l'ensemble des fichiers .h et .cc de NS.

- ***nam-1*** *:* l'outil de visualisation des résultats de la simulation : l'animateur réseau.

- **tclcl** : sources du code assurant la liaison entre l'interpréteur et le simulateur. Citons l'un des principaux fichiers : tcl-object.tcl.

Dans le répertoire ns-2, on trouve les répertoires suivant :

tcl	pour tous les codes interprétés
bin	pour les utilitaires et les exécutables, pour la réalisation du binaire ns-2
lib	pour la bibliothèque de lib g++
gen	pour les sources générées lors de la réalisation du binaire ns-2 par le makefile
test_output	pour les résultats des simulations
tous les fichiers **.h** et **.cc** des classes C++ du simulateur.	

*- **Tableau** ch3.1 : les répertoires de ns-2-*

I.2.4. Liaisons entre l'interpréteur et le simulateur

La classe TclObject : est la racine de toutes les autres classes à la fois dans l'arborescence compilée et interprétée. La classe TclObject constitue donc la classe de base de la plupart des autres classes. Elle possède les interfaces nécessaires aux liaisons entre les variables qui doivent être visibles à la fois en C++ et OTcl. Elle définit la fonction *command*() qui est très utile pour ajouter des commandes à l'interpréteur.

La classe NsObject : est une sous-classe de la classe TclObject mais reste cependant une super-classe aux autres classes. La principale différence avec la classe TclObject tient à ce qu'elle est capable de recevoir des paquets et traiter des événements. Elle hérite de la classe Handler.

La classe Handler : Cette classe constitue la principale racine des objets dans NS. Elle contient les fonctions communes nécessaires au simulateur et définit la fonction virtuelle "void recv ( )".Cette fonction est une fonction générique utilisée par tous les objets dérivés de NsObject pour recevoir un paquet.

L'utilisateur crée les objets de sa simulation via l'interpréteur (les objets existeront à la fois dans l'interpréteur et dans le simulateur). La création d'un objet est en fait l'instance de deux objets qui sont chacun au même niveau dans leur arborescence de classe.

La classe TclClass sert à établir l'arborescence de classes OTcl en fonction de celle de C++. Au lancement du simulateur toutes les variables de TclClass sont créées.

Remarque :

Pourquoi faire cette copie? La réponse est l'héritage. Quand un objet est créé, il a les attributs de sa classe mais aussi ceux de ses ascendants. Les objets sont créés via l'interpréteur et sont d'abord des instances de classes OTcl. Pour que ces objets aient les mêmes attributs que leur clone dans le simulateur, nous devons avoir la même hiérarchie en OTcl et C++. La classe Trace/Enq sert à créer un objet qui écrit une trace à chaque arrivée de paquet.

I.2.4.1. Attributs :

Un attribut est une variable d'instance. NS offre 5 types d'attributs**: réel, entier, débit, temps et booléen.**

La liaison entre l'attribut des objets de l'interpréteur et du simulateur est établie par le constructeur de l'objet compilé au moyen de la fonction bind( ):

bind*("<nom variable OTcl>", &<variable C++>);*

Pour que la liaison fonctionne, il faut que l'attribut existe préalablement dans l'interpréteur. Pour cela,les attributs sont définis pour la classe avec une valeur par défaut. La plupart de ces initialisations sont effectuées dans le fichier *tcl/lib/ns-default.tcl*. La syntaxe est la suivante:

<classe> set <attribut> <valeur>

I.2.4.2. Commandes

I.2.4.2.1Classe Tcl

L'accès à l'interpréteur à partir du C++ se fait par les méthodes fournies par la classe Tcl.

Tous les objets créés dans le simulateur sont enregistrés dans une table avec comme clé l'ID attribué à l'objet lors de sa création. Cette classe comporte également les méthodes pour ajouter, supprimer et retrouver un élément dans la table.

I.2.4.2.2Méthodes Command

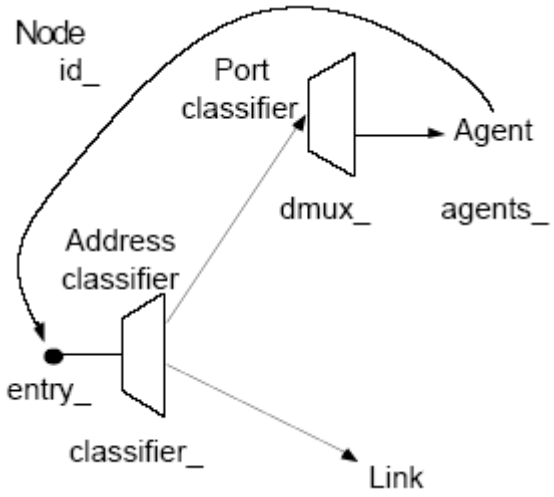
Pour chaque objet de la classe TclObject, NS fournit la méthode *cmd{}*. Cette procédure sert à exécuter des méthodes pour l'objet interprété qui sont définies pour l'objet compilé.

I.2.5. Architecture du réseau :

L'architecture réseau sous NS repose sur le modèle OSI. Présentons les classes de bases utilisables pour définir l'architecture et la topologie du modèle. Les classes Node et Link servent à la composition de la topologie du réseau. Elles modélisent les nœuds et les arcs d'un graphe.

I.2.5. 1. Le nœud :

La classe Node est une classe Otcl elle n'a donc pas d'existence en tant que telle dans le simulateur. Cette classe et ses méthodes sont définies dans le fichier *tcl/lib/ns-node.tcl*. Un noeud est une collection de classifiers et d'agents. Le classifier démultiplexe les paquets. L'agent est habituellement l'entité d'un protocole. L'assemblage des composants est représenté par la figure ci-dessous :



-Figure ch3.4 : *Composants d'un nœud-*

Les mots se terminant par le caractère "_" indique des instances variables du noeud. Ce sont pour la plupart des références à un composant du noeud. L'entrée du noeud est indiquée par la référence entry_ qui pointe sur le premier objet à l'entrée du noeud.

Le trafic émis par l'agent est transmis au point d'entrée du noeud. Que ce soit des paquets reçus de ses voisins ou émis par ses agents. Quand un noeud reçoit un paquet, il examine les champs du paquet (habituellement sa destination) et le commute vers la bonne interface de sortie. Cette tâche est exécutée par le classifier. La table des adresses du classifier est remplie par la procédure de routage *(add-routes{}).* Celle du "port multiplexer" se remplit à chaque ajout d'un agent au noeud.

Une adresse (ou *node id_)* est affectée au nœud lors de sa création. L'adresse est un entier codé sur 8 bits contenu dans la variable d'instance *id_* et aussi dans la variable *address.* L'identification d'un agent est composée par un mot de 16 bits: les 8 bits de poids forts identifient le nœud et les 8 bits de poids faible servent de numéro de port et identifient l'agent dans le nœud. Une simulation peut comporter donc 256 nœuds au maximum.

I.2.5. 1. 1. Classifier

Un classifier a pour objectif de retrouver une référence à un autre objet de la simulation à partir d'une comparaison sur un critère dont la valeur est contenue dans le paquet. Il a un rôle de démultiplexeur en quelque sorte. Le classifier sert notamment pour un nœud à modéliser la

fonction de forwarding. Le classifier contient une table de *n* slots. A chaque slot correspond la référence à un NsObject. Quand un paquet est reçu par le classifier, il identifie le slot de sortie par *classify().* Si le slot est invalide, le classifier lance *no-slot{}.* Les méthodes du classifiers sont:

- *install{},* installe une référence à un élément dans le classifier.

- *element{},* retourne la liste des éléments installés dans le classifier.

- *slot{},* retourne la référence installée à un slot spécifique.

- *clear{},* efface le contenu d'un slot spécifique.

Il existe plusieurs sortes de classifiers dans NS. Il y a ceux qui font :

- des classifications avec plusieurs critères ou des critères spécifiques comme par exemple le flow id, l'adresse de destination.
- des traitements particuliers comme le classifier replicator. Ce classifier duplique un paquet sur tous ses slots.

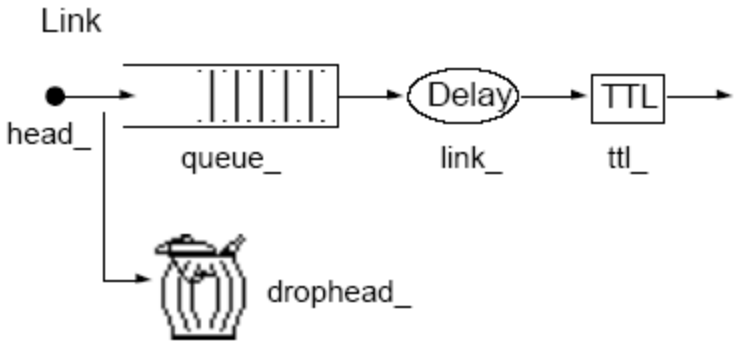
Concernant le classifier d'adresse du nœud, la table de slots est indexée par *l'id_* du nœud destination. A la réception d'un paquet et suivant l'adresse de destination, le classifier détermine quel est l'objet de simulation suivant c'est à dire à qui donner ce paquet. C'est soit au port classifier du nœud, soit à un lien de sortie.

I.2.5. 1.2. Détail sur le nœud

Quand un paquet est reçu par un nœud, ce dernier le conserve, détermine sa route et l'envoie à un autre NsObject. Il n'y a pas de durée ou de temps consommé pour ces opérations. Pour prendre en compte ces temps, il convient d'ajouter un élément de délai dans le nœud comme c'est fait pour le lien.

I.2.5. 2. Le lien :

Le lien sert à relier les nœuds. Il modélise le système de transmission. Le lien est principalement caractérisé par un délai de propagation et une bande passante. Quelque soit le type du lien, il comporte 5 références sur des objets qui le composent. Le lien peut se représenter selon la figure suivante :



- Figure ch3.6 : Composants d'un lien-

Les 5 instances variables suivantes ont la sémantique:

- ***head_ :*** point d'entrée du lien, il pointe sur le premier objet du lien.

- ***queue_ :*** référence la file d'attente de paquets.

- ***link_ :*** référence l'élément qui caractérise le lien souvent en terme de délai et bande passante.

- ***ttl_,*** : référence l'élément qui manipule la durée de vie de chaque paquet.

- ***drophead_ :*** référence l'élément qui traite les pertes de paquets au niveau de la file d'attente.

Parmi les variables du lien, une référence est gardée sur le noeud amont (*fromNode_)* et sur le nœud aval (toNode_).

- **Connector**

Tous les objets du lien sont dérivés de la classe C++ Connector. Un Connector à la différence d'un classifier, envoie les paquets à un seul récepteur. Dans le cas normal le Connector a un voisin connu sous la référence *Connector::target_*. Si il doit détruire le paquet il le passe à l'objet référencé par *Connector::drop_*. En somme, le Connector délivre le paquet à son voisin indiqué par *target_* ou *drop_*. Si *drop_* est indéfini, le paquet à jeter est supprimé de la simulation.

La variable *target_* définit l'objet auquel le paquet sera envoyé. Ainsi tous les paquets reçus par l'objet Connector une fois traités, sont passés à la *target_* du Connector.

I.2.5. 3. Agent :

L'agent est un autre composant d'un nœud. Il modélise les constructeurs et les consommateurs de paquets IP. La classe agent fournit des méthodes utiles au développement de la couche transport et à d'autres protocoles du plan de signalisation ou de gestion. Cette classe est à la fois dans l'interpréteur et dans le simulateur. C'est la classe de base pour définir des nouveaux protocoles dans NS. Elle fournit l'adresse locale et de destination, les fonctions pour générer

les paquets, l'interface à la classe Application. Actuellement NS comporte de nombreux agents citons: UDP, protocoles de routage, différentes versions de TCP, RTP, etc.

Les principales commandes sont celles qui ont trait à des attachements:

- *$ns_ attach-agent $node_ $agent_ :* attache l'agent au nœud. Concrètement cela signifie que la référence de l'agent est mis dans le port classifier pour qu'il puisse recevoir des paquets.

La variable *target_* est initialisée avec la valeur *entry_* (point d'entrée du noeud). C'est par cette commande qu'un numéro de port (variable *OTcl portID_)* est attribué et que l'adresse de l'agent est déterminée. La variable *Agent::addr_* contient cette adresse.

- *$ns_ connect $agent1 $agent2 :* établit une connexion entre les deux agents. Cette commande consiste uniquement à initialiser la variable *Agent::dst_* (adresse de destination) de chaque agent.

- $*agent_ attach-source $source_* : attache une source de trafic à un agent.

- *$source_ attach-agent $agent_* : idem

a. Les variables utiles sont :

*addr_* : adresse du noeud (ou l'agent se trouve= l'adresse source dans le paquet)

dst_ : ou je vais envoyer le paquet.

*size_* : taille du paquet en bit (placer dans l'entête commune du paquet)

*type_* : type du paquet.

fid_ : c'est l'identificateur du flot ip.

*prio_* : le champ de la priorité IP.

*flags_* : drapeau du paquet.

*defttl_* :valeur ttl par défaut.

b. Les fonctions utiles sont:

- *allocpkt()* : crée un paquet et initialise ses différents champs avec des valeurs nulles.

- *trace()* :constitue la fonction de trace (ou tracer) d'une variable d'un agent. Cette fonction est très utilisée pour les statistiques et l'étude de performance d'un algorithme.

- *send()* et *recv ()* :servent à envoyer et recevoir des paquets constitués.

La méthode *recv ()* est le point d'entrée principal pour un agent qui reçoit des paquets, elle est appelée par des nœuds désirant d'envoyer un paquet. Dans la plupart des cas, les agents n'utilisent pas le deuxième argument de la fonction *recv( ).*

- *sendmsg()* :envoie une suite binaire.

I.2.5. 4. Paquet et en-tête de paquet:

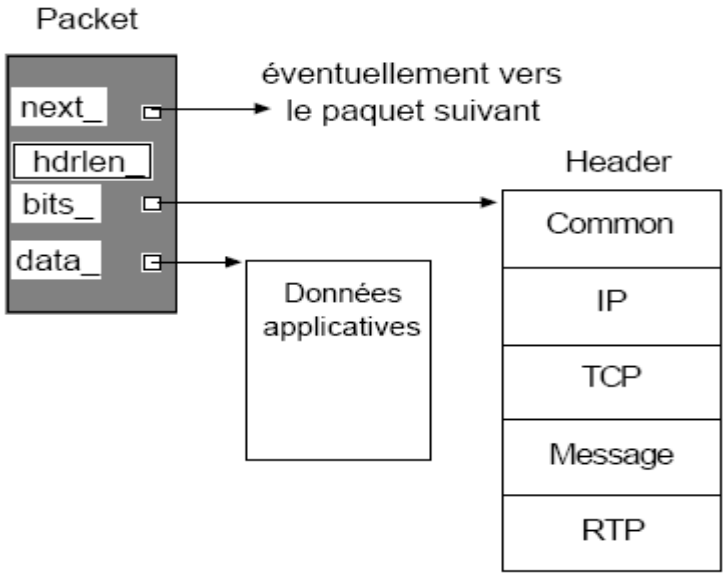
La gestion des paquets dans NS met en œuvre trois classes:

- **la classe Packet** modélise les unités de données échangées entre les composants du réseau.

- **la classe PacketHeaderClass** est dérivée de la classe TclClass et fournit les méthodes pour accéder et communiquer avec l'interpréteur. Elle sert à localiser dans le paquet l'en-tête d'un protocole particulier.

- **la classe PacketHeaderManager** sert à gérer les en-têtes de paquets qui seront disponibles dans le simulateur.

La classe Packet est dérivée de la classe Event. Seules les instances de la classe NsObject peuvent recevoir et émettre des paquets. Un paquet NS se schématise selon la figure ci-dessous et contient : (un élément liant pour pouvoir les chaîner entre eux, la longueur de la zone mémoire du paquet dédiée aux en-têtes de protocoles, une zone mémoire dédiée spécialement aux en-têtes de protocoles et une zone mémoire pour les éventuelles données d'application).



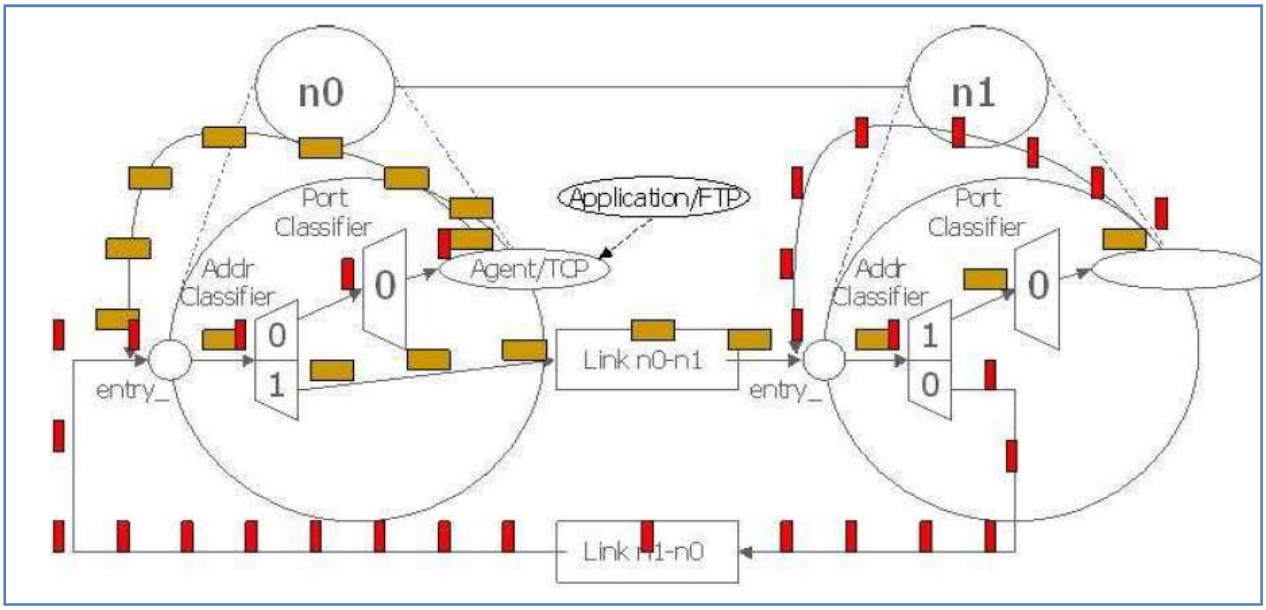
-***Figure ch3.7:*** *Le paquet NS.*-

La zone mémoire dédiée spécialement aux en-têtes de protocoles est considérée comme un tableau d'octets et a une taille suffisante pour tous les en-têtes. Cette zone est la concaténation de tous les entêtes de paquets disponibles dans le simulateur. Le contenu et la taille de cette zone sont définis à l'initialisation de l'objet Simulator. Sa taille est conservée dans la variable *Packet::hdrlen_*. L'entête d'un protocole particulier est accédé par une adresse relative dans la zone mémoire à partir de l'adresse indiquée par *bits_*.

IL est à noter que l'en-tête *"common*" doit toujours être présent. Elle sert à la fonction de trace et à d'autres fonctions indispensables à la simulation. Les principaux champs de l'en-tête

"common" sont : (***size_* :** taille simulée du paquet ; ***ptype_ :*** type du paquet ; ***ts_* :** timestamp ; ***ref_count_*** *:* durée de vie en nombre de sauts).

La classe PacketHeaderClass contient 2 variables privées *(* ***hdrlen_* :** taille de la structure de l'en-tête ; ***offset_*** *:* un pointeur sur la variable static *offset_* de l'en-tête du paquet. Le constructeur initialise ce pointeur par l'appel à offset).



- Figure ch3.8*: *représentation d'un flot de paquet-

I.2.6. Eléments de la simulation

I.2.6.1. Simulateur

La simulation est configurée, contrôlée et exploitée via l'interface fournie par la classe OTcl Simulator. Elle n'existe que dans l'interpréteur. Un script de simulation commence toujours par créer une instance de cette classe par la commande ***set ns_ [new Simulator]***

L'objet Simulator (ici représenté par la référence *ns_)* contiendra une référence sur chaque élément de la topologie du réseau simulé et sur les options mises pour la simulation comme par exemple les noms des fichiers de trace. La topologie du réseau est construite avec des instances des classes OTcl, Node et Link. Les méthodes de la classe Simulator sont définies dans le fichier *tcl/lib/ns-lib.tcl.* L'initialisation du Simulator par la méthode *init{}* initialise le format des paquets de la simulation et crée un ordonnanceur.

I.2.6.2. Ordonnanceur : L'ordonnanceur est défini dans le fichier *scheduler.{h,cc}.* L'ordonnanceur a en charge de choisir l'événement le plus proche en terme de temps, d'exécuter les traitements, de faire progresser le temps simulé et d'avancer à l'événement suivant etc. Les événements sont rangés dans un échéancier. Un seul événement est traité à la fois. Si plusieurs événements doivent être traités au même instant. Ils sont exécutés en série

mais au même instant en termes de temps simulé. Le temps simulé est l'échelle de temps du modèle de simulation. Le fonctionnement par défaut de l'ordonnanceur est "Calendar Scheduler". D'autres méthodes de gestion existent..

Les principales commandes sont: (***now{ }* :** retourne le temps simulé ; ***at* <*time*> *"<action>",* *at{ }* :** ajoute un événement dans l'échancier ; ***run{ }* :** lancement de la simulation ; ***halt{ } :*** arrêt de l'ordonnanceur ; ***cancel $eventID*** : annule un événement préalablement placé avec "at". EventID est la valeur retournée lors du placement de l'événement par set eventID [$ns_ at <time> "<action>"] L'événement est défini par la classe Event et se caractérise par l'heure de déclenchement et par l'événement à réaliser (handler): c'est l'objet qui va consommer l'événement.

La classe NsObject se dérive en premier en deux sous-classes: (connector ;classifier).

La différence se situe sur la capacité de connexion. A la réception d'un paquet, un connector fait suivre le paquet au NsObject référencé par *target_*, le classifier doit déterminer à quel NsObject est destiné ce paquet. Il utilise une table *slots_[]* dont chaque slot contient une référence à un NsObject. Le connector sert à assembler les objets en relation 1-1 tandis que le classifier établit des relations 1-N entre les objets.

I.2.6. 3. Consommation de temps

Aucun objet dans la simulation ne peut faire avancer le temps. Pour consommer du temps, il faut obligatoirement passer par l'ordonnanceur.

Avec scheduler*::schedule() ; void Scheduler::schedule(Handler*, Event*, double delay);*

Mais avant de pouvoir faire appel à cette fonction, il faut obtenir la référence sur le simulateur par l'appel à la fonction *Scheduler::instance(). scheduler::schedule()* consiste à insérer un événement dans l'échéancier de la simulation avec une heure de déclenchement. L'heure de déclenchement correspond au temps simulé actuel plus le temps à consommer (exprimé par le paramètre delay). Lorsque le temps simulé arrive à l'heure de déclenchement de l'événement, l'objet référencé par la variable de type Handler est activé. Cette méthode est également valable pour passer un paquet à un voisin en un temps non nul. Quelque soit l'utilisation de *schedule(),* le paramètre Handler doit avoir une valeur non nulle.

I.2.6.4. Traitement séquentiel en temps simulé

Le temps simulé est découplé du temps réel. Si aucun objet ne fait de consommation de temps, vis à vis du temps simulé tous les traitements se font en même temps (mais par rapport au temps réél ils sont exécutés en série). Un simulateur est naturellement une machine pseudo-parallèle. Pour modéliser un objet à traitement séquentiel, il est nécessaire d'adjoindre un mécanisme de blocage qui empêche l'objet de commencer un nouveau traitement avant d'avoir fini le précédent (en termes de temps simulé).

Exemple de file d'attente :

Pour illustrer ce mécanisme, prenons, l'exemple du modèle classique de file d'attente: une file d'attente et un serveur. Tant que le serveur est occupé en transmission de paquet, aucune autre transmission ne peut avoir lieu. La durée de la transmission dépend à la fois de la longueur du paquet et du débit du lien. Le serveur effectue la transmission du paquet selon un mode séquentiel. Le composant modélisant un serveur a en charge de:

- Empêcher de sortir des paquets de la file tant qu'il y en a un en transmission. La transmission dure Tt Temps de transmission.

- Autoriser une nouvelle transmission à la terminaison de la transmission du précédent paquet

(t0 + Tt)

- Activer l'objet récepteur d'un paquet à l'instant de réception Tt+Tp. On pose Tp Temps de propagation.La classe *Queue* et *LinkDelay* représente respectivement la file d'attente et le serveur. Dans le lien, *LinkDelay* est l'élément de délai qui se situe en aval de la queue. Une queue est bloquée jusqu'à ce qu'elle soit re-activée par son voisin aval (ici délai). C'est avec ce mécanisme que le délai de transmission est simulé.

I.2.6.5. Temporisateur

Il existe deux mécanismes de temporisateurs dans NS:

- un pour l'ordonnancement des événements générés par l'interpréteur. Il est défini dans le fichier *tcl/ex/timer.tcl.*
- un second défini en C++ pour les composants du simulateur.

Les temporisateurs C++ sont dérivés de la classe *TimerHandler*

Exemple : un temporisateur de détection d'inactivité pour un agent TCP Le temporisateur Tcl sert à appeler des méthodes OTcl de l'interpréteur durant la simulation. Il est réalisé par *at{}* de Simulator.

I.2.7. Interprétation

NS fournit deux moyens pour extraire des données de la simulation:

- La ***trace*** enregistre dans un fichier les changements d'états d'un paquet ou de valeur d'une variable.
- Le ***moniteur*** est un objet pouvant faire des calculs sur différentes grandeurs tel que le nombre de paquets ou d'octets arrivés, etc.

I.2.7.1. Trace

Il existe deux types de traces: les traces effectuées à partir d'une file d'attente d'un lien et les traces de variables. Le code source des fonctions de trace sont respectivement dans *trace.cc* et *tclcl/tracedvar.cc*. Le fichier de traces est un fichier de texte structuré en lignes.

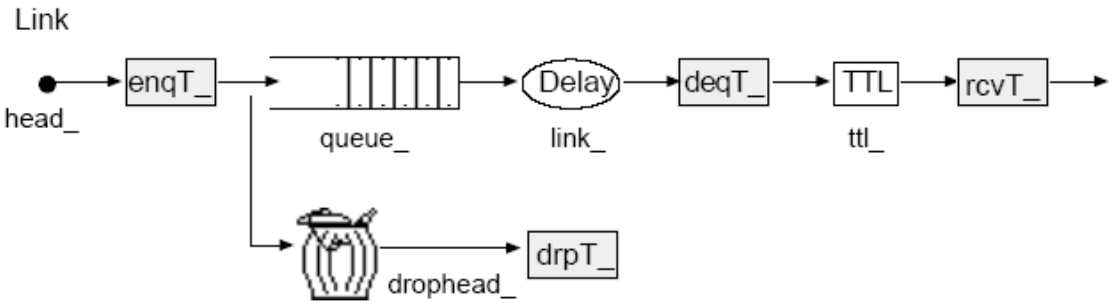
a. Queue : Chaque type d'événement pouvant faire l'objet d'un enregistrement se définit par une sous-classe de Trace. La classe Trace consiste à la réception d'un paquet à effectuer l'écriture des principales caractéristiques du paquet dans le fichier et à passer le paquet à l'objet aval. Une ligne de trace commence par une lettre indiquant le type de trace afin de différencier chaque sous-classe de Trace:

lettre	**Type de trace**
+	mise en file d'attente
-	sortie de la file d'attente
d	suppression de la file d'attente
r	réception au nœud
l	perte (suite à une erreur binaire)

*-**Table ch3.2:** type de trace correspondant à chaque lettre-*

Pour effectuer la trace d'un lien, il faut établir la configuration de lien représentée par la figure ci-dessous. 4 objets de trace sont ajoutés et sont présentés par leur référence dans la figure:

(***enqT_*** : pour l'arrivée des paquets ; ***deqT_* :** pour le départ des paquets de la file d'attente ; ***drpT_* :** pour la perte des paquets due à la congestion de la file d'attente ; ***rcvT_* :** pour la réception des paquets au nœud suivant.)



*-**Figurech3.9 :** Composition du lien avec les objets de tracage.-*

Une ligne de trace se présente sous le format suivant:

opération	temps	nœud source	Nœud Destination	type de paquet	taille du paquet	Dra-peaux	flot id

suite

adresse source du paquet	adresse destination du paquet	numéro du paquet	id du paquet

***Tableau ch3.3 :** Le format d'une ligne de trace-*

Exemple d'un fichier trace :

Action	Temps	Nœuds		Paquet	Taille	Flag	ID flux	Adresses		N°= de séq.	uid
		Source	Dest.					Source	Dest.		
+	1.84375	0	2	cbr	210	---	0	0.0	3.1	225	610
-	1.84375	0	2	cbr	210	---	0	0.0	3.1	225	610

b. Variable

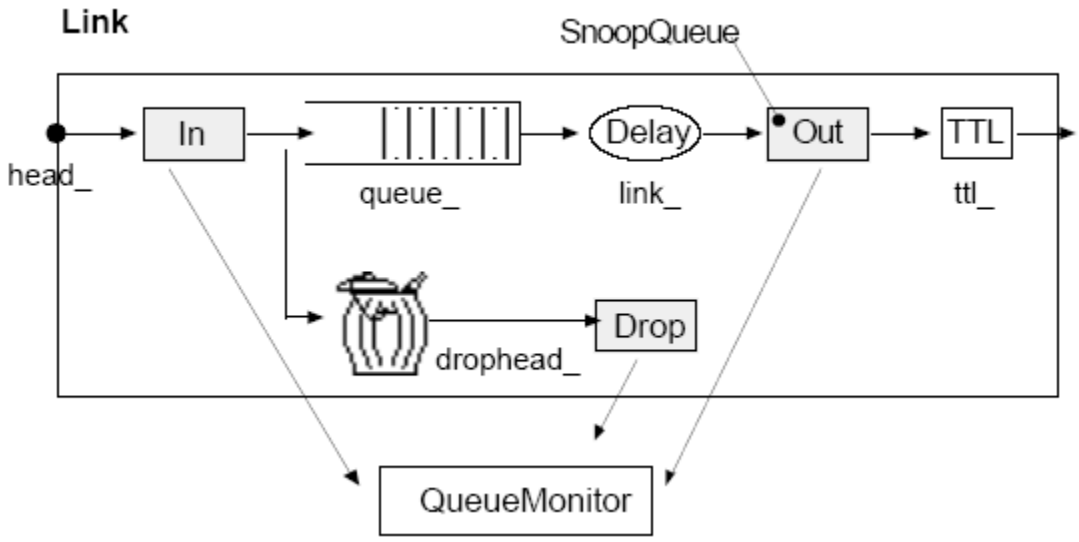
Les variables doubles ou int peuvent être tracées. C'est à dire que chaque fois que leur valeur

Change, une trace est écrite dans le fichier de traçage.

Dans NS, les variables pouvant être tracées sont dérivées de la classe de base *TracedVar*. Deux classes sont fournies: *TracedInt* et *TracedDouble*

I.2.7.2. Moniteur

Un moniteur utilise des objets observateurs dits sniffeurs (snoop) qui sont insérés dans la topologie réseau. Le rôle de ces petits sniffeurs est de faire remonter les informations d'états du réseau au moniteur. Celui-ci est un point de ralliement de ces informations, où des calculs sont effectués. La figure ci-dessous montre l'assemblage des composants du lien lorsqu'il y a un *QueueMonitor*. La classe *QueueMonitor* est définie dans le fichier *queue-monitor.cc*. Le moniteur est un composant qui effectue des calculs relatifs au lien.



- ***Figure ch3.10:*** *Installation d'un moniteur sur un lien.-*

Un moniteur peut servir à collecter des statistiques sur l'ensemble des flots ou par flot. Dans ce dernier cas, le moniteur est de la classe *FlowMon* et est défini dans le fichier *flowmon.cc*. Les statistiques collectées sont par exemple: (***parrivals_*** : mesure combien de paquets ont été reçus par la file ; ***barrivals_*** : mesure combien d'octets ont été reçus par la file.)

I.2.8. Routage:

Une fois les différents éléments topologiques créés, il reste encore l'assemblage des liens avec les nœuds et la constitution de la table de routage pour chaque nœud. Cette tâche est du ressort du routage. Le routage par défaut dans NS consiste en un routage *unicast statique* c'est-à-dire déterminé avant le démarrage de la simulation. Le routage utilise l'algorithme *SPF* de *Dijkstra.* Un lien coûte par défaut 1. Les fonctions et procédures du routage unicast statique peuvent être trouvées dans les fichiers *tcl/lib/ns-route.tcl*, *tcl/rtglib/route-proto.tcl* et *route.cc*. Le calcul de route est effectué dans la classe *RouteLogic*. Une instance de cette classe est créée au lancement de la simulation par la fonction *run{}* de simulator. Pour configurer la table de routage manuellement, un exemple est proposé dans *tcl/ex/many_tcp.tcl.*

I.2.8. Autres composants et fonctionnalités

I.2.8.1. Agent/Loss

Cet agent est un puits de trafic. Il sert à comptabiliser la réception des paquets. En surveillant les numéros de séquence des paquets, il détecte les pertes. Il comporte les compteurs suivants:

- ***nlost_ :*** nombre de paquets perdus

- ***npkts_ :*** nombre de paquets reçus

- ***nbytes_ :*** nombre d'octets reçus

- ***lastPktTime_ :*** heure du dernier paquet arrivé

- ***expected_ :*** numéro du prochain paquet attendu

I.2.8.2. Agent/Message

L'agent Message est très simple. Il permet de véhiculer des messages de taille maximum de 64 octets dans des paquets.

I.2.8.3. Applications

La classe Application modélise l'application en terme de source de trafic. L'application est associée avec un agent qui correspond à l'entité de transport. L'interface entre l'agent et l'application reprend celle des sockets (*cf agent.h*). Les applications sont de deux types:

- sources, elles sont employées à générer un flux pour un transport TCP. On y trouve *Telnet* et *Ftp*.

- générateur de trafic, ils sont employés pour les transports en mode non connecté comme *UDP*. On y trouve *CBR*, Exponentiel, Pareto. Le trafic peut également être généré à partir d'un fichier de traces contenant l'heure et la longueur des données à générer.

I.2.9. Ajout d'éléments dans NS

a. L'ajout d'un nouveau protocole :

Lorsque ce protocole est au-dessus de IP ou utilise IP, il est codé sous forme d'un agent (classe dérivée de la classe agent). Si ce n'est pas le cas, il doit être codé comme une classe dérivée de la racine (classe NsObject).

b. Les étapes de l'ajout d'un protocole dans NS sont:

- déclaration de l'en-tête du paquet du protocole (ou unité de données du protocole)
- déclaration de la classe du protocole (méthodes et attributs du protocole). Le nom des attributs est suffixé par le caractère "_".
- définition de la liaison entre le code C++ et le code OTcl par la déclaration des variables statiques dérivées de la TclClass et de PacketHeaderClass respectivement pour l'agent et pour le paquet (ou unité de données) du protocole.
- ajout du protocole ID dans packet.h par un #define. L'indicateur de fin de liste des protocoles ID est *PT_NTYPE*. Le protocole ID sert à identifier le type de paquet.
- ajout du nom du protocole dans *PT_NAMES* (*packet.h*). L'ajout doit se faire à la position indiquée par la valeur du protocole ID définie précédemment. Pour les traces, la liste sert à remplacer l'ID du protocole par une chaîne de caractères.
- ajout du paquet dans la liste gérée par le packet manager. Ceci est rendu nécessaire pour le calcul de l'offset de l'en-tête du paquet.
- enfin modification du *makefile.in* par l'ajout du fichier objet du protocole à la liste des fichiers objet de ns.
- recompilation de ns

c. Conventions

nom{} signifie une fonction ou méthode OTcl. Elle se nomme dans ce langage des "instance procédure".

nom() signifie une fonction ou méthode C++.

"_" Dans le code, ce caractère est appliqué comme suffixe au nom des variables; il

indique une variable d'instance.

II. Notions pour l'utilisation de l'interpréteur

II.1. Tcl

a. Présentation

Tcl est un langage de commande comme le shell UNIX mais qui sert à contrôler les applications. Son nom signifie *Tool Command Language*. Tcl offre des structures de programmation telles que les boucles, les procédures ou les notions de variables. En pratique, l'interpréteur Tcl se présente sous la forme d'une bibliothèque de procédures C qui peut être facilement incorporée dans une application. Cette application peut alors utiliser les fonctions standards du langage Tcl mais également ajouter des commandes à l'interpréteur.

b. Lancement

Toutes les applications qui utilisent Tcl créent et utilisent un interpréteur Tcl. Cet interpréteur est le point d'entrée standard de la bibliothèque. L'application *tclsh* constitue une application minimale ayant pour but de familiariser un utilisateur au langage Tcl. Elle ne comporte que l'interpréteur Tcl. On retrouve cet interpréteur dans l'application NS. Lorsque l'on tape ns, une série d'initialisations est faite puis l'application passe en mode interactif. On peut alors entrer les commandes Tcl.

c. Concepts de base

Chaque commande consiste en un ou plusieurs mots séparés par des espaces ou des tabulations. Tous les mots sont des chaînes de caractères. Le premier mot de la commande est le nom de la commande, les autres mots sont les arguments passés à la commande. Chaque commande Tcl retourne le résultat sous forme d'une chaîne de caractères. Le caractère de retour à la ligne termine une commande et lance son interprétation. Le caractère de séparation de commandes sur une même ligne est ";". Une fois la commande exécutée et terminée, l'application retourne en mode interactif c'est à dire que l'interpréteur est de nouveau prêt à recevoir une nouvelle commande.

Tcl n'est pas un langage compilé, c'est un langage interprété. Il n'y a pas de grammaire fixe qui traite l'ensemble du langage. Tcl est défini par un interpréteur qui identifie les commandes par des règles d'analyse syntaxique. Seules les règles d'analyse des commandes sont fixées. Tcl évalue une commande en deux étapes: analyse syntaxique et exécution.

L'analyse syntaxique consiste à identifier les mots et effectuer les substitutions. Durant cette étape, l'interpréteur ne fait que des manipulations de chaînes. Il ne traite pas la signification des mots.

Pendant la phase d'exécution, l'aspect sémantique des mots est traité comme par exemple déduire du premier mot le nom de la commande, vérifier si la commande existe et appeler la procédure de cette commande avec les arguments.

d. Substitutions

- Substitution de variable : *$<variable>*

Le contenu d'une variable est obtenu en faisant précéder le nom de variable par le symbole $

Exemple : *>set a 20* ; *>expr $a*2* ; *< 4O*

- Substitution de commande : *[<commande>]*

La substitution de commande permet de remplacer la commande par son résultat. Les caractères entre les crochets doivent constituer un script Tcl valide. Le script peut contenir un nombre quelconque de commandes séparées par des retours à la ligne et des ";".

- Anti-slash substitution: "\"

Il est utilisé devant les caractères retour à la ligne, $, pour indiquer qu'il faut les interpréter comme des caractères ordinaires.

e. Inhibitions

Il existe plusieurs façons d'empêcher l'analyseur syntaxique de donner une interprétation spéciale aux caractères tel que $ ou ;. Ces techniques se nomment inhibitions. Tcl fournit deux formes d'inhibitions:

- double cotes (") inhibe les séparations de mots et de commandes. Il reste les substitutions de variable et de commande. Les caractères espaces, tabs, retour à la ligne et point-virgule sont traités comme des caractères ordinaires.

Exemple : *>set msg " Aujourd'hui: $date " **;** >set msg <Aujourd'hui: 22 Décembre*

- accolade inhibe toutes les substitutions et tous les caractères spéciaux. Les caractères compris entre les crochets sont considérés comme un mot.

>set msg {Aujourd'hui: $date} ; >set msg ; <Aujourd'hui: $date

II.2. OTcl

OTcl est une extension orientée objet de Tcl. Les commandes Tcl sont appelées pour un objet. En OTcl, les classes sont également des objets avec des possibilités d'héritage. Les analogies et les différences avec C++ sont:

- C++ a une unique déclaration de classe. En OTcl, les méthodes sont attachées à un objet ou à une classe.
- Les méthodes OTcl sont toujours appelées avec l'objet en préfixe
- L'équivalent du constructeur et destructeur C++ en OTcl sont les méthodes *init{}/destroy{}*
- L'identification de l'objet lui-même: *this (C++), $self (OTcl). $self* s'utilise à l'intérieur d'une méthode pour référencer l'objet lui-même. A la différence de C++, il faut toujours utiliser $self pour appeler une autre méthode sur le même objet. C'est à dire *"$self xyz 5" serait "this->xyz(5)"* ou juste *"xyz(5)"* en C++.

- Les méthodes OTcl sont tout le temps "virtual". A savoir, la détermination de la méthode à appeler est effectuée à l'exécution.
- En C++ les méthodes non définies dans la classe sont appelées explicitement avec l'opérateur de portée "::". En OTcl, les méthodes sont implicitement appelées dans l'arbre d'héritage par *"$self next"*. Cette commande appelle la méthode de même nom de la classe parent.
- L'héritage multiple est possible dans les deux langages

II.3. Liens C++ et Tcl :

L'interpréteur effectue l'analyse syntaxique et appelle la fonction C correspondant à la commande Tcl. Ajouter une commande Tcl consiste à établir un lien entre un mot et une fonction C. Le mot sera le nom de la commande Tcl. La fonction C est définie dans le code source de l'application.

III. Architecture et topologie d'un réseau ad hoc sous NS :

Nous avons déjà parlé de l'architecture des réseaux sous NS, mais pour les réseaux ad hoc, la notion de liens n'existe pas et la topologie est définie uniquement par un ensemble de nœuds mobiles.

a. Nœud mobile :

Un nœud mobile est un nœud avec des fonctionnalités supplémentaires comme le mouvement, capacité de transmettre et recevoir sur un canal cela lui permet d'être employé pour créer les environnements mobiles et sans fil de simulation. La classe *MobileNode* est dérivée de la classe base *node*. Les dispositifs de mobilité comprenant le mouvement de nœud, mises à jour périodiques de position, maintien des frontières de la topologie sont implémenté en C++, tandis que les composants du nœud mobile eux-mêmes (comme classifier, dmux, LL, imper, Manche etc..) sont implémenté en Otcl.

b. Donner du mouvement aux nœuds mobiles :

- Indiquer le point d'origine, la destination et la vitesse explicitement pour chaque nœud mobile. Les mises à jour sont déclenchées chaque fois que l'on exige la position du nœud mobile à un moment donné. Cette solution plutôt faite pour des petites simulations.
- Générer des mouvements aléatoires : à l'appel d'une procédure, le nœud mobile démarre à partir d'une position aléatoire et exécute des déplacements. Le nœud mobile exécute des mises à jour de routage pour changer de destination et de vitesse.
- Indépendamment des méthodes utilisées pour générer les mouvements des nœuds mobiles, il faut définir une topographie ; L'espace est considéré comme étant une grille dont il faut donner les frontières (valeurs de x abscisse et y ordonnée).

option	valeur	signification	Valeur Par défaut
addressType	plate, hierarchique	Type d'adresse	flat
liType	LL	Type couche liaison	""
macType	Mac/802_11	Type de la sous couche mac	""
ifqType	Queue/DropTail, Queue/DropTail/PriQueue	Type de la file d'attente	""
phyType	Phy/WirelessPhy,	Type de l'interface réseau	""
adhocRouting	DSR, AODV, TORA, DSDV,	Protocole de routage adopté	""
propType	Propagation/TwoRayGround, Propagation/Shadowing		""
propInstance	Propagation/TwoRayGround, Propagation/Shadowing	Modèle de propagation	""
antType	Antenna/OmniAntenna	Type de l'antenne	""
channel	Channel/WirelessChannel, Channel/Sat	Type du canal	""
topoInstance	<topology file>	La topologie	""
mobileIP	ON, OFF		off
energyModel	EnergyModel	Modèle d'énergie	""
initialEnergy	<value in Joules>	énergie initiale	""
rxPower	<value in W>	Énergie pour la réception	""
txPower	<value in W>	Énergie pour la transmission	""
idlePower	<value in W>		""
agentTrace	ON, OFF	Indique le niveau de récolte d'une trace pour le fichier trace	off
routerTrace	On, off		off
macTrace	On, off		off
movementTrace	On, off		off
errProc	UniformErrorProc		""
FECProc	?		?
toraDebug	On, off	Modèle d'erreur	off

-tableau ch3.4: *Ensemble de caractéristiques d'un nœud mobile-*

c. Les composants d'un nœud mobile :

Les composant d'un nœud mobile sont :(*link layer=LL ; ARP=adress resolution protocol* ; *interface priority queue=IFq ; mac layer=MAC* ; *network interface=netIF*)

- **La couche liaison de données (LL) :**

Elle est similaire à celle utilisée par un nœud ordinaire (dans les réseaux filaires), la seule différence est quelle possède un module ARP. Chaque paquet sortant vers le canal est transmis de l'agent de routage à la couche liaison. Celle-ci le transmet vers l'interface de la file. Chaque paquet entrant (du canal) est transmis par la couche MAC à la couche LL, qui le transmis ensuite au point d'entrée du nœud .elle est implémenté dans *~ns/ll.{cc,h}* et *~ns/tcl/lan/ns-ll.tcl.*

- **ARP (Adress Resolution Protocol) :**

Il reçoit des paquets de la couche liaison et il résout toutes les conversions d'adresses IP en adresses physiques (MAC). Il traite des demandes de l'objet *LinkLayer* en utilisant une table de correspondance entre l'adresse IP et l'adresse MAC.si l'ARP a l'adresse physique de la destination, il l'écrit dans l'en-tête MAC du paquet. Si non, il diffuse une requête ARP et stocke temporairement le paquet dans le buffer (les paquets pour lesquels il ne connaît pas de correspondance). Une fois l'adresse physique du prochain nœud du paquet trouvée, le paquet est inséré dans l'interface de la file.

- **L'interface de la file :**

Elle fonctionne suivant un algorithme de gestion de file. Le plus utilisé des algorithmes est PriQueue qui donne la priorité aux paquets de routage.

- **La couche MAC :**

Pour les nœuds la seule norme implémentée est la norme IEEE 802.11.

- **l'interface réseaux** :

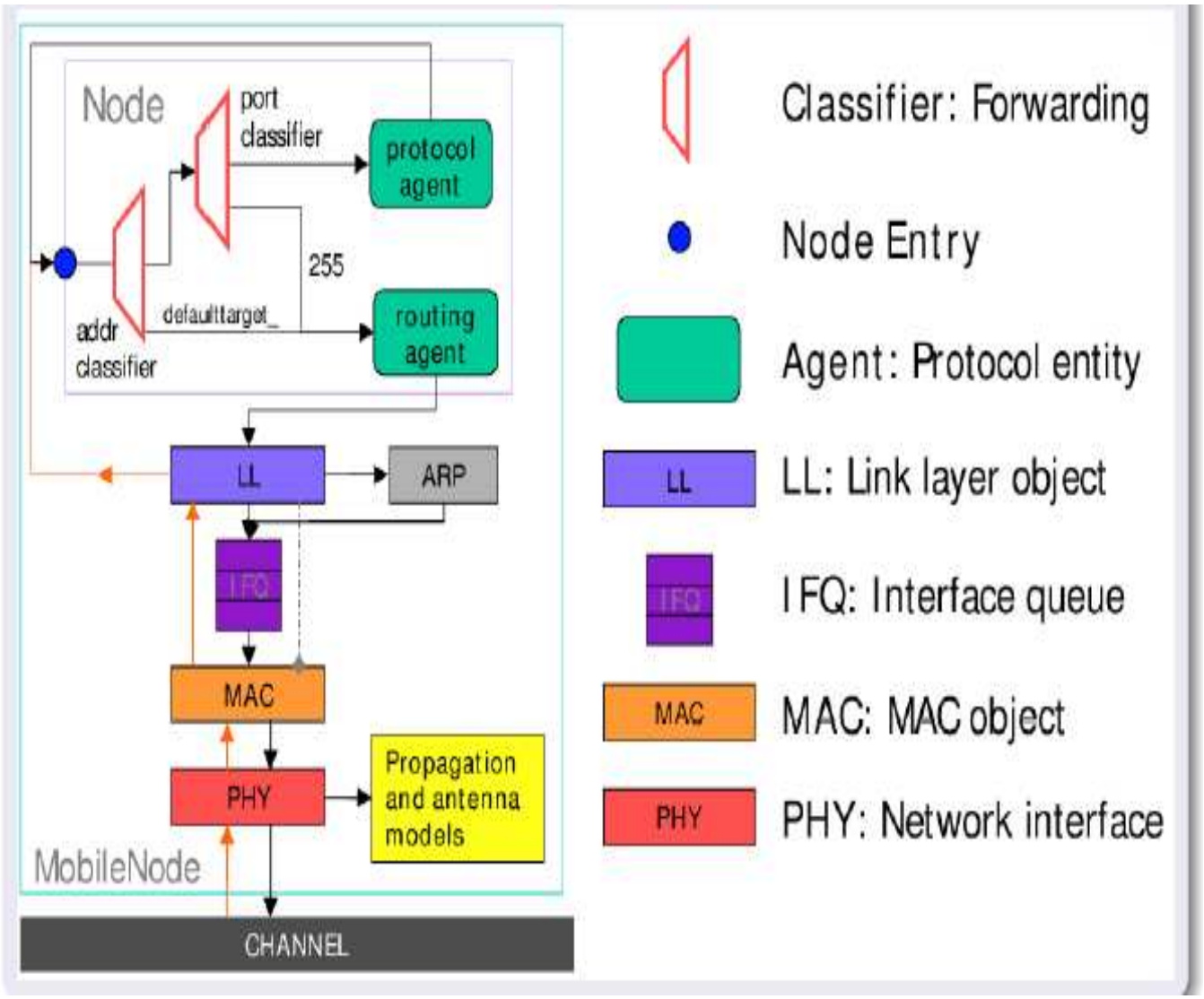
L'objectif est de simuler l'interface matérielle qu'un nœud mobile utilise pour accéder au canal. Ces interfaces sont implémentées dans les fichiers *wirelessPhy.{cc,h}.* Ces interfaces sont soumises aux collisions et enregistrent l'intensité du signal, la longueur d'onde…

- **Le modèle de propagation radio :**

Pour les courtes distances c'est le modèle d'atténuation *'Friss-space'* qui est utilisé, pour les grandes distances c'est le *'TwoRayGround'*.

- **L'antenne :**

La seule antenne implémentée pour les réseaux sans fils est l'antenne omnidirectionnelle.



*-**Figure**ch3.11 : nœud mobile dans NS-*

IV.1. Script de simulation [R16]

Le script de simulation consiste à indiquer la topologie du réseau, à activer des traces aux endroits pertinents, à engendrer des événements particuliers à des instants donnés. De nombreux exemples sont disponibles dans la distribution, citons les plus simples:

example.tcl ; simple.tcl ; test-tbf.tcl ; vlantest-flat.tcl.

On peut avoir également recours à des générateurs de scripts. Le générateur crée et exécute une simulation pour des spécifications données d'agent, de topologie et de routage.

Nous allons écrire un programme simple. Ou toutes les commandes les plus utiles vont apparaître (création d'un simulateur, création d'un nœud, création d'un agent)

Il faut instancier un objet *Simulator* qui permettra de créer et gérer tout autre objet. Sa création est assurée par :

```
set ns_ [new Simulator]
```

Ensuite, il faut définir la topologie, c'est-à-dire les nœuds et les liens. Voici la description de deux nœuds et d'un lien les reliant :

```
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

Le lien ainsi crée est un lien « duplex » (communication en double sens), avec une bande passante de 10Mb, un délai de 10ms et une file d'attente « DropTail ». Pour créer une plus grande topologie, on peut créer les nœuds avec une boucle itérative. Chaque nœud sera alors stocké dans un tableau accessible par un indice. Le code correspondant à la création d'un tableau « n » de sept nœuds est le suivant

```
for {set i 0} {$i < 7} {incr i} {
      set n($i) [$ns node]
}
```

Ensuite il faut définir les transferts possibles entre chaque nœud. Pour cela, il faut attacher des agents aux différents nœuds et les connecter ensemble. Voici la création d'un agent CBR :

```
set cbr0 [new Agent/CBR]
$ns_ attach-agent $n0 $cbr0
```

On peut également régler les paramètres des paquets envoyés (taille et période) :

```
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
```

De l'autre côté, il faut créer un agent récepteur sur l'autre nœud et enfin les connecter :

```
set null0 [new Agent/Null]
$ns_ attach-agent $n1 $null0
$ns_ connect $cbr0 $null0
```

Pour voir les effets des tests sur les protocoles de transport, on peut attacher un agent UDP ou TCP à un nœud générateur de trafic. Voici la mise en place d'un trafic CBR par un agent UDP :

```
set udp0 [new Agent/UDP]
$ns_ attach-agent $n0 $udp0
set cbr0 [new Agent/Traffic/CBR
$cbr0 attach-agent $udp0
$udp0 set packetSize_ 536
```

Puis en admettant qu'un agent « Null » null0 a déjà été attaché au nœud 1, on peut connecter les agents :

```
set null0 [new Agent/Null]
$ns_ attach-agent $n1 $null0
$ns_ connect $udp0 $null0
```

Ensuite on définit les évènements du scénario, c'est-à-dire quand le trafic commence, se termine :

```
$ns_ at 0.5 "$cbr0 start"
$ns_ at 4.5 "$cbr0 stop"
```

Enfin il faut lancer la simulation par la commande :

```
$ns_ run
```

Bien entendu toutes ces commandes constituent la base des simulations. Pour simuler des ruptures de liens pour voir comment les protocoles réagissent à une panne, on peut indiquer au simulateur quand « détruire » une liaison et quand la remettre en place. On peut alors mettre un protocole de routage dynamique en place :

```
$ns_ rtmodel-at 1.0 down $(n1) $n(2)
$ns_ rtmodel-at 2.0 up $(n1) $n(2)
$ns_ rtproto DV
```

Ces quelques lignes indiquent que le lien entre les nœuds 1 et 2 ne seront pas utilisables du temps 1.0 au temps 2.0. Le protocole de routage qui se chargera de trouver une nouvelle route est celui à Vecteur de Distance.

La simulation de la mobilité dans NS-2 passe par la déclaration de nœuds mobiles. Celle-ci se passe comme pour les autres nœuds en précisant toutefois la configuration des paramètres des nœuds mobiles. L'attache des agents pour simuler un protocole de transport ou une application est identique. Cependant, on peut définir la position initiale des nœuds mobiles ainsi que leur mouvement de manière très précise :

```
set topo [new Topography]
$topo load_flatgrid 500 500
```

Pour définir la grille, frontière de la simulation, puis :

```
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$ns_ at 10.0 "$node_(0) setdest 20.0 18.0 13.0"
```

Les deux premières lignes indiquent la position initiale du mobile dans la grille. La dernière ligne signifie que le nœud 0 au temps 10.0 va se déplacer en direction de la position (x=20, y=18) sur la grille à une vitesse de 13 m/s. Dans un souci de clarté des scripts, surtout pour les grandes simulations, les spécifications de mouvement ou les définitions des agents de trafic peuvent se faire dans des fichiers séparés.

Pour visualiser une animation ou simplement enregistrer tous les évènements dans un fichier il faut le spécifier au début du script. Les commandes utilisées pour créer un fichier de sortie sont :

```
set tracefd [open sortie.tr w]
$ns_ trace-all $tracefd
```

Et pour envoyer les évènements à NAM :

```
set tracenam [open out.nam w]
$ns_ namtrace-all $tracenam
```

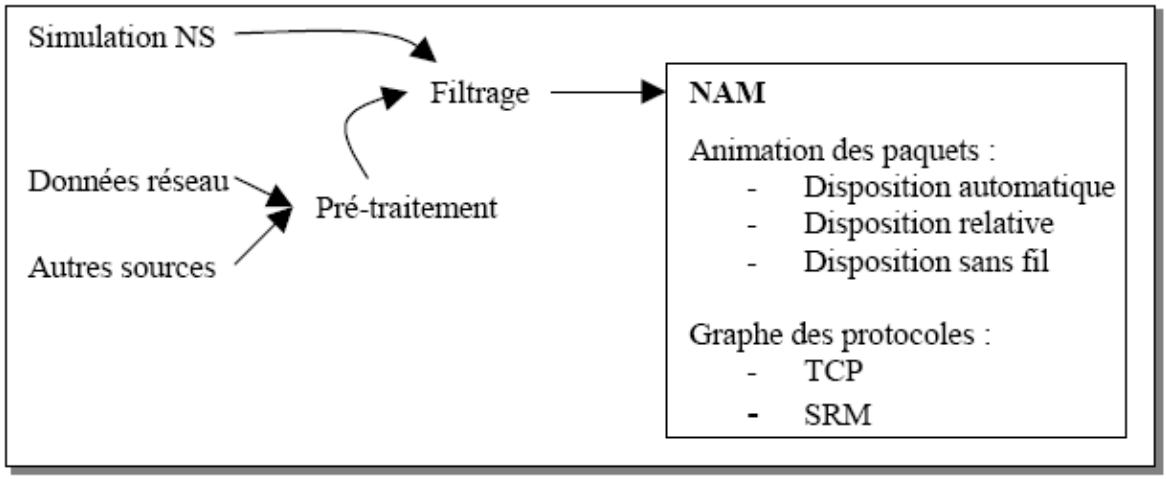
IV.2. NAM[R16]

Les traces ont deux inconvénients majeurs : elles présentent un nombre important de détails, ce qui peut compliquer la compréhension des données, et elles sont statiques, ce qui cache une dimension importante du comportement des protocoles. Les outils de visualisation adressent ce problème en permettant à l'utilisateur de prendre en considération plusieurs informations très rapidement, d'identifier visuellement les modèles de communication et de mieux comprendre les interactions et les causalités.

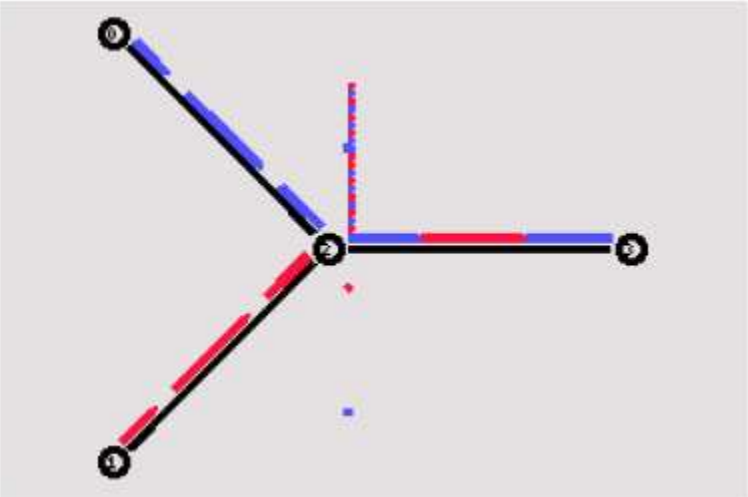
NAM est un outil d'animation basé sur *Tcl/TK* pour l'observation des traces de paquet. Les données utilisées par NAM peuvent provenir d'un simulateur ou de tests sur des réseaux réels (par exemple : utilisation de *tcpdump*). Il supporte l'affichage de la topologie, l'animation des échanges de paquets et des outils d'inspection de données divers. NAM a été crée par le laboratoire LBL et s'est considérablement développé durant les dernières années. Le

développement de NAM est en collaboration avec le projet VINT. Plus tard, il est développé à ISI dans les projets CONSER29 et SAMAN30.

NAM interprète un fichier de trace contenant des évènements réseau indexés par le temps. Ces évènements sont principalement les arrivées, départs et suppression de paquets, rupture de lien. Pour les simulations de réseau sans fil, la localisation et les mouvements des nœuds s'ajoutent aux évènements interprétés.



NAM est exécuté avec comme paramètre le fichier enregistré. Lorsqu'on exécute NAM, une fenêtre de travail NAM est créée. Il est possible de faire tourner plusieurs animations avec une seule instance, ce qui permet de mieux comparer certain protocole. On peut entre autre régler le pas de la simulation (de 8s à 800ms), zoomer sur des zones de la simulation, et manipuler la lecture : on peut mettre pause à tout moment ce qui donne un « arrêt sur image », revenir, avancer sur les étapes de la simulation ce qui permet d'examiner des occurrences particulières. La taille des objets dépend de leurs caractéristiques : l'épaisseur des liens dépend du débit du lien et la taille des paquets dépend de leur longueur en bits et de la bande passante sur le lien. La couleur des paquets peut être utilisée pour plusieurs raisons ; dans ce cas, elle différencie deux flux de données différents (celui du nœud 0 et celui du nœud 1). Les paquets se déplacent de nœuds en nœuds le long des liens et sont mis en file d'attente quand un lien est saturé (la file d'attente à côté du nœud 2 correspond à la saturation du lien entre les nœuds 2 et 3). Les points bleus et rouges que l'on voit dans la partie inférieure de la fenêtre représentent les paquets détruits. Ces paquets sont détruits car la file d'attente est remplie et le lien entre 2 et 3 n'a pas assez de débit pour écouler tous les paquets émis par le nœud 0.



-figure ch3.12: *animation de paquets dans NAM-*

V.Conclusion:

NS-2 est un simulateur réseau très utilisé dans la communauté de recherche en réseau. Il implémente en C++ et OTcl tout un ensemble de classes pour définir les concepts de nœuds, liaisons et agents. La structure d'un nœud est fortement inspirée par le modèle en couche OSI. La mobilité a été introduite dans NS-2 dans sa version 2 par le CMU. Au départ, il s'agissait uniquement de simuler des LAN ad hoc, c'est-à-dire tous les nœuds de la simulation devaient être mobiles. Plus tard est venue l'implémentation de MIPv4 qui a permis de faire des simulations sur des topologies avec à la fois des nœuds mobiles et des nœuds fixes. Plusieurs extensions sont notamment disponibles sur différents sites de recherche.

Deuxième partie : NS3

I.1.Introduction :

NS3 est un simulateur réseau à événements discrets dédié à la recherche scientifique. NS3 ainsi que NS2 sont très utilisés et reconnus par la communauté scientifique mais il faut noter que NS3 n'est pas une extension de NS2.
Dans cette parie du chapitre nous allons nous intéresser au simulateur NS3 et, quelques fois, le comparer à NS2.

I.2. Description de NS3 :

Le simulateur de réseaux NS3 a été créé en 2006, par les étudiants de l'université de Washington Thomas R.Henderson et Sumit Roy. NS3 est un puissant outil de recherche pour étudier la conception et les interactions des protocoles, et les problèmes de performance à grande échelle. Sa conception est orientée vers des pratiques de logiciels de développement communautaire et open source pour encourager la participation de la communauté de recherche. Conçu pour améliorer l'évolutivité, la modularité, le style de codage et la documentation. Son noyau et ses modèles sont implémentés en C++, mais avec une interface de script Python, construit comme une bibliothèque qui peut être statique ou dynamique liée à un programme principal C++ qui définit la topologie de simulation et démarre le simulateur.

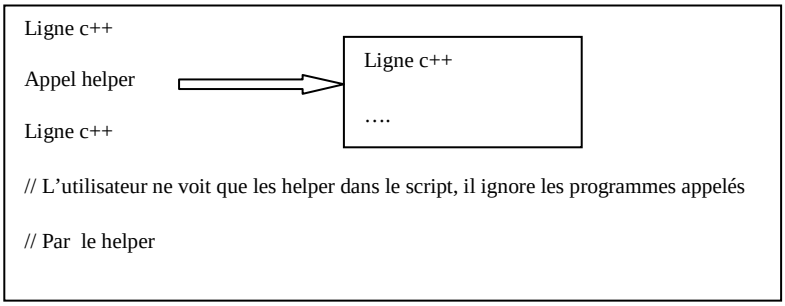
Sous les termes de la GNUGPLv2 licence ; étant open-source le projet s'efforce de maintenir un environnement ouvert pour les chercheurs pour contribuer et partager leur logiciel.
Le système NS3 utilise plusieurs composants de la GNU "toolchain" pour le développement, par exemple : gcc, binutils GNU, et gdb. Il fonctionne seulement sous un environnement de type Linux. Pour ceux fonctionnant sous Windows, il existe des environnements qui simulent l'environnement Linux à des degrés divers (cygwin).

NS3 comprend un support limité pour le script en python ainsi que certaines tâches de haut niveau telles que la visualisation. Un ensemble de compilation peut être générer pour permettre à un script python d'interagir avec l'API du NS3 qui sera normalement accessible à partir d'un script C++.

D'une manière générale :

- NS3 est un simulateur spécialisé dans les réseaux sans fils et les réseaux connectés à internet.
- NS3 n'est pas une extension de NS2
- NS3 est un ensemble de librairies qui travaillent ensemble et avec des librairies externes
- NS3 modélise les réseaux WIFI et les protocoles IP, même si la majorité de ses utilisateurs se concentrent sur les simulations sans fil /IP grâce ou modèles performant existant dans NS3 sur le wifi et le wimax.

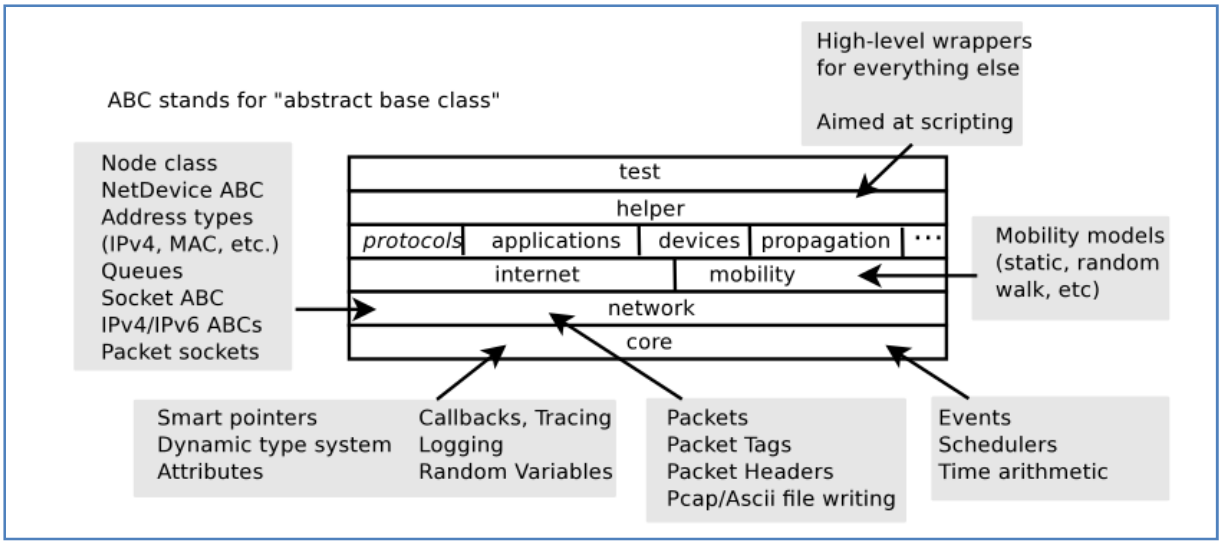
- NS3 travail avec des outils externes, des animateurs, des outils de visualisation, des analyseurs de données, même si pour les scripts on utilise toujours le terminal de commande (en langage C++ ou en python).
- NS3 est conçu pour travailler sous linux mais on peut utiliser des émulateurs linux pour travailler sous Windows (cygwin, machine virtuel).
- NS3 est écrit entièrement en C++, les scripts sont en C++ ou python.
- NS3 ne dispose pas de l'animateur NAM comme NS2, d'autres outils sont disponibles dans NS3.
- Les protocoles de routage spécial vanet ne sont pas encore implémentés dans NS3, c'est pour cela que dans ce mémoire on a choisi DSR comme protocole de routage même si ce n'est pas l'idéal pour les réseaux vanet car le protocole voulu 'GYTAR' n'est pas encore implémenté dans NS3.
- NS3 supporte l'ordonnancement temps réel ce qui est un atout majeur dans la conception d'applications temps réel dans les réseaux.
- Parce que NS3 est développé par toutes la communauté scientifique, une application 'release manager (RM)' gère d'une manière très efficace tous les ajouts et extension de ce grand logiciel.
- Une preuve de la puissance de ce logiciel, on peut intégrer tout un système linux dans un seul nœud de NS3.
- NS3 dispose de helper pour facilité son utilisation. C'est comme l'utilisation d'un langage haut niveau dans un autre langage haut niveau. On verra un exemple de ces helper quand on utilisera un protocole de routage dans NS3.



-figure ch3.13: l'utilisation des helper dans un script NS3-

I.3.Modules du simulateur NS3 :

NS3 est une librairie C++ constitué d'un ensemble de modèles pour simuler des réseaux. Ces modèles sont implémentés en objets C++ enveloppés par python.
Les évènements dans le simulateur NS3 sont tout simplement des fonctions appelées à être exécutées dans un temps donné par l'ordonnanceur du simulateur. Voici les modules de base de NS-3 sur la figure ci-dessous :



-figure ch3.14 : modules de NS3-

Core Module :

Il permet de créer une structure de classes hiérarchiques qui dérive de la classe Object de base. Ce module de base est indépendant.

Common Module :

Il permet de gérer les actions liées à l'émission et à la réception des paquets. Centré sur la classe Paquets utilisé pour la simulation de réseau au niveau des paquets, son utilisation permet d'optimiser la gestion de la mémoire en utilisant la méthode copier/coller. La plupart des autres modules utilisent ses fonctionnalités.

Simulator Module :

Il permet de gérer la simulation des événements, il prévoit explicitement la possibilité de planifier ces derniers à des moments différents et ensuite de les exécuter.
La classe Time représente le temps simulé à haute résolution en utilisant un entier de 128 bits, et cette classe est la classe la plus importante du simulateur.

Mobility Module :

Ce module permet de définir la position des nœuds et associe un modèle de mouvement aux agents de la simulation.

Node Module :

La classe Node peut contenir plusieurs NetDevices, Chacun d'entre eux est attaché à un canal par lequel il envoie et reçoit des paquets. Un nœud peut contenir plusieurs gestionnaires de protocole qui acceptent les paquets reçus par le NetDevice. Pour lancer la transmission des paquets, chaque nœud peut également contenir une liste d'applications.

Helper Module :

Ce modèle peut être considéré comme un emballage de haut niveau. Il facilite la construction de scénarios complexes de simulation et l'installation des différents modules dans des agents différents.

Application Module :

Certaines applications sont intégrées et fournies par NS-3. Elles sont installées dans les nœuds et peuvent être démarrées/arrêtées à des moments précis de la simulation.

Internet Stack Module :

Les classes de ce module définissent les protocoles TCP/IP des couches réseaux trois et quatre (TCP/IP).

Devices Module :

Les composants de ce type représentent des périphériques réseaux et transmettent des paquets via un canal virtuel à d'autres instances de la même classe NetDevice.

Routing Module :

Deux algorithmes de routage sont disponibles dans NS-3. Le premier appelé GlobalRouter, utilise des routes statiques et le deuxième met en œuvre le protocole OLSR pour les réseaux dynamiques ad-hoc.

I.4. Comparaison entre NS2 et NS3 :

Différences :

NS2 :
Crée en 1996 par DARPA VINT ; SAMAN & NSF ; CONSER, NS-2 est lourdement basé sur l'architecture du NS original qui est développé en 1989.
NS2 a été mis en œuvre en utilisant une combinaison de OTCL (un prolongement orienté objet de TCL (Tool Command Language), et C++ (noyau entièrement écrit en C++). NS2 ne possède aucune interface graphique. Il existe une extension nam (Network Animator) qui permet de visualiser les résultats d'une simulation une fois achevée.

NS3 :
Crée en 2006, NS3 n'est pas une extension de NS2, c'est un nouveau simulateur. Il ne dispose pas de l'outil NAM pour l'animation et on ne peut pas utiliser le langage otcl. Bien que les

deux simulateurs soient tous deux écrits en C++, NS3 ne supporte pas les API de son prédécesseur. En plus du C++, NS3 utilise Python (interface de script). Quant à la simulation sous NS3, elle se fait à l'aide d'un package basé sur python appelé PyViz.
Un script NS2 ne peut pas s'exécuter sous NS3 et un script NS3 ne peut pas s'exécuter sous NS2.

Points communs:

- Quelques modèles écrit en C++ dans NS2 sont portés dans NS3, OLSR et le model d'erreur sont écrits pour NS2 mais ils sont utilisés dans NS3.
- Tous deux sont des simulateurs de réseaux qui permettent, comme leurs noms l'indiquent, de réaliser des simulations. Toutefois ces derniers ciblent principalement un domaine qui se résume à la recherche et l'utilisation pédagogique qui affectent de manière directe la communauté de la recherche qui contribue d'une façon permanente et continue au développement de nouveaux modèles.
- Tout les deux permettent d'anticiper sur la topologie d'un réseau car lorsque les résultats d'une simulation ne sont pas satisfaisants, il est facile de modifier la topologie dans le but de corriger les problèmes avancés par la simulation précédente.
- nous pouvons aussi nous servir de l'un de ces deux simulateurs pour tester un nouveau protocole (la facilité de l'intégration dépend du simulateur utilisé) avant de l'utiliser réellement.
- NS2 et NS3 sont d'une grande aide dans la recherche scientifique et le développement d'applications.
- Le premier point que l'on peut relever est le coût (financier). Une simulation est peu coûteuse car elle ne nécessite qu'un seul terminal, la simulation se faisant graphiquement. On peut ainsi simuler aisément un réseau complexe mettant en ouvre un grand nombre de postes. Ils sont aussi facilement abordables puisque ils sont distribués sous licence, ce qui implique que l'accès aux sources est possible.

Les résultats sont aussi détaillés : consommation de l'énergie dans le routage d'un réseau ad hoc, propagation des ondes radio, sachant que ces résultats sont difficiles à obtenir dans une application réelle.

Critiques :

- NS2 est critiqué sur ces modèles trop complexes, la lenteur dans l'exécution des taches, le fait d'être obliger d'apprendre un nouveau langage pour écrire un script et parce que l'utilisateur doit avoir beaucoup de notions théoriques sur les files d'attente.

- NS3 est aussi critiqué sur la non implémentation de certains protocoles. Surtout que ces mêmes protocoles sont implémentés dans NS2.

I.5.Pour installer NS3 on a besoin de ressources :

I.5.1. Le Web :

Il existe plusieurs ressources importantes que tout utilisateur de NS3 doit connaitre. Le site principal http : //www.nsnam.org donne accès à des informations de base sur le système NS3. Une connexion Internet est indispensable pour l'installation de NS3.

I.5.2. Mercurial :

Les systèmes complexes ont besoin d'un moyen pour gérer l'organisation et l'évolution de leur ligne de code et leur documentation. Le plus connu de ces systèmes de gestion de code est 'concurrent version system CVS'.

Le projet NS3 utilise Mercurial comme système de gestion de code source. Pour plus d'info voir le site *'http/ :www.selenic.com/mercurial'*.

I.5.3. Bake :

Bake n'est pas *make* n'est pas *autoconfig* ou *automake.* Bake ne remplace pas les package de gestion qui se trouve dans notre système, c'est un outil intégré par les développeurs de software pour automatiser la construction d'un nombre de projets qui dépendent les uns des autres.
Bake a été développé pour automatiser la construction reproductible de NS3 en tenant compte du fait que cette construction est composée d'un nombre de projets dépendants entre eux.
Bake est là pour simplifier l'assemblage de toutes ces pièces de software d'une manière cohérente et faire de cet ensemble quelque chose d'utile.
Dans notre travail on utilisera bake pour la construction de NS3.

I.5.3.1. Caractiristique de bake :

- Manipulation automatique des dépendances entre les librairies.
- Téléchargement automatique des composants nécessaires.
- Construction automatique des composants requis.
- Possibilité d'ajout de modules à l'arbre des composants.
- Eviter tous les détails de la construction a l'utilisateur.

I.5.3.2. limitations :

- Bake est utilisé uniquement sous linux.
- Si des outils ont été oubliés ils doivent être installés par l'utilisateur.

I.5.3.3. prérequis :

- Bake est implémenté en python donc python est requis. Pour voir si python est installé on exécute la commande « *python –v* ».
- On a besoin que mercurial soit installé dans notre machine.
- Pour utiliser bake on besoin d'une série de commandes pour faciliter son utilisation.
- Pour vérifier les composants manquants on utilise la commande « *bake.py chech* ». on verra comment utiliser bake dans le chapitre réalisation est simulation.
- Une fois que tout est vérifié on peut installer bake.

I.5.3.4. installation de bake : voir dans le chapitre réalisation et simulation.

I.5.4. Waf :

Une fois que nous avons téléchargé le code sur notre système local, nous aurons besoin de compiler cette source pour produire des programmes utilisables. Tout comme dans le cas de la gestion du code source, il existe de nombreux outils disponibles pour remplir cette fonction (make le plus connu).

Récemment, ces systèmes ont été développés en utilisant le langage Python. Le système Waf qui est l'un des systèmes de construction de nouvelle génération à base de Python est utilisé sur le projet NS3.

II. Environnement de développement :

Les scripts dans NS3 sont faits en C++ ou Python. La plupart des API NS3 sont disponible en Python, mais les modèles sont écrits en C++ dans les deux cas. Le système NS3 utilise plusieurs composants de la GNU "toolchain" pour le développement."toolchain" est l'ensemble des outils de programmation disponibles dans l'environnement déjà cité. NS3 utilise gcc, binutils GNU, et gdb.
Cependant, nous n'utilisons pas le GNU pour construire des outils système, ni make ni autotools. Nous utilisons Waf pour ces fonctions. Un utilisateur de NS3 travaillera sous Linux ou un environnement Linux. Pour ceux qui travaillent sous Windows, des environnements existent pour simuler l'environnement Linux à des degrés divers. NS3 peut s'installer dans un environnement de machines virtuelles telles que VMware en installant une machine virtuelle Linux.

II.1. L'installation de NS3 :

L'installation du simulateur NS3 se fait en trois étapes : le téléchargement puis la construction qui, elle même, se fait en deux étapes. Seulement avant, l'utilisateur doit télécharger des package pour pouvoir faire ces étapes correctement. (Voir chapitre réalisation et simulation pour plus de détails).

Conclusion :

La simulation est très importante pour l'étude des réseaux, c'est une solution très économique et elle évite les déplacements, même si on doit ajouter l'erreur de la simulation à l'erreur réelle d'une solution.

On a vue deux simulateurs de réseau, NS2 et NS3, dans le chapitre suivant on verra comment les installer et comment les utiliser.

I.1. Introduction :

La simulation des VANET est pratiquement séparée en une simulation du trafic et une simulation du réseau.
Des simulateurs du trafic routier sont utilisés pour générer des traces du mouvement des véhicules qui sont injectées dans un simulateur NS2 ou NS3 pour mesurer les performances du réseau. Pour faciliter l'interaction entre les simulateurs du trafic et les simulateurs du réseau, d'autres outils sont également développés.
Dans notre cas, on évitera l'utilisation de simulateurs externes en intégrant l'application 'vanet-highway' dans NS3.

I.2. matériels et logiciels de base pour réaliser notre travail:

- Télécharger linux debian 6.0.3 32bit
- un micro-ordinateur de type notebook avec processeur 'Atom N150 ' et de RAM 2GO.

L'installation et très facile, choisir le mode graphique pour pouvoir utiliser la sourie. Pour les débutant télécharger et consulter le pdf '*debian-edu-wheezy-manual.pdf*' ou bien consulter le site *www.debian.com*.
Remarque : debian est un système linux complet, pratique pour le développement. Même les outils pour les bases de données sont installés avec le système, il suffit de les coché pendant l'installation. L'interface graphique est simple et efficace.

La difficulté pour nous les débutants sur débian est que le niveau de sécurité est très élevé. Aussi, debian n'a pas de session graphique pour l'administrateur 'root'. L'accès à root est uniquement sur terminal administrateur.

I.3. Installation de l'environnement de simulation (sumo, NS2, NS3):

I.3.1.Installation de sumo :

Télécharger et voir l'annexe A du fichier pdf (Optimisation dans les réseaux de capteur véhicules).

I.3.2.Installation de NS2 : sur linux debian

Etape1 : télécharger NS2.35 sur le site officiel 'http://www.isi.edu/nsnam/ns/'

Etape2 : placer le dossier de NS2 dans le répertoire de travail et décompresser avec la commande *$ tar -xzvf ns-allinone-2.35.tar.gz*.

Etape 3 : aller au répertoire NS2 avec la commande

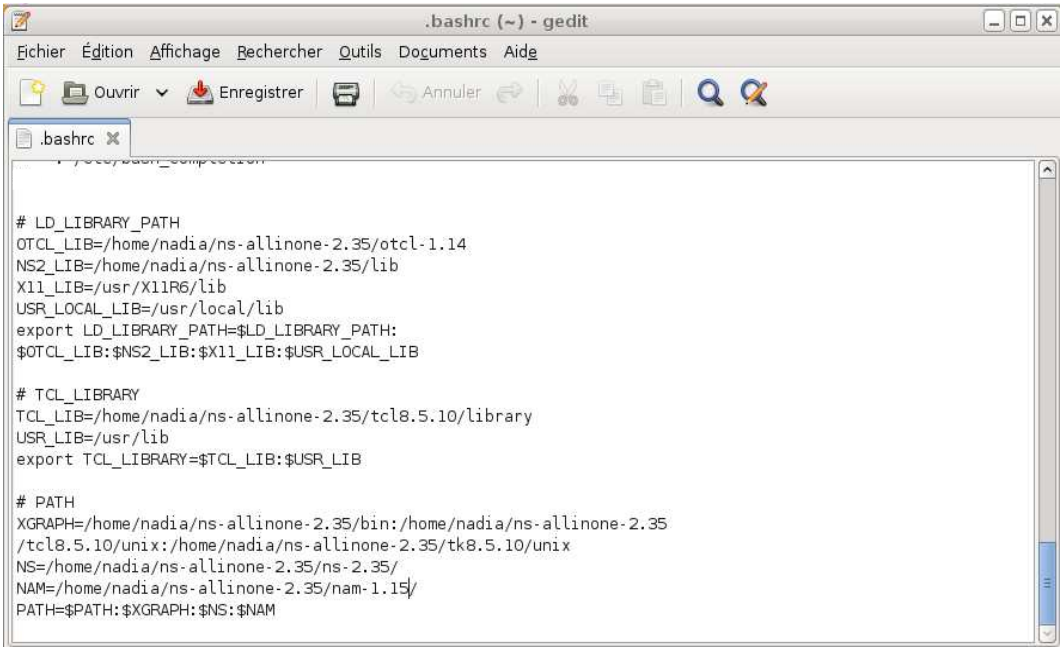
$ cd ns-allinone-2.35

Etape4 : insérer le dvd du système linux debian pour récupérer les packages manquant. Puis exécuter les commandes suivantes.

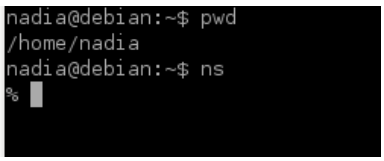
$ sudo apt-get install build-essential autoconf automake libxmu-dev
$ sudo apt-get install -f build-essential libxt-dev libxt6 libsm-dev libsm6 libice-dev libice6 libxmu-dev

Etape 5 : lancer l'installation avec la commande
$. /install
Cela peut prendre un peu de temps selon la capacité de la machine.

Etape6 : changer les variables de l'environnement avec la commande
$ gedit ~/.bashrc
Une fenêtre s'affiche. On ajoute les lignes suivantes (voir figure ci-dessous).

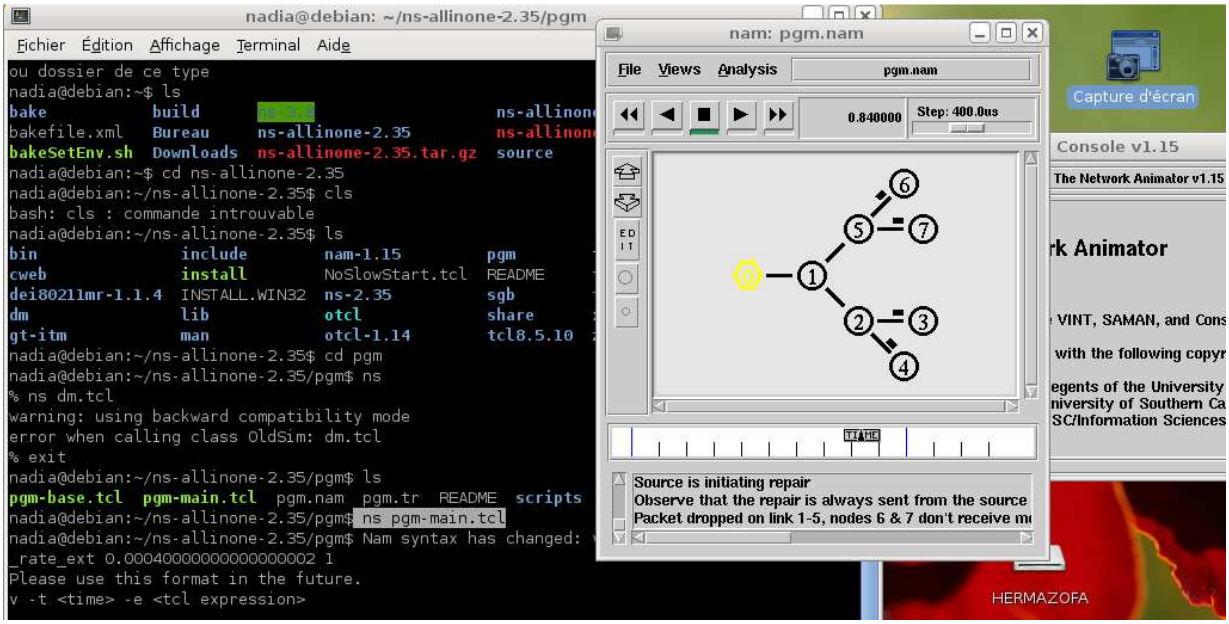


Etape 7 : pour voir si NS2 est installé, on exécute la commande suivante.
$ ns
On verra sur le terminal :



Etape8 : validation de NS2 avec la commande suivante
$ cd ns-2.35
$ ./validate
L'exécution va prendre un peu de temps selon la capacité de la machine

Etape9 : un exemple. On va copier un script *'pgm-main.tcl'* dans le répertoire NS2 puis on lance l'exécution avec la commande '*ns nom-fichier.tcl'* dans notre cas '*ns pgm-main.tcl'* (voir la figure ci-dessous)



I.3.3. Installation de NS3.20 sur linux ubento et linux debian :

L'installation de NS3 n'est pas évidente comme celle de NS2. Ce qui complique notre travail c'est que NS3 n'est pas encore une bibliothèque prête à l'utilisation. On doit télécharger tous les composants nécessaires, puis compiler et assembler le tout pour avoir l'arbre des composants pour utiliser NS3. Cela se fera grâce au software 'bake'.

Étape 1 : télécharger les composants manquant :

Le noyau de NS3 nécessite un gcc / g ++ version 3.4 ou plus, et Python 2.4 ou plus. Mettre le dvd debian dans le lecteur. Executer les commandes suivantes une par une.

- **exigences minimales pour C ++ :** C'est le minimum des packages nécessaires à l'exécution NS3

  sudo apt-get install gcc g ++ python

- **exigences minimales pour Python :** C'est le minimum de packages nécessaires pour travailler avec Liaisons Python

  sudo apt-get install gcc g ++ python python-dev

- **Mercurial est nécessaire pour travailler avec NS3 :** Les systèmes complexes ont besoin d'un moyen pour gérer l'organisation et l'évolution de leur ligne de code et leur documentation. Le plus connu de ces systèmes de gestion de code est 'concurrent version system CVS'

Le projet NS3 utilise Mercurial comme système de gestion de code source. Pour plus d'info voir le site *'http/ :www.selenic.com/mercurial'* .voici en italique la commande pour le télécharger et l'installer :

sudo apt-get install mercurial

- L'exécution des liaisons de python dans l'arbre de développement NS3 (ns-3-dev) nécessite bazar

  sudo apt-get install bzr

- **debogage:**

  sudo apt-get install gdb valgrind

- **GNU Scientific Library (GSL) pour les modèles d'erreur WiFi**

  sudo apt-get install GSL-bin libgsl0-dev libgsl0ldbl

- **Besoin d'un analyseur syntaxique et d'un analyseur lexical ' bison et flex':**

  sudo apt-get install bison flex libfl-dev

- **Pour installer gcc-3.4 pour une simulation Réseau Cradle (NSC) :**

  sudo apt-get install g ++ - 3.4 gcc-3.4

- **Pour lire les traces de paquets pcap**

  sudo apt-get install tcpdump

- **support de base de données pour les statistiques :**

  sudo apt-get install sqlite3 sqlite libsqlite3-dev

- **Version basé sur XML (nécessite libxml2 version> = 2.7)**

  sudo apt-get install libxml2 libxml2-dev

- **Un système de configuration basé sur GTK**

  sudo apt-get install libgtk2.0-0 libgtk2.0-dev

- **Pour expérimenter avec des machines virtuelles et ns-3**

  sudo apt-get install vtun lxc

- **Prise en charge des outils check-style.py / programme de contrôle de style de code**

  sudo apt-get install uncrustify

- **Doxygen et la documentation en ligne connexes:**

  sudo apt-get install doxygen graphviz imagemagick
  sudo apt-get install texlive texlive-extra-utils texlive-latex-extra

- **Le manuel et le tutoriel ns-3 sont écrits en reStructuredText Sphinx (doc / tutorial, doc / doc / manuels, modèles), et les chiffres généralement avec dia:**

  sudo apt-get install python-sphinx dia

- **Support pour ns-3-pyviz le visualiseur de Gustavo Carneiro**

  sudo apt-get install python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev

- **Support pour module de OpenFlow (nécessite certaines bibliothèques boost)**

  sudo apt-get install libboost-signaux-dev libboost-filesystem-dev

- **Prise en charge de l'émulation distribuée à base de MPI**

  sudo apt-get install openmpi-bin-openmpi commun openmpi-doc libopenmpi-dev

- **Prise en charge des génératrices modifié liaisons python**

  sudo apt-get install gcc-multilib

Étape 2 installation avec bake : c'est cette méthode qu'on va utiliser

- Télécharger NS3 avec bake
- Construction avec bake
- installation

Bake n'est pas *make* n'est pas *autoconfig* ou *automake.* Bake ne remplace pas les package de gestion qui se trouve dans notre système, c'est un outil intégré par les développeurs de software pour automatiser la construction d'un nombre de projets qui dépendent les uns des autres.
Bake a été développé pour automatiser la construction reproductible de NS3 en tenant compte du fait que cette construction est composée d'un nombre de projets dépendants entre eux.
Bake est là pour simplifier l'assemblage de toutes ces pièces de software d'une manière cohérente et faire de cet ensemble quelque chose d'utile.
Dans notre travail on utilisera bake pour la construction de NS3.

Pour utiliser Bake on a besoin d'avoir Python (de préférence 2.6 ou plus) et mercurial dans notre machine.

- Premièrement on a besoin de télécharger bake en utilisant mercurial avec la commande suivante :

  hg clone http://code.nsnam.org/bake

- Télécharger bake, cette commande va télécharger bake et créer un répertoire bake à l'intérieur du répertoire ou nous sommes comme on le voit sur la figure ci-dessous:

Hg clone http ://code.nsnam.org/bake bake

- ajouter bake dans path:

export BAKE_HOME=`pwd`/bake
export PATH=$PATH:$BAKE_HOME

export PYTHONPATH=$PYTHONPATH:$BAKE_HOME

- pour voir si python est installé exécutez la commande suivante :

python -v

- on va appeler 'bake check' pour trouver ce qui manque dans notre système pour l'installation de ns-3.

bake.py check

- on doit avoir sur le terminal :

```
> Python - OK
  > GNU C++ compiler - OK
  > Mercurial - OK
  > CVS - OK
  > GIT - OK
  > Bazaar - OK
  > Tar tool - OK
  > Unzip tool - OK
  > Unrar tool - OK
  > 7z  data compression utility - OK
  > XZ data compression utility - OK
  > Make - OK
  > cMake - OK
  > patch tool - OK
  > autoreconf tool - OK
  > Path searched for tools: /usr/lib64/qt-3.3/bin
```

- configurer bake

  bake.py configure -e ns-3.20

- Pour voir les modules ajoutés, et les besoin du système pour cette configuration, appeler 'bake show' avec la commande suivante:

  bake.py show

- Pour télécharger les modules, construire et installer NS3, on appelle la commande 'bake deploy' :

  bake.py deploy

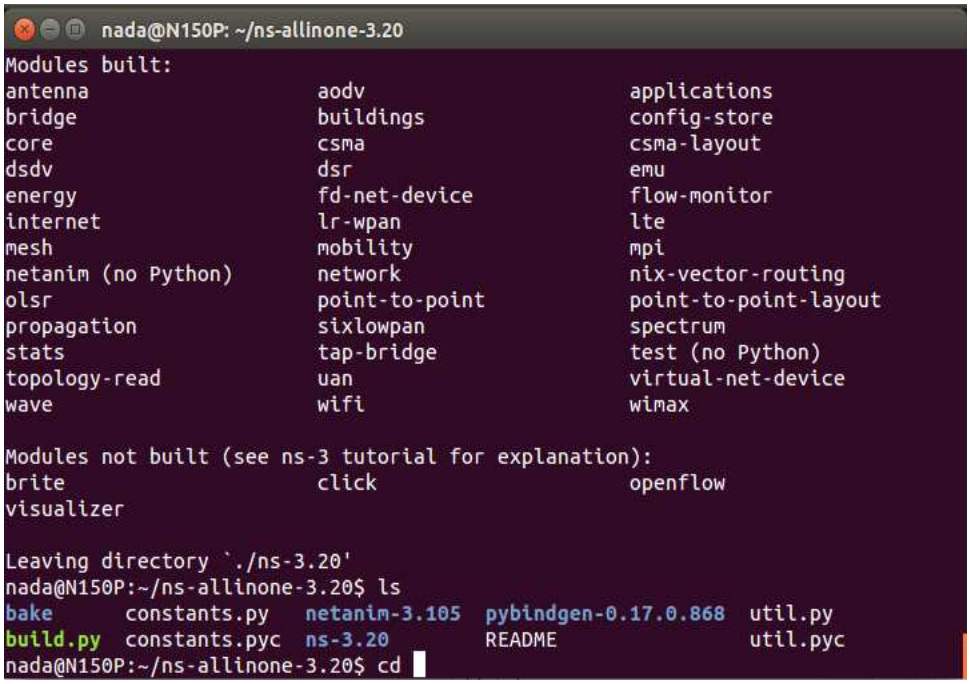
Ceci va télécharger les modules et tous les composants logiciels dont il dépend et construire NS3. On peut le faire étape par étape en exécutant les commandes suivantes :

bake.py download

- En suite la compilation

  bake.py build

- on verra sur le terminal



- Voici l'arbre des modules dépendants :

```
-- Enabled modules dependency tree --
  +-enabled/
   |
   +-click-dev
   |
   +-gccxml-ns3
   |
   +-libxml2-dev
   |
   +-ns-3-allinone/
   | |
   | +-click-dev (optional)
   | |
   | |
   | +-nsc-dev (optional)
   | |
   | +-openflow-dev/ (optional)
   | | |
   | | +-libxml2-dev (mandatory)
   | |
   | +-pybindgen-dev/ (optional)
   | | |
   | | +-pygccxml/ (optional)
   | | | |
   | | | +-gccxml-ns3 (mandatory)
   | | |
   | | +-python-dev (optional)
   | |
   | +-pyviz-prerequisites/ (optional)
   |  |
   |  +-pygoocanvas (optional)
   |  |
   |  +-pygraphviz (optional)
   |  |
   |  +-python-dev (optional)
   |
   +-nsc-dev
   |
```

```
    +-openflow-dev/
    | |
    | +-libxml2-dev (mandatory)
    |
    +-pybindgen-dev/
    | |
    | +-pygccxml/ (optional)
    | | |
    | | +-gccxml-ns3 (mandatory)
    | |
    | +-python-dev (optional)
    |
    +-pygccxml/
    | |
    | +-gccxml-ns3 (mandatory)
    |
    +-pygoocanvas
    |
    +-pygraphviz
    |
    +-python-dev
    |
    +-pyviz-prerequisites/
    |
    +-pygoocanvas (optional)
    |
    +-pygraphviz (optional)
    |
    +-python-dev (optional)
--end tree--
```

Etape3 la construction avec waf:

./waf clean

./waf --build-profile=optimized --enable-examples --enable-tests configure

On va voir sur le rerminal :

```
Setting top to : .
Setting out to : build
```

```
Checking for 'gcc' (c compiler) : /usr/bin/gcc
Checking for cc version : 4.2.1
Checking for 'g++' (c++ compiler) : /usr/bin/g++
Checking boost includes : 1_46_1
Checking boost libs : ok
Checking for boost linkage : ok
Checking for click location : not found
Checking for program pkg-config : /sw/bin/pkg-config
…
'configure' finished successfully
```

Etape 4 Compilation avec Waf :

On peut aussi faire la configuration (la compilation) avec waf

- Executer les commandes suivantes:

  CXX="distcc g++" ./waf configure
  ./waf build

Etape 5: Validation

- Exécuter la commande suivante

  ./test.py
  ./test.py -c core // pour tester uniquement le cœur

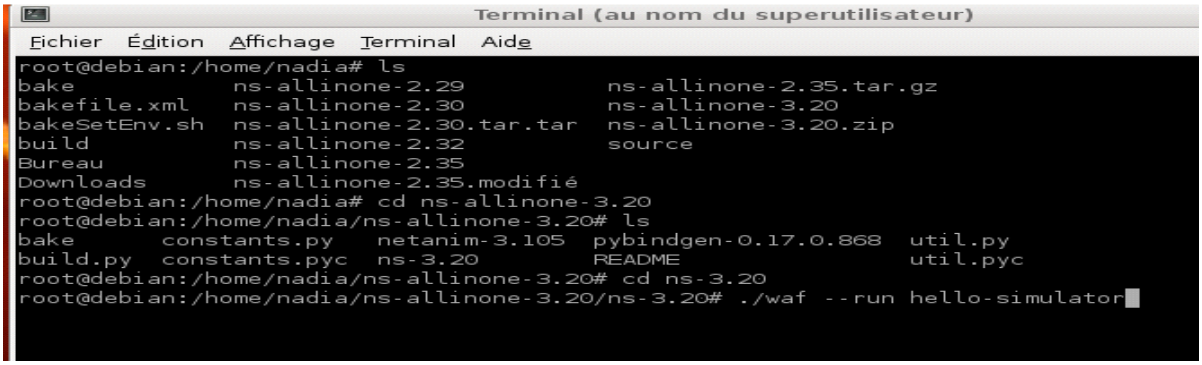
On voit sur le terminal:

```
PASS: TestSuite histogram
PASS: TestSuite ns3-wifi-interference
PASS: TestSuite ns3-tcp-cwnd
PASS: TestSuite ns3-tcp-interoperability
PASS: TestSuite sample
...
```

Etape 6 : executer un premier script sur NS3.20

Une fois la construction et les teste terminés, on va lancer la commande '*run*' comme suite : dans le repertoire '*home /nadia/ns-allinone-3.20/ns-3.20*', on utilise la commande '*cd nom-de-repertoire*' (voir la figure ci-dessous) :

./waf –run h ello-simulateur



Attendre la fin de l'exécution qui prend un peu de temps selon la capacité de la machine. On voit sur le terminal la phrase "hello simulateur " signe que NS3 fonctionne correctement.

Si le message 'hello simulateur' ne s'affiche pas, télécharger le fichier "ns-3 Tutorial Release ns-3.20.pdf" puis allez à la page 18 chapitres 3 sections 3.5 "running a script" pour voir la solution.

Méthode 1 : simulation d'un réseau vanet en utilisant NS3 et SUMO

I.1.Introduction :

Cette méthode est la méthode classique pour la simulation des VANET. Dans cette méthode, deux types de simulations sont nécessaires, une simulation du trafic et une simulation du réseau.

I.2.Première partie : simulation du trafic

I.2.1. définition SUMO (Simulation of Urbain MObility) :

'Simulation of Urban Mobility' (SUMO) est un package open source pour la simulation du trafic. Le développement de SUMO (simulation of urbain mobility) en français « Simulation de la

Mobilité Urbaine » a commencé dans l'année 2000. La raison majeure qui a mené au développement d'un outil de simulation du trafic routier était de faciliter les recherches dans ce domaine sans être obligé de mettre un réseau de véhicules en pratique.

I.2.2.Création d'un scenario avec SUMO :

Pour créer un scenario dans sumo, on a besoin de créer des fichiers xml spécifiques. On général, on a besoin de trois fichiers comme entrées dans sumo. Les fichiers avec nod.xml et edg.xml extension contiennent les informations du réseau, ces informations seront converties dans sumo pour avoir les nœuds et les liens. On va décrire brièvement ces fichiers.

- **Le fichier 'node' :** dans ce fichier on trouve les coordonnées (x, y) des nœuds, chaque nœud aura son numéro identifiant ID et aussi les informations sur la communication entre les nœuds

```
<nodes>
 <node id="id" x="x-coordinate" y="y-coordinate" />
</nodes>
```

- **le fichier 'edge' :** il contient les informations sur les routes. Une route est considérée comme une connexion entre deux nœuds. Chaque route a un numéro ID et un nombre de lignes.

```
<edges>
    <edge id="id" from="source node" to="destination node" priority="priority no." numLanes="no. of lanes"
    speed="vehicle speed" />
</edges>
```

- **le fichier 'net file' :** le fichier 'net file' dans sumo est un fichier avec l'extension net.xml. avec l'utilisation des fichiers précédents, le fichier 'net file' sera généré avec la commande '*NETCONVERT*'.
- **Le fichier 'flow file' :** en utilisant le fichier nœud et le fichier 'edge' on crée le fichier 'flow'.

```
<flows>
    <flow id="id" from="source edge" to="destination edge" begin="begin time" end="end time" number="no. of
    vehicles" />
</flows>
```

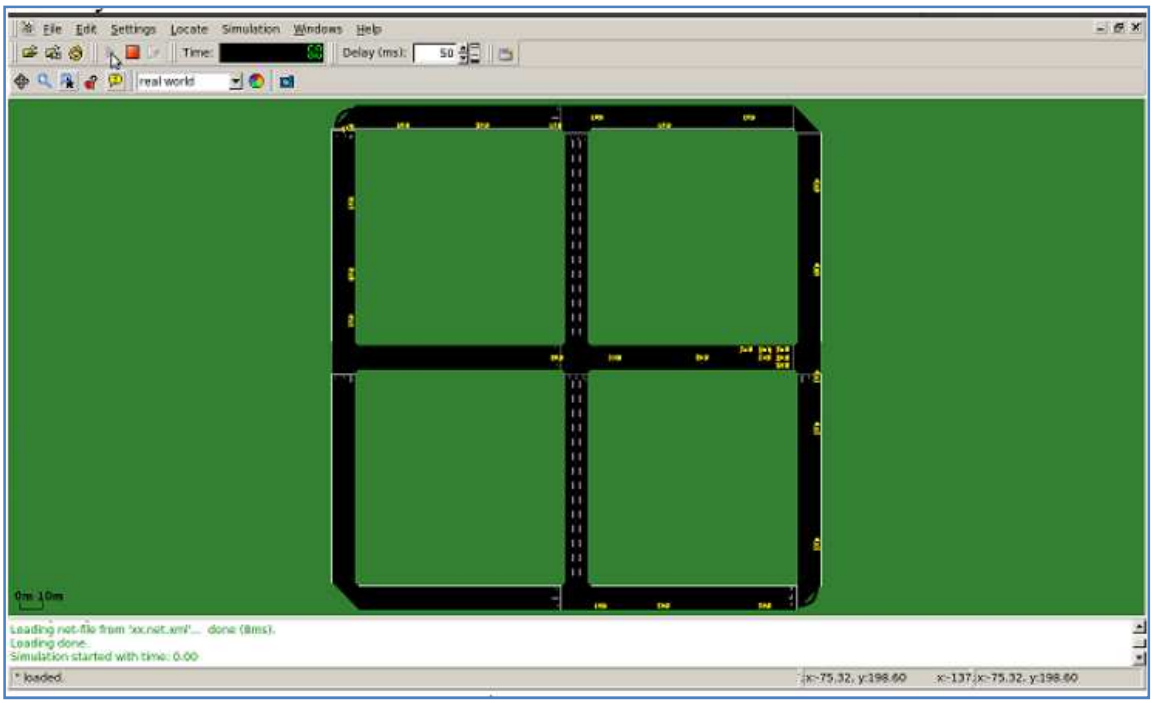
- **Le fichier route :** en donnant à sumo comme entrée le fichier ‘flow’ et le fichier ’net file’, le fichier route sera crée avec la commande suivante :

duarouter –flows=xx.flow.xml –net=xx.net.xml –output-file=newrou.rou.xml

- **Le fichier ‘sumocfg’ :** le fichier route et le fichier ‘net file’ seront utilisés comme entrées dans sumo.

```
<input>
    <net-file value="xx.net.xml"/> // net file
    <route-files value="newroute.rou.xml"/> // route file
</input>
<time>
    <begin value="0"/> // simulation start time
    <end value="600"/> // simulation end time
</time>
</configuration>
```

I.2.3.SUMO GUI*:* SUMO GUI est pratiquement la même application que sumo, mais avec une interface graphique comme extension.



-figure ch4.1 : l’interface graphique de SUMO GUI-

I.2.4.la génération des fichiers traces :

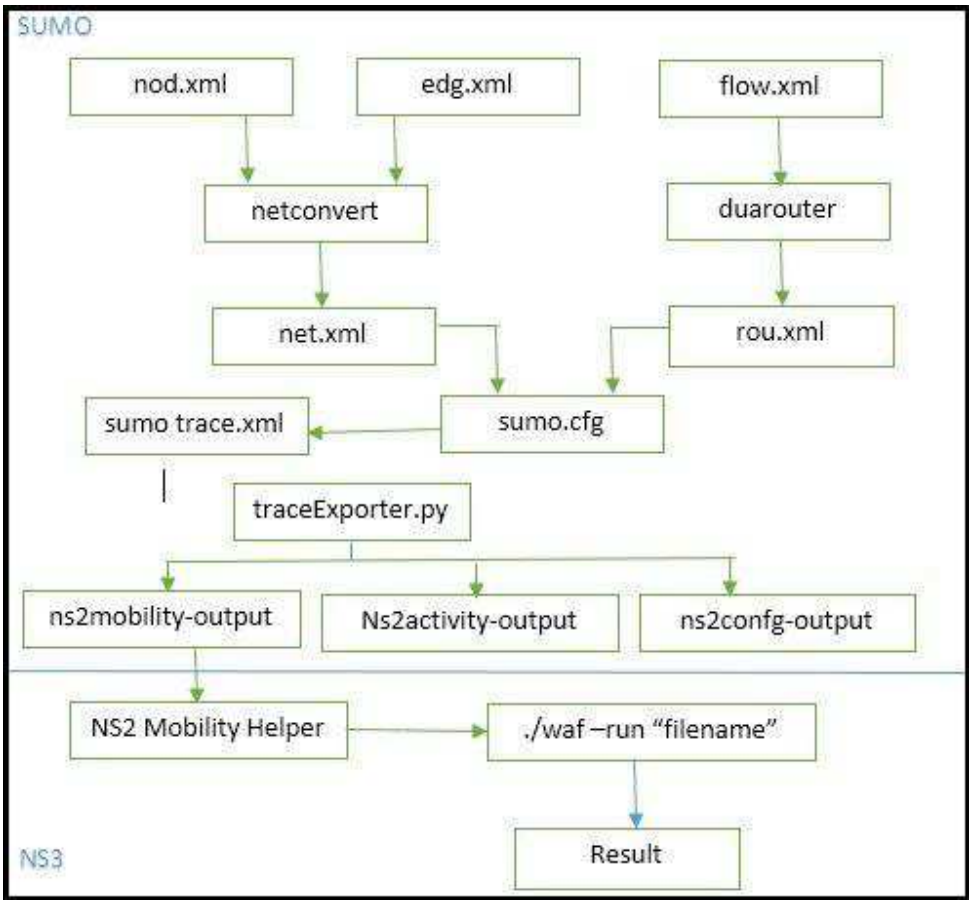
La création des fichiers traces est une étape très importante pour la création d'un scenario réseau pour le simulateur NS3.
Pour générer les fichiers traces, on peut procéder de différentes manières. En utilisant 'SUMO trace exporter', 'MOVES' ou 'iTETRIS'. Dans notre cas on va utiliser SUMO trace exporter.
Sumo dispose de cet utilitaire pour la génération des fichiers trace. On le trouvera dans le dossier 'tools' de package SUMO. D'abord on crée le fichier trace en utilisant la commande suivante:

sumo -c My-scenario.sumo.cfg --fcd-output sumoTrace.xml

en suite on convertit le fichier trace crée vers un autre format, dans notre cas '.tcl' qui correspond à NS2 ; on aura en sortie trois fichiers (mobility file, activity file et config file) :

```
./traceExporter.py –fcd-input=smpleTrace.xml –ns2mobility-output=mobsmple.tcl
./traceExporter.py –fcd-input=smpleTrace.xml –ns2activity-output=actsmple.tcl
./traceExporter.py –fcd-input=smpleTrace.xml –ns2config-output=consmple.tcl
```

On doit ensuite convertir ce fichier .tcl en .cc pour qu'il puisse être utilisé par NS3. C'est de cette manière qu'on peut utiliser le fichier trace de sumo dans NS3.



-Figurech4.2 : diagramme des flow de donnée dans sumo-

I.3. Deuxième partie : la simulation réseau avec NS3

Une fois la simulation du trafic routier est terminée on passe à la simulation du réseau avec NS3. On passe directement au script de simulation (dans NS3 les scripts sont en langage c++ ou python. NS3 est écrit en C++).

- **importer des modules dans NS3**

Cette partie du script consiste à définir les modules externes à ajouter pour effectuer la simulation. NS3 propose un ensemble de classes 'helper' pour aider l'utilisateur à ne pas se perdre dans des lignes de programme. Par exemple "dsr-helper.h" peut être importé de cette manière :

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
 #include "ns3/mobility-module.h"
```

```
#include "ns3/ns2-mobility-helper.h"
#include “ns3/dsr-module.h”
#include “ns3/dsr-helper.h”
```

- **La fonction Main :**

Ceci est le cœur de notre script. On peut configurer un scenario et implémenter un protocole dans cette fonction. Voici ce qu’il faut configurer dans un script NS3 :

1) *WiFi Channel:*

Pour une simulation VANET, la première chose à faire est de créer ‘Wifi chanel’. Il y a plusieurs modèles à configurer selon les besoins, WiFi-802.11, WiMax-802.16, Point To Point Channels, CSMA links. On sélectionne ‘WiFi-802.11b’ dans la classe ’ WiFiHelper’ :

```
WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
```

En suite, on doit fixer le modèle de propagation pour le canal choisi, le délai de propagation et le modèle de propagation loss. On sélectionne ‘YansWiFiChannelHelper’ pour configurer le ‘wifi channel’ et ‘FriisPropagationLossModel’.

```
YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss("ns3::FriisPropagationLossModel","MinDistance",
DoubleValue (250));
wifiPhy.SetChannel (wifiChannel.Create ());
```

- **Création des nœuds :**

On va créer maintenant les nœuds et on va leurs ajouter la mobilité. NS3 dispose d’une classe ‘ns2mobilityhelper class’ qui permet l’utilisation de fichiers de mobilité externe, de sumo par exemple. Le fichier externe doit avoir l’extension .tcl. La classe ‘NetDeviceContainer’ est utilisée pour configurer les composants réseau utilisés dans la simulation.
Le nombre de nœuds dans la simulation réseau doit être égal au nombre de véhicules dans la configuration de sumo.

```
NoeContainer c;
c.Create (nNodes);
Ns2MobilityHelper ns2 = Ns2MobilityHelper ("scratch/nom_fichier_extern_trace.tcl");
ns2.Install ();
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);
```

- **Le routage :**

Pour le routage des classes helper sont disponibles pour chaque protocole de routage implémenté dans NS3. Exemple 'AodvHelper class' , 'DsrHelper class'.
Malheureusement les protocoles de routage spéciaux pour les VANET ne sont pas encore implémentés dans NS3. Dans notre cas on va utiliser le protocole de routage DSR (voir chapitre 3).

```
DsrHelper Dsr;
InternetStackHelper internet;
internet.SetRoutingHelper (Dsr);
 internet.Install (c);
```

- **L'adressage IP :**

NS3 supporte les deux modes d'adressage IPv4 et IPv6. Pour les scenarios VANET on utilise en général IPv4

```
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
 Ipv4InterfaceContainer i = ipv4.Assign (devices);
```

- **Application :**

Les applications dans NS3 sont attachées aux nœuds. Chaque nœud à sa propre liste de références pour les applications. Beaucoup d'applications sont disponibles dans NS3 comme OnOff, PacketSink, UdpEcho, BulkSend, UdpClientServer etc. Un utilisateur peut aussi écrire ses propres applications.

Pour les scenarios VANET en utilise UDP. Pour utiliser l'application 'UdpClientServer', un nœud doit être sélectionné comme 'source socket' et un autre comme 'receive sink'.

```
uint16_t Port = 12345;

 Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (71), tid);
InetSocketAddress local = InetSocketAddress ((Ipv4Address::GetAny()), Port);
recvSink->Bind (local);
recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

 Ptr<Socket> source;
source = Socket::CreateSocket (c.Get (23), tid);
InetSocketAddress remote = InetSocketAddress (("10.1.1.72"), Port);
source->Connect (remote);
```

- **Callbacks:**

Main utilise la fonction 'SetRecvCallback' pour la collection de données.

void ReceivePacket (Ptr<Socket> socket){
NS_LOG_UNCOND ("PACKET RECIEVED"); }

- **Logging***:*

Chaque fois que 'recvSink' reçoit un paquet, elle appelle la procédure 'ReceivePacket' qui affiche le message "PACKET RECIEVED". Elle est utilisée pour faire sortir des informations utiles de la simulation.
Cette fonction peut être activée ou désactivée selon le vouloir de l'utilisateur. Le 'logging' et le traçage sont des mécanismes différents. Le 'logging' peut aussi être utilisé par le débogueur pour avoir les informations sur les erreurs qui se produisent durant la simulation.

- **Exécution de la simulation***:*

Une fois que les composants réseau, le canal, les nœuds et les applications sont configurés on passe à la simulation. Dans cette partie en fixe la durée (exemple120S) de la simulation, sinon la simulation s'exécutera à l'infini.

Simulator::Run ();
Simulator::Stop(Seconds (120));

- **La collection de données:**

Parmi les principaux objectifs de la simulation, le traçage. Une des caractéristiques de NS3 est la variété des méthodes de traçage. Exemple : l'utilisation de 'FlowMonitor', l'utilisation de 'Statistical Framework', la méthode traditionnelle etc.

- **La méthode de traçage traditionnelle dans NS3 :**

Cette méthode est la plus connue. Pour le traçage, NS3 propose plusieurs classe 'helper' . NS3 génère deux type de fichiers trace.

a) **Les fichiers '.pcap' (packet capture)** : ces fichiers peuvent être lus avec différents utilitaires de visualisation (Wireshark , tcpdump…).

NS3 génère un fichier .pcap pour chaque composant de la simulation.

WifiPhy.EnablePcapAll ("myfirst");

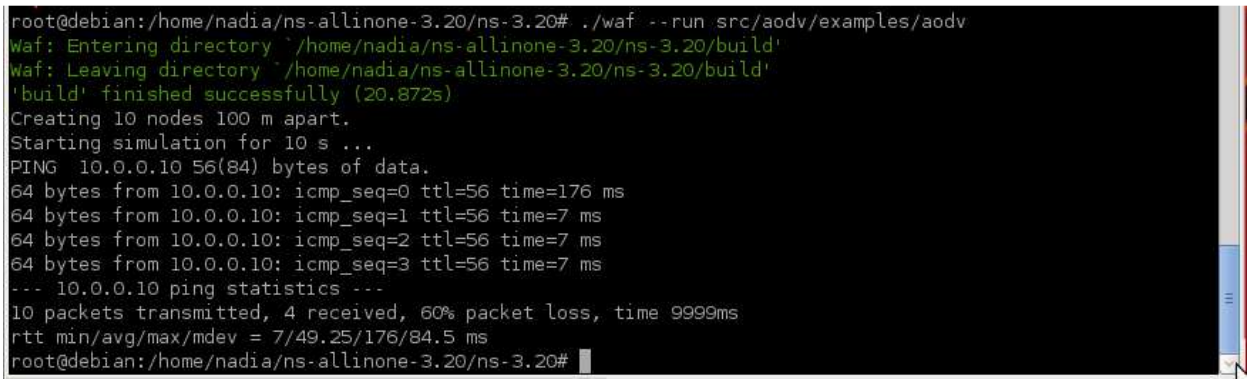
b) **Les fichiers ASCII:** NS3 génère au même temps un fichier '.tr' très similaire à celui généré par NS2. Pour cela NS3 dispose d'une classe helper 'AsciiTraceHelper'.

Les lignes du script suivantes générerons un fichier trace nommé ‘fichiertrace.tr’. Pour extraire les données qui nous intéressent on utilise l’utilitaire ‘AWK’.

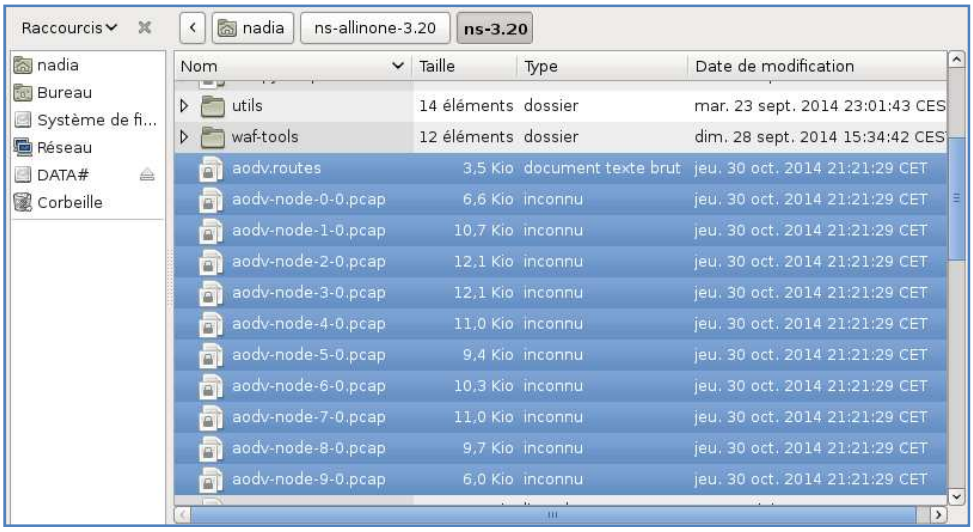
AsciiTraceHelper ascii;
WifiPhy.EnableAsciiAll (ascii.CreateFileStream ("fichiertrace.tr"));

c) Exemple d’utilisation des fichiers trace:

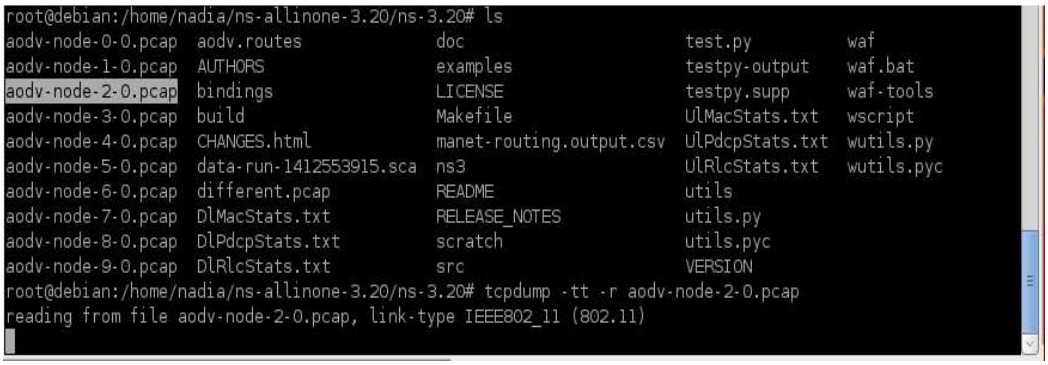
On va exécuter le protocole aodv dans NS3.20. On va dans le répertoire de travail ‘/home/nadia /’ Puis avec la commande ‘cd’ on va vers le répertoire ‘/home/nadia /ns-allinone-3.20/ns-3.20’. puis on lance la commande ‘*./waf –run src/aodv/examples/aodv*’(voir la figure ci-dessous).



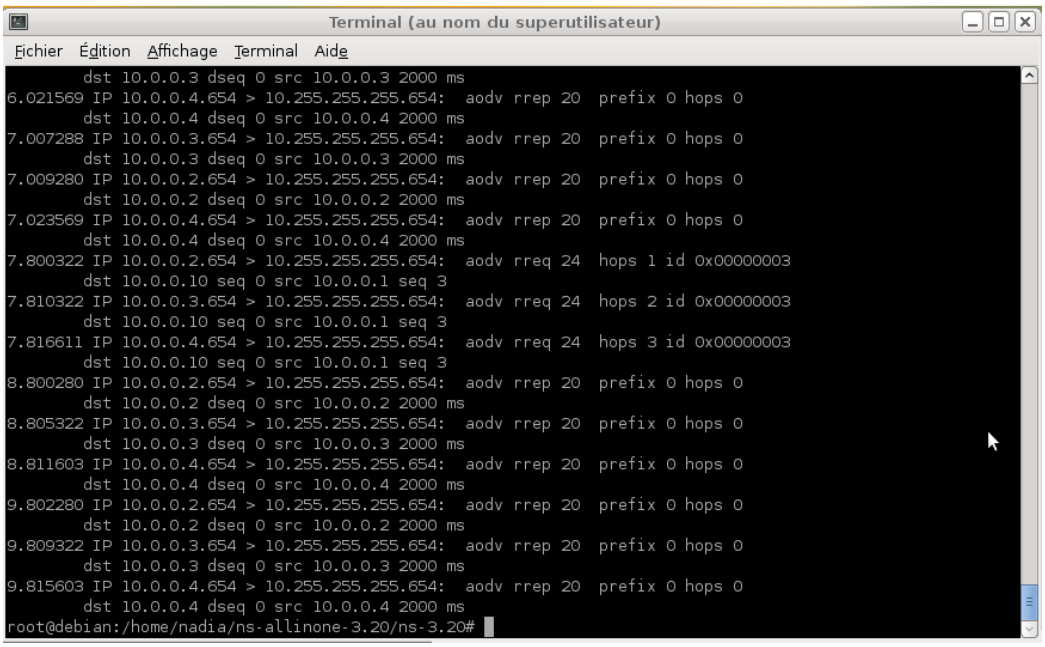
Ensuite, comme sortie de l’exécution, on a les fichiers trace suivant (voir la figure ci-dessous):



Pour lire le contenu du fichier trace on utilise l’utilitaire ‘tcpdump’ avec la commande suivante (voir la figure ci-dessous) :
Tcpdump –tt -r nom-fichier.pcap



Apres la lecture on aura :



- **Visualisation:**

Pour visualiser on utilise la commande suivante :

./waf –run "nom-fichier" --visualize

I.4.inconvénients :

on voit que cette méthode n'est pas l'idéal pour les VANET.

- Trop longue
- Elle demande la maitrise de deux simulateurs.
- Le réseau VANET final n'est pas réaliste c.-à-d. dans la réalité les messages dans le réseau influe sur le trafic. Le réseau n'est pas indépendant du trafic.
- Toutes les études faites en se basant sur cette méthode simplifient soit le scenario du trafic, soit celui du réseau.

Méthode 2 : intégration de l'application 'vanet highway' dans NS3 et simulation d'un réseau vanet avec 'vanet-highway'

I . Présentation de l'application 'VANET-HIGHWAY' pour NS3 :

I.1. introduction :

L'étude des réseaux VANET requière des outils de simulation puissants et efficaces. Parce que les messages envoyés dans le réseau VANET peuvent affecter le mouvement des véhicules et le comportement du conducteur, le modèle de mobilité doit être intégré dans le simulateur réseau.

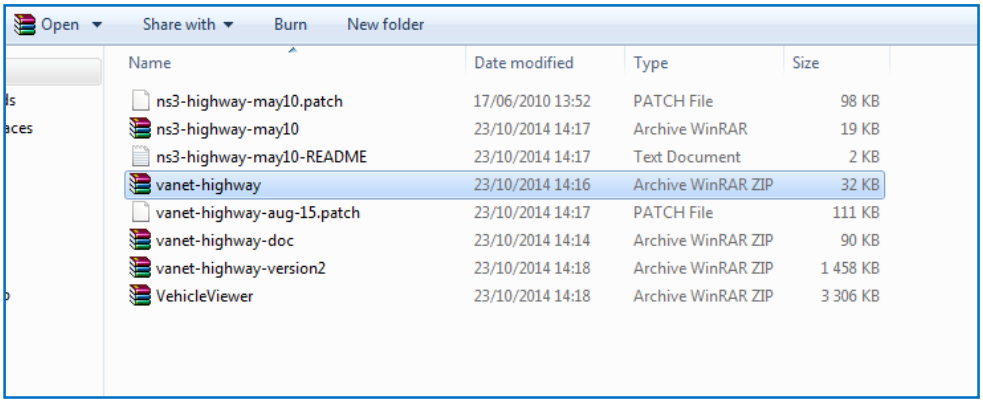
Ce modèle de mobilité des véhicules ne doit pas être générer indépendamment du simulateur réseau comme cela se fait traditionnellement.

Nous allons présenter une application pour les VANET faite pour NS3 (package téléchargeable gratuitement sur le net).

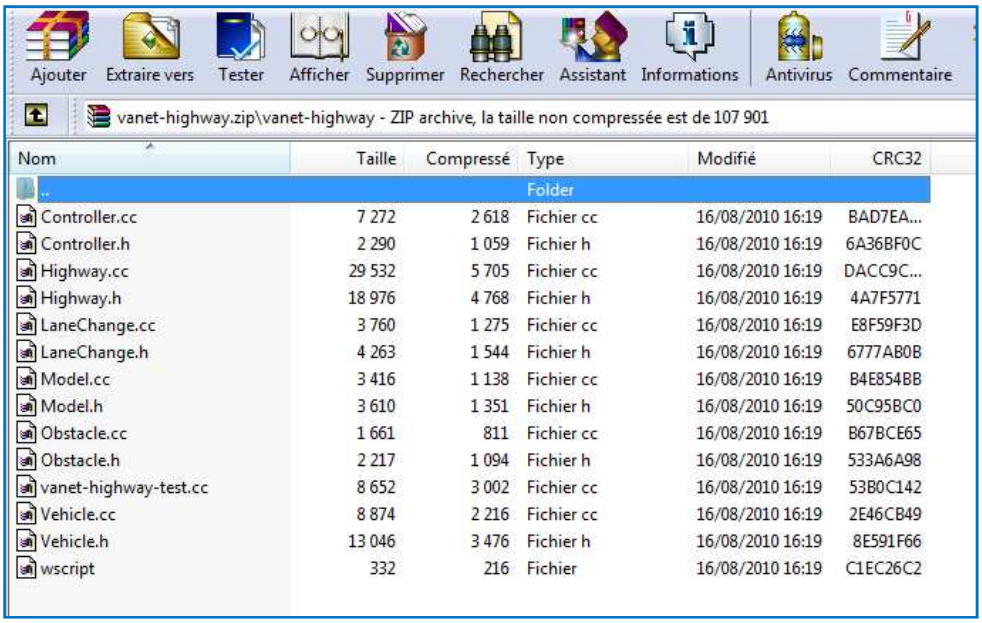
Dans cette application (vanet-highway mobily) le mouvement des véhicules et la communication dans le réseau sont intégrés à travers des événements dans NS3.

Dans cette application le plus petit grain de simulation est une autoroute qu'on peut attacher à d'autres autoroutes pour former un arbre de routes.

I.3.le package de projet vanet-highway : voir figure ci-dessous.



On ouvre le package :



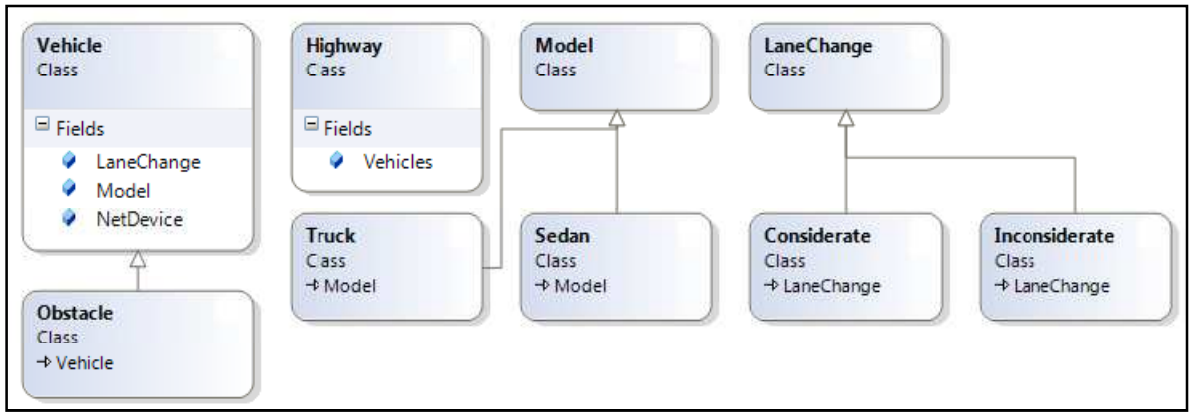
Nom	Taille	Compressé	Type	Modifié	CRC32
..			Folder		
Controller.cc	7 272	2 618	Fichier cc	16/08/2010 16:19	BAD7EA...
Controller.h	2 290	1 059	Fichier h	16/08/2010 16:19	6A36BF0C
Highway.cc	29 532	5 705	Fichier cc	16/08/2010 16:19	DACC9C...
Highway.h	18 976	4 768	Fichier h	16/08/2010 16:19	4A7F5771
LaneChange.cc	3 760	1 275	Fichier cc	16/08/2010 16:19	E8F59F3D
LaneChange.h	4 263	1 544	Fichier h	16/08/2010 16:19	6777AB0B
Model.cc	3 416	1 138	Fichier cc	16/08/2010 16:19	B4E854BB
Model.h	3 610	1 351	Fichier h	16/08/2010 16:19	50C95BC0
Obstacle.cc	1 661	811	Fichier cc	16/08/2010 16:19	B67BCE65
Obstacle.h	2 217	1 094	Fichier h	16/08/2010 16:19	533A6A98
vanet-highway-test.cc	8 652	3 002	Fichier cc	16/08/2010 16:19	53B0C142
Vehicle.cc	8 874	2 216	Fichier cc	16/08/2010 16:19	2E46CB49
Vehicle.h	13 046	3 476	Fichier h	16/08/2010 16:19	8E591F66
wscript	332	216	Fichier	16/08/2010 16:19	C1EC26C2

I.4. Architecture de l'application 'vanet-highway mobility' :

Ici on décrit les composants de l'application vanet-highway, ce modèle est conçu et programmé par HADI.ARBABI<email:marbabi@cs.odu.edu>.et MICHELEC.WEIGLE<email :mweigle@cs.odu.edu>.

Ce package est constitué de 5 classes principales (figure ci-dessous) :

1. ***Vehicle*** : c'est le nœud mobile qui contient les composants de la communication sans fil.
2. ***Obstacle*** : c'est un véhicule sans mobilité.
3. ***Model*** : c'est 'IDM car-following mobility ' modèle.
4. ***LaneChange*** : c'est le modèle qui représente le changement de ligne dans une route.
5. ***Highway*** : 'holds *Vehicle'* et '*Obstacle* objects' et l'utilisation de '*Vehicle*'s *Model' et* '*LaneChange'* propriétés pour le contrôle de la mobilité.



-Figure ch4.3: le diagramme des classes dans le projet highway-

L'application *Highway* utilise les 4 premières classes pour générer le trafic sur une autoroute.

I.4.1. Vehicle

Vehicle est un nœud mobile qui contient les composants de la communication sans fil. La classe *vehicle* contient les propriétés suivantes :

- VehicleID
- width : la largeur du véhicule en mettre.
- length : la longueur du véhicule en mettre
- lane : c'est le numéro de la voie où se trouve le véhicule
- direction : {-1, 1} (convention l'Est est 1 et West est -1).
- position : c'est un vecteur (*x*, *y*, *z*), ou *x* est la position de l'arrière du véhicule, *y* le centre du véhicule, et *z* est l'altitude du véhicule (toutes les unités sont en mettre)
- velocity : m/s
- acceleration : m/s2
- model : l'affectation des valeurs.
- lanechange : l'affectation du changement de la ligne.

Procédures:

- C'est l'objet *highway* qui se charge de la gestion des positions, directions et numéro de la ligne ou se trouve le véhicule.
- On peut poser la valeur de la vitesse ou de l'accélération manuellement ou bien elles peuvent être calculées en se basant sur les règles du modèle de mobilité IDM (modèle existant dans NS3).

- La classe véhicule est capable de changer de ligne si c'est possible et si nécessaire, en se basant sur le modèle '*MOBIL lane change'* .
- L'objet *Vehicle* peut être créé manuellement puis insérer dans *Highway* ou injecter automatiquement.
- Les véhicules peuvent communiquer (send/receive) à travers le standard NS3 'WiFi channels'.
- Dans l'application '*highway'* différents moyens sont mis à la disposition des utilisateurs pour effectuer le traçage.

I.4.2. Obstacle :

La classe '*Obstacle'* est un nœud statique qui contient des composants pour la communication sans fil. Cette classe hérite tout de la classe véhicule sauf la mobilité (i.e., velocity = acceleration = model = lanechange = 0)
Un obstacle peut être utilisé comme une barrière pour fermer une ligne temporairement. Un obstacle peut aussi être utilisé comme station ou une autre infrastructure sur l'autoroute.

Si un obstacle est placé dans l'autoroute, il doit avoir une direction, un numéro de ligne. Tout ce qu'on peut faire avec la classe véhicule on peut le faire avec la classe obstacle (sauf la mobilité). C'est pour cela qu'à partir de cette section on n'utilisera que le terme véhicule.

I.4.3. Le modèle de mobilité :

Model est la classe qui implémente le modèle de mobilité. Le projet highway utilise le modèle Intelligent Driver Model (IDM), ce modèle est basé sur les équations et les paramètres développés par Treiber (Treiber et al. 2000, Treiber 2006).

IDM est un modèle qui suit les véhicules, c'est-à-dire que l'accélération et la décélération d'un véhicule dépend de sa propre vitesse, de la vitesse désirée ainsi que de la position et de la vitesse du véhicule suivi sur la même ligne. Ce véhicule suivi est appelé *front vehicle.*
Chaque véhicule dans IDM a une 'desired velocity', 'safety time headway' (le temps nécessaire pour couvrir la distance entre deux véhicules)…
IDM utilise toutes les informations du véhicule et de son prochain pour calculer la nouvelle accélération.
La fonction '*CalculateAcceleration'* dans la classe *Model* utilise les équations IDM pour calculer et retourner la nouvelle accélération en chaque unité de temps. la nouvelle vitesse du véhicule et sa position sont ajustés en se basant sur la nouvelle accélération.

Pour simuler des expériences spéciales. L'utilisateur peut créer son propre type de véhicule en utilisant les différentes valeurs dans les paramètres d'IDM. Par exemple ajouter des véhicules de sport.

Lane Change Model :

LaneChange est la classe qui implémente le modèle de changement de ligne d'un véhicule dans une autoroute. Cette classe est implémentée en se basant sur les équations et les paramètres développés par Treiber (Treiber 2006b, Treiber et Helbing 2002).
Chaque changement de ligne dans ce modèle doit satisfaire les critères de sécurité et les critères de la mobilité.
Dans cette classe, pour savoir si un conducteur est dangereux ou pas on regarde la valeur de sa civilité. Si $p \geq 1$, le conducteur n'est pas un danger pour les autres.

La fonction *'CheckLaneChange'* dans la classe *'LaneChange'* retourne un booléen pour indiquer si le changement de ligne peut ou ne peut pas s'effectuer.

I.4.5. la classe Highway :

Highway est la classe qui gère les *Véhicules* et leur mobilité. On va discuter maintenant des propriétés physiques de *Highway*

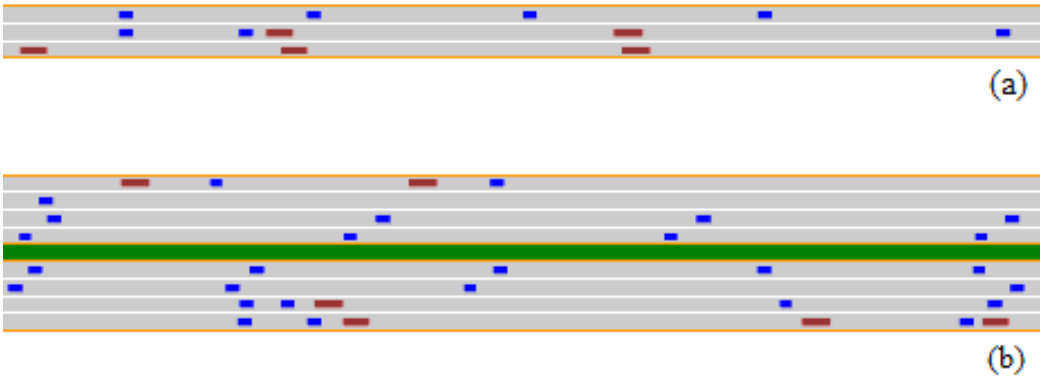
.

I.4.5.1. Propriétés physique :

Highway représente la topologie d'une autoroute droite, et a les propriétés physiques suivantes :

- length : longueur de l'autoroute en mètre (plus de 10.000 m).
- number of lanes : dans chaque direction il y a [1,5].
- lane width : la largeur de la ligne en mètre.
- median gap : largeur de trait qui sépare les deux directions en mètre.
- bidirectional : true si la route contient deux directions, false si la route a une seule direction.

La figure (a) est une autoroute unidirectionnelle avec trois lignes. La Figure (b) est une autoroute bidirectionnelle avec 4 lignes dans chaque direction. Une séparation des deux routes avec le vert.



-Figure ch4.4: un petit segment de l'autoroute implémenté par l'application highway.-

Les véhicules sont représentés par de petits rectangles bleus, les camions sont représentés par des longs rectangles rouges.

I.4.5.2 Gestion de véhicule :

Il y a plusieurs fonctions pour la gestion de véhicules. L'application Highway peut automatiquement créer l'objet véhicule avec ces propriétés, puis insérer ces objets créés dans des lignes et faire circuler le véhicule en accords avec sa mobilité et son model de changement de ligne.

I.4.5.2.1 la création et l'injection automatique des vehicules :

Quand la propriété '*AutoInjection*' du projet highway est sur 'true', un objet véhicule est crée et injecté sur l'autoroute. C'est pour cette raison que l'application 'highway' crée un model de mobilité par défaut avec les propriétés nécessaires pour des véhicules et des camions standards, respectivement 'SedanModel' et 'TruckModel'.

Highway crée aussi un modèle de changement de ligne par défaut avec l'ensemble des propriétés nécessaires pour les véhicules et les camions injectés dans l'autoroute. Le pourcentage des véhicules par rapport au camions est contrôlé par le paramètre *'injectionMix'*. Tout les véhicules crées sont munies automatiquement de 'WiFi Phy/Mac' approprié pour les VANET.

Le projet Highway enregistre chaque ligne dans une structure liste. Quand un objet véhicule est ajouté à l'autoroute, il est inséré dans sa propre place en se basant sur sa ligne, sa direction et position x.

Dans le cas de l'auto injection, il y a un paramètre 'minGap' qui spécifie la distance minimum entre deux véhicules qui entrent dans l'autoroute. Les véhicules sont ajoutés avec le même pourcentage dans les deux sens de l'autoroute.

I.4.5.2.2 la mobilité des véhicules :

Chaque '*DeltaT*' seconde, highway appelle les fonctions qui mettent à jour la position x, la vitesse et l'accélération de chaque véhicule en se basant sur le modèle de mobilité.

La probabilité qu'un véhicule change de ligne est évalué chaque '10 * *DeltaT*' secondes.

Le temps de réaction d'un conducteur est de 0.7 seconde (Green, 2000). La position des véhicules doit être m-à-j plus souvent que le temps de réaction d'un conducteur. C'est pour cela que la valeur par défaut de '*DeltaT*' est 0.1 secondes.

I.4.5.3. Le contrôle des véhicules par l'utilisateur l'application highway :

L'application highway permet à l'utilisateur de modifier certains aspects de son fonctionnement selon les résultats obtenus c.-à-d. on peut passer du réseau au model de mobilité pour obtenir des résultats. De cette façon le programme (script) de l'utilisateur peut interagir avec l'objet véhicule.

- L'application *Highway* permet à l'utilisateur d'accéder à n'importe quel objet véhicule à travers la variable *'VehicleID'* en utilisant la fonction ' *FindVehicle()'*.

- La fonction '*FindVehiclesInRange()'* retourne la liste de tout les objet véhicules. La fonction '*FindVehiclesInSegment()'* retourne la liste des véhicules se trouvant dans le tronçon de route *x1 - x2*.

- Pour accéder à un objet véhicule dans un temps 't', l'application highway dispose d'événements comme *'InitVehicle'*, '*ControlVehicle'*, et '*ReceiveData'* .

- Les événements '*DevRxTrace'* et '*PhyRxErrorTrace'*, sont prévus pour le traçage de la communication sans fil et le comportement des composants réseau des véhicules.

- L'événement *'InitVehicle'* est prévu pour l'initialisation.

Toutes ces fonctions et événements donnent la possibilité à l'utilisateur de contrôler le scenario VANET et de modifier les affectations initiales. De cette façon l'utilisateur peut créer et positionner un véhicule à n'importe quel moment.

II.1. L'installation de l'application 'vanet-highway' dans NS3 : on a besoin de

- télécharger le package spécial vanet-highway. Malheureusement on ne peut pas utiliser NS3.20. Il n'est pas compatible avec l'application vanet-highway. Donc il faut télécharger NS3.8

- installer vanet-highway.

Etape 1 : télécharger et installer NS3.8

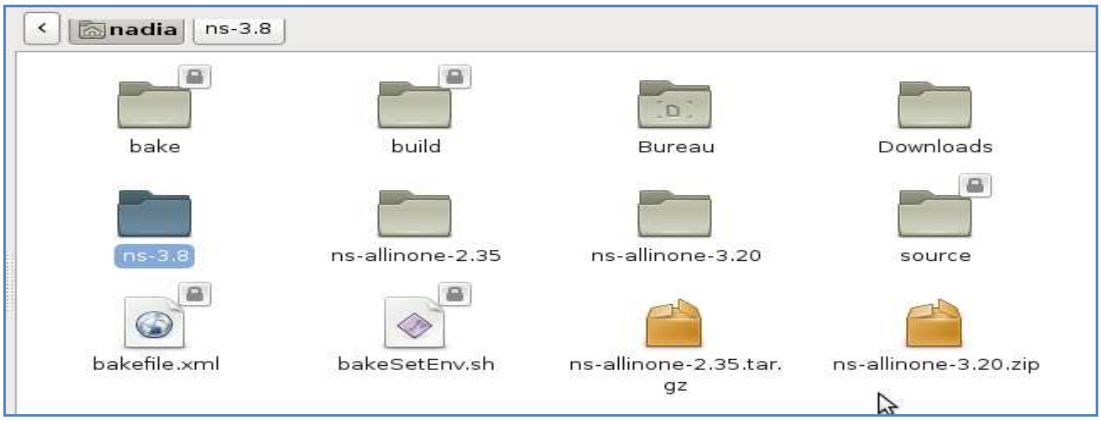
Remarque : on n'a pas besoin d'installer NS3.8, il suffit de ne pas désinstaller NS3.20 et de télécharger NS3.8 dans le même répertoire et la construction se fera automatiquement après le téléchargement. Pour télécharger NS3.8 on connecte la machine à internet puis on exécute la commande suivante.

hg clone http://code.nsnam.org/ns-3.8

On verra sur le terminal :

« … [765/943] cxx: src/helper/wifi-helper.cc -> build/debug/src/helper/wifi-helper_1.o
[766/943] cxx: src/helper/olsr-helper.cc -> build/debug/src/helper/olsr-helper_1.o
[767/943] cxx: src/helper/point-to-point-helper.cc -> build/debug/src/helper/point-to-point-helper_1.o
[768/943] cxx: src/helper/csma-helper.cc -> build/debug/src/helper/csma-helper_1.o
[769/943] cxx: src/helper/mobility-helper.cc -> build/debug/src/helper/mobility-helper_1.o
...
/contrib/topology-read/inet-topology-reader_1.o build/debug/src/contrib/topology-read/orbis-topology-reader_1.o -> build/debug/libns3.so
[943/943] cxx_link: build/debug/vanet-highway/vanet-highway-test_1.o build/debug/vanet-highway/Highway_1.o build/debug/vanet-highway/Controller_1.o build/debug/vanet-highway/Vehicle_1.o build/debug/vanet-highway/Obstacle_1.o build/debug/vanet-highway/Model_1.o build/debug/vanet-highway/LaneChange_1.o -> build/debug/vanet-highway/vanet-highway-test
Waf: Leaving directory `/home/nadia/ns-3.8/build'
*'**build' finished successfully** (9m4.455s) »*

- Une fois téléchargé on voit le répertoire NS3.8 dans notre répertoire de travail (voir la figure ci-dessous) :

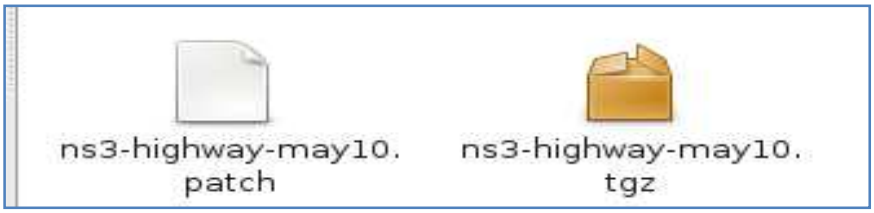


Etape2 : télécharger les packages manquants

Maintenant qu'on a terminé avec NS3.8. On va télécharger sur internet tous les packages suivant.

- Puis on ouvre le dossier NS3.8 et on colle le patch *'highway-may10.tgz'* à l'intérieur :
- Puis on va deziper le patch avec la commande suivante:
  (Voir la figure ci-dessous) :

  tar -xzvf ns3-highway-may10.tgz



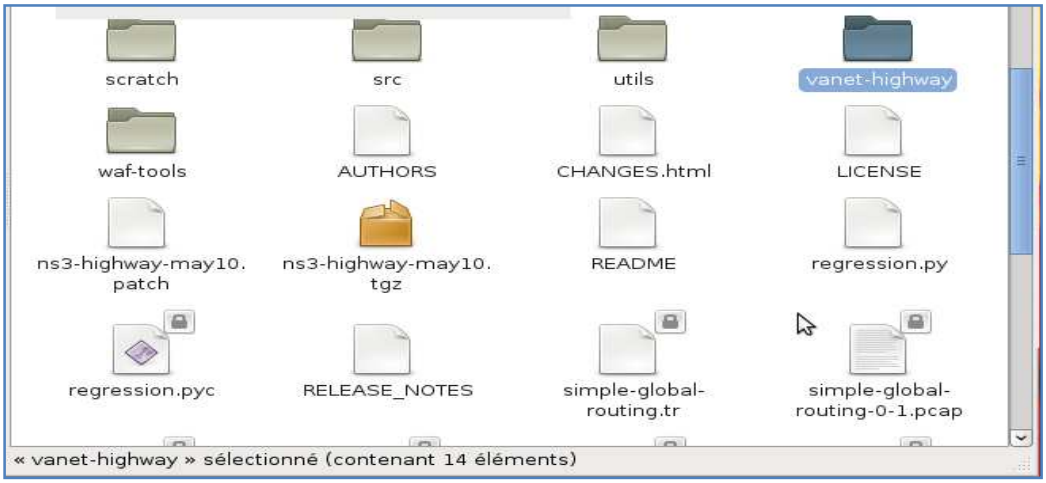
- appliquer le patch dans le répertoire ns-3.8 avec la commande suivante :

hg import --no-commit ns3-highway-may10.patch

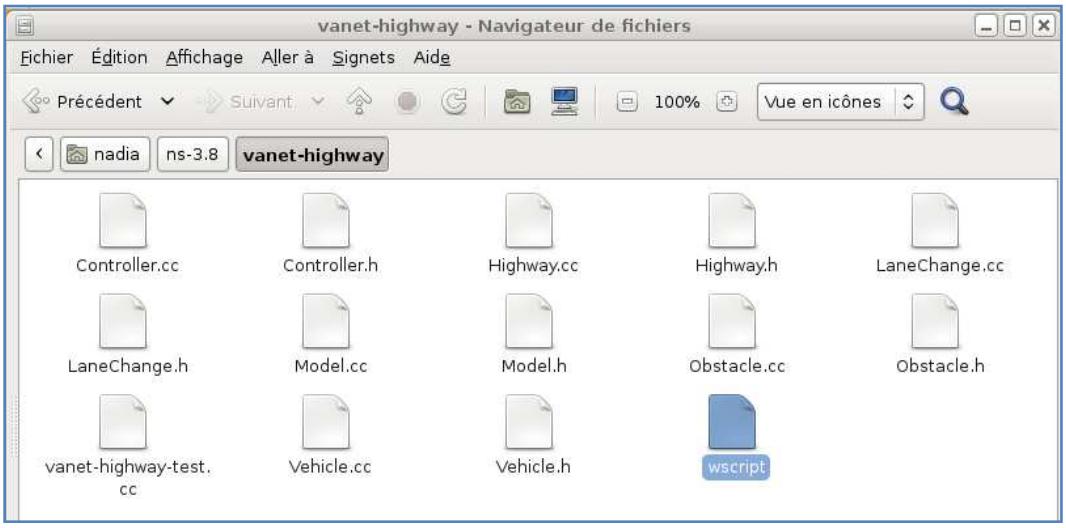
Le patch va créer les fichiers suivant dans le dossier 'vanet-highway'. (Ce patch est compatible avec ns-3.8, ns-3.6 et ns-3.7). Il n'est pas compatible avec ns3.20:

vanet-highway/Controller.{cc,h}
vanet-highway/Highway.{cc,h}
vanet-highway/LaneChange.{cc,h}
vanet-highway/Model.{cc,h}
vanet-highway/Obstacle.{cc,h}
vanet-highway/Vehicle.{cc,h}
vanet-highway/vanet-highway-test.cc
vanet-highway/wscript

On verra apparaitre le dossier 'vanet-highway' dans le répertoire NS3.8 (voir les figures ci-dessous):



Le contenu du package :



- avant de modifier le script dans NS3.8. On doit changer les permissions de système pour pouvoir écrire dans le script de NS3.8. Pour cela on exécute la commande suivante :

  chmod -R a+rwx NS3.8

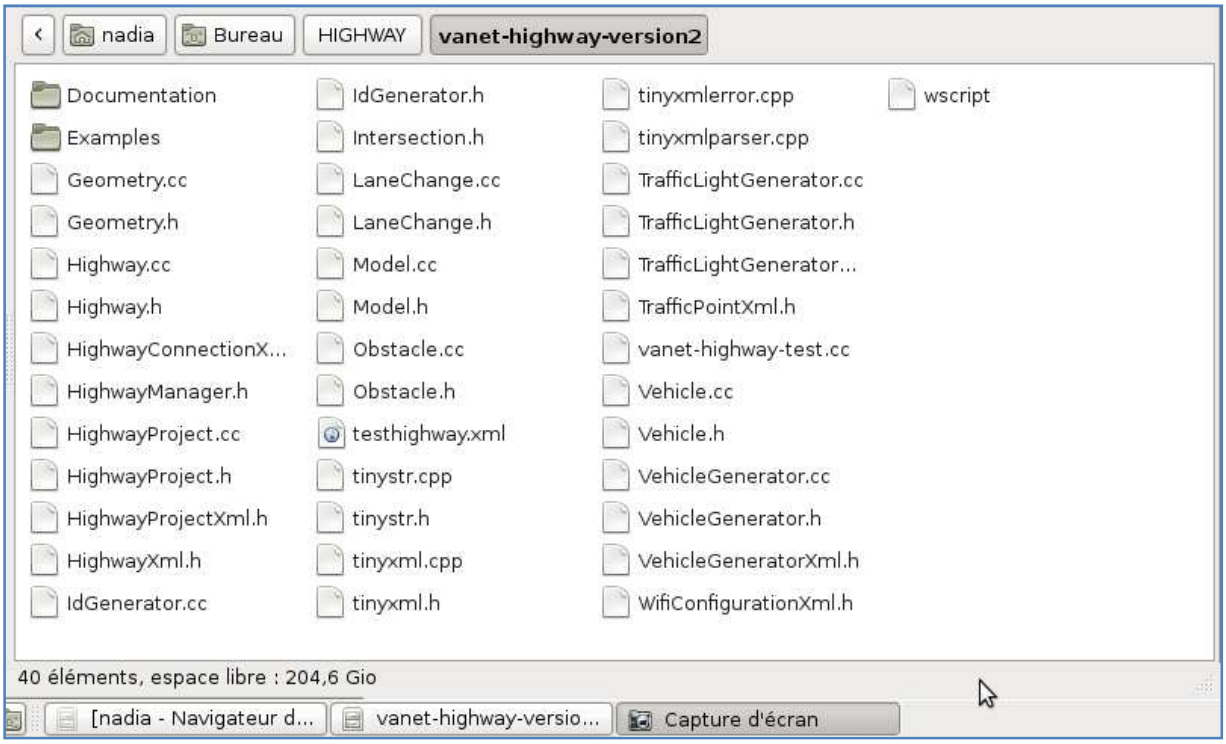
- puis on ajoute la ligne suivante du programme dans le scripte de NS3.8. Puis on clique sur 'enregistrer' dans le menu fichier. Après, l'application 'vanet-higway' est intégré dans NS3.8 (voir la figure ci-dessous) :

  bld.add_subdirs('vanet-highway') // ligne du code à ajouter dans le script

```
bld.all_envs['default'] = variant_env

# process subfolders from here
bld.add_subdirs('src')
bld.add_subdirs('samples')
bld.add_subdirs('utils')
bld.add_subdirs('vanet-highway')
add_examples_programs(bld)
add_scratch_programs(bld)

## if --enabled-modules option was given, we disable building the
## modules that were not enabled, and programs that depend on
## disabled modules.
env = bld.env

if Options.options.enable_modules:
```

Python Largeur des tabulations: 8 Lig 565, Col 37 INS

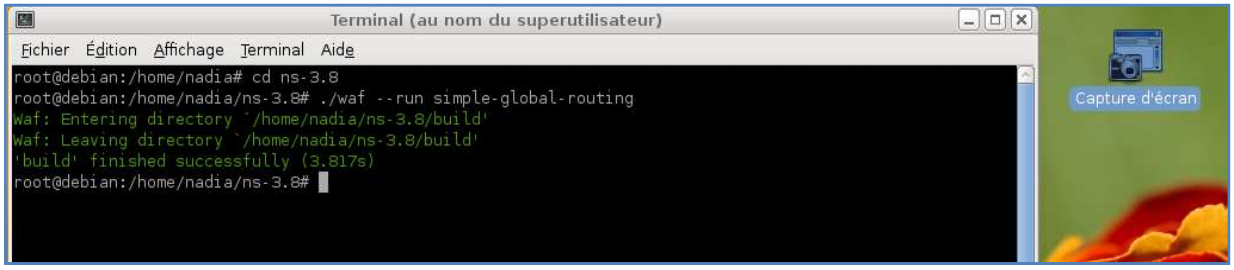
- pour utiliser la nouvelle version de l'application 'vanet-highway' il suffit de remplacer le contenu du dossier 'vanet-highway' par le contenu du dossier 'vanet-highway-version2' (voir la figure ci-dessous) :



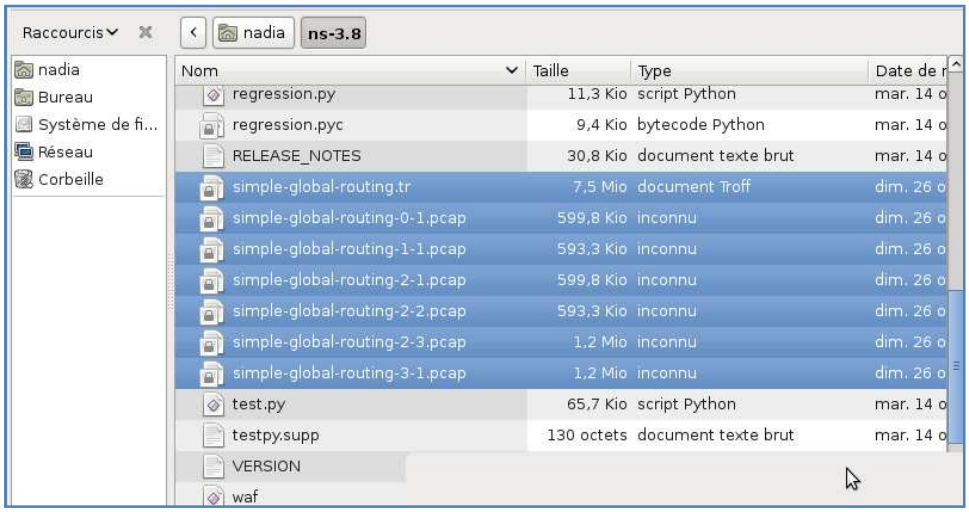
II.2.Exécuter un script dans NS3.8 :

Pour tester l'installation de NS3.8. On exécute le script 'simple-global-routing' dans le répertoire NS3.8 avec la commande suivante :

./waf --run simple-global-routing



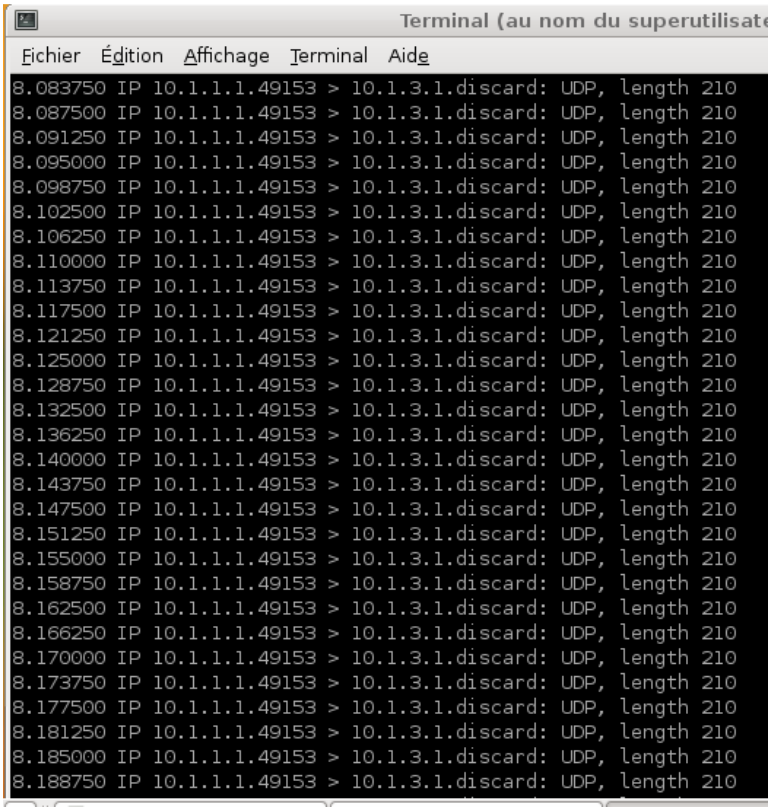
L'exécution est réussi et a produit des fichiers trace (voir figure ci-dessous) :



Pour voire le contenu des fichiers trace. On utilise la commande suivante (voir la figure ci-dessous) :

```
root@debian:/home/nadia/ns-3.8# tcpdump -tt -r simple-global-routing-0-1.pcap
```

On verra sur le terminal ceci (voir la figure suivante) :



II.3. simulation d'un scenario vanet avec l'application 'vanet-highway'

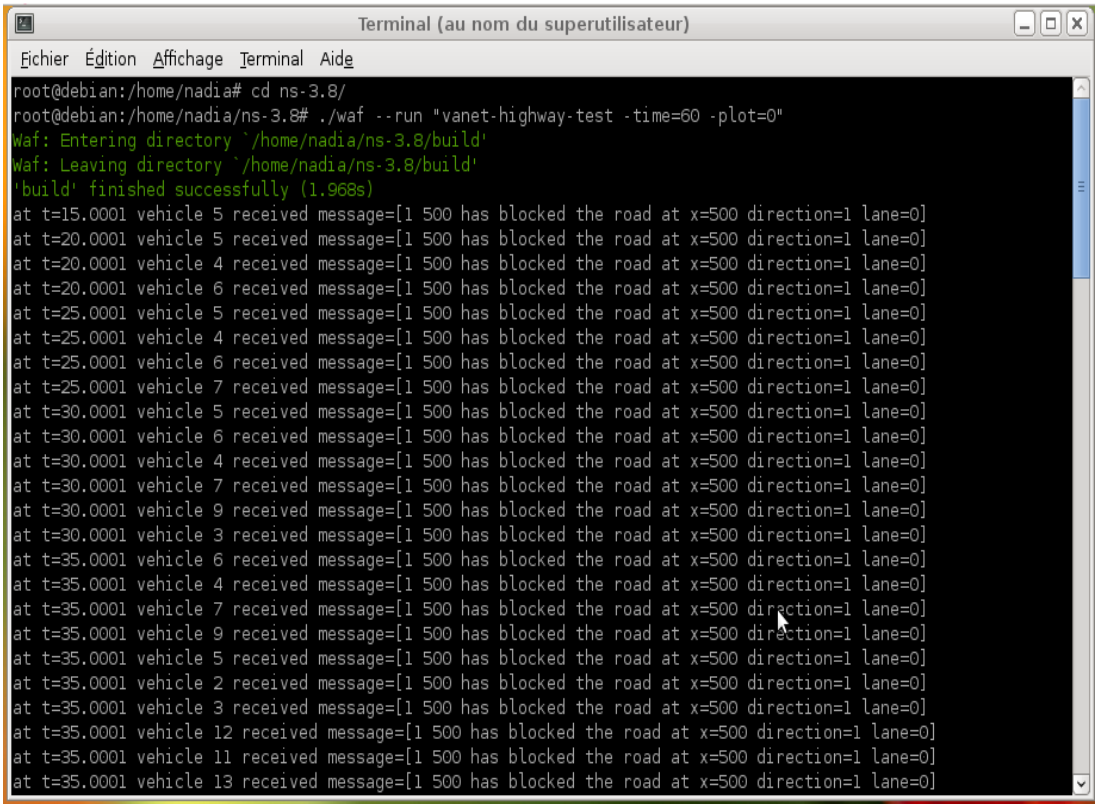
Description du scenario :

On veut placer des véhicules obstacles pour simuler une congestion pour voir le comportement du réseau.

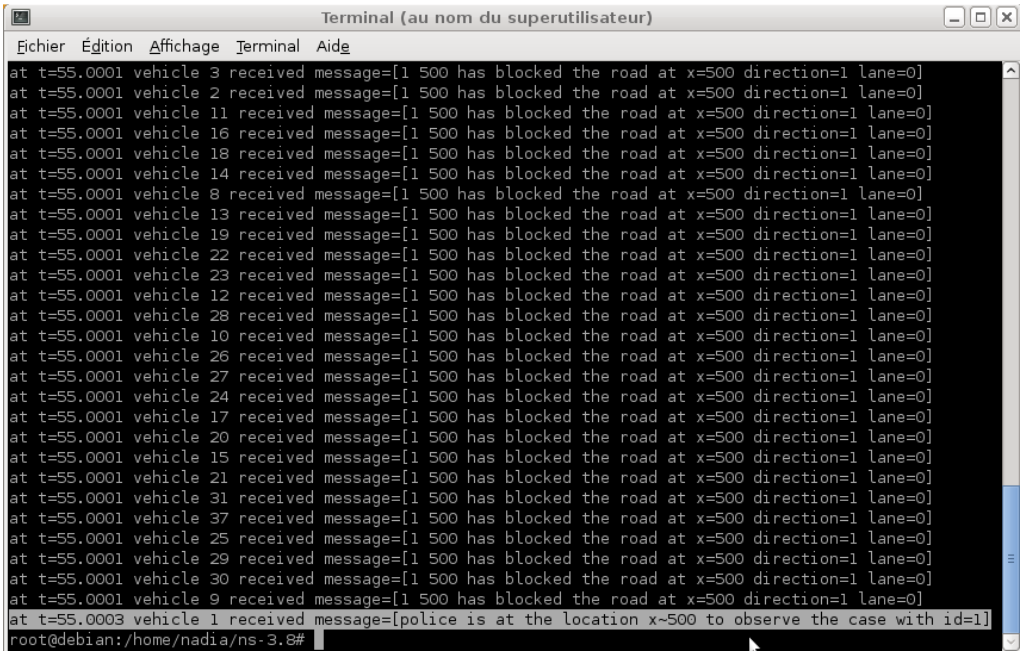
L'utilisateur peut avoir le contrôle total des événements pour produire le scenario désiré. Cet exemple génère des fichiers de sortie qu'on peut transformer en graphe avec *gnuplot* ou un autre outil de visualisation.

- La classe '*Controller'* permet de manipuler les événements pour la création de véhicules spéciaux.
- L'autoroute dans cet exemple est bidirectionnelle : 1KM avec deux voies dans chaque direction.
- La largeur d'une voie = la largeur de la ligne séparatrice= 5 mètre.
- Le pourcentage *'sedan-truck'* est de 80% c.-à-d. 80% de véhicules et 20% de camions.
- Les véhicules seront injectés automatiquement, au maximum, chaque 10 mètres.
- On place un véhicule obstacle dans le milieu de l'autoroute (lane=0, direction=1, x=500). Ce véhicule obstacle diffusera un message de sécurité toutes les 5 secondes.
- On a aussi un véhicule de police avec ID=2. Le véhicule de police a des caractéristiques différentes, il est plus rapide qu'un véhicule normal, sa transmission sans fil a une plus grande portée que les autres véhicules. Il peut écouter et rependre à tous les messages. Le véhicule de police ralentit à l'approche du véhicule obstacle puis il s'arrête.
- Sur le terminal on exécute le script du scenario avec la commande :

./waf --run ''vanet-highway-test -time=60 -plot=0''

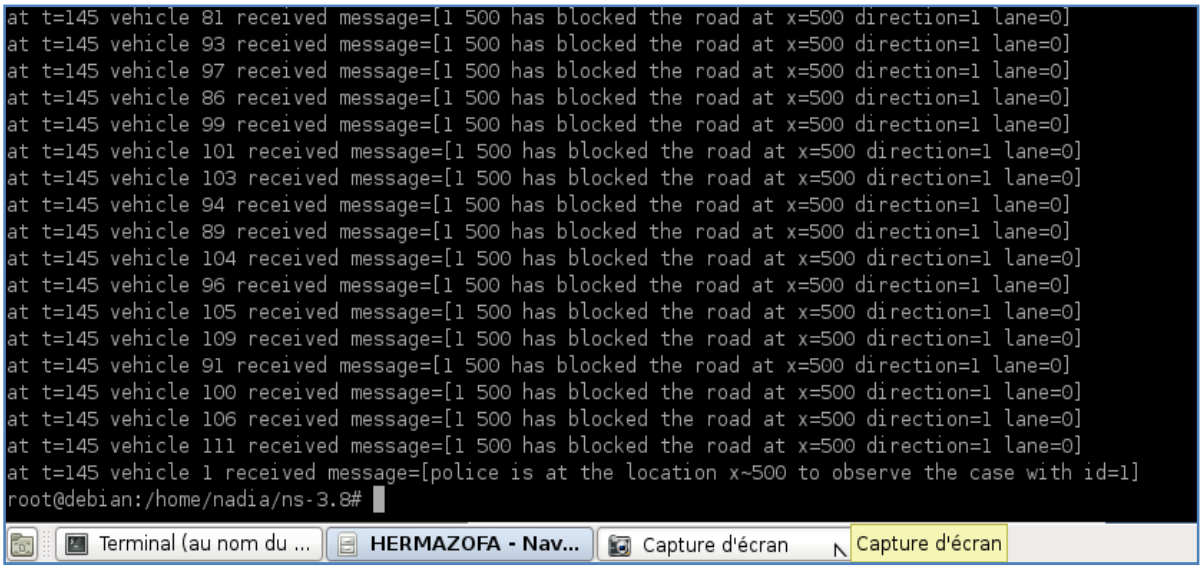


Suite (voir la figure ci-dessous):



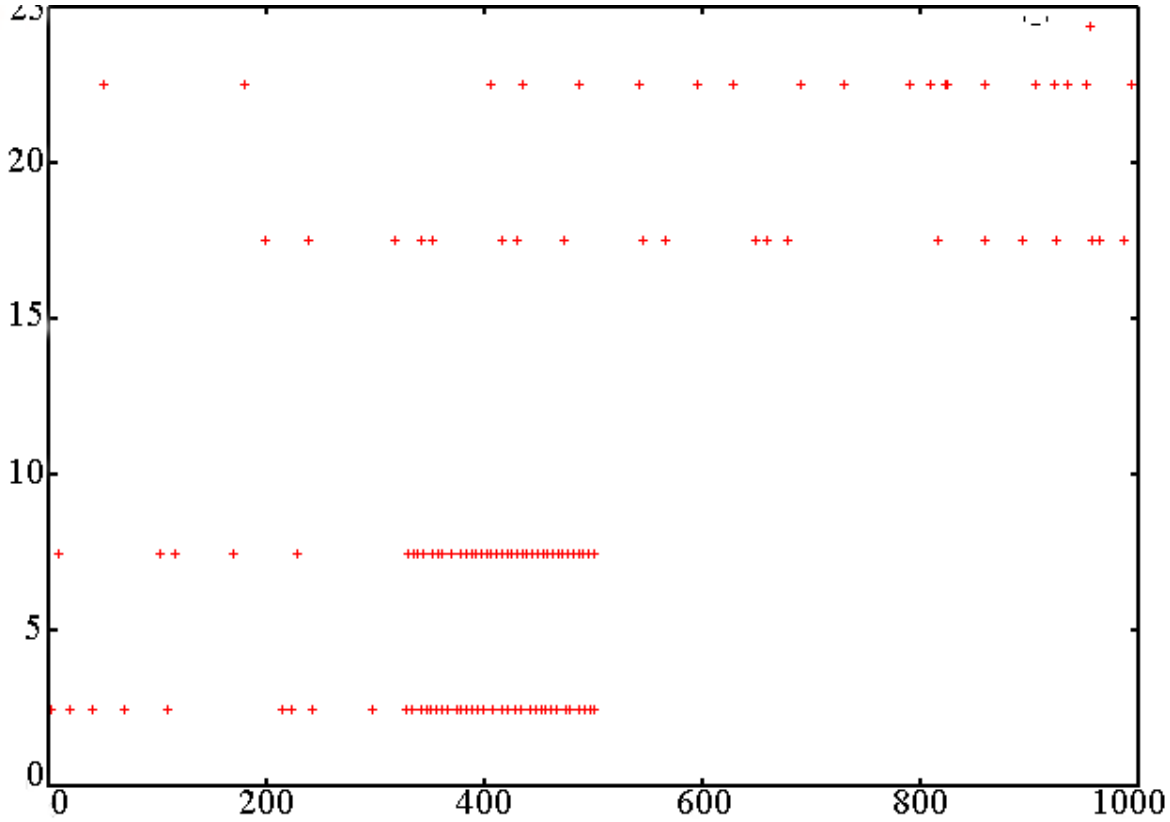
- Pour voir le phénomène de l'injection automatique on lance la simulation 150 seconde.

On remarquera que plus de 109 véhicules sont injectés dans l'autoroute par l'application 'vanet-highway' jusqu'à ce que la voiture de police bloque la circulation pour régler le problème de la voiture obstacle (voir la figure suivante).



- Les sorties de la simulation sont dirigées vers *gnuplot* puis affichées et animées. La Figure ci-dessous montre le graphe (après 2 minutes et 40 secondes de la simulation).

La voiture de police atteint la voiture obstacle à 500 mètres après 20 secondes et s'arrête dans la deuxième voie ce qui provoque une congestion.



-Figure ch4.5: visualisation du scénario-

Interprétation du graphe :

Les signes + représentent des véhicules.

On voit sur la ligne 0 (entre y=0 et y=5) de l'autoroute : le véhicule obstacle a créé un blocage de circulation entre 350m et 500m.

On voit sur la ligne 1 (entre y=5 et y=10) de l'autoroute : l'intervention de la voiture de police a créé un blocage de circulation entre 350m et 500m.

Sur la ligne 2 (entre y=15 et y=20) de l'autoroute et sur la ligne 3 (entre y=20 et y=25) de l'autoroute : la circulation est fluide car les véhicules ne sont pas touchés par le blocage.

Conclusion :

Dans ce chapitre on a vu comment installer NS2 et NS3, étudier l'architecture et le fonctionnement de l'application 'vanet-highway'. Avant de l'intégrer dans le simulateur NS3 pour exécuter un scenario VANET.

Conclusion générale:

La réalisation de ce modeste travail nous a permis de découvrir le domaine des réseaux ad hoc véhiculaire VANET dont on ignorait jusque là l'existence. C'est un nouveau type de réseau issu des réseaux ad hoc mobiles (MANET). Leur particularité provient des communications qui peuvent s'échanger entre véhicule et véhicule ou bien entre véhicule et une infrastructure situé au bord de la route.

Aussi, nous avons pu connaitre et maitriser des outils conçus spécialement pour la simulation des réseaux à l'instar de NS2 et NS3 (network simulateur). C'est des logiciels très complexes et l'unique documentation existante est en anglais. Mais en tant qu'étudiante en informatique l'étude (théorique et pratique) des simulateurs NS2 et NS3 est une chance qu'on ne peut que saisir.

L'application 'vanet-highway' n'est qu'un début dans le domaine de la simulation du trafic routier. Le plus petit grain de simulation est une autoroute qu'on peut attacher à d'autres autoroutes pour former une carte routière complète.

Nous estimons que l'objectif initialement fixé est atteint surtout que la réalisation de ce travail a pu nous instruire dans ce vaste domaine.

Perspectives :

Comparaison d'algorithmes de routage dans NS3 en utilisant l'application 'vanet-highway' pour simuler le trafic routier.

Les principales références :

[1] S . Rimour, « Généralités sur les réseaux : chapitre 1 »

[2] Tayeb Lemlouma, « Le routage dans les réseaux mobiles ad hoc »

[3] Sorin Paun Laurentiu « Gestion de la mobilité dans les réseaux ambiants » Thèse de doctorat, Institut National Polytechnique de Grenoble 2005.

[4] Jérémie Defaye, « les différents types de réseaux sans fil » Conservatoire des arts et métiers Rhône –Alpes- Centre de Lyon 2007.

[5] P Santi and D Blough, « The critical transmitting range for connectivity in sparse wireless ad hoc networks ». IEEE Transactions on Mobile Computing 2003.

[6] Thése de magister : M^{r} Meraihi Yassine : routage dans les reseaux vehiculaires (vanet)cas d'un environnement type ville

[7] Jérôme Vandermeerschen « Hybridation entre les modes Ad Hoc et infrastructure dans les réseaux de type Wifi-Fi »

[8] Hassnaa Moustafa « Routage unicast et multicast dans les réseaux mobiles Ad Hoc » Thèse doctorat.

[9] Touati nadia « intégration de la q.o.s dans le protocole DSR » mémoire d'ingénieur.

[10] R. Meraihi, Mohamed Senouci, Moez Djebri « Réseau mobile Ad Hoc et réseaux de
capteurs sans fil »

[11] James Bernsen, and D. Manivannan, "Unicast Routing Protocols for Vehicular Ad Hoc Networks: A Critical Comparison and Classification"

[12] Magnus Frodigh, Per Johansson and Peter Larsson. « Wireless ad hoc networking –The art of networking without a network ».

[13] Tayeb Lemlouma, « Le routage dans les réseaux mobiles ad hoc »

[14] Amina Naimi-Meraihi « Délai de routage dans les réseaux Ad Hoc 802.1 » Thèse doctorat.

[15] Fabrice Theolyeyre, Fabrice Valois « Routage Hybride sur structure virtuelle dans un
réseau mobile Ad Hoc »

[16] A. Munaretto, H. Badis, K. Al Agha and G. Pujolle. A Link-state QoS Routing Protocol for Ad Hoc Networks. In the proceedings of IEEE MWCN2002, Stockholm,
Sweden, September 2002.

[17] Moez JERBI « Protocoles pour les communications dans les reseaux de vehicules en environnement urbain :
Routage et GeoCast bases sur les intersections » thése de doctorat

[18] Draft-ietf-manet-dsr-10.text
www.ietf.org/ietf/lib-abstracts.txt (internet-drafts)

[19] ns-3 tutorial *Tom Henderson University of Washington(en L. englais)*

[20] NS 3 gustavo J A M. pdf (inescporto)(en englais)

[21] ns-3 Tutorial Release ns-3.18.116 novembre 2013 (119 page en L.englais très bonne référence)

[22] Modeling and Simulating ITS Applications with iTETRIS_
Jérôme Härri, EURECOM,Sophia-Antipolis, France

[23] iTETRIS - A Large Scale Integrated Simulation

Platform for V2X Communications: Application
to Real-Time Traffic Management
V. Kumar1i, R. Bauza2 (pdf trouvé sur le net)

[24] mohamed yacine ghamri doudane **contributions a l'amelioration de l'utilisation** des ressources dans les reseaux de paquets sans fi

[25] simulation d'une extension de mobile ip sous network simulator Djaoumel Ramdan, Merdjane Tahar Mémoire d'ingénieur 2005.

[26] Autour de la réservation de bande passante dans les réseaux ad hoc Clude Chaudi 2004. (thèse de doctorat)

[27] Linux initiation et utilisation Jean-Paul Armspach, Olinpierre, Frederique Ostre-Waerze

[28] Programmation système en c sous linux
Christophe Blaess [Eyrolles]

[29] NS-2: Principes de conception et d'utilisation par P. Anelli & E. Horlait

[30] The *ns* Manual The VINT Project

[31] C++ par John R. Hubbard (Mini Schaum's)

[32] cours C++ de Patrick Trau, Université Louis Pasteur Strasbourg, email : http://www-ipst.u-strasbg.fr/pat/program/cpp/Patrick.Trau (à) http://www-ipst.u-strasbg.fr/pat/program/cpp/ipst-ulp.u-strasbg.fr)

[34] Tutorial for the network simulator "ns" devlopé par Marc Greis Marc puis maintenu VINT group .

[35] [R13]Leila Toumi(these de doctorat) : Algorithmes et mécanismes pour la qualité de service dans des réseaux hétérogènes