# Análisis de Capacidad en Redes VANET Implementando Procedimientos de Asignación de Espectro y Control de Potencia


Daniel Felipe Méndez Bonilla
Anderson David Rodríguez Narváez

**ANEXOS**

**Monografía presentada como requisito para optar por el título de Ingeniero en Electrónica y Telecomunicaciones**

Director: Víctor Fabián Mirama Pérez

*Universidad del Cauca*
**Facultad de Ingeniería Electrónica y Telecomunicaciones**
**Departamento de Telecomunicaciones**
**Línea de Investigación de Gestión integrada de redes, servicios y arquitecturas de comunicaciones**
Popayán, Agosto de 2014

# TABLA DE CONTENIDO

# LISTA DE TABLAS

# LISTA DE FIGURAS

# ANEXO A

## A.1 GUIA INSTALACION HERRAMIENTAS DE SIMULACION EN LINUX

### A.1.1 Instalación de SUMO

1) Descargar la versión más estable y reciente de esta página http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Downloads

2) Por lo general el archivo se guarda en la carpeta /home/user/Downloads, en este caso es /home/Usuario/Descargas

***$ cd***

***$ cd Descargas***

3) Instalar los siguientes paquetes Para integrar SUMO con OMNET y VEINS.

***$ sudo aptitude install bison flex build-essential zlib1g-dev tk8.4-dev blt-dev libxml2-dev sun-java6-jre libpcap0.8-dev autoconf automake libtool libxerces-c2-dev proj libgdal1-dev libfox-1.6-dev***

***$ sudo apt-get install libgdal1-dev proj libxerces-c2-dev***
***$ sudo apt-get install libfox-1.6-dev libgl1-mesa-dev libglu1-mesa-dev***

- Importante: Ubuntu 12.04 no se distribuye con ***libgdal.so***, solo con ***libgdal1.7.0.so.*** Por lo que debe crearse un enlace simbólico:

***$ sudo ln -s /usr/lib/libgdal1.7.0.so /usr/lib/libgdal.so***

- Después de realizar este paso se continua con la instalación del resto de paquetes

***$ sudo apt-get install build-essential gcc g++ bison flex perl \ tcl-dev tk-dev blt libxml2-dev zlib1g-dev default-jre \ doxygen graphviz libwebkitgtk-1.0-0 openmpi-bin libopenmpi-dev libpcap-dev***

- En este caso instalamos todos los paquetes mencionados, cuando aparezca la opción [s/n] damos S e instalamos. Recordar que sin estos prerrequisitos ninguno de los tres programas funcionara.

- Descomprimir el paquete de SUMO (se supone que se está en el directorio donde se descarga):

***$ tar -xzvf sumo-src-0.17.1.tar.gz***

4) Mover la carpeta que fue descomprimida al directorio /usr/local/src con:

***$ sudo mv -v sumo-0.17.1 /usr/local/src***

5) Dirigirse al directorio donde se movió, se configura y se hace la respectiva instalación:

***$ cd /usr/local/src/sumo-0.17.1***

***$ ./configure --with-fox-includes=/usr/include/fox-1.6*** \
***--with-gdal-includes=/usr/include/gdal --with-proj-libraries=/usr*** \
***--with-gdal-libraries=/usr --with-proj-gdal***

***$ make***
***$ sudo make install***

6) Finalmente ya instalado, se procede a ejecutar el programa por medio de líneas de comando o por interfaz gráfica:

- Por terminal o líneas de comando:

***$ sumo***

- Ejecutar SUMO con interfaz gráfica

***$ sumo-gui***

### A.1.2 INSTALACIÓN DE OMNET++

1) Se cierra el terminal y se abre nuevamente, se procede a descargar la versión más reciente y estable desde http://omnetpp.org/, en este caso fue omnetpp-4.3-src gz .

2) Se va al directorio en donde se descargó, en este caso el archivo estaba ubicado en la carpeta personal así que no hay que hacer nada más, en caso contrario es recomendable mover el archivo a la carpeta personal así de esta forma se tiene la carpeta de OMNET separado de otros archivos ya que así es más ordenado.

3) Se descomprime y se ejecuta las siguientes líneas:

***$ tar xvfz omnetpp-4.3-src.tgz***

***$ . setenv***

4) abrir y editar el archivo .bashrc

***$ gedit ~/.bashrc***

5) Al final del archivo se añade la siguiente línea y se procede a guardar los cambios:

***export PATH=$PATH:$HOME/omnetpp-4.3/bin***

6) se cierra y se abre nuevamente la terminal para que tomen efecto los cambios.

7) Ahora hay que ir al directorio donde se descomprimió OMNET++ de nuevo y realizamos la configuración e instalación:

***$ cd omnetpp-4.3***

***$ . /configure***

8) Todo debería salir normalmente, sin embargo es probable que al final aparezca una advertencia que dice que no se ha encontrado la librería TCL que es la que maneja la parte gráfica. Para corregir el problema se teclea en la terminal lo siguiente:

***$ gedit ~/.bashrc***

- Al final del archivo se añade la siguiente línea y se guarda los cambios:

***export TCL_LIBRARY=/usr/share/tcltk/tcl8.5***

- Nota: esta línea aparece en el mensaje de error que da la configuración

9) Se configura nuevamente y el problema estará solucionado:

***$ ./configure***

10) Nota: se despliega en pantalla que falta una librería llamada akaroa, pero esta es opcional así que por ahora no es necesaria en la instalación.

11) Se ejecuta la instrucción make:

***$ make***

- El proceso puede demorar dependiendo de la capacidad del computador, si se quiere aprovechar los múltiples procesadores (ejemplo un portátil i5, con 6 núcleos de procesamiento) se añade al comando make -j2 así:

***$ make -j2***

12) Para probar que omnet funciona:

***$ cd samples/dyna***

***$ ./dyna***

- Debe desplegarse el ejemplo con la interfaz gráfica y funcionando normalmente

13) . Para entrar a OMNET por el terminal:

***$ omnetpp***

14) Para añadir un icono para ingresar por la barra de menú o por un icono de escritorio, se añade uno de estos comandos o ambos respectivamente

***$ make install-menu-item***

***$ make install-desktop-icon***

### A.1.3 INSTALACIÓN DE VEINS

Terminado el paso anterior, se cierra y se abre nuevamente el terminal

1) Se Descarga la versión más reciente que en este caso es veins-2.1, s descomprime y la se mueve al directorio /usr/local/src como se hizo con sumo anteriormente.

***$ cd Descargas***

***$ unzip veins-2.1.zip***

***$ sudo mv -v veins-2.1 /usr/local/src***

2) Abrir omnet++, se va a la opción: File > Import > General: Existing Projects into Workspace y se añade la carpeta veins que está ubicada en /usr/local/src/veins-2.1 y damos aceptar.

3) Ejecutar project > Build all para compilar el proyecto. La carpeta que contiene el módulo de veins se llama mixim como muestra la figura.

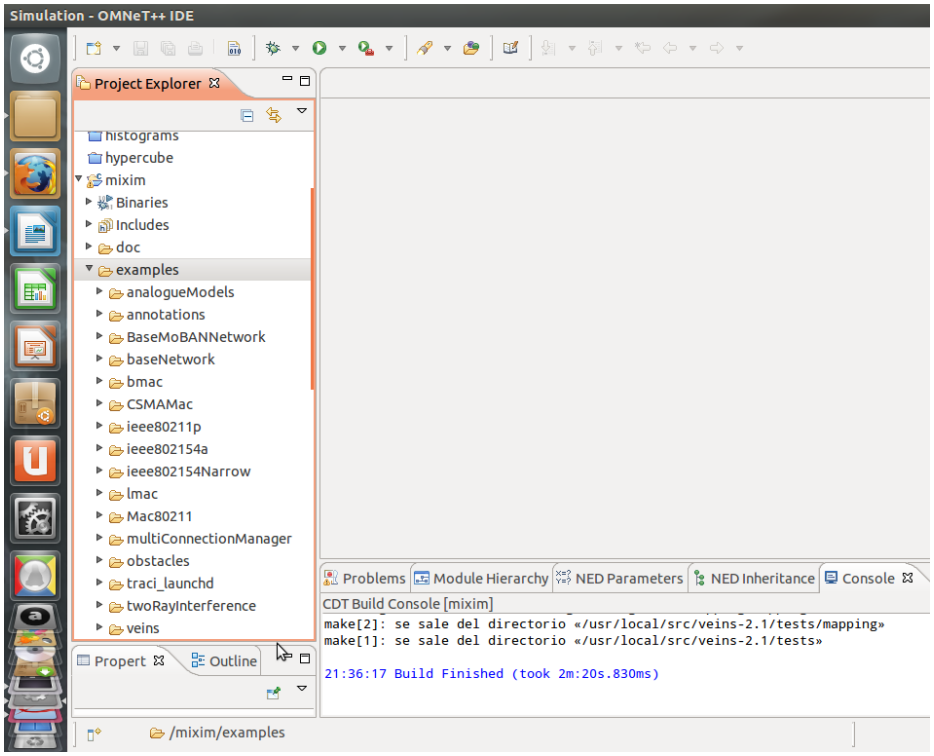

Figura 1 Ubicación de VEINS en MIXIM. Por los autores

4) Asegurarse que SUMO funcione junto con VEINS así que en la terminal se tipea lo siguiente:

***$ cd /usr/local/src/veins-2.1/examples/veins***

***$ /usr/local/src/sumo-0.17.1/bin/sumo -c erlangen.sumo.cfg***

- Aparece un mensaje "Loading configuration... done." con un contador que 0 a 1000 donde al llegar al paso #999 detiene la simulación.

5) verificar que el módulo mixim funciona, para eso se hace click derecho en mixim/examples/baseNetwork/omnetpp.ini y se elige: Run As > OMNeT++ simulation, aparece varias opciones de cómo se quiere correr el archivo así que siempre escogeremos la simulación del mismo tipo que en este caso se debe tomar baseNetwork. Ahora debería Correr una simulación de una red inalámbrica sencilla como muestra la Figura 2.

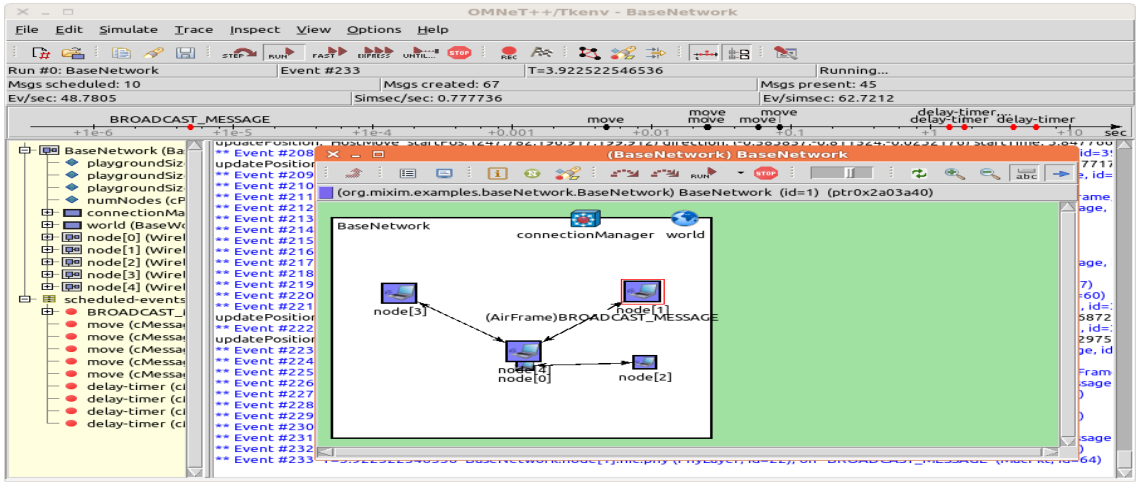


Figura 2 Simulación red inalámbrica en OMNET con MIXIM. Por los Autores

6) Ahora el paso final para correr veins junto con SUMO:

***$ /usr/local/src/veins-2.1/sumo-launchd.py -vv -c /usr/local/src/sumo-0.17.1/bin/sumo***

- El anterior comando crea un puerto con conexiones proxy TCP entre OMNET y SUMO. Siempre que iniciemos VEINS desde el principio tenemos que colocar este script o no funcionara. Si no se quiere colocar lo mismo cada vez que se inicia se puede intentar lo siguiente. (Nota: puede funcionar pero en caso de no hacerlo probar con el primer método)

7) Se abre otra terminal y se ejecuta las siguientes líneas:

***$ cd omnetpp-4.3***

***$ gedit ~/.bashrc***

- Se añade la siguiente línea al final y se guarda:

***export PATH=$PATH:/usr/local/src/sumo-0.17.1/bin***

8) finalmente se cierra el terminal y se concluye la instalación de VEINS, OMNET++ y SUMO.

# ANEXO B

## B.1 API DE REFERENCIA DEL MODELO VANET SA/PC

El modelo de simulación desarrollado simula el intercambio de paquetes entre vehículos mediante el Protocolo de Mensajes Cortos (WSMP, *WAVE Short Messages Protocol*). El modelo toma el *framework* de código abierto de Simulación en Redes Vehiculares (VEINS, *Vehicles in Network Simulation*) como punto de partida, a este se le adicionan nuevas clases desarrolladas en C++, que permiten la simulación de un modelo más realista y la implementación de los procedimientos de Control de Potencia (PC, *Power Control*) y Asignación de Espectro (SA, *Spectrum Allocation*) seleccionados en trabajo de grado.

## B.2 ARQUITECTURA DEL MODELO DE SIMULACION

La Figura 3 muestra la arquitectura del modelo de simulación.

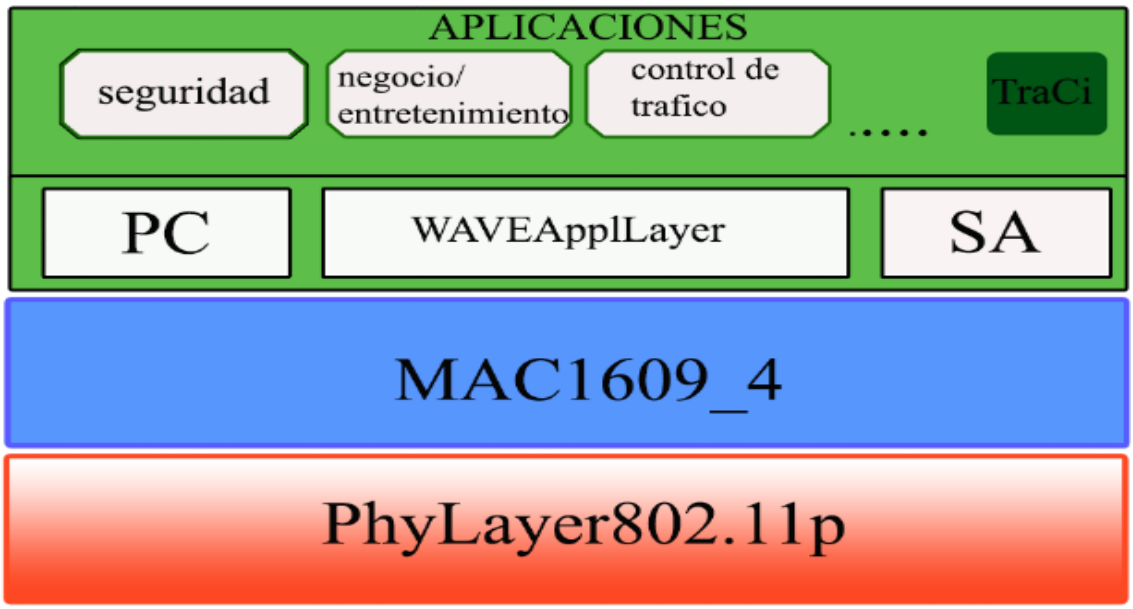


Figura 3 Arquitectura modelo de simulación. Por los autores

- **APLICACIONES**: Son las clases C++ encargadas de modelar las aplicaciones y servicios.
- **TraCi**: Son las clases C++ que permiten la integración de VEINS con el simulador de movilidad SUMO, el cual se comunica a través de sockets mediante el puerto 9999.

- **WAVEApplLayer**: Es la capa encargada de modelar el despliegue de las aplicaciones mediante Mensajes Cortos WAVE (WSM, *WAVE Short Messages*) y del procesamiento de los mismos.

- **PC**: Clases C++ encargadas de la implementación de (MCA) como proceso de control de potencia.

- **SA**: Clases C++ encargadas de la implementación de (NACSA) como proceso de asignación de espectro.

- **MAC1609_4**: Son las clases C++ encargadas de modelar funcionamiento del nivel MAC de una de un NIC80211p. Entre los principales procesos modelados en esta capa se encuentran la operación multicanal, la conmutación de canal, El tratamiento de Calidad de Servicio (QoS, *Quality of Service* ), mediante el Acceso al Canal Distribuido Mejorado (EDCA, *Enhanced Distributed Channel*,), el empaquetamiento y des empaquetamiento de la trama MAC, el Acceso Múltiple por Detección de Portadora con Evasión de Colisiones (CSMA/CA, *Carrier Sense Multiple Access/Collision Avoidance*) y la comunicación con la capa física y capa de aplicación.

- **PhyLayer802.11p:** Son las clases C++ encargadas de modelar la creación, transmisión y recepción de señales a nivel físico, así como el canal de radio.

## B.3 API de Referencia modelos VANET SA/PC y VANET

En las Tabla 1, 2, 3 y 4 se realiza una breve descripción de las principales clases y funciones que se añadieron o modificaron en VEINS[1] para la implementación de los modelos VANET y VANET SA/PC.

### B.3.1 APLICACIONES

- En esta capa se desarrollaron las siguientes funciones:

- Modelado del acceso a los servicios WAVE.

- Modelado del intercambio de paquetes entre nodos.

- Aplicación de los algoritmos de Mínima Área de Cesado (MCA, *Minimum Cesased Area*) y Nuevo Algoritmo de Asignación de Espectro Cooperativo (NACSA, *New*

[1] La documentación oficial de VEINS se encuentra en la paginas http://veins.car2x.org/ y

http://mixim.sourceforge.net/doc/MiXiM/doc/neddoc/index.html

*Algorithm for Cooperative Spectrum Allocation*) como procesos de PC y SA respectivamente.

- La Tabla 1 muestra una breve descripción de las principales clases y funciones referentes a la capa de aplicaciones.

| CLASE | PRINCIPALES METODOS Y EVENTOS | FUNCIONAMIENTO |
|---|---|---|
| **ASPCApplBase**<br><br>**Descripción:**<br><br>Trabaja conjuntamente con la clase ASPCAppl11p para el manejo de la capa de aplicación, responsable del despliegue de servicios sobre la red VANET.<br><br>**Archivos:**<br><br>ASPCApplBase.h<br><br>ASPCApplBase.cc | Inicialize() | Inicializa el módulo appl |
| | handleSelfMsg() | Maneja los eventos a lo largo de la simulación. |
| | handleLowerMsg() | Maneja los mensajes provenientes de la capa MAC. |
| | prepareWSA() | Preparación y transmisión de los mensajes WSA. |
| | updateProvider() | Actualiza el registro proveedores tras la llegada de cada mensaje WSA |
| | prepareJoined() | Preparación y transmisión de los mensajes JOIN. |
| | prepareWSM() | Preparación y transmisión de los mensajes WSM. |
| | prepareDisc() | Preparación y transmisión de los mensajes DISCONECT |
| | calcPower() | Calcula la potencia MCA |
| | addPower() | Agrega la potencia calculada en la cabecera |

| CLASE | PRINCIPALES METODOS Y EVENTOS | FUNCIONAMIENTO |
|---|---|---|
| | | del mensaje WSM a transmitir. |
| | chooseProvider() | Selecciona un proveedor registrado en el registro de proveedores. |
| | chooseChannel() | Selecciona el canal de servicio mediante la aplicación el procedimiento ASPC |
| **ASPCAppl11p**<br><br>**Archivos:**<br><br>ASPCAppl11p<br><br>ASPCAppl11p | onWSA() | Este evento se lanza tras la llegada de cada mensaje WSA. |
| | onJoinChannel() | Este evento se lanza tras la llegada de cada mensaje JOIN |
| | onData() | Este evento se lanza tras la llegada de cada mensaje DATA |
| | onDisconect() | Este evento se lanza tras la llegada de cada mensaje DISCONECT |
| **MatrixBase**<br><br>**Descripcion:**<br><br>Maneja la información del registro de nodos cercanos en cada canal de servicio. | setValue() | Registra un usuario en el registro nodos. Se ejecuta con la llegada de los mensajes WSA y JOIN |
| | deleteValue() | Borra un usuario en el registro nodos. Se ejecuta con la llegada de los mensajes WSA y JOIN |

| CLASE | PRINCIPALES METODOS Y EVENTOS | FUNCIONAMIENTO |
|---|---|---|
| **Archivos**<br><br>MatrixBase.h<br><br>MatrixBase.cc | ChooseMinSCH() | Selecciona del registro nodos el canal que contenga menor número de usuarios. |
| | calcUsers() | Calcula el número de usuarios de un canal determinado |
| | getChannel() | Obtiene el actual canal de servicio usado por un usuario. |

Tabla 1 Clases C++ capa de aplicaciones

### B.3.2 MAC 1609_4

En esta capa se desarrollaron las siguientes funciones:

- Ajuste de la frecuencia y potencia de transmisión a la señal de transmisión.
- Cambio de canal de servicio .requerido por el algoritmo NACSA.
- La Tabla 2 muestra una breve descripción de las principales clases y funciones referentes a la capa MAC 1609_4

| CLASE | PRINCIPALES METODOS Y EVENTOS | FUNCIONAMIENTO |
|---|---|---|
| **ASPCMac 1609_4**<br><br>**Descripcion:**<br><br>Responsable de las operaciones de la NIC a nivel MAC, tales como: CSMA/CA, EDCA, conmutación de canal y | Initialize() | Inicializa el módulo 1609_4 |
| | handleUpperMsg() | Maneja los mensajes provenientes de la AU. |
| | handleLowerMsg() | Maneja los mensajes provenientes de la capa física |
| | handleSelfMsg() | Responsable del manejo de eventos, entre los cuales se |

| CLASE | PRINCIPALES METODOS Y EVENTOS | FUNCIONAMIENTO |
|---|---|---|
| operación multicanal<br><br>**Archivos:**<br><br>ASPCMac 1609_4.h<br><br>ASPCMac 1609_4.cc | | encuentra la conmutación de canal |
| | createQueue() | Crea un buffer donde se almacenan los mensajes de acuerdo a la prioridad hasta ser tranmsitidos. |
| | changeServiceChannel() | Cambia de canal de servicio |
| | attachSignal() | Crea la señal a ser enviada, de la cabecera del mensaje la frecuencia y la potencia de transmisión y los ajusta en la creación de la señal a ser transmitido. |

Tabla 2 Clases C++ capa MAC 1609_4

### B.3.3 PhyLayer80211p

La Tabla 3 muestra una breve descripción de las principales clases y funciones referentes a la capa PhyLayer80211p.

| CLASE | PRINCIPALES METODOS Y EVENTOS | FUNCIONAMIENTO |
|---|---|---|
| **PhyLayer80211p**<br><br>**Descripcion:**<br><br>Trabaja en cooperación de Decider80211p para el manejo y control de las operaciones de la NIC a nivel físico, procesa las señales entrantes por el canal de radio y transmite las tramas provenientes de la capa MAC**.** | changeListeningFrecueny() | Cambia la frecuencia de escucha. |
| | handleUpperMsg() | Maneja las tramas provenientes de la capa MAC |
| | initializeDecider80211p() | Inicializa el decider |
| **Decider80211p** | processNewSignal() | Procesa la señal entrante |
| | processSignalEnd() | Computa si el paquete llega en buenas condiciones o contiene errores. |
| | calcChannelSenseRSSI() | Calcula el umbral de recepción, la trama es descartada si el umbral es muy bajo o se produce una colision |

Tabla 3 Clases C++ capa PhyLayer80211p

### B.3.4 CapacityMeter

La Tabla 3 muestra una breve descripción de las principales clases y funciones referentes al módulo CapacityMeter.

| CLASE | PRINCIPALES METODOS Y EVENTOS | FUNCIONAMIENTO |
|---|---|---|
| **CapacityMeter** | RegisterNode() | Registra todas las trasmisiones en la red. |
| | RegisterChannel() | Registra las transmisiones en cada canal de servicio. |

Tabla 4 Clases C++ módulo CapacityMeter

# ANEXO C

## C.1 FORMATO MENSAJES USADOS EN MODELOS VANET SA/PC Y VANET

### C.1.1 FORMATO MENSAJES WSA

```
packet WaveServiceAdversing {
	//Version of the Wave Short Message
	int wsaVersion = 2;
	//Determine which security mechanism was used
	int securityType = 0;
	//Channel Number on which this packet was sent
	int channelNumber;
	//Data rate with which this packet was sent
	int psid = 0;
	//Provider Service Context
	string psc = "Service with some Data";
	//Length of Wave Short Message
	int wsaLength;
	//Data of Wave Short Message
	string wsaData = "Some Data";

	int senderAddress = 0;
	int recipientAddress = -1;
	int serial = 0;
	Coord senderPos;
	simtime_t timestamp = 0;
	double power=18;
	int sch=1;
	int lastSch=1;
	int idDisconect=0;
	int server=-1;
	bool lastPacket= false;
}
```

### C.1.2 FORMATO MENSAJES WSM

```
packet WaveShortMessage {
	//Version of the Wave Short Message
	int securityType = 0;
	//Channel Number on which this packet was sent
	int channelNumber;
	//Data rate with which this packet was sent
	int psid = 0;
	//Provider Service Context
```

```
        string psc = "Service with some Data";
        //Length of Wave Short Message
        int wsmLength;
        //Data of Wave Short Message
        string wsmData = "Some Data";

        int senderAddress = 0;
        int recipientAddress = -1;
        int serial = 0;
        Coord senderPos;
        simtime_t timestamp = 0;
        double power=18;
        int sch=1;
        int lastSch=1;
        int idDisconect=0;
        int server=-1;
        bool lastPacket= false;
}
```

### C.1.3 FORMATO TRAMA MAC

```
packet Mac80211Pkt extends MacPkt
{
    int address3;
    int address4;
    int fragmentation; //part of the Frame Control field
    int informationDS; //part of the Frame Control field
    int sequenceControl;
    bool retry;
    simtime_t duration;   //the expected remaining duration the current
transaction
}
```

### C.1.4 FORMATO AIRFRAME11P

```
packet AirFrame
{
    Signal signal;        // Contains the physical data of this AirFrame
    simtime_t duration;   // time the AirFrames takes to be transmited (without
propagation delay)
    int state = 1;        // state of the AirFrames, used by the physical layer
                                  // as state machine for delay and
transmission duration
                                  // simulation

    int type = 0;         // If type isn't null then this is a control-AirFrame
                                  // and type specifies the control type.

    long id;              // Unique ID of the AirFrame used as identifier for
                                        // related control-AirFrames

    int protocolId;       //the id of the phy protocol of this airframe

    int channel;          //the channel of the radio used for this transmission
}
```