
**UNIVERSIDAD DE SEVILLA**
Escuela Superior de Ingenieros

Titulación en

INGENIERÍA DE TELECOMUNICACIÓN

# Modelado de un Sistema de Información Turística Móvil para la ciudad de Florencia

Proyecto Fin de Carrera de:

Javier Gala González

Tutorado por:

Juan Manuel Vozmediano Torres

Diciembre del Año 2005

*A mis padres.*
*Gracias mamá, porque siempre has estado*
*a mi lado para que nada me falte.*
*Gracias papá, porque tu confianza en mí*
*ha sido fundamental.*

*A mis hermanos.*
*Gracias Luis, por enseñarme a convivir,*
*a madurar y*
*a superar una carrera.*
*Gracias Ana Belén, sin tu alegría y tu interés*
*no habría sido lo mismo.*

*A mis compañeros de clase.*
*Gracias Fran, tú ya lo sabes. Si tuviera que*
*quedarme con algo de la escuela…*
*sin duda serían tu amistad y apoyo.*
*Gracias Rob, porque es todo un lujo haberte*
*tenido como compañero.*

*A mis compañeros de piso.*
*Gracias Miguelito, por aportarme*
*la humildad y la nobleza…*
*Y por tantas tardes de risas.*
*Gracias Jose, porque con un Zipi como tú*
*es fácil pasarlo de escándalo.*
*Gracias Colino, porque compartir dos años contigo*
*ha sido toda una experiencia.*

*A ti, Piliti.*
*Gracias, porque has sabido animarme en cada examen,*
*en cada momento difícil… Por involucrarte como si*
*fuera tu propia carrera.*

*A todos:*
*¡Lo conseguí!*

# Índice general

# Sección I

# Memoria

# Capítulo 1
# Introducción

## 1.1 Presentación

En este libro se describe el Proyecto Fin de Carrera titulado: ”Modelado de un sistema de información turística móvil para la ciudad de Florencia”. Más concretamente, se trata de la realización de un programa en lenguaje C++ que modela una red de tráfico Streaming y Elastic para terminales móviles en el centro de una ciudad. Además, en el presente Proyecto se comprueba el correcto funcionamiento del programa para una situación real y de gran exigencia, como lo es el centro histórico de Florencia (Italia).

El modelo creado no exige que la red a emular sea de un tipo concreto, pues es el propio usuario el que decidirá los parámetros del modelo y por tanto del tipo de red. La única característica que se presupondrá a la red es la de ser sin cables, para permitir la libre movilidad del terminal por todo el territorio.

Por su parte, el tráfico Streaming es tráfico de tipo real-time (comunicaciones vocales, audio-video real time...), caracterizado por la necesidad de retrasos End-to-End inapreciables. El tráfico Elastic comprende todos los mensajes que forman parte de un proceso background (SMS, solicitudes a bases de datos, e-mail,...), que no requieren valores tan exigentes de retraso en la entrega, siendo así menos prioritarios.

Para la implementación del programa se ha hecho uso de la plataforma de simulación NePSi (Network Protocol Simulator), creada en el LENST (*Laboratorio di Elaborazione Numerica e Telemática*) de la Facultad de Ingeniería de Florencia como una de las partes del proyecto INeSiS (Integrated Network Signal Processing Simulator). Se trata de un conjunto de clases escritas en C++ que actúan como marco en simulaciones de Eventos Discretos y Asíncronos (DEA). Este tipo de eventos se corresponde de forma unívoca con los eventos que se generan en el programa.

## 1.2 Motivaciones y objetivos

SITI (*Servizi e Itinerari Turistici Integrati*) es el portal creado dentro de la solución mTourist promovido por la Región Toscana en colaboración con la Asociación de empresas hoteleras (A.I.A.) de Florencia y provincia, y la Facultad de Ingeniería de la Universidad de Florencia. SITI representa un cambio innovador para estimular un turismo más activo y dinámico, proporcionando a los turistas un sistema por el que, gracias a una serie de dispositivos (teléfonos móviles, PDA, etc.) se encuentran conectados interactivamente a todos los servicios ofrecidos (recepción hotelera, etcétera). Es decir, el turista es libre de navegar por Florencia a través del propio dispositivo móvil, descubrirla y conocerla.

El proyecto SITI ofrece un soporte informativo completo a la clientela turística, contemplando:

- Acceso y obtención de la información referente a los puntos de principal interés turístico (Point of Interest, PoI):
  - El patrimonio cultural y artístico.
  - Las exposiciones y eventos culturales.
  - Las estructuras hoteleras.
  - Tareas comerciales.
- Geolocalización.
- Reservas on-line.
- Pagos on-line.

La información será accesible a un turista que se encuentre directamente sobre el territorio en el cual se ofrece el servicio.

El fragmento que arriba se expone, presenta brevemente el proyecto del que nace la idea que da vida al presente Proyecto Fin de Carrera. Se trata de un proyecto que se está llevando a cabo actualmente en la ciudad de Florencia (Italia), y en el que el alumno

responsable de este Proyecto Fin de Carrera participó activamente durante su estancia en la ciudad italiana como parte del programa Sócrates-Erasmus de intercambio europeo.

Por tanto, el presente Proyecto nace para modelar la red que se propone en el proyecto SITI y la mayoría de sus principales características.

Así, en el modelo los terminales móviles solicitan información de tipo Streaming y Elastic cuando llegan a uno de los lugares que arriba se denominan Puntos de Interés (PoI). Un Punto de Interés es un lugar de valor histórico, cultural, y/o artístico de una ciudad, que reclama la atención del visitante. Pueden ser considerados Puntos de Interés de una ciudad sus monumentos, museos, iglesias, catedrales, plazas, teatros, puentes..., pero también pueden serlo los lugares en que se desarrollen eventos, muestras, exposiciones... Entonces, los Puntos de Acceso presentes en la propia red se encargan de gestionar las peticiones recibidas, hacérselas llegar al Servidor del sistema y finalmente proceder a su envío de forma ordenada. El Servidor se encarga de crear mensajes Streaming y Elastic e introducirlos en el buffer de memoria oportuno. Habrá un buffer de memoria en modo FIFO (First Input First Output) para cada tipo de tráfico y para cada Punto de Acceso, garantizando con ello las prioridades de los mensajes (el modelo da prioridad a los mensajes Streaming frente a los Elastic por sus propias características) y evitando la inactividad en Puntos de Acceso con paquetes a transmitir.

Además, el modelo pretende guiar a los terminales por la ciudad durante el tiempo que dura la simulación, con el objetivo de hacer recorridos eficientes y atractivos para los mismos. Para ello se creará un modelo de Movilidad que hará bordear los obstáculos presentes en el recorrido de forma lógica, y además, seleccionará de entre los más cercanos al terminal, el siguiente Punto de Interés a visitar.

Por último, se modelará a su vez un mecanismo de “deambulación” o traspaso de célula. Con él se desea garantizar la calidad del servicio cuando un terminal cambie de una región a otra mientras está recibiendo la información que ha solicitado.

El programa que describe el modelo está escrito en el lenguaje de programación C++, y el listado de sus bloques y clases se expone en uno de los Anexos adjuntos al Proyecto.

## 1.3 La plataforma de simulación NePSi

NePSi es un conjunto de clases en C++ que actúan como marco en las simulaciones de Eventos Asíncronos (DEA). En los eventos de este tipo, el flujo de datos entre los diversos bloques que componen la simulación es establecido de manera dinámica durante la misma.

En NePSi, el sistema simulado es representado por una clase "System" que hereda sus propiedades de una clase general llamada DESytem. De esta forma, el usuario puede añadir bloques hasta conseguir la funcionalidad deseada. Estos bloques pueden ser tanto bloques de librerías, bloques desarrollados a partir de otros o simplemente nuevos bloques.

Cada bloque del sistema puede generar eventos que podrán ser enviados a otros integrantes del sistema, o bien a sí mismo, o bien recibir de los demás. Ejemplos de eventos generados pueden ser la transmisión de un paquete por parte de un terminal móvil, la recepción de un asentimiento,... Los eventos vienen representados por clases que contienen información referente a ellos, como quién ha generado el evento o cuándo ha ocurrido. Para la gestión de los mismos, NePSi dispone de un gestor de recursos que es el encargado de ordenar los eventos según su tiempo de generación, disponerlos en el buffer, extraerlos cuando sea oportuno..., en resumen, gestionar el funcionamiento e interconexión de los eventos entre sí.

Las dos ventajas principales de esta plataforma son, en primer lugar, la flexibilidad y comodidad a la hora del paso de parámetros en una simulación y, en segundo lugar, la disponibilidad de un conjunto de clases cuya funcionalidad es la de recoger resultados estadísticos provenientes de la simulación. Provee también un conjunto de clases generadoras de números aleatorios.

De cualquier forma, la plataforma NePSi se explica mejor en otro de los Anexos que acompaña a este Proyecto Fin de Carrera. Incluso existe una completa documentación en Internet, en la página http://nepsing.sourceforge.net/.

# Capítulo 2
# Descripción del programa

## 2.1 Introducción

En este capítulo se pretende explicar el programa que da forma al modelo, y que se ha descrito brevemente en el apartado 1.

Para ello, se ha distribuido el capítulo en una serie de apartados que pretenden estudiar cuidadosamente cada una de las características del programa. Así pues, los apartados establecidos tienen los siguientes objetivos:

- **Apartado 2.2**: Presentar el modelo que emula la movilidad de los terminales, haciendo un estudio individualizado del escenario en que se desarrolla, cómo se eligen los próximos destinos de los terminales, y cómo se llega hasta ellos.

- **Apartado 2.3**: Explicar el proceso de transmisión - recepción de paquetes, incluyendo el mecanismo de deambulación del que se puede hacer uso, así como la estructura de los paquetes y el modelo usado para las fuentes de tráfico.

- **Apartado 2.4**: Exponer dos diagramas de flujo con los que se pretende organizar y esquematizar las tareas de los elementos de la comunicación.

- **Apartado 2.5**: Presentar cada uno de los parámetros de entrada que recibe el programa, sus significados y sus unidades de medida (cuando proceda).

- **Apartado 2.6**: Explicar las diferentes clases que componen el programa, así como sus principales variables de estado y funciones miembro. Asimismo, se exponen un par de diagramas de clases para mostrar las clases NePSi de que se hereda y la interconexión entre las clases del programa.

## 2.2 Modelo de Movilidad

Con el objetivo de efectuar una emulación del movimiento de los turistas (o terminales) en la ciudad lo más fiel posible a la realidad, ha sido necesario desarrollar un modelo de Movilidad de los mismos. Dicho modelo se explica a continuación con una división en sub-apartados en la que se muestran sus características principales.

### 2.2.1 Descripción del escenario a transitar

El escenario elegido en este proyecto está representado por un área geográfica limitada, de forma rectangular, en cuyo interior están presentes obstáculos físicos tales como palacios, edificios, ríos o iglesias. Cada obstáculo está representado por un cuadrilátero de forma indefinida descrito por las coordinadas de sus cuatro vértices y por un identificador numérico. Para simplificar el modelo, los obstáculos serán creados de forma cuadrangular, lo que no descarta la posibilidad de crear dos obstáculos unido para formar obstáculos de más lados.

Por otro lado, la idea base ha sido la de valorar las prestaciones del algoritmo propuesto en un escenario que represente el centro histórico de una ciudad, en el cual el flujo de turistas es considerable. El modelo de Movilidad desarrollado busca, como ya se ha introducido, aproximar lo más verosímilmente posible el movimiento de personas en el centro de la ciudad, ya sea a pie o en autobús. No obstante, los resultados obtenidos y las simulaciones pueden ser aplicados sin pérdida de generalidad a escenarios de tipo diverso, como podría ser el interior de un centro comercial o de un aeropuerto; es suficiente modificar el archivo que se pasa como parámetro y en el que se encuentran las posiciones de los obstáculos, así como los parámetros que representan el alto y el ancho del mapa.

### 2.2.2 Elección de una destinación

Al inicio de la simulación, para cada terminal se elige casualmente una posición en el interior del área de trabajo. Dicha posición está elegida con distribución uniforme sobre todas las posiciones "alcanzables", entendiendo por posición alcanzable un punto en el área de trabajo que no caiga en el interior de algún obstáculo.

Posteriormente, para elegir las destinaciones se crea un algoritmo. En este, en primer lugar se seleccionarán, de entre todos los Puntos de Interés de la ciudad, los que se encuentren en un rango inferior a 1/6 de una distancia que se pasa como parámetro y que representa la distancia máxima a la que se puede encontrar un destino para dirigir al nodo hacia él. De los Puntos de Interés que cumplan esta condición se descartan aquellos por los que ya ha pasado ese turista y se hace una elección aleatoria de uno de los sobrantes.

Si no se encontrase ningún Punto de Interés a esta distancia o ya hubieran sido visitados todos, se prueba con una distancia superior (1/3 de la distancia ya nombrada) y se sigue el mismo proceso.

Si tampoco quedasen Puntos por visitar a esta distancia, entonces se usaría ya el total de la distancia parámetro, y si ni siquiera a esta distancia hubiesen Puntos de Interés válidos, entonces habría que considerar el escenario completo para buscar un destino al que dirigir al turista. El hecho de buscar destino a una distancia mayor que la que el usuario del programa indica puede parecer inadecuado, pero se usa para tener la certeza de que el terminal va a continuar moviéndose durante la simulación, pues puede darse el caso de que el usuario del modelo no haya elegido bien el parámetro, o que la simulación sea demasiado larga como para haber visitado todos los Puntos de Interés cercanos.

### 2.2.3 Cómo alcanzar la destinación

Naturalmente, el modelo expresado arriba por sí solo no basta para simular el movimiento de usuarios móviles en áreas en las que están presentes uno o más obstáculos (en el centro histórico de una ciudad puede haber cientos o miles de éstos); por tanto, ha sido necesario desarrollar un algoritmo de búsqueda del mejor camino entre dos puntos que prevea el rodeo de los obstáculos físicos.

Si la destinación, que será la posición en el mapa donde se encuentra el Punto de Interés seleccionado, es alcanzable en línea recta desde la posición actual del nodo, el usuario comenzará a moverse en dicha dirección de acuerdo al modelo precedente. Por inciso, una destinación es alcanzable en línea recta si la recta que une la posición actual y la destinación no se corta con ningún obstáculo presente en el área de trabajo. Si por el contrario la destinación no es alcanzable en línea recta desde la posición actual, el terminal a examen elegirá una destinación intermedia alcanzable hacia la cual dirigirse (ver figuras 2.1 y 2.2) determinando el obstáculo $O_V$ más cercano a la posición actual que es cortado por la recta que une la posición con la destinación.

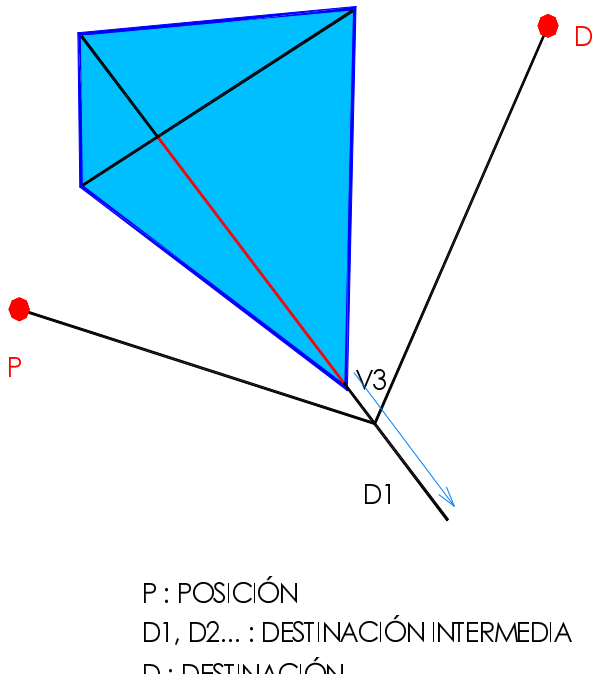



Figura 2.1: Elección de la posición cercana al vértice a sortear.

Posteriormente, se determina el vértice de dicho obstáculo más cercano a la destinación que sea visible desde la posición actual no teniendo en cuenta los otros obstáculos presentes, indicaremos con $V_S$ dicho vértice. Si $V_S$ no es alcanzable en línea recta desde la posición actual, el algoritmo es repetido recursivamente, se determina la destinación intermedia considerando como destinación actual el vértice $V_S$.

De la figura 2.2 se observa que si desde P debe desplazarse al nodo hacia D según el algoritmo propuesto se elegirá como primera destinación intermedia el punto D2; dado que D2 no es alcanzable en línea recta desde P, se determina D1 como destinación intermedia para desplazarme hacia D2. Haciendo la secuencia de destinaciones intermedias que permiten desplazar al nodo de P a D, es consecutivamente P, D1, D2, D3, D.

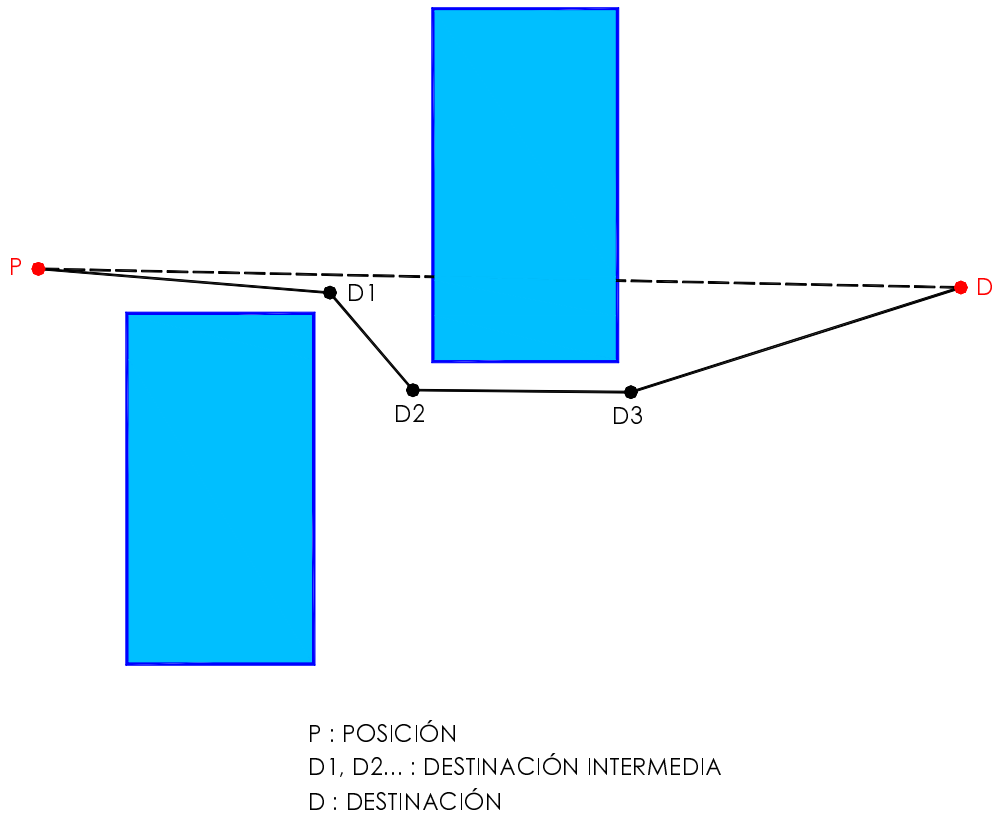



Figura 2.2: Elección de las posiciones intermedias (a).

Determinado $V_S$, se elige un punto $D_i$ próximo a $V_S$ que esté en la recta que une el centro (intersección de las diagonales) del obstáculo C con $V_S$ y que sea exterior al obstáculo (ver figura 2.1). Indicando con $D_V$ la distancia de $V_S$ desde el centro C, el

punto $D_i$ será elegido de forma que su distancia a C sea uniformemente distribuida en el intervalo ($D_V$, $D_V$ + range) donde *range* es una constante numérica expresada en píxeles que se pasa como parámetro al programa. Claramente, el valor de *range* debe ser elegido de forma coherente con la distribución espacial de los obstáculos en el área de trabajo. El terminal móvil iniciará, por tanto, a moverse hacia $D_i$ (destinación intermedia) hasta alcanzar dicho punto. Una vez alcanzada dicha posición, el proceso se itera hasta que el terminal no llegue a la destinación (ver figura 2.3).

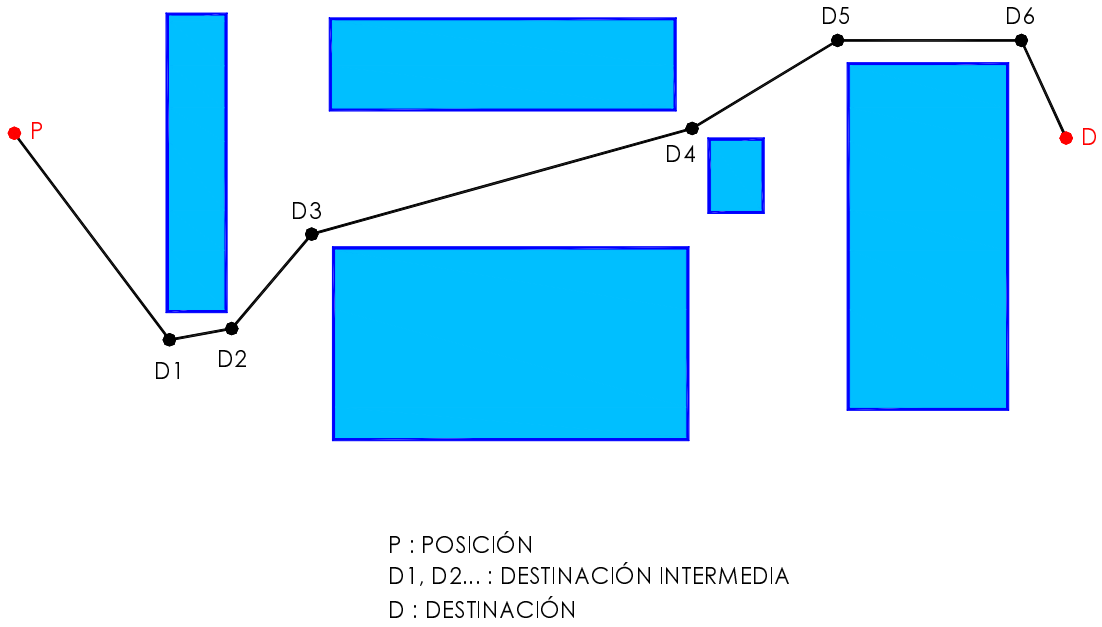



Figura 2.3: Elección de las posiciones intermedias (b).

Cuando finalmente el terminal móvil llega a su destino, permanecerá allí durante un tiempo llamado *StopTime*, al término del cual se elegirá una nueva destinación y el algoritmo recomenzará desde el inicio.

## 2.3 Protocolo de comunicación y fuentes de tráfico

### 2.3.1 Comunicación entre los principales agentes

En el protocolo de comunicación desarrollado, se han utilizado como agentes el usuario móvil, representado por una instancia de la clase *Node*, y un Punto de Acceso (representado por un objeto *AccessPoint*) único para cada región. La comunicación Usuario – Punto de Acceso comienza en el momento en que el nodo llega a la destinación, concretamente a uno de los Puntos de Interés del mapa. En este punto, el nodo se para y efectúa una solicitud de información al Punto de Acceso de la región en la que se encuentra. Tal solicitud se implementa en el código llamando desde el evento *Ev_Muovinodo* al evento *Ev_MsgRequest*.

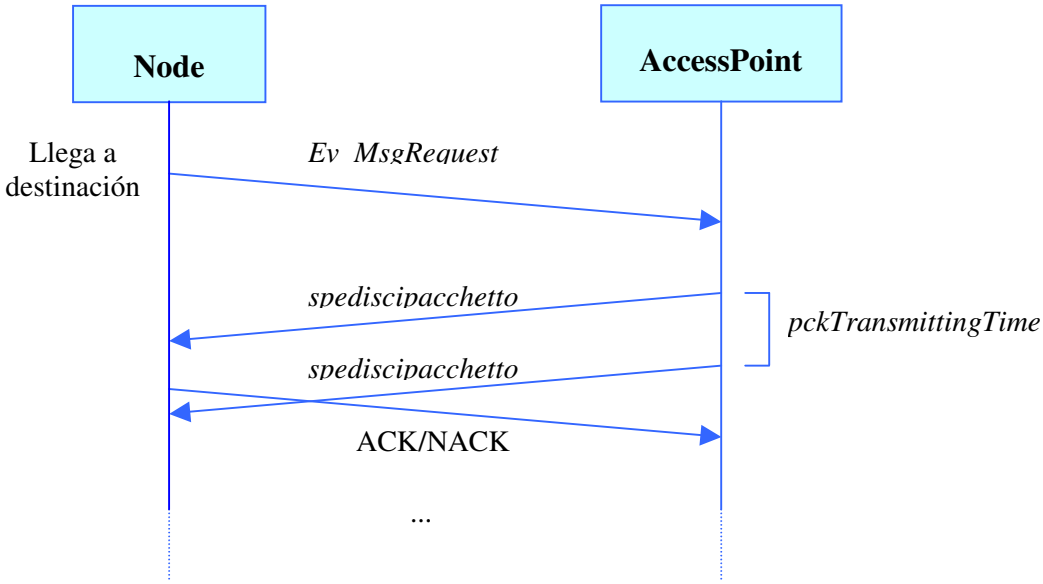



Figura 2.4: Diagrama UML (Unified Modelling Language) del esquema de comunicación entre entidades de red.

Una vez llega la solicitud al Punto de Acceso, éste se la manda al servidor del servicio, el cual recopila la información solicitada por el usuario accediendo a la base de datos del sistema. El servidor, por cada solicitud de información recibida generará un número aleatorio de mensajes Streaming, uniformemente distribuido entre 1 y m mensajes, además de un número aleatorio de mensajes Elastic, uniformemente distribuido entre 0 y n mensajes (los valores m y n son escogidos por el usuario del programa). El tamaño

de cada mensaje se obtiene de los modelos de las fuentes de tráfico analizados posteriormente.

Después de esto, los mensajes son fragmentados en paquetes de tamaño fijo para insertarlos en el buffer correspondiente (buffer Streaming o Elastic) del Punto de Acceso del que llegó la solicitud de información. El Punto de Acceso extrae estos paquetes con la llamada al evento *Ev_TryToSend*. El tiempo considerado entre extracciones de paquetes es únicamente el tiempo de transmisión del paquete, *pckTransmittingTime*, por considerar que otros tiempos como el tiempo de propagación, son despreciables frente a éste. Dicho tiempo se calcula como el cociente entre la longitud del paquete en bits y la tasa de transmisión, como ya es de sobra conocido.

En dicha extracción, lo primero que hace el servidor es buscar en el buffer de paquetes Streaming del Punto de Acceso correspondiente. Si hay algún paquete lo extrae para su posterior envío. Solo en el caso de que la cola de los paquetes Streaming estuviese vacía, se procedería a la extracción de paquetes Elastic, siempre y cuando hubiese algún paquete en esta cola. Antes de mandar el paquete al Punto de Acceso hay que verificar que el usuario destinatario del paquete (*pck.Iddestinazione*) se encuentra en su región, puesto que si por cualquier razón hubiera cambiado a otra región , se iniciaría el proceso de deambulación que se explica en el siguiente sub-apartado..

Antes de proceder al envío, se comprueba a su vez que los paquetes no hayan superado tanto el tiempo máximo que pueden permanecer en cola como el número máximo de transmisiones permitidas. En el caso de que se produjera una de ambas circunstancias, el paquete se daría por perdido y se procedería al análisis del siguiente paquete en el buffer de memoria.

Para simular el envío del paquete se hace uso del parámetro FER (Frame Error Rate), que representa el porcentaje de paquetes que llegan con errores a su destino. De esta forma, se genera en el programa un número aleatorio entre 0 y 1; si dicho número es superior al valor de FER, entonces se considera que el paquete llega correctamente al nodo, y por tanto se borra de la cola. En caso contrario, el paquete llega con errores y se mantiene en la cola. En ambos casos se espera un tiempo 2**pckTransmittingTime* para proceder al siguiente envío, emulando que se espera a recibir el ACK/NACK oportuno.

### 2.3.2 Proceso de deambulación

Podría darse el caso de que el usuario del programa eligiese que los terminales apenas se detuviesen en los Puntos de Interés, pues hay parámetros del modelo que configuran el tiempo de parada del terminal; o bien que se emplease mucho tiempo en el envío de la información del Punto de Interés por estar los búferes llenos... En tales situaciones, es posible que a causa del movimiento del terminal sobre el mapa, éste haya cambiado de región en el tiempo transcurrido desde el momento de la solicitud hasta la expedición del paquete. Para casos de ese tipo, se ha creado un algoritmo de deambulación, por el que si un terminal que está recibiendo información se desplaza a la región de cobertura de otro Punto de Acceso, entonces se pasa la información que debe recibir a los buffer del nuevo Punto de Acceso. Hay que señalar que no se considera cambio de región hasta que el terminal no se aleja una distancia prudencial de la primera región, para no provocar histéresis.

Si efectivamente el nodo ha cambiado de región, el servidor selecciona todos los paquetes que tiene en la cola del Punto de Acceso inicial con destino el nodo desplazado, y a través de la función *ChangePacket ()* se transfieren a la cola del Punto de Acceso que cubre la nueva región en la cual se encuentra el nodo. Estos paquetes se situarán justo después de los paquetes correspondientes al primer mensaje de la cola del nuevo Punto de Acceso para no degradar la calidad del servicio, es decir, para no provocar un retraso superior al debido. Se podría pensar en situar los paquetes justo al inicio del buffer, pero entonces se podría provocar un corte en el envío, y a su vez una degradación del retraso en la transmisión de dicho primer mensaje.

### 2.3.3 Estructura de los paquetes

Los paquetes generados en el sistema, tienen todos ellos una estructura simple y común: la cabecera y el campo de información. En la cabecera de cada paquete, el protocolo de nivel de red debe inserir las informaciones que se exponen a continuación. El tamaño de la cabecera, añadiendo códigos cíclicos y bits de inicio, asciende a 30 bytes. Por su parte, el campo de información es también fijo (con lo que se introducirán rellenos), pero su dimensión dependerá del valor de uno de los parámetros de entrada, el que representa la longitud de los paquetes (se explica en apartado 2.5).

- *IDpacket*: identificador numérico del paquete;
- *IDMessage*: identificador numérico del mensaje al que el paquete pertenece;
- *lunghezza*: tamaño del paquete en bytes;
- *IDNodosorgente*: identificador numérico del Punto de Acceso que solicita este paquete;
- *IDNododestinazione*: identificador numérico del terminal al que el paquete está destinado;
- *IDRegioneDestinazione*: identificador numérico de la región en la que se encuentra el nodo *IDNododestinazione* en el momento de la generación del paquete;
- *posdestinazione*: coordenadas de la posición del nodo *IDNododestinazione*, dicha información se inserta en el momento de la generación del paquete;
- *TTL*: intervalo de tiempo máximo dentro del cual el paquete debe ser entregado a su destinación (Time-To-Live);
- *TimeBorn*: referencia temporal al instante en el que el paquete se ha generado;
- *priority*: indica si se trata de un paquete de tipo Stream o de tipo Elastic;

- *numpckinmsg*: número total de paquetes que constituyen el mensaje *IDMessage*;
- *tx_number*: indica el número de veces que ha sido transmitido el paquete;
- *max_num_tx*: número máximo de veces que el paquete puede ser transmitido.

### 2.3.4 Modelado de las fuentes de tráfico

Para conseguir un modelo de una fuente de tráfico hay que estudiar y comprender el comportamiento de la propia fuente según una aproximación estadística. En general, el modelo está constituido por un conjunto finito de estados que representan la actividad de la fuente.

Un tipo importante de modelo es el que constituye el modelo a dos estados, llamado ON-OFF, que contempla periodos de ON y periodos de inactividad. Una fuente caracterizada por este modelo se conoce como fuente ON-OFF. El estado ON representa el periodo en el que la fuente genera paquetes, mientras que el estado OFF está caracterizado por la inactividad de la propia fuente. En nuestra simulación, la duración del periodo OFF no podrá ser descrita por una variable aleatoria con una cierta distribución de probabilidad porque el tiempo de llegadas entre los mensajes no es síncrono; depende fuertemente de la velocidad con la que se desplaza el usuario y la distancia a la cual se encuentra el Punto de Interés. Por tanto, no es posible elegir un tiempo aleatorio de llegadas entre mensajes. En el estado ON se crean los mensajes siguiendo una distribución de Pareto con parámetros diferentes en función del tipo de mensaje. Se elige la distribución de Pareto por considerarla la más acorde a nuestro tipo de mensajes.

## 2.4 Diagramas de flujo de los agentes de la comunicación

Para dejar claramente definidos los elementos de la comunicación, se exponen en este apartado los diagramas de flujo con las principales tareas ejecutadas en ellos. Así, en el siguiente diagrama se presentan tanto las tareas asociadas a la movilidad como las tareas asociadas a la transmisión - recepción producidas en el nodo.

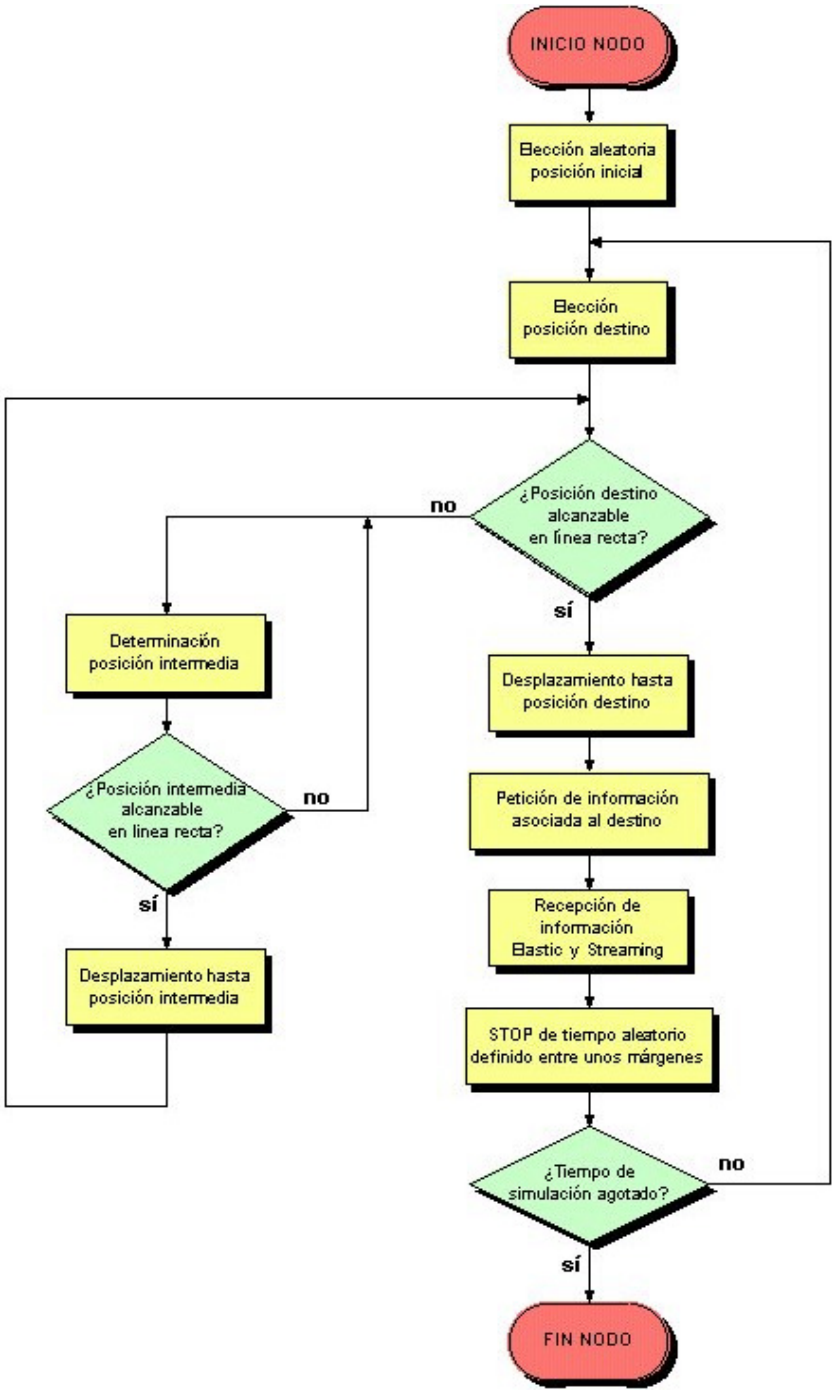



Figura 2.6: Diagrama de flujo del Nodo.

En el programa se han unido el Servidor y el Punto de Acceso en una sola clase (clase *AccessPoint*), aunque en la clase Mappa se realiza alguna tarea del primero. El objetivo es reducir así el número de clases que lo componen; además, de otra forma la funcionalidad de una clase para el Punto de Acceso quedaría bastante reducida. El siguiente diagrama de flujo expone las tareas del Servidor – Punto de Acceso y el orden en el que se ejecutarán en el programa.

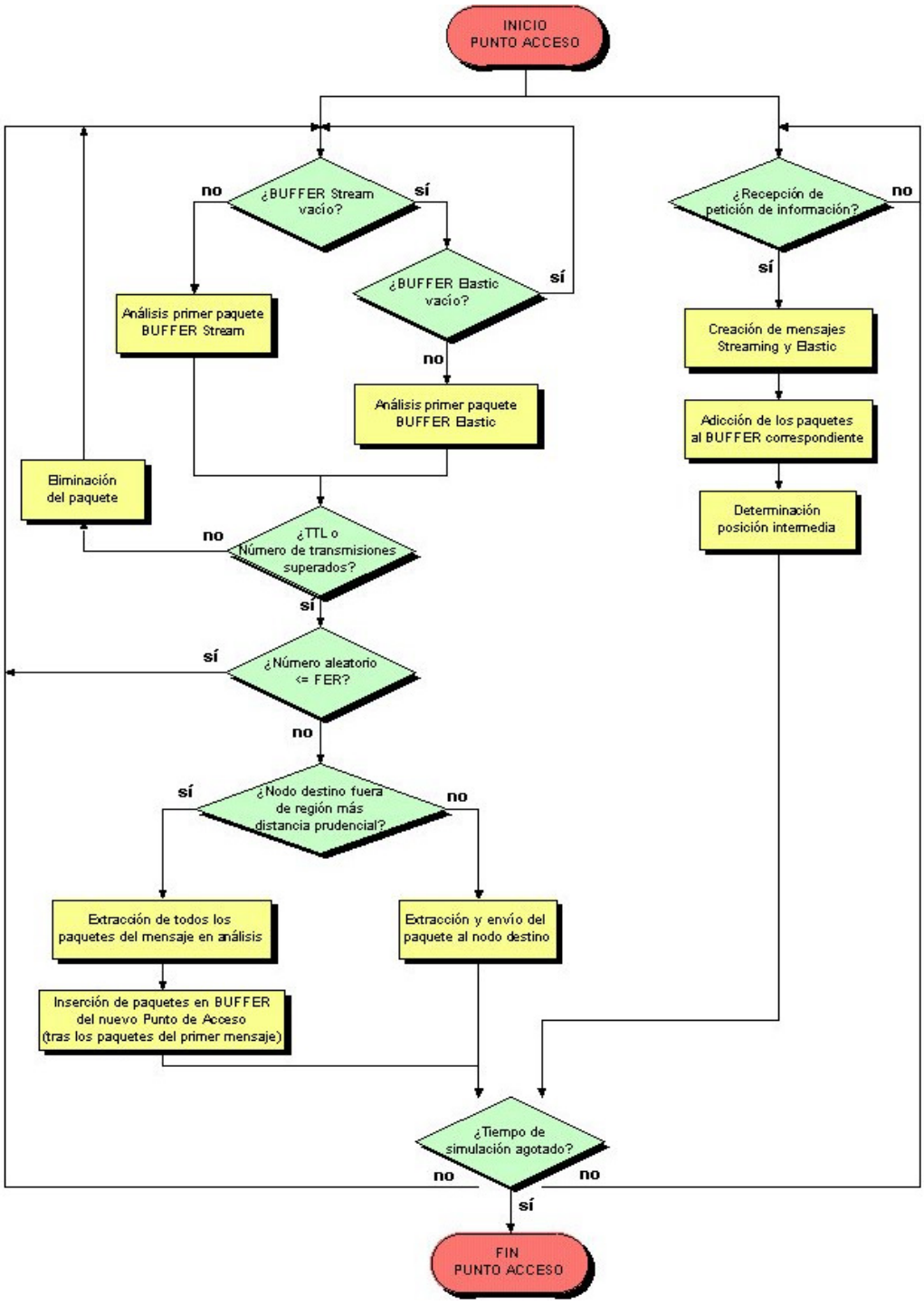



Figura 2.5: Diagrama de flujo del Punto de Acceso y el Servidor.

## 2.5 Parámetros de entrada al programa

No tendría sentido hacer una completa explicación del programa si no se hace una completa explicación de los parámetros que se le pasan al mismo, pues sin su conocimiento nunca podríamos alcanzar una simulación con sentido.

Con la siguiente orden:

*Nombre_ejecutable –p –np Nombre_ param*

Se obtiene un archivo de nombre *Nombre_param* en el que se muestran el nombre de todos los parámetros que se le pasan al sistema y que hay que rellenar. En él se muestran los parámetros separados según la clase a la que pertenecen, y en algunos de ellos aparece un valor por defecto que se puede modificar, pero que generalmente es el valor idóneo. Comentemos a continuación el significado de los parámetros más importantes:

### CLASE System

- *SimulationTime*: representa la duración en segundos de la simulación, es decir, el tiempo durante el cual los turistas van a estar paseando y queremos recoger sus muestras.

- *WriteStatsInterval*: representa el intervalo de tiempo en segundos que ha de transcurrir para que se escriban los resultados y estadísticas de la simulación en los archivos correspondientes, es decir, cada WriteStatsInterval, se obtiene la media y la varianza de cada variable a examen y se escribe el resultado en el fichero que representa a la misma.

El resto de parámetros de esta clase llevan valores por defecto y no es necesario hacerlos variar, pues corresponden a parámetros del sistema.

## CLASE Probe

- *BaseFileName*: es el nombre del directorio en el que queremos almacenar todos los archivos que se obtienen como resultado de la simulación. Este directorio tiene que estar creado antes de la simulación para que puedan ser creados dentro del mismo los archivos.

- *StartCollectAt*: es el valor de tiempo en segundos en el que se empezarán a tomar muestras de las variables que se observan en la simulación. Por regla general comienza en '0', pero se puede variar si se conoce cuando empiezan a variar.

## CLASE Test

- *TimeResolution*: este parámetro indica la precisión que queremos obtener en los resultados de la simulación. Su valor, que se eleva a la base decimal (10), nos sirve para conocer el número de cifras menos significativas con que queremos precisar.

## CLASE Network region map

Realmente representa la clase Mappa y aquí se encuentran, como su propio nombre indica, los parámetros que definen el área de trabajo. Comentar que las coordenadas de todos los puntos se han definido con el eje X de izquierda a derecha y el eje Y de arriba abajo, como se hace en el programa Paint del que se ha hecho uso para averiguar las coordenadas de los puntos que han sido necesarios. Es muy importante que se introduzcan de esta forma para asegurar el correcto funcionamiento del programa.

- *limitX*: representa el ancho del mapa seleccionado en píxeles. Se escoge el píxel como medida pues se usa en los programas de dibujo y simplifica la búsqueda de coordenadas en el mapa.

- *limitY*: representa la altura del mapa seleccionado en píxeles.

- *numnodi*: este parámetro indica el número de usuarios totales que se encuentran en el área de trabajo y que participan en la simulación.

- *rangevertx*: es la distancia máxima a la que un usuario puede bordear un vértice. Su sentido se explicó claramente en el apartado 'Modelo de Movilidad'.

- *range_tra_poi*: es la distancia máxima a la que se puede encontrar un Punto de Interés del usuario cuando éste solicita un nuevo destino, a no ser que el usuario ya haya visto todos los Puntos vecinos, caso en el que sí podría ser enviado a uno más lejano. Debe escogerse teniendo en cuenta la distancia entre los Puntos de Interés, para que sirva como una limitación.

- *FrequenzaLavoro*: es la frecuencia a la cual se envía la información en la simulación.

- *BandaTx*: representa la tasa de transmisión del sistema. Se solicita en 'bits por segundo'.

- *DistanzaHisteresi*: es un parámetro del que se hace uso en la deambulación. De hecho, se trata de la distancia en píxeles que se tiene que salir un usuario de su región para hacer efectivo el cambio a otra y así no provocar histéresis.

- *numostacoli*: indica el número de obstáculos presentes en el sistema.

- *archiviOstacoli*: deberá ser el nombre de un archivo de texto. En cada línea de este fichero se escribirán las 8 coordenadas de los cuatro vértices que conforman cada uno de los obstáculos, separadas por tabulación. Recordar para esto que la coordenada 'x' se expresa de izquierda a derecha y la coordenada 'y' de arriba abajo. Además, deberán ir ordenadas de la siguiente forma para asegurar el correcto funcionamiento del programa:

- 1ª y 2ª coordenada: coordenadas ‘x’ e ‘y’ del vértice superior izquierda.
- 3ª y 4ª coordenada: coordenadas ‘x’ e ‘y’ del vértice superior derecha.
- 5ª y 6ª coordenada: coordenadas ‘x’ e ‘y’ del vértice inferior derecha.
- 7ª y 8ª coordenada: coordenadas ‘x’ e ‘y’ del vértice inferior izquierda.

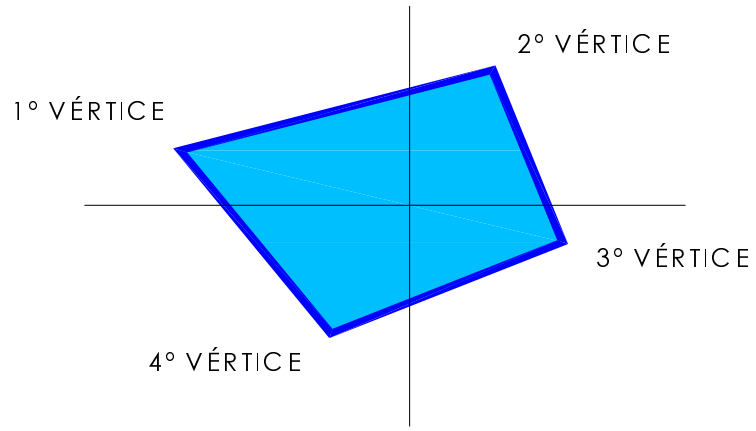



Figura 2.7: Enumeración correcta de los vértices de los obstáculos.

- *archiviPoi*: contiene de igual modo el nombre de un archivo de texto. En cada línea de este fichero se escribirán las dos coordenadas de cada Punto de Interés, primero la coordenada ‘x’ y después la coordenada ‘y’, separadas por un tabulador.

- *numpoi*: indica el número de Puntos de Interés presentes en el sistema.

- *numzone_col*: representa el número de columnas en que se divide el mapa para crear las regiones.

- *numzone_row*: representa el número de filas en que se divide el mapa para crear las regiones.

- *numAP*: indica el número total de regiones (o igualmente de Puntos de Acceso) en que se divide el área de trabajo. Por tanto, y teniendo en cuenta los dos parámetros anteriores, su valor será ‘*numzone_col * numzone_row*’.

## CLASE Node

- *velocitymax*: representa la máxima velocidad en píxeles por segundo (para lo que hay que tener en cuenta la equivalencia píxel-metro) a la que se mueve un turista. Esto ofrece la posibilidad de varios escenarios en función de este parámetro y del que le sigue a continuación: todos los turistas a pie, todos los turistas en autobús, o incluso en bicicleta...

- *velocitymin*: indica la velocidad mínima en píxeles por segundo.

- *HopTime*: es el tiempo que se tarda en comprobar la posición del turista cuando está en movimiento. Hay que buscar un compromiso para su valor, pues si se escoge demasiado pequeño se sobrecarga la simulación, pero si se escoge demasiado grande se corre el riesgo de que el usuario llegue a su destino (bien un vértice o un Punto de Interés), y se quede parado. Su valor dependerá por tanto de cómo se desplacen los turistas (a pie o en autobús).

- *ObserveTime*: valor de tiempo (en segundos) considerado entre cada una de las adquisiciones de muestras que originan los ficheros resultantes de la simulación.

- *MaxStopTime*: tiempo máximo que se considera permanece el turista en el mismo Punto de Interés.

- *MinStopTime*: tiempo mínimo que dedica un turista a visitar y contemplar uno de los Puntos de Interés.

## CLASE APoint

- *FER*: representa el porcentaje de tramas que llegan al destino con errores.

- *max_num_tx4pack*: indica el número máximo de veces que el paquete puede ser transmitido.

- *maxttlxpck*: es el intervalo de tiempo máximo dentro del cual un paquete de tipo Elastic debe ser entregado a su destino.

- *maxttlxpckStream*: indica el intervalo de tiempo máximo dentro del cual un paquete de tipo Stream debe ser entregado a su destino.

- *maxLunPck*: tamaño del paquete en bytes.

- *PCKtransmittingTime*: representa el tiempo de transmisión de un paquete en segundos. Su valor se calcula como ya se describió en el apartado 2.3.1.

- *Mean_Message_Length*: valor medio de la longitud en bytes de los mensajes Elastic. Es uno de los parámetros requeridos para el modelo de fuente de tráfico, en este caso, para parametrizar la distribución Pareto de la cual se hace uso en los instantes ON.

- *Mean_Message_LengthStream*: es el valor medio de la longitud en bytes de los mensajes Stream.

- *Minim_Message_Length*: longitud mínima que puede tener un mensaje Elastic. Es el otro parámetro necesario para modelar la distribución de Pareto.

- *Minim_Message_LengthStream*: longitud mínima de un mensaje Stream.

- *maxMsg4Req*: indica el número máximo de mensajes de tipo imagen que se pueden enviar por cada solicitud de información. O de otra forma, es el número máximo de mensajes Elastic que representan a un solo Punto de Interés.

- *maxMsg4ReqStream*: indica el número máximo de mensajes de tipo Stream que se pueden enviar por cada solicitud de información. O de otra forma, es el número máximo de mensajes Stream que representan a un solo Punto de Interés.

## 2.6 Implementación del programa

El ambiente de desarrollo software seleccionado para la creación del sistema propuesto es la plataforma NePSi (Network Protocol Simulator) que se explica en el apartado 1. Puesto que esta plataforma está realizada en lenguaje C++, el proceso de compilación se ha llevado a cabo con el compilador de libre distribución GNU C++ y la tarea de depuración se ha efectuado con el depurador GDB (a su vez, de GNU). Por otra parte, las compilaciones y ejecuciones se han realizado bajo el sistema operativo Debian GNU/Linux y en una computadora basada en procesador i686.

La plataforma NePSi pone a disposición del desarrollador un conjunto de clases relacionadas entre ellas que permiten simular el comportamiento de los protocolos de nivel de red. Gracias a ellas y al mecanismo de herencia obtenido de la programación orientada a objetos, ha sido posible generar las clases que implementan el programa:

- Clase Mappa.
- Clase Node.
- Clase AccessPoint.
- Clase Ostacolo.
- Clase Regione.
- Clase Common.
- Clase SysTest.
- Clases DEevent.

La organización de las clases se presenta en la página siguiente (figura 2.8) en un diagrama que expone de forma conjunta las clases realizadas por el alumno (recuadros amarillos) y las clases NePSi de las que se hereda y/o se hace uso (recuadros verdes). Como se puede comprobar, las clases NePSi de las que se hereda son Probe, GenericParameter, DESystem, DEDevice, DevTerminal (a su vez hereda de DEDevice) y DEEvent. De éstas, las cuatro últimas son las clases de las que se hereda para la generación del código fuente del programa desarrollado.

En la clase obtenida por herencia de DESystem se definen las clases que operan en el simulador, y contiene el programa principal o *main*. La clase DEDevice es una clase virtual de la que se derivan todos los bloques para terminales móviles. Por su parte, de DevTerminal derivan los bloques para crear dispositivos asociados a una red móvil. Por último, DEEvent es la clase virtual de la que derivan los eventos.

El resto de clases NePSi se utilizan en la implementación de las clases del programa, no con el objetivo de ser heredadas, pero sí para simplificar algunas tareas.

Las clases ParetoRndGen, UniformIntRndGen y UniformDoubleRndGen sirven para la generación de números casuales. El grupo de clases Probe sirve para gestionar las estadísticas a observar. Finalmente, el grupo de clases Parameter se ocupa de la lectura y escritura de los parámetros de entrada al simulador.

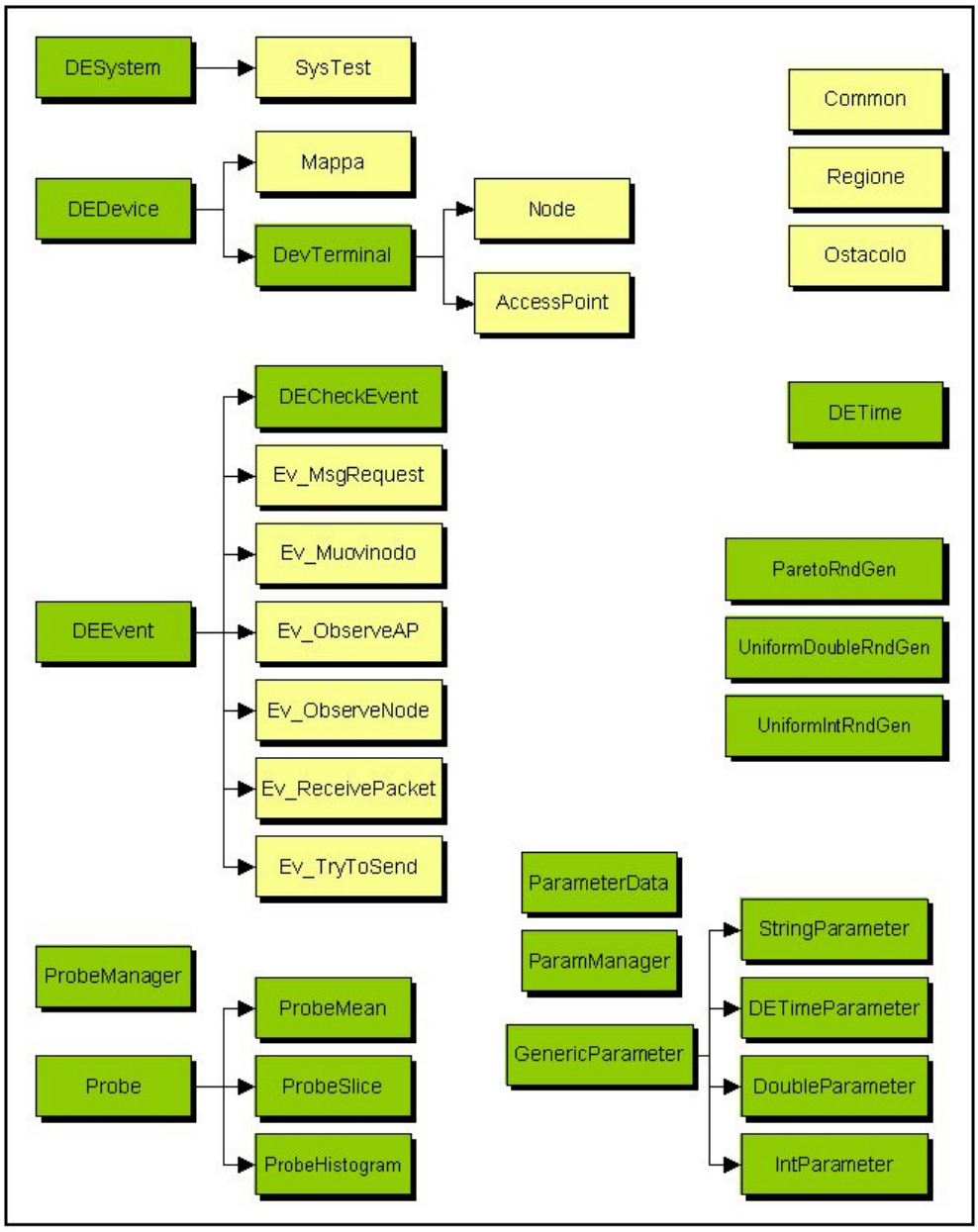



Figura 2.8: Diagrama de clases.

Por otra parte, también se puede observar que algunas de las clases implementadas no hacen uso de la herencia. Son clases que realmente sirven de apoyo al resto, como se observa en el siguiente diagrama. En éste se presentan las relaciones entre las diferentes clases implementadas para hacer una idea de la interconexión entre ellas.

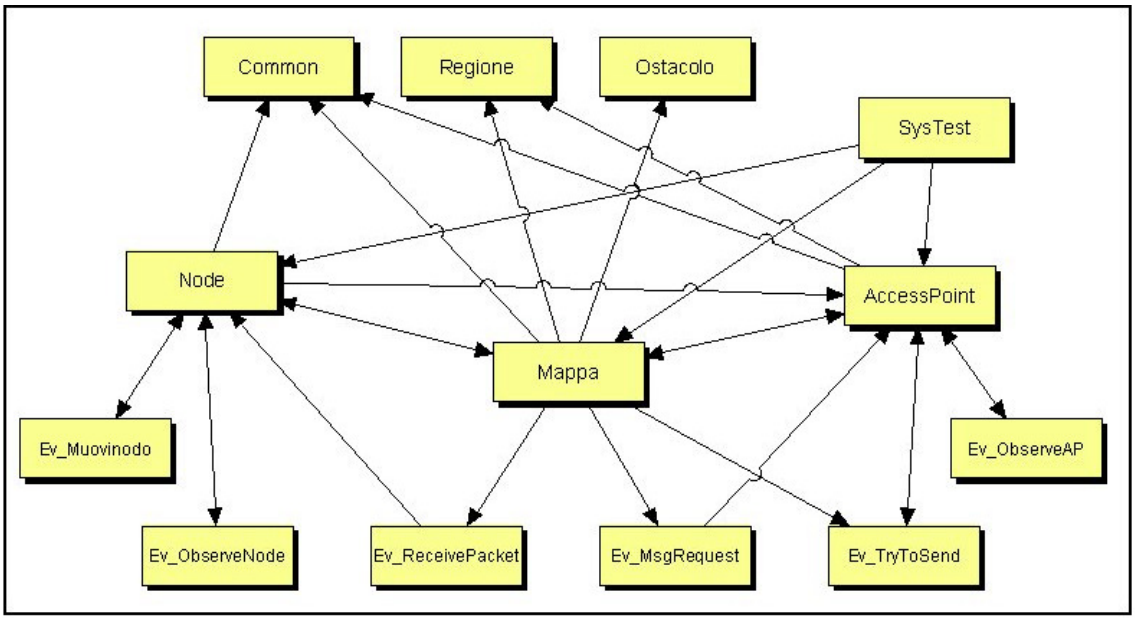
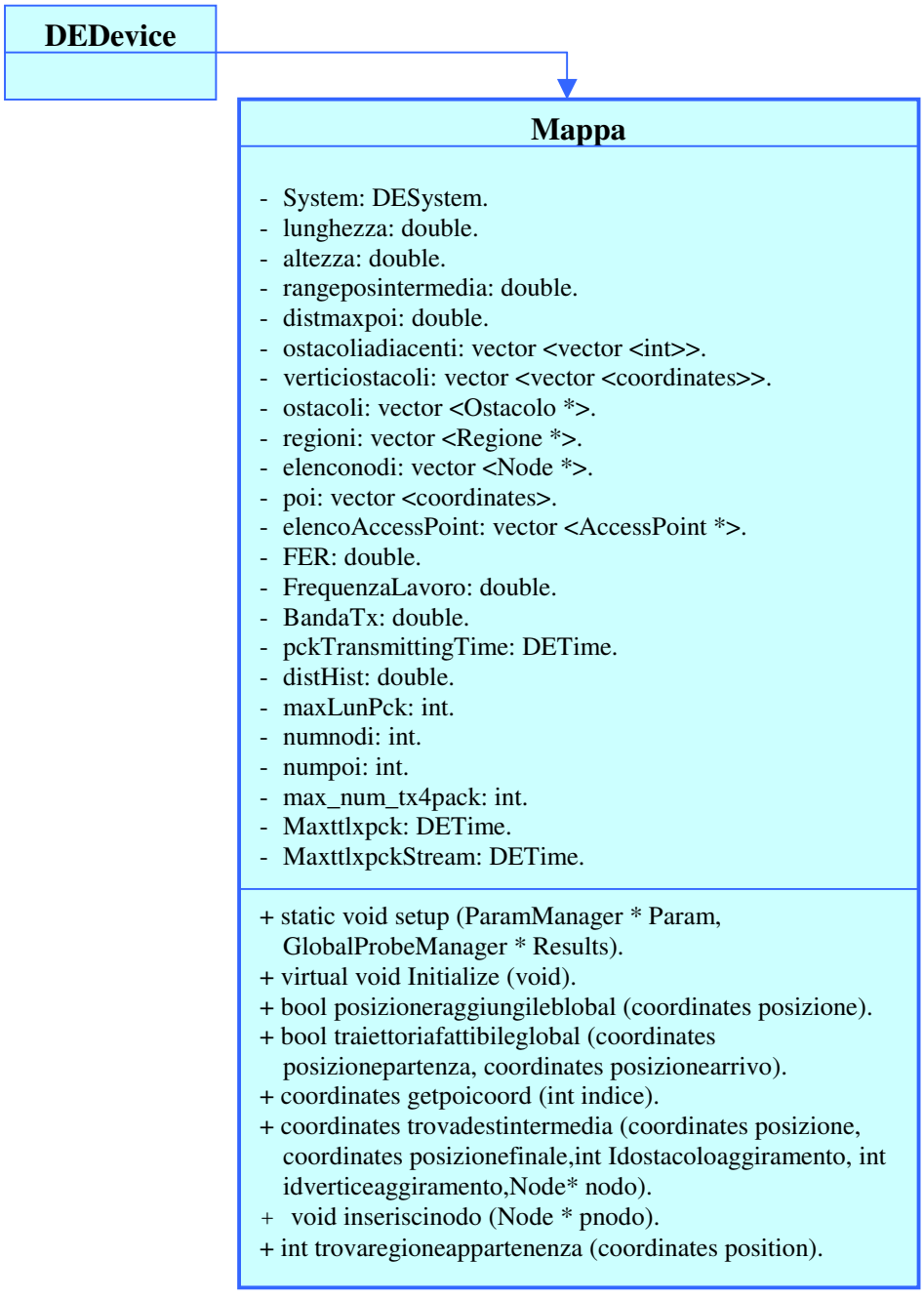



Figura 2.9: Diagrama de las clases implementadas y la relación entre ellas.

Cada flecha del diagrama indica que la clase origen de la propia flecha hace uso en algún momento de la clase destino. Lógicamente, las clases Common, Regione y Ostacolo solo reciben flechas (o mejor dicho, llamadas), al ser clases de apoyo como ya se ha comentado. Por su parte, en la clase SysTest sucede lo contrario ya que es la clase con la que se inicia el programa (contiene el *main*), y por tanto debe invocar a otras para poder proseguir con la ejecución.

### 2.6.1 CLASE Mappa

La clase *Mappa* tiene la función de coordinar las interacciones entre los varios objetos. En particular, permite la transmisión de mensajes y paquetes entre los Punto de Acceso y los nodos, y simula algunas operaciones del servidor. Además, en ella se encuentran todos los elementos que se necesitan para definir el mapa. Para adaptarse a la coordinación de todos los objetos, la clase Mappa contiene referenciadores (punteros) a las instancias de los objetos creados. Así, las variables privadas y funciones públicas más importantes de la clase Mappa se muestran en la figura:



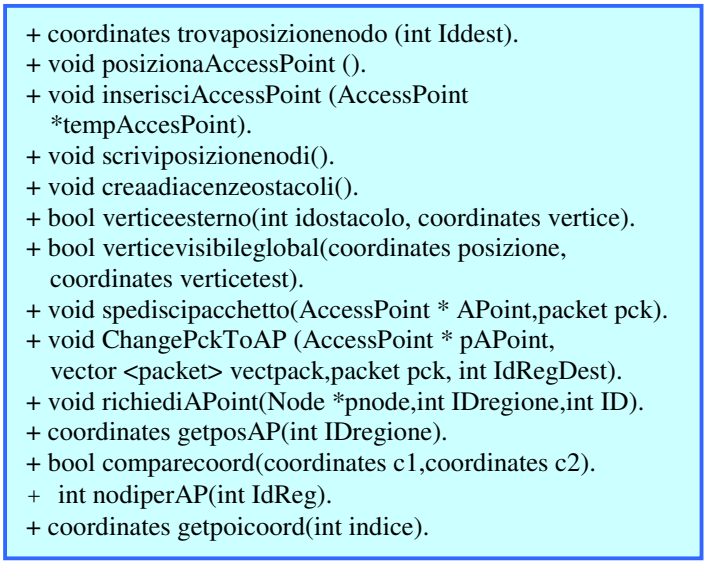



Figura 2.10: Diagrama UML de la clase Mappa; se muestran las variables privadas y las funciones miembro.

Veamos con más detenimiento las **variables privadas** de que hace uso la clase:

- *System*: es un puntero a la clase DESystem de NePSi, pues es en esta clase donde se definen todas las clases que operan en el simulador y además es donde se define el *main*.

- *lunghezza*: variable en la que se guarda el ancho del mapa seleccionado en píxeles. Todos los valores de este tipo relacionados con características del mapa se obtienen de los parámetros que se le pasan al sistema.

- *altezza*: en esta variable se guarda el alto en píxeles del mapa.

- *rangeposintermedia*: es la variable que contiene la distancia máxima a la que un usuario puede bordear un vértice. Se explica mejor en el apartado 2.2.2.

- *distmaxpoi*: representa la distancia máxima a la que se puede encontrar un Punto de Interés desde un turista para que el sistema lo haga dirigirse hacia éste.

- *ostacoliadiacenti*: es un vector de vector de enteros en el que se guardan para cada obstáculo los índices de los obstáculos que se encuentran pegados a él aunque solo sea por un vértice. Si no hay ninguno, se escribe el valor '-1'.

- *verticiostacoli*: en este caso se guardan para cada 'macroobstáculo' creado los vértices que componen al mismo. Se define macroobstáculo como el obstáculo que se crea por la unión de dos o más obstáculos rectangulares que se encuentran pegados. Por tanto, no ha de tener solo 4 vértices, puede tener algunos más. Los macroobstáculos se crean para hacer más real el sistema, pues una ciudad no está compuesta solo de rectángulos.

- *ostacoli*: vector de punteros a todas las instancias creadas de la clase Ostacolo.

- *regioni*: vector de punteros a todas las instancias creadas de la clase Regione.

- *elenconodi*: vector de punteros a todas las instancias creadas de la clase Node.

- *poi*: vector de coordenadas en el que se guardan de forma ordenada las coordenadas de todos los Puntos de Interés.

- *elencoAccessPoint*: vector de punteros a todas las instancias creadas de la clase Punto de Acceso.

- *FER*: variable que, como las posteriores, obtiene su valor de los parámetros de entrada al programa y representa el Frame Error Rate (probabilidad de que una trama llegue con error).

- *FrequenzaLavoro*: su valor representa la frecuencia de trabajo a la que emite el sistema.

- *BandaTx*: define el ancho de banda en que puede transmitir nuestro sistema.

- *pckTransmittingTime*: es el tiempo de transmisión de un paquete y se calcula dividiendo la longitud de un paquete entre el ancho de banda de la transmisión.

- *distHist*: es una distancia que se usa para la deambulación. Un usuario no cambia de región hasta que no se haya desplazado esta distancia de la primera.

- *maxLunPck*: define el tamaño máximo de un paquete en bytes.

- *numnodi*: en esta variable se guarda el número de usuarios que se encuentran visitando el centro histórico de la ciudad representado en el mapa.

- *numpoi*: representa el número de Puntos de Interés presentes en el área de trabajo.

- *max_num_tx4pack*: cifra que representa la cantidad de veces que puede ser enviado un paquete antes de ser descartado y eliminado del buffer.

- *Maxttlxpck*: es el intervalo de tiempo máximo que puede estar un paquete de tipo Elastic en su buffer correspondiente antes de ser descartado.

- *MaxttlxpckStream*: es el intervalo de tiempo máximo que puede estar un paquete de tipo Stream en su buffer correspondiente antes de ser descartado.

Las **funciones miembro** públicas de la clase Mappa tienen las siguientes tareas:

- *Setup*: En esta función se crea una instancia de la clase ParamManager de NePSi, para poder gestionar los parámetros de entrada que se necesitan para la clase Mappa. Posteriormente, se añaden a esta clase el nombre y rango permitido para dichos parámetros. De esta forma, se permite al usuario del programa visualizar los parámetros que se necesitan en un archivo de texto. Posteriormente, dicho usuario invoca al programa con los parámetros de todas las clases que se le indiquen ya rellenos.

- *Initialize*: En primer lugar recoge el valor de todos los parámetros solicitados en la función *Setup* y se los asigna a las correspondientes variables privadas de la clase. Posteriormente, subdivide el área del mapa en las regiones rectangulares que se hayan indicado con los parámetros de entrada 'numzone_col' y 'numzone_row'. Crea a su vez las instancias de la clase Regione, a las que se le pasan dos vértices opuestos de la región y el centro de la misma. Su siguiente labor es la de extraer las coordenadas de los vértices de los obstáculos del fichero de texto correspondiente que se indica como parámetro. Además, crea cada una de las instancias de la clase Ostacolo con las coordenadas de sus cuatro vértices. Finalmente establece un vector con las coordenadas de todos los Puntos de Interés a través de otro fichero de texto que se pasa también como parámetro.

- *getpoicoord*: Función que recibe un número de Puntos de Interés y devuelve las coordenadas de ese Punto en el mapa.

- *posizioneraggiungileblobal*: Esta función recibe una copia de las coordenadas que individualizan un punto sobre el mapa, por tanto efectúa un control sobre todos los obstáculos presentes y establece si tal punto cae dentro de un obstáculo físico, o incluso fuera del mapa. En caso afirmativo devuelve un valor FALSE, de otro modo devuelve TRUE. Se usa sobretodo en funciones relacionadas con el desplazamiento del nodo para ver si la nueva posición asignada es válida.

- *traiettoriafattibileglobal*: Esta función, por su parte, sirve para determinar si la recta que une dos puntos P1 y P2 se corta o no con cualquier obstáculo físico presente. Recibe las coordenadas de los dos puntos P1 y P2 y restituye un valor booleano indicando si la trayectoria es factible o no. Por tanto, se usa en funciones para describir la trayectoria del terminal, para averiguar si es posible o no llegar hasta el destino con el camino más corto (la recta).

- *inseriscinodo*: Función que inserta el nodo apuntado por la referencia que se pasa como entrada en *elenconodi*, variable privada de la clase Mappa.

- *trovadestintermedia*: Esta función miembro implementa el algoritmo de *pathing* (movilidad) explicado en el apartado 2.2.3. Se invoca cuando un nodo debe desplazarse hacia una destinación física a la que no puede llegar en línea recta porque existen obstáculos en medio. Recibe como parámetros de entrada la posición de partida y la posición que representa la destinación hacia la cual el terminal se desea mover. El parámetro restituido es el punto que representa la destinación intermedia que se define en el algoritmo de movilidad.

- *trovaregioneappartenenza*: Función que devuelve el identificador (ID) de la región que contiene las coordenadas de la posición que se pasa como parámetro. Se usa para tener controlada en todo momento la región en la que se encuentra un terminal cuando se desplaza.

- *trovaposizionenodo*: Esta función devuelve la posición actual del nodo cuyo identificador se pasa como parámetro. Se usa sobretodo para completar la cabecera del paquete.

- *inserisciPunto de Acceso*: Función que inserta la instancia de Punto de Acceso que se pasa por referencia en *elencoPunto de Acceso*, variable privada de la clase Mappa.

- *creaadiacenzeostacoli*: Función que se invoca al inicio del programa (en el Initialize de esta misma clase). Examina cada uno de los obstáculos del mapa seleccionado para ver si está unido por cualquier borde a algún otro obstáculo del mapa, pues si es así crea uno de los 'macroobstáculos' mencionados anteriormente para asegurar el correcto funcionamiento del algoritmo de movilidad. Se encarga, por tanto, de rellenar los vectores *ostacoliadiacenti* e *verticiostacoli*, variables privadas de esta clase.

- *verticeesterno*: Esta función sirve de apoyo a la anterior, y su labor es detectar en los 'macroobstáculos' qué vértices son los exteriores, pues habrá vértices de los obstáculos unidos que coincidan o que no se tengan que considerar al no formar un borde en dicho macroobstáculo. Así su función es dejar únicamente los vértices reales del nuevo obstáculo formado.

- *verticevisibileglobal*: Procedimiento en el que se apoya la función *trovadestintermedia* en el caso de que el obstáculo que el turista tenga que sortear sea un macroobstáculo ya conocido. Se llama a esta función para primero averiguar a qué 'miniobstáculo' pertenece el vértice y entonces poder probar si dicho vértice es o no el que hay que bordear desde la posición actual del nodo. Para esta última tarea se llama al procedimiento *verticevisibile* de la instancia correspondiente de la clase Ostacolo, clase en la que no se tratan los macroobstáculos. Por esto último se selecciona primero el 'miniobstáculo' al que pertenece el vértice. Por tanto, esta función recibe la posición del nodo y las coordenadas del vértice a testar, y devuelve un valor booleano en función de que sea el vértice a sortear o no.

- *spedisciPacchetto*: Este procedimiento simula la transmisión de un paquete desde un Punto de Acceso A hasta un nodo B. Dicha función se llama desde el manejador de evento *HEV_TryToSend* de cada instancia de la clase *Punto de Acceso*, que se explica posteriormente. La primera tarea de este procedimiento es la de determinar si el usuario se ha movido a una nueva región la distancia suficiente (*distHist*) como para poder iniciar el proceso de deambulación. En caso afirmativo, se llama a las funciones *ChangePckToAP*, y *searchPackets* de la clase Punto de Acceso para completar el proceso. En caso negativo se procede al envío del paquete. Para ello, en primer lugar se comprueba que tanto el número de transmisiones como el tiempo de vida del mensaje no superan los valores permitidos indicados en la cabecera del paquete (*pck.max_num_tx* y *pck.TTL*). Si es así, se procede al envío del paquete, que en función de un valor aleatorio entre 0 y 1 (que se compara con la tasa *FER*), llegará a su destino correctamente o deberá ser reenviado. Si no es así, se descarta el envío de dicho paquete y se elimina del buffer.

- *ChangePckToAP*: Esta función se utiliza en el proceso de deambulación. Como se indica en el apartado 2.3.2, cuando un usuario que está recibiendo información se cambia completamente de Punto de Acceso, entonces saca el paquete que se está intentando enviar y el resto de paquetes que pertenecen a ese mismo mensaje de la cola en la que se encontraban y se insertan en la cola del

Punto de Acceso que se encuentra en la nueva región , justo después de todos los paquetes del mensaje que se estaba enviando. Se coloca justo ahí para no entorpecer el envío de ningún mensaje y a su vez no provocar una parada larga en el envío. Así pues, esta función recibe la referencia a la instancia de la clase Punto de Acceso a la que se tienen que enviar los paquetes que forman el vector que también se recibe. Con esto, se encarga de buscar entre el listado de Punto de Acceso cuál es el requerido y apoyándose en la función *CopyPacket* del Punto de Acceso, introduce los paquetes en su cola correspondiente (Stream o Elastic).

- *richiediAPoint*: Esta función sirve para buscar en el elenco de los Puntos de Acceso (variable privada *elencoAccessPoint*), aquel cuyo identificador se pasa como parámetro y que es el que debe satisfacer la petición de información de un nodo cuyo identificador numérico también se pasa como parámetro. Tras localizar el Punto de Acceso correspondiente, se le manda un evento (*Ev_Msg_Request*) para que actúe en consecuencia con dicha petición.

- *NodiperAP*: Función que devuelve el número de nodos que se encuentran en una región determinada en el instante de invocación de la función.

### 2.6.2 CLASE Node

La clase *Node* representa, por su parte, el terminal móvil genérico. En su interior se implementan funciones útiles sobretodo para el movimiento del nodo, para la recepción de paquetes y para la adquisición de resultados. Así, las instancias de la clase Node están constituidas por variables privadas que indican la posición del nodo y el estado de los paquetes recibidos en cada instante de la simulación, además de referencias a objetos (Probes) que tienen la función de adquirir resultados (ver figura 2.11).

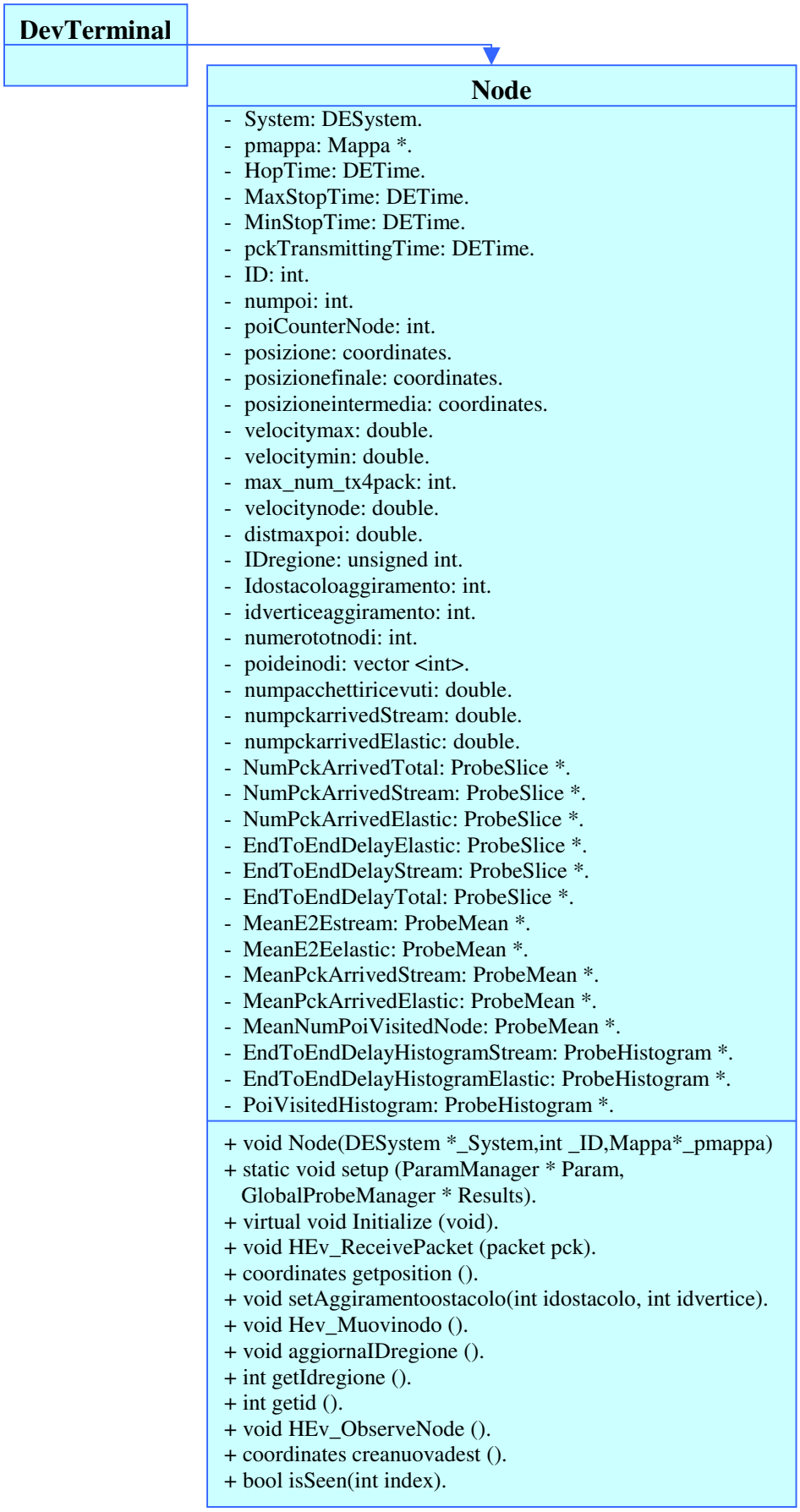



Figura 2.11: Diagrama UML de la clase Node; se muestran las variables privadas y las funciones miembro.

Como se observa en la figura 2.11, muchas de las variables de las que hace uso la clase Node son objetos hereditarios de la clase Probe que sirven para el análisis de resultados, y que forman parte de la plataforma NePSi. Así, antes de explicar estas variables se aclarará la labor de estos tipos de objetos:

- *ProbeMean*: presenta en un fichero de texto las estadísticas (media, varianza y desviación cuadrática) de cualquier parámetro que queramos presentar como resultado, cada cierto intervalo de tiempo fijo. Así, el objeto va cogiendo las muestras de este parámetro cuando se indique en el programa y modificando los valores estadísticos, que se analizan desde el inicio de la simulación hasta el instante en que se recoge la última muestra. El formato del fichero de resultados es el siguiente:

  [Tiempo de observación] TAB $\overline{x}$ TAB $\overline{x^2}$ TAB $\sigma^2$

- *ProbeSlice*: muy semejante al anterior Probe, solo que ahora en lugar de analizarse las estadísticas desde el origen de la simulación, se analizan entre cada intervalo de tiempo fijo (que coincide con el 'tiempo de observación', en que se escriben las medidas). Este tiempo fijo es un parámetro de entrada al programa de la clase System que se llama WriteStatsInterval, con lo que es el propio usuario quien lo decide. El formato del fichero de resultados es idéntico.

- *ProbeHistogram*: aquí se presenta un histograma del parámetro que se ha considerado, es decir, se representan los valores que ha tomado el parámetro frente a las probabilidades con las que los ha tomado. Para ello, debe elegirse como un parámetro uno que vaya moverse en un intervalo de valores, pero que no sea por ejemplo siempre creciente... El formato del fichero en que se presentan los resultados es complicado, pero se usa para crear una gráfica en 'gnuplot' que los muestra claramente.

Después de esto, ya sí se está en disposición de comentar el significado de cada una de las **variables privadas** de la clase Node:

- Poicounter: realmente no es una variable privada, sino todo lo contrario. Es una variable externa a todas las clases, pero la definimos aquí puesto que es en esta clase únicamente donde se modifica su valor. Es un contador que almacena el número de nodos que han visto entre todos los nodos de la simulación, y no un solo nodo como se podría pensar. Se usa al final de la simulación para mostrar la carga que ha tenido el sistema, pues de este parámetro depende el número de solicitudes al sistema. De cualquier forma, la carga se explicará mejor en el capítulo 4.

- *System*: es un puntero al objeto de la clase DESystem, pues en esta clase se definen todas las clases que operan en el simulador y además es donde se define la función *main*.

- *pmappa*: sirve para tener una referencia a la instancia de la clase Mappa, pues es la que coordina las instancias de las demás clases del programa, y además pertenecen a ella varias funciones de interés de las que se hará uso en esta clase.

- *HopTime*: variable en la que se almacena el tiempo que se va a esperar entre cada comprobación de la posición del terminal. Obviamente, solo se hará efectiva esta comprobación si el terminal está en movimiento.

- *ObserveTime*: almacena un valor temporal, el cual representa el intervalo de tiempo que ha pasar para introducir en ciertos objetos Probe las muestras correspondientes.

- *MaxStopTime*: contiene el tiempo máximo que se considera va a permanecer un terminal visitando un Punto de Interés.

- *MinStopTime*: esta variable almacena el tiempo mínimo de visita a un Punto de Interés. Dicho tiempo de visita siempre será aleatorio, limitado por los dos márgenes que le marcan las variables recién explicadas, pues así se da un mayor realismo al modelo.

- *pckTransmittingTime*: almacena el tiempo de transmisión de un paquete, que se pasa como parámetro. Este tiempo se usa para llamar a eventos de recepción de paquetes.

- *ID*: es la variable que guarda el identificador numérico de la instancia del nodo del que se trate, y es fundamental para identificar al nodo en funciones de la clase AccessPoint y de la clase Mappa.

- *numpoi*: guarda el número de Puntos de Interés que se hallan en el mapa, pues se usa en algunos bucles para cerciorarnos que se han examinado todos los ellos.

- *poiCounterNode*: es un entero que se encarga de contabilizar el número de Puntos de Interés que ya ha visitado cada nodo. Por tanto, se inicializa a cero. Se usa como variable para la adquisición de resultados, que posteriormente se pasará a los objetos tipo Probe para el análisis de resultados.

- *posizione*: almacena en cada momento las coordenadas x e y de la posición en la que se encuentra el terminal.

- *posizionefinale*: variable que contiene las coordenadas del Punto de Interés al que se está dirigiendo en cada momento el terminal.

- *posizioneintermedia*: aquí se guardan las coordenadas del vértice de un obstáculo que tiene que salvar el terminal para poder llegar a su destinación (ver apartado 2.2.3). Se utiliza siempre que no se puede llegar al Punto de Interés destino en línea recta, por ello se considera un destino intermedio que se tiene que alcanzar.

- *velocitymax*: su valor se obtiene de un parámetro de entrada, y representa la máxima velocidad media en píxeles por segundo a la que consideramos se va a mover el usuario del sistema.

- *velocitymin*: en este caso se representa la mínima velocidad media del usuario.

- *velocitynode*: entre las dos variables recién comentadas se escoge aleatoriamente un valor, que es el que se almacena en esta variable y que representa la velocidad del nodo durante un recorrido particular, con lo que este valor se reelige siempre que se llegue a un Punto de Interés, puesto que se considera que el usuario no va siempre a la misma velocidad.

- *distmaxpoi*: almacena la distancia máxima a la que se encuentra un Punto de Interés desde un terminal para considerarlo como posible próximo destino de dicho terminal (ver apartado 2.2.2), a no ser que dicho terminal ya haya visto todos los Puntos vecinos, caso en el que sí se podría enviar a un destino más lejano para que pueda contemplar la visita a la ciudad.

- *IDregione*: representa el identificador numérico de la región en la que se encuentra el nodo en cada momento.

- *Idostacoloaggiramento*: almacena el identificador del último obstáculo que ha bordeado el terminal en su trayectoria.

- *idverticeaggiramento*: aquí se guarda el identificador del vértice del último obstáculo sorteado. Esta variable y la anterior se memorizan para evitar que el algoritmo de movilidad entre en un bucle.

- *numerototnodi*: variable en que se almacena el número total de instancias de la propia clase.

- *poideinodi*: vector en el que se almacenan los identificadores de los Puntos que ya ha visitado el terminal, con el objetivo de no hacerlo pasar por el mismo.

- *numpacchettiricevuti*: variable que contabiliza los paquetes recibidos por cada nodo. Se usa como variable para la adquisición de resultados, y se pasará a los objetos Probe puesto que son los que se encargan del análisis de los resultados.

- *numpckarrivedStream*: variable del mismo tipo que la anterior, pero contabiliza solo los paquetes recibidos de tipo Streaming por cada instancia de la clase.

- *numpckarrivedElastic*: como las anteriores, pero contabiliza solo los paquetes tipo Elastic.

- *NumPckArrivedTotal*: objeto ProbeSlice para analizar las estadísticas de la variable contador *numpacchettiricevuti* (explicada anteriormente) en cada intervalo de tiempo sugerido por el usuario del programa.

- *NumPckArrivedStream, NumPckArrivedElastic*: son ProbeSlice similares al anterior, pero ahora analizando las variables *numpckarrivedStream* y *numpckarrivedElastic*, respectivamente.

- *EndToEndDelayElastic, EndToEndDelayStream*: objetos ProbeSlice que analizan el retraso que sufre un paquete tipo Elastic o Stream respectivamente, desde el momento de su creación hasta que llega de forma correcta al terminal que lo solicita (retraso End-To-End). Las medidas estadísticas se reinician cada vez que pasa un intervalo de tiempo *WriteStatsInterval* (parámetro de entrada al programa).

- *EndToEndDelayTotal*: es una instancia ProbeSlice en la que se analizan los retrasos End-To-End de todo tipo de paquetes, ya sean Stream o Elastic.

- *MeanE2Estream, MeanE2Eelastic*: instancias del tipo ProbeMean para analizar las estadísticas del retraso End-To-End de los paquetes Stream y Elastic respectivamente. Las estadísticas se tienen en cuenta desde el inicio de la simulación.

- *MeanPckArrivedStream, MeanPckArrivedElastic*: objetos ProbeMean que se encargan de hallar la media, varianza y desviación cuadrática desde el inicio de la simulación de los contadores de paquetes recibidos tipo Stream o Elastic respectivamente.

- *MeanNumPoiVisitedNode*: ProbeMean en el que se analiza la variable global *poiCounterNode*, para poder medir así los Puntos de Interés en término medio que visita cada terminal y comprobar el correcto funcionamiento del algoritmo.

- *EndToEndDelayHistogramStream, EndToEndDelayHistogramElastic*: objetos de la clase ProbeHistogram para representar la probabilidad con la que los paquetes Stream-Elastic (respectivamente) sufren un retraso End-To-End determinado.

- *PoiVisitedHistogram*: objeto ProbeHistogram para representar la cantidad de veces que se ha visitado un nodo. Es útil para comprobar que los Puntos de Interés más vistos son los más céntricos de la ciudad, pues están más cercanos entre ellos (se demostrará en el apartado 4.7).

En esta clase se encuentran las primeras funciones manejadoras de eventos de las que se hace uso en el programa. Como su propio nombre indica, todas las funciones de este tipo se invocan siempre como respuesta a un evento, y realizan todas las tareas que se le solicitan al evento. Una vez dicho esto, analizaremos al detalle las **funciones miembro** de la clase:

- *Node*: Constructor de la clase, del cual se obtienen las referencias a los objetos de la clase Mappa y System, además del identificador numérico del nodo. En esta función se igualan a las variables globales de la clase que los representan.

- *setup*: Función en la que se crea una instancia de la clase ParamManager para gestionar los parámetros de entrada que se necesitan para la clase. Tras ello, se añaden a esta clase el nombre y rango permitido para dichos parámetros. De esta forma, se permite al usuario del programa visualizar los parámetros que se necesitan en un archivo de texto. Esta función se invoca automáticamente tras la creación de la instancia, sin necesidad de ser llamada en el programa.

- *Initialize*: Es la función que se ocupa de inicializar los valores de las variables privadas de estado y de lanzar los eventos iniciales. En particular, se llama a los eventos *Ev_Muovinodo*, *Ev_ReceivePacket*, *Ev_ObserveNode*. Cada uno se volverá a llamar por otra función después de un intervalo de tiempo oportuno en cada caso. Esta función se debe invocar justo después de la creación de cada instancia de la clase.

- *HEv_ReceivePacket*: Función manejadora del evento *Ev_ReceivePacket.* Se encarga de analizar los paquetes recibidos por el terminal correspondiente, pues contienen información útil para los resultados. Para ello, examina la cabecera del paquete recibido, pues de ella obtiene parámetros tan importantes como el retraso y el tipo de paquete que porta. El retraso lo introduce directamente como muestra en las variables Probe que correspondan, puesto que es necesario analizar todos los retrasos. Por su parte, el tipo de paquete sirve para aumentar el contador de paquetes conveniente, pero no se introduce en el objeto Probe su valor para independizarlo de la recepción de paquetes, lo que podría falsear los resultados.

- *getposition*: Es un procedimiento cuya única tarea es devolver la posición del nodo en el momento en que se llama. Se llama desde la clase Mappa, cuando tiene que encontrar una destinación intermedia al nodo.

- *setAggiramentoostacolo*: Es una función que también se llama desde Mappa y que se encarga de memorizar en las variables privadas del objeto Node el último vértice y obstáculo bordeado, para evitar volver a bordearlo.

- *HEv_Muovinodo*: Manejador del evento *Ev_Muovinodo* y que se encarga del control del movimiento del nodo. Hace mover al nodo teniendo en cuenta el Modelo de Movilidad, durante un tiempo indicado en la variable *HopTime*. Tras ese tiempo solicita de nuevo el evento *Ev_Muovinodo*, con lo que se vuelve a llamar a esta función. Si en la posterior llamada se observa que el nodo no ha llegado aún a un punto reseñable, hace que siga moviéndose. En caso contrario,

crea un nuevo destino para el nodo, ya sea el siguiente obstáculo a bordear (*posizioneintermedia*), o bien un nuevo Punto que visitar (*posizionefinale*).

- *aggiornaIDregione*: Función llamada desde el manejador *HEv_Muovinodo*, y que se encarga de actualizar la región en la que se encuentra el propio nodo después de haber sufrido un desplazamiento.

- *getid*: Se usa en la función que transmite los paquetes de la clase Mappa, para averiguar cuál es el nodo al que tiene que mandar cada paquete. Devuelve el identificador numérico del objeto Node que se solicita.

- *getIdregione*: Este procedimiento devuelve el identificador numérico de la posición actual del nodo. Se usa en la clase Mappa a la hora de mandar un paquete, para comprobar si el nodo destino sigue encontrándose en la misma región que se indica en el paquete, o por el contrario se ha cambiado a otra.

- *HEv_ObserveNode*: Manejador del evento *Ev_ObserveNode*. Es un evento que se lanza cada cierto tiempo (*ObserveTime*) con independencia de los demás. Se encarga de introducir las muestras en los objetos Probe, para buscar no falsear la adquisición de los resultados. Busca por tanto adquirir los resultados en instantes desvinculados de la transmisión o más bien recepción de paquetes.

- *creanuovadest*: Esta función es invocada desde *HEvMuovinodo* cuando el nodo llega al destino, y restituye la posición de la nueva destinación, elegida de forma inteligente. Esto quiere decir que hay que tener en cuenta los Puntos de Interés que hasta ese momento ha visitado el nodo indicado, para no volver a dirigirlo a ese destino, así como las limitaciones de distancia ya conocidas (*distmaxpoi*).

- isSeen: Función auxiliar a la anterior que recibe el identificador de un Punto de Interés y devuelve TRUE o FALSE en función de que dicho Punto esté o no en la lista de Puntos de Interés visitados por el nodo.

## 2.6.3 CLASE AccessPoint

La clase *AccessPoint* representa, por su parte, la estación base de cada región. En su interior se implementan la mayoría de las funciones relacionadas con la recepción de peticiones de información y el envío de paquetes desde los buffer como respuesta a dichas peticiones.

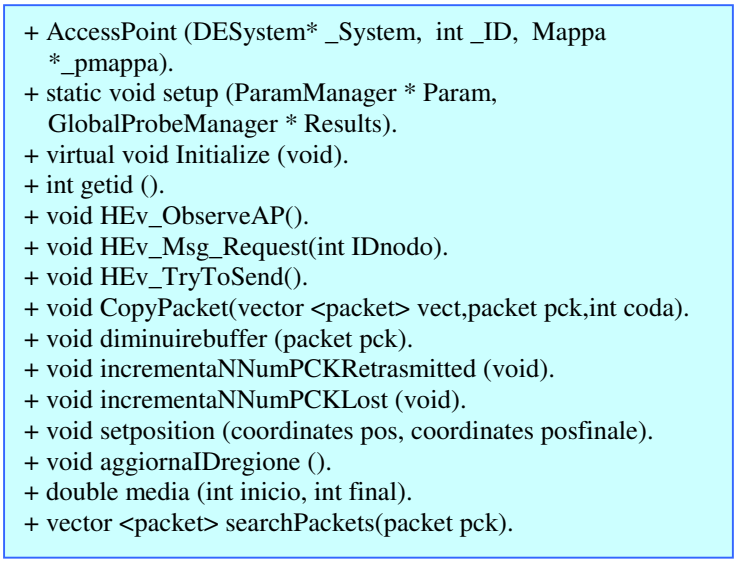



Figura 2.12: Diagrama UML de la clase Punto de Acceso; se muestran las variables privadas y las funciones miembro.

Se comenta a continuación el significado de las **variables privadas** de la clase AccessPoint:

- *System*, *pmappa*, *ObserveTime*, *max_num_tx4pack*, *pckTransmittingTime, distmaxpoi*,: están igualmente definidos en la clase Node y tienen el mismo significado que en dicha clase, ya explicada anteriormente.

- *MaxttlxpckStream*, *Maxttlxpck*, *numnodi*, *maxLunPck*: se encuentran igualmente definidos en la clase Mappa.

- *ID*: identificador numérico de cada instancia de la clase AccessPoint.

- *maxMsg4ReqStream*, *maxMsg4Req*: son variables en las que se almacenan el máximo número de mensajes de Stream o Elastic respectivamente que se pueden generar en una solicitud de información. Su valor se pasa como parámetro de entrada.

- *IDregione*: identificador numérico del objeto Regione al que pertenece este AccessPoint. En realidad posee el mismo valor que la variable *ID*.

- *messaggiocorrente*: variable que se usa en el manejador *HEv_Msg_Request*. En esta variable, que incremento siempre que el objeto AccessPoint crea un nuevo mensaje, escribo el identificador del nuevo mensaje que se creará; así cada mensaje está unívocamente determinado por la copia IDAccessPointsorgente – IDmessaggio.

- *BufferPCKStream*, *BufferPCK*: son listas de paquetes en las que se almacenan los paquetes Stream o Elastic respectivamente. Por tanto, representan realmente los dos buffer del servidor para cada Punto de Acceso. Hay que reseñar aquí que los paquetes se tratan con la modalidad FIFO.

- *meanMessageLength, meanMessageLengthStream*: representan el número medio de bytes por mensaje de tipo Elastic o Stream respectivamente.

- *minimMessaggeLength, minimMessageLengthStream*: representan el valor mínimo de bytes que va a tener un mensaje creado por la variable de Pareto correspondiente.

- *NNumPCKElasticBorn*, *NNumPCKStreamBorn*: son variables que contabilizan el número de paquetes creados en cada objeto de la clase Punto de Acceso. En la primera variable se cuentan los paquetes Elastic y en la otra los de tipo Stream. Se usan como variable para la adquisición de resultados, y se pasarán a los objetos tipo Probe puesto que son los que se encargan del análisis de los resultados.

- *NNumMSGElasticBorn, NNumMSGStreamBorn*: en este caso las variables se encargan de contabilizar el número de mensajes Elastic y Stream creados en el Punto de Acceso correspondiente. Son también variables de apoyo para los Probe.

- *NNumPCKRetrasmitted*, *NNumPCKLost*: son contadores para el análisis de resultados de los Probe. Se encargan de contar el número de paquetes retransmitidos y perdidos respectivamente. Los paquetes a retransmitir dependen

exclusivamente de la variable *FER*, mientras que los paquetes perdidos se generan al superarse el número de retransmisiones permitidos (*maxnumtx4pack*), o bien el tiempo de vida en cola de un paquete (*MaxttlxpckStream*, *Maxttlxpck*).

- *TimeResolution*: parámetro que se obtiene de la resolución temporal que se pasa como parámetro de entrada, y que como indica, expresa la cantidad de cifras que queremos que tenga la cantidad menos significativa de las variables temporales.

- *NumMSGStreamBorn*, *NumMSGElasticBorn*: son objetos de la clase ProbeSlice para analizar las estadísticas de las variables contador *NNumMSGStreamBorn, NNumMSGElasticBorn* (explicadas anteriormente), en cada intervalo de tiempo sugerido por el usuario del programa.

- *BufferElasticSize*, *BufferStreamSize*: objetos ProbeSlice que se encargan de analizar, en cada intervalo de tiempo, el tamaño de las colas de paquetes tipo Elastic y Stream de cada Punto de Acceso.

- *NumPckRetrasmitted, NumPckLostTotal*: instancias ProbeSlice como las anteriores, pero en este caso encargadas de mostrar las estadísticas asociadas a las variables *NNumPCKRetrasmitted* y *NNumPCKLost* en el intervalo de tiempo indicado en el parámetro de entrada *WriteStatsInterval*.

- *NumPckBornElastic*, *NumPckBornStream, NumPckBorn*: objetos ProbeSlice para crear las estadísticas de los paquetes Elastic, Stream y totales creados en el intervalo de tiempo ya conocido. Se apoyan en las variables *NNumPCKElasticBorn*, *NNumPCKStreamBorn y en la variable suma de ambas, respectivamente.*

- *MeanBufferElasticSize*, *MeanBufferStreamSize*: objetos ProbeMean que se encargan de hallar la media, varianza y desviación cuadrática desde el inicio de la simulación del tamaño de las colas de paquetes tipo Elastic o Stream respectivamente. Las estadísticas se toman desde el inicio de la simulación y para los dos buffer de cada Punto de Acceso.

- *MeanPckRitrasmessi*: objeto ProbeMean encargado de analizar la variable *NumPckRetrasmitted desde el instante inicial.*

- *MeanPckBornElastic*, *MeanPckBornStream*: instancias ProbeMean para las variables que contabilizan el número de paquetes creados en el Punto de Acceso (*NNumPCKElasticBorn*, *NNumPCKStreamBorn*).

- *MeanNodiPerAP*: ProbeMean que obtiene las estadísticas del número de terminales que se encuentran en cada momento en cada región.

- *LoadFactor*: instancia de la clase ProbeMean para analizar la carga a la que se ve sometido cada Punto de Acceso. Se representa por tanto de la siguiente manera:

$$LoadFactor : nodeAP * poixSecxNode * numMeanPck * pckTrasmittingTime$$

  - nodeAP: es el número de nodos de los que se encarga el AccesPoint en el momento solicitado.
  - poixSecxNode: es el número medio de Puntos de Interés que visita cada terminal por segundo. Se halla por tanto dividiendo la variable global *Poicounter* entre el número total de nodos de la simulación y el tiempo de simulación transcurrido hasta ese momento.
  - numMeanPck: su valor representa el número medio de paquetes que se solicitan por cada Punto de Interés visitado. Por tanto, multiplica el número medio de mensajes Stream por solicitud, por el número medio de paquetes por mensaje Stream y se lo añade al mismo valor calculado para los mensajes Elastic.
  - pckTrasmittingTime: expresa el tiempo de transmisión de un paquete. Es de hecho la variable que contiene ese valor.

En la clase Punto de Acceso, como en la clase Node, se definen **funciones miembro** públicas que coordinan todas las funcionalidades de la estación. A continuación se analizan al detalle:

- *Punto de Acceso*: Constructor de la clase, del cual se obtienen las referencias a los objetos de la clase Mappa y System, además del identificador numérico del propio Punto de Acceso.

- *Setup*: Función con la misma labor que los *Setup* de las clases anteriores.

- *Initialize*: Función que se ocupa de inicializar los valores de las variables privadas de estado y de lanzar los eventos iniciales. En particular, se llama a los eventos *Ev_TryToSend*, *Ev_ObserveAP*, que se volverán a llamar después de un intervalo de tiempo oportuno en función del tipo de evento. Por otra parte, en esta función se leen del archivo de parámetros los datos necesarios para el modelado de las fuentes de tráfico.

- *HEv_Msg_Request*: Manejador del evento *Ev_Msg_Request*, que se encarga de crear los mensajes y a su vez paquetes cuando recibe una solicitud de información. Una vez creados en función de una serie de parámetros aleatorios, guarda los paquetes en el buffer correspondiente. Finalmente, se toman las muestras en esta función para el objeto *LoadFactor*. Aunque el objetivo real es independizar los momentos de recogida de muestras del sistema, se hace aquí esta medición por considerar que no tendría sentido hacerla independiente.

- *HEv_TryToSend*: Manejador del evento *Ev_TryToSend*, que tiene como función intentar proceder al envío de un paquete. Si hay paquetes en la cola Stream del Punto de Acceso, se mandan; si no, se buscan en la cola de paquetes Elastic (debido a la prioridad). Si no hay paquetes en ninguna cola, se llama al propio evento tras un brevísimo espacio de tiempo. Para mandar cada paquete se hace uso de la función *spediscipacchetto* de la clase Mappa, que se encarga de localizar el nodo destinatario.

- *CopyPacket*: Esta función introduce en el buffer correspondiente del Punto de Acceso en cuestión, un vector de paquetes debido a la activación del procedimiento de deambulación comentado en el apartado 2.3.2. Este vector de

paquetes se coloca al final de los paquetes del mensaje que se esté enviando en ese momento, como ya se explicó.

- *diminuirebuffer*: Función que cancela de la cola de paquetes el último paquete que ha sido transmitido. Se usa en la función *spediscipacchetto* de Mappa después de enviar un paquete, para borrarlo.

- *incrementaNNumPCKRetrasmitted*, *incrementaNNumPCKLost*: Funciones que se utilizan también en *spediscipacchetto* de Mappa, a la hora de enviar un paquete. Si el paquete es erróneo se llama a la primera función, que aumenta el contador *NumPckRetrasmitted* de la instancia Punto de Acceso correspondiente. En caso de la pérdida del paquete se llama a la segunda función, que incrementa el número de paquetes perdidos (*NumPckLostTotal*) por el Punto de Acceso. No se incrementan directamente las variables porque son privadas, y por tanto, desde la clase Mappa no serían visibles.

- *getid*: Típica función utilizada para obtener el identificador numérico del objeto. Se usa sobretodo en la clase Mappa, cuando se está buscando un Punto de Acceso identificado por su propio identificador o por un puntero al objeto.

- *media*: Esta función halla la media aritmética de un intervalo de valores enteros cuyo extremo superior e inferior son los parámetros de entrada a la función. Se usa para averiguar el factor 'numMeanpck' del ProbeMean *LoadFactor*, es decir, el número medio de mensajes por petición.

- *searchPackets*: Procedimiento usado para extraer los paquetes de la cola correspondiente cuando se entra en el procedimiento de deambulación. Por tanto, se encarga de buscar todos los paquetes que pertenezcan al mismo mensaje, los saca del buffer en el que se encuentran y los introduce en un vector de paquetes. La función devuelve el vector de paquetes.

- *HEv_ObserveAP*: Manejador del evento *Ev_ ObserveAP*. Introduce las muestras en los objetos Probe, para buscar no falsear la adquisición de los resultados.

## 2.6.4 CLASE Ostacolo

Las instancias de la clase *Ostacolo* representan los obstáculos definidos en el área de trabajo, es decir, los edificios que existen en esta parte de la ciudad.

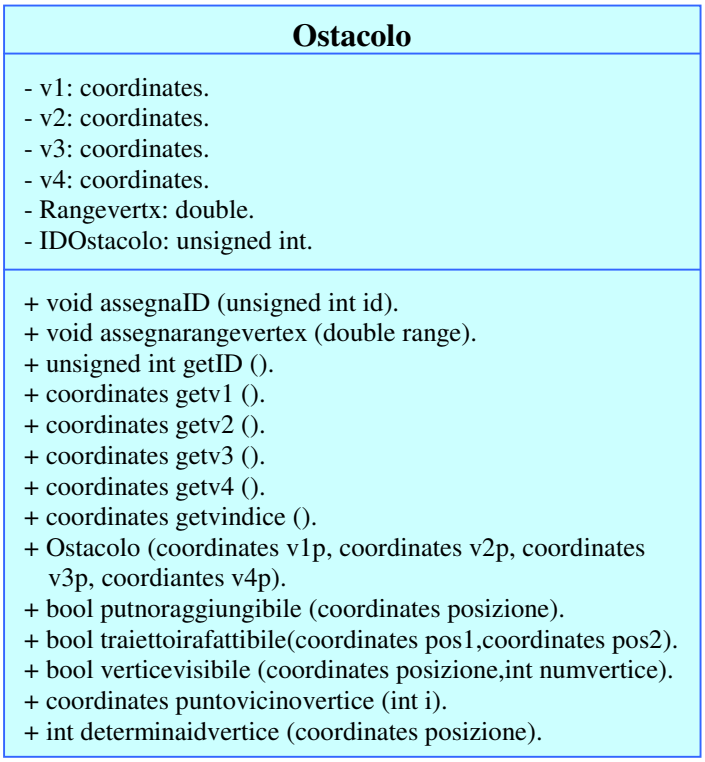



Figura 2.13: Diagrama UML de la clase Ostacolo; se muestran las variables privadas y las funciones miembro.

Cada instancia de la clase viene caracterizada por las **variables privadas** que se muestran a continuación:

- *v1, v2, v3, v4*: variables en las que se guardan las coordenadas de cada vértice del obstáculo representado en la instancia. Respectivamente, representan el vértice superior izquierdo, el vértice superior derecho, el vértice inferior derecho y el vértice inferior izquierdo del obstáculo tal y como viene representado en el mapa.

- *IDOstacolo*: identificador numérico para distinguir los objetos de la clase.

- *Rangevertx*: variable que obtiene su valor del parámetro de entrada que lleva su mismo nombre y cuyo significado ya se explicó.

A continuación se da una breve explicación de las **funciones miembro**:

- *assegnaID, assegnarangevertex*: Son funciones que únicamente recogen un valor y se lo asignan respectivamente al identificador numérico del objeto y al la variable privada *Rangevertx.* Se usan en la inicialización del objeto por parte de la clase Mappa.

- *Ostacolo*: Constructor de la clase. Recibe como parámetros las coordenadas de cada uno de los vértices en el orden oportuno, y los asigna a las variables privadas correspondientes.

- *getID*, *getv1*, *getv2*, *getv3*, *getv4*: Son procedimientos que devuelven el valor del identificador y de cada uno de los cuatro vértices, respectivamente, del objeto referido.

- *getvindice*: Función que devuelve las coordenadas del vértice del obstáculo indicado en el parámetro de entrada. Se usa en la función *trovadestintermedia* de la clase Mappa, para averiguar las coordenadas del vértice que hay que bordear.

- *puntoraggiungibile*: Esta función acepta como entrada las coordenadas de una posición y devuelve TRUE o FALSE en función de que dicha posición caiga dentro del obstáculo o no. Para ello, calcula la intersección de las rectas v1-v2 (recta que pasa por vértices 1 y 2) y la recta v3-v4 con la recta vertical que pasa por la posición dada. Mira por tanto si la ordenada de la posición se encuentra en medio de las 2 intersecciones. Hace lo mismo con las rectas v2-v3 y v1-v4, considerando ahora el corte con la recta horizontal y la abcisa de la posición. La posición cae dentro del obstáculo si en ambos casos se encuentra dentro del intervalo de las intersecciones. Función auxiliar de *posizioneraggiungileblobal*.

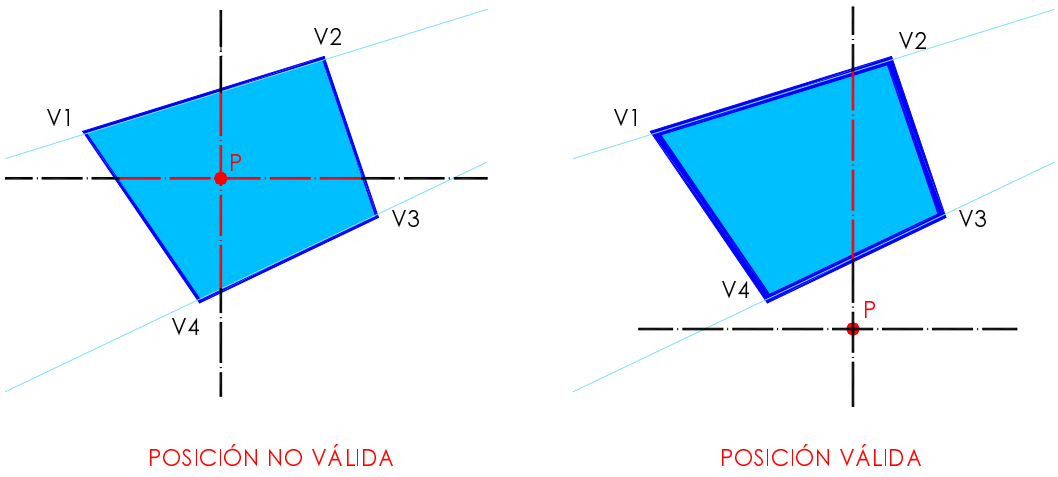



Figura 2.14: Explicación gráfica de la función '*puntoraggiungibile*'.

- *traiettoriafattibile*: Función que recibe dos posiciones que definen una recta (origen y destino del terminal). Esta función devuelve verdadero o falso en función de que la recta no corte al obstáculo indicado o sí lo haga. Para establecer si la recta es una trayectoria factible o no, se calcula la intersección de la trayectoria con los lados del obstáculo y se comprueba si el punto de intersección, por ejemplo con el lado v1-v2, cae entre v1-v2 y entre pos1-pos2 (origen-destino). Esto se hace para los otros 3 lados. Si la recta cumple la condición anterior para uno o más lados del obstáculo, significa que la trayectoria no es factible. Se usa en *traiettoriafattibileglobal* de la clase Mappa.

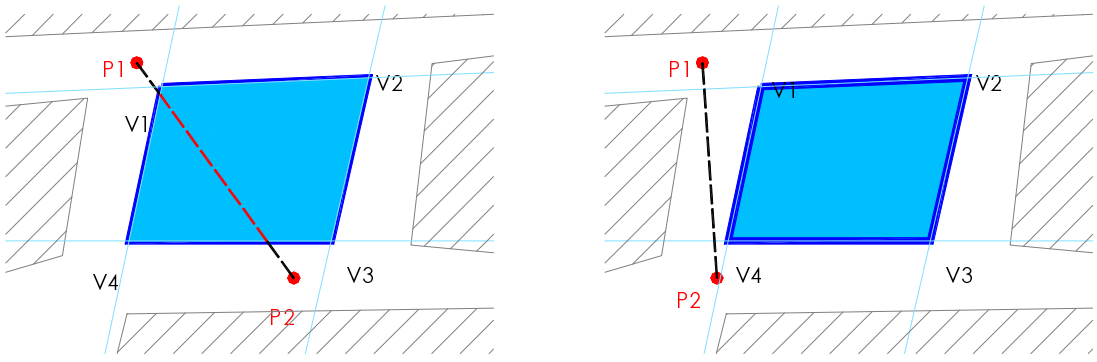



Figura 2.15: Explicación gráfica de la función '*traiettoriafattibile*'.

- *verticevisibile*: Esta función recibe la posición en la que se encuentra un terminal y el número del vértice que se quiere examinar. Devuelve en un valor booleano si el vértice es visible desde la posición o por el contrario no lo es. Para ello,

busca la intersección de la recta que pasa por la posición y el vértice solicitado con la recta que forman dos de los vértices opuestos a este. Se comprueba para las dos rectas que pueden formar los vértices opuestos. Si la intersección está comprendida entre posición y vértice, y además entre los dos vértices opuestos, entonces el resultado devuelto es FALSE. Si no está comprendida en ninguna de las dos posibilidades que se indican, entonces el resultado devuelto es TRUE. Se usa en *trovadestintermedia* de Mappa para averiguar el vértice más cercano al terminal, pues éste ha de ser visible a su vez para el terminal.

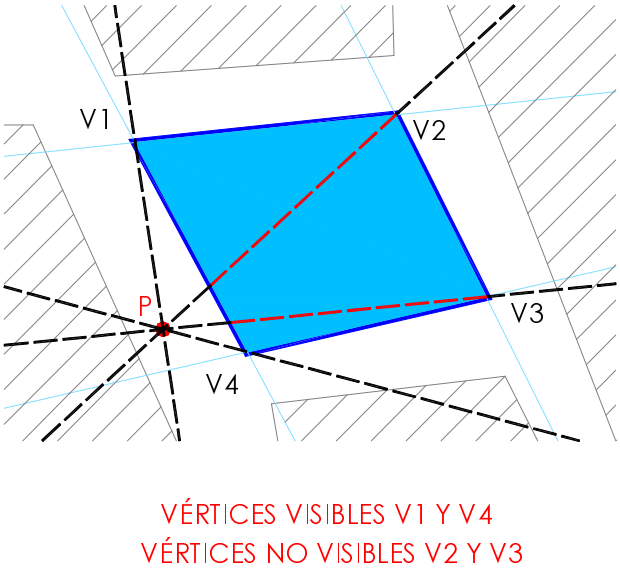



Figura 2.16: Explicación gráfica de la función '*verticevisibile*'.

- *puntovicinovertice*: Función que restituye las coordenadas del punto exterior al obstáculo que esté situado en las proximidades del vértice cuyo índice se pasa como parámetro. Es decir, si por ejemplo el parámetro lleva el valor 1, se calcula la recta que pasa por los vértices v1-v3 y se coge un punto sobre dicha recta que sea exterior al obstáculo y que diste como máximo un valor *Rangevertx* de v1. Se usa en *trovadestintermedia* de Mappa, para sortear un obstáculo.

- *determinaidvertice*: Se usa en algunas funciones de la clase Mappa en las que se conocen las coordenadas del vértice pero no su índice. Por tanto, la función devuelve el número de vértice a partir de sus coordenadas.

## 2.6.5 CLASE Regione

Las instancias de la clase *Regione* representan los *clusters* definidos en el área de la simulación. Dichas instancias son útiles sobretodo para la gestión de la deambulación y el área de cobertura.

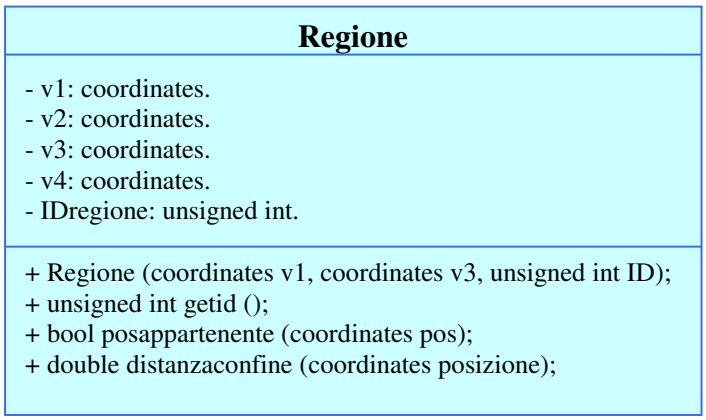



Figura 2.17: Diagrama UML de la clase Regione; se muestran las variables privadas y las funciones miembro.

A la vista de la figura, se observa que las únicas **variables privadas** de la clase son las coordenadas de los cuatro vértices que definen el rectángulo (*v1, v2, v3, v4*) y el identificador numérico de la región (*IDregione*). Por su parte, las **funciones miembro** de la clase son las siguientes:

- *Regione*: Constructor de la clase. Únicamente recibe las coordenadas del vértice superior derecha y el vértice inferior izquierda, además del identificador numérico de la región. Solo se reciben dos vértices porque al ser un rectángulo son suficientes para definir todas las coordenadas del mismo. Hay que comentar a su vez que la numeración de las instancias Regione respecto al mapa es de abajo hacia arriba y de izquierda hacia derecha.

- *getid*: Función que únicamente devuelve el identificador del objeto. Se usa para conocer la región en la que se encuentra un nodo que se hace en la clase Mappa. Desde allí, se busca la región a través de un puntero a su objeto y se desconoce el identificador de la misma.

- *posappartenente*: Función que sirve para saber si un terminal, del que se pasa su posición, se encuentra en la región (u objeto de la clase). Se usa, como la función anterior, para conocer en qué región se encuentra el propio terminal.

- *distanzaconfine*: Esta función restituye la distancia de la posición que se pasa como parámetro de entrada, al lado más cercano a ésta de la región. Es una función clave en la deambulación, puesto que comprueba si el terminal se ha separado lo suficiente (*distHist*) de su región de partida. Si es así, habría que considerar el envío de sus paquetes desde el Punto de Acceso de la nueva región en la que se encuentra en ese instante.

### 2.6.6 CLASE Common

La clase Common se usa para definir una serie de variables y funciones que van a ser comunes a las demás clases, de ahí su nombre. Puesto que van a ser utilizadas por algunas de las demás clases, las variables de esta clase se declaran como públicas. De hecho, son únicamente estructuras de uso general en el sistema, como se puede observar en la figura 2.18.

En esta clase, además, se introducen los <include> comunes y necesarios para que sea posible la ejecución del programa. Dichos <include> pueden ser librerías del C++ para el tratamiento de caracteres, tiempo, funciones matemáticas, listas, vectores... Estamos hablando de librerías del tipo string, ctime, cmath, list o vector. Los <include> también pueden ser ficheros .h pertenecientes a la plataforma NePSi usados para la creación de distribuciones aleatorias o para herencias, caso de los archivos rndgen.h o DevTerminal.

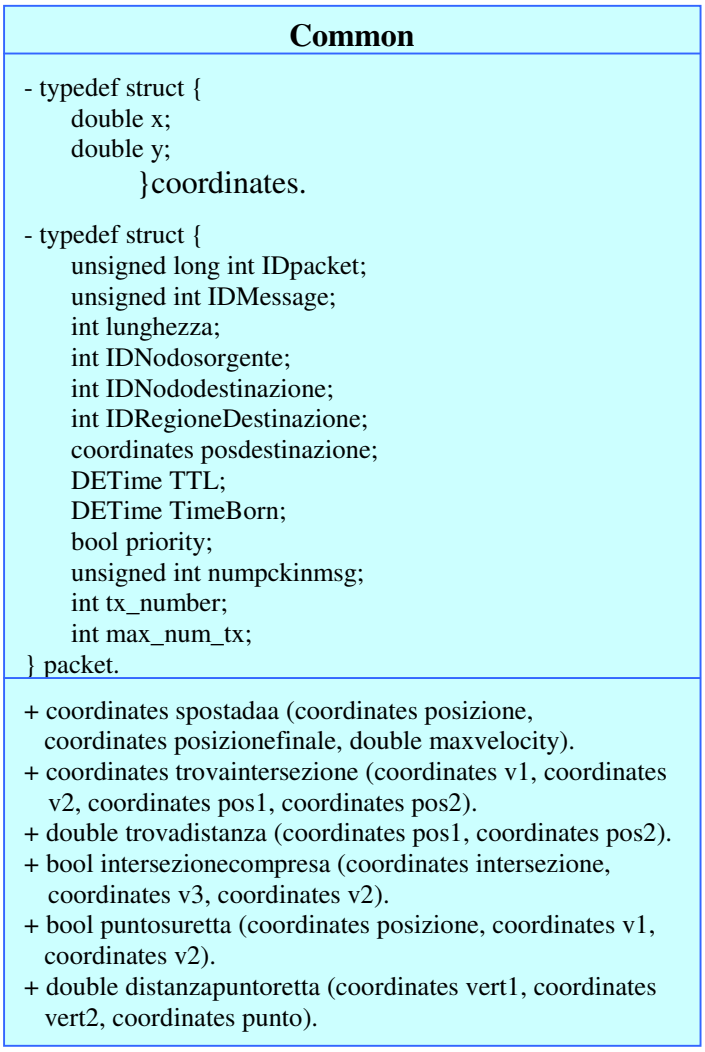



Figura 2.18: Diagrama UML de la clase Common; se muestran las variables públicas y las funciones miembro.

A continuación se explican los parámetros de las estructuras y a su vez **variables públicas** de la clase Common:

- *coordinates*: Estructura ya mencionada en varias ocasiones y que almacena las variables x e y, que son las coordenadas de un punto sobre el mapa.

- *packet*: Estructura que almacena la cabecera del paquete que se va a mandar en nuestro sistema. Comentar que la cabecera tiene una longitud de 26 bytes y que su composición campo a campo se estudia en el apartado 2.3.4.

Las **funciones miembro** que definen a la clase Common son de tipo matemático y se usan para el correcto funcionamiento del Modelo de Movilidad propuesto. Suelen ser funciones auxiliares con tareas muy concretas, pero que se usan en varias clases:

- *spostadaa*: Función llamada desde el manejador *HEv_Muovinodo* en la clase Node, y que recibe la posición en ese instante de un nodo, la posición a la que se quiere dirigir (sin obstáculos intermedios) y la distancia que se quiere desplazar. Se encarga por tanto de desplazar al nodo la distancia indicada en el parámetro de entrada, hacia la posición destino en línea recta. Esta posición destino no tiene porqué ser un Punto de Interés, puede ser un punto cercano a un obstáculo que hay que bordear para llegar a dicho Punto.

- *trovaintersezione*: Se usa como función auxiliar para algunas funciones de la clase Ostacolo. Devuelve las coordenadas de la posición intersección de dos rectas. Cada una de estas rectas están definidas por dos puntos que se pasan como parámetro de entrada. Si las rectas son paralelas, devuelve (0,0), y si son coincidentes devuelve (-1,-1) para evitar confusiones.

- *intersezionecompresa*: Función auxiliar utilizada en varias funciones que se encarga de comprobar si el punto intersección de dos rectas se encuentra entre los dos puntos que definen a una de ellas y que se pasan como entradas. Devuelve un valor booleano, TRUE si se encuentra entre los dos puntos.

- *puntosuretta*: De nuevo función auxiliar, en este caso encargada de comprobar si un punto está en la recta que definen otros dos puntos que se pasan como parámetro de entrada.

- *distanzapuntoretta*: Esta función devuelve la distancia de un punto a una recta. Se usa por ejemplo en la gestión de la deambulación, para comprobar si un punto se ha alejado lo suficiente de una región como para cambiarlo de región.

- *trovadistanza*: Función auxiliar que sirve para hallar la distancia entre dos puntos que son los parámetros de entrada.

## 2.6.7 CLASE SysTest

Clase que se obtiene por herencia de la clase DESystem. Se encarga de definir las clases que intervienen en el simulador y además contiene el programa principal o *main*.

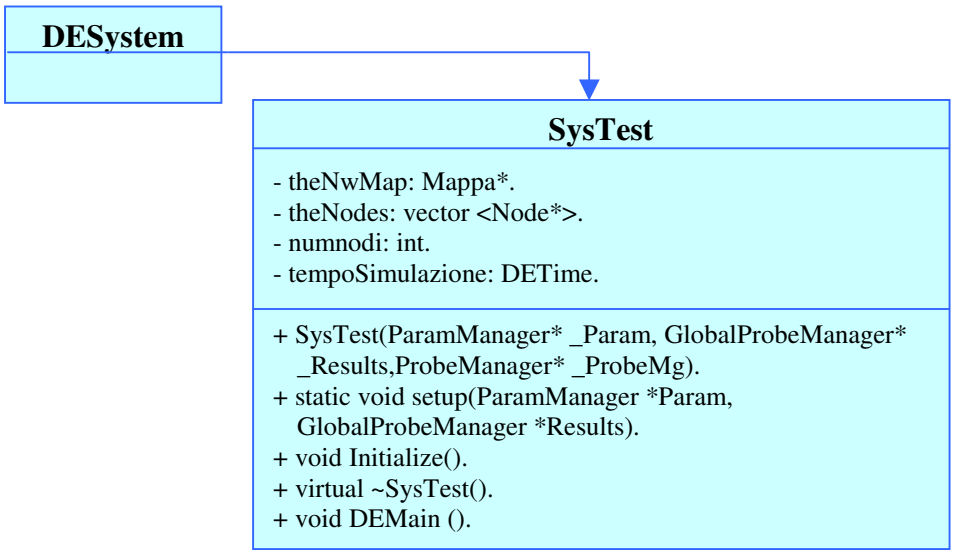



Figura 2.19: Diagrama UML de la clase SysTest; se muestran las variables privadas y las funciones miembro.

De esta forma, algunas de las **variables privadas** son referencias a las clases principales (*NwMap* a la clase Mappa y *theNodes* a un vector de la clase Node), que se crean e inicializan en esta clase. La variable *numnodi* se usa para asegurar la creación del número correcto de instancias de la clase Node. Por su parte, *tempoSimulazione* almacena el tiempo de duración de la simulación, que interviene aquí puesto que en esta clase se crea todo el entorno de la propia simulación.

Las **funciones miembro** desempeñan las siguientes tareas:

- *SysTest*: Es el constructor de la clase, y donde se pone de manifiesto la herencia desde la clase DESystem adquiriendo sus parámetros. Inicializa las variables privadas que son referencias a otras clases.

- *setup*: Función como las que llevan su nombre en las demás clases. Se crea una instancia de la clase ParamManager para gestionar los parámetros de entrada que se necesitan para la clase.

- *Initialize*: Esta función, en primer lugar adquiere el valor para algunas de sus variables de los parámetros de entrada correspondientes. Posteriormente, procede a la creación del objeto Mappa, a la creación de los objetos Punto de Acceso y por último a la creación de los Node.

- *~SysTest*: Es el destructor de la clase. Es la última función que se ejecuta en la simulación, y por tanto, contiene las que queremos que sean las últimas órdenes. Son órdenes para liberación de la memoria, para mostrar el resultado final de algunas variables...

- *DEMain*: Es la función *main* propiamente dicha. Únicamente se ejecuta una orden en esta función (*RunSimul*), pero que es la encargada de dar inicio a la simulación.

### 2.6.8 CLASES DEevent

Las clases Evento (DEevent) sirven para definir la sucesión correcta de eventos síncronos y asíncronos que se llaman durante la simulación.

La plataforma NePSi, de hecho, pone a disposición de cada clase derivada una referencia temporal común. La creación dinámica de más instancias de las clases DEevent permite, en consecuencia, la llamada a las funciones miembro de los objetos en instantes temporales precisos. Ese mecanismo consiente la sincronización temporal entre las varias funcionalidades de los objetos a examen.

En nuestro caso ha sido necesario definir las clases DEevent que se explican a continuación. Se hace una breve explicación de ellas puesto que sus manejadores, que

son los que realmente realizan la tarea, se han ido explicando en las clases anteriores que los poseen. Son las siguientes:

- *Ev_Muovinodo*: Este evento gestiona la movilidad de los terminales. Se llama cada intervalo de tiempo HopTime (especificado en el archivo de parámetro) de cada instancia de la clase Node. Desde aquí se invoca a la función HEv_Muovinodo que se encarga de cambiar la posición actual del nodo en función de los parámetros del modelo de movilidad.

- *Ev_ReceivePacket*: Dicho evento se lanza desde la clase Mappa y gestiona la recepción en el nodo de un paquete expedido por el Punto de Acceso. Se lanza por tanto cuando un Punto de Acceso va a enviar un paquete.

- *Ev_ObserveNode*: Este evento se llama periódicamente desde su propio manejador de la clase Node para la adquisición de muestras que después muestran el resultado a través de los Probes. Efectivamente, hay una llamada inicial al evento en la función *Initialize* de Node para poder crear el bucle de llamadas.

- *Ev_Msg_Request*: Tal evento se lanza de forma asíncrona desde una instancia de Node cuando el nodo llega a un Punto de Interés. El Punto de Acceso que recibe el evento procede a la elaboración de la información (lo que realmente se hace en el servidor).

- *Ev_TryToSend*: Este evento se llama síncronamente desde cada instancia de la clase Punto de Acceso para la expedición de paquetes de las colas.

- *Ev_ObserveAP*: La misma función que Ev_ObserveNode, aplicada en este caso a los Punto de Acceso.

# Capítulo 3
# Pruebas del simulador

## *3.1 Introducción*

En el presente capítulo se pretende demostrar, con simulaciones que lo corroboren, las principales características del simulador que se expone en este Proyecto Fin de Carrera:

- Modelo de movilidad.
- Correcta transmisión y recepción de paquetes y mensajes.

Por este motivo, en el programa se colocarán líneas de código de forma estratégica que permitan visualizar por pantalla los resultados más pertinentes en cada caso. Esto es, se introducen en el código una serie de comandos de salida que irán variando en función del apartado, pues se pretenden demostrar diferentes cosas para cada uno de ellos.

Antes de comenzar a ilustrar los resultados de las simulaciones de prueba, comentar que dichas simulaciones se han desarrollado sobre el mismo escenario en el que se han desarrollado el resto de simulaciones y que se presenta en el apartado 4.1. De hecho, la mayoría de los parámetros de entrada también se han escogido idénticos a algunos de los que se muestran en el apartado 4.2.

De todos modos, hay que indicar que se ha elegido un escenario complejo para realizar estas pruebas (centro histórico de una ciudad, con muchos obstáculos...) para probar el simulador en situaciones de extrema complejidad. Esto garantizará que funcione para todo tipo de situaciones. Por otro lado, comentar que la adquisición de los parámetros de entrada del siguiente apartado se lleva a cabo para comprobar que el programa funciona en situaciones reales y con parámetros reales.

## 3.2 Pruebas asociadas al Modelo de Movilidad

Para comenzar con las pruebas, se harán sobre el modelo de Movilidad explicado en el apartado 2.2. Sin duda alguna, ésta es la característica del programa más compleja y que más esfuerzos ha supuesto, tanto en la creación de su modelo como en el desarrollo práctico del mismo. No obstante, al final se han alcanzado los resultados esperados y que se muestran posteriormente.

Para las dos primeras pruebas realizadas sobre la Movilidad se presentarán ambas imágenes de la interfaz del programa en el que se han hecho las simulaciones. Esto se hace con el objetivo de mostrar los resultados que se presentan por pantalla. Efectivamente, estos resultados estarán asociados a valores de posición de un nodo de la simulación elegido al azar. Los valores de posición presentados serán los asociados a posición inicial, posiciones intermedias y finales que alcanza un nodo durante la simulación.

Así pues, el resultado asociado a la primera prueba es el que se muestra debajo:

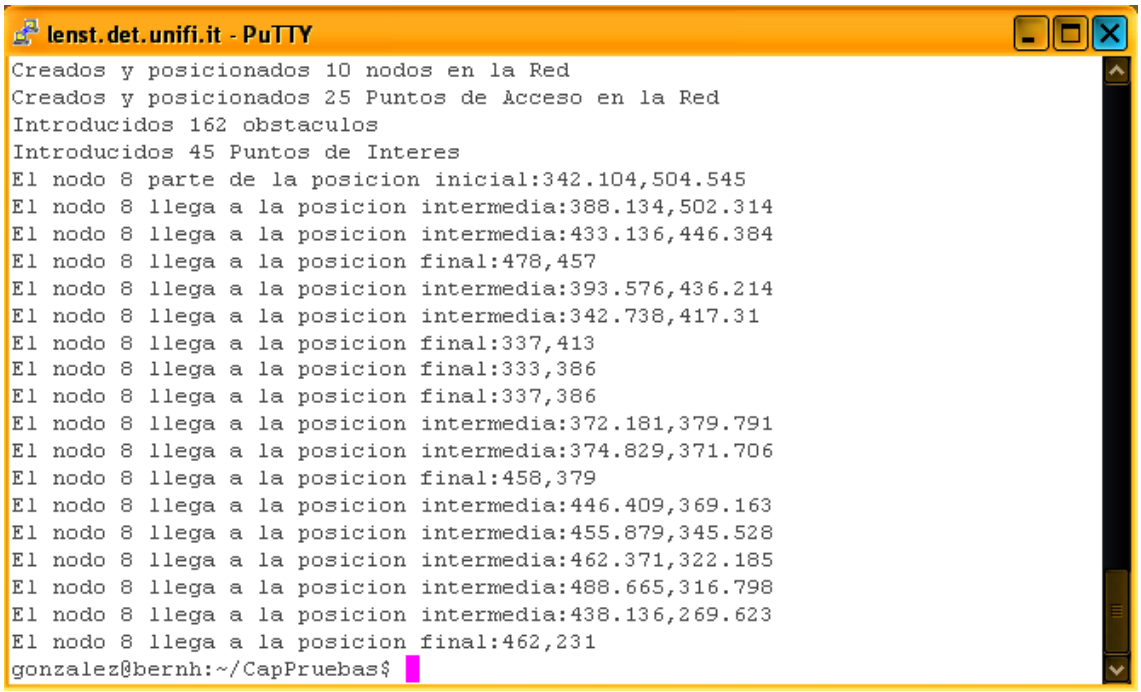



Figura 3.1: Resultados asociados a la primera prueba de la Movilidad.

Como ya se comentó, es una imagen de la interfaz del programa con el que se han hecho las simulaciones. Efectivamente se trata del programa Putty, un programa usado para hacer operaciones en servidores y computadores remotos. Se ha hecho uso de este programa para poder emplear los potentes computadores de que el alumno dispuso durante su colaboración en el Laboratorio de Telemática de la Facultad de Ingeniería de Florencia, y además disponer con mayor facilidad de los recursos de que allí hizo uso.

Respecto a los resultados mostrados, observar cómo inicialmente se exponen los valores de algunos de los principales parámetros de la simulación. Estos resultados se muestran durante la fase de inicialización de la simulación con el objetivo de que el usuario compruebe la correcta captación de datos. El resto de líneas de resultados presentan las posiciones ya indicadas en píxeles. Así expuestas no aportan nada, pero se muestran para que el lector pueda verificar, al menos visualmente, que las posiciones que más abajo se señalan en el mapa elegido son reales.

A continuación se presentan los resultados asociados a la segunda prueba, y que junto a los obtenidos en la primera, se exponen posteriormente en el mapa ya mencionado:

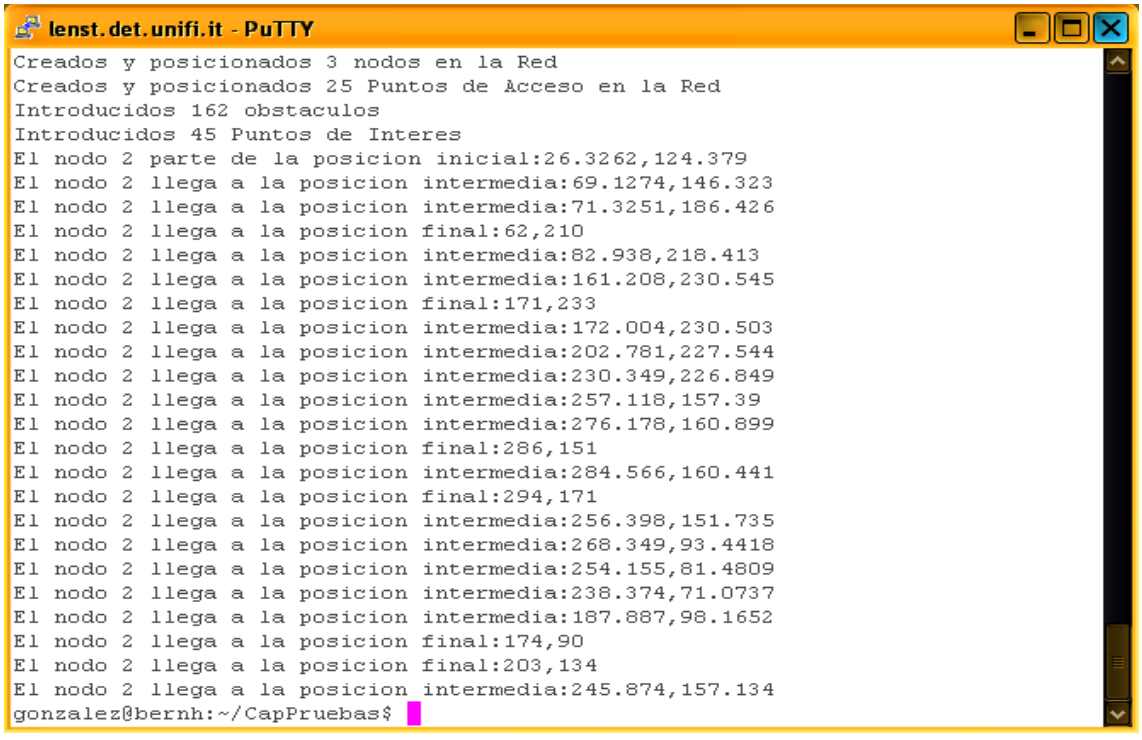



Figura 3.2: Resultados asociados a la segunda prueba de la Movilidad.

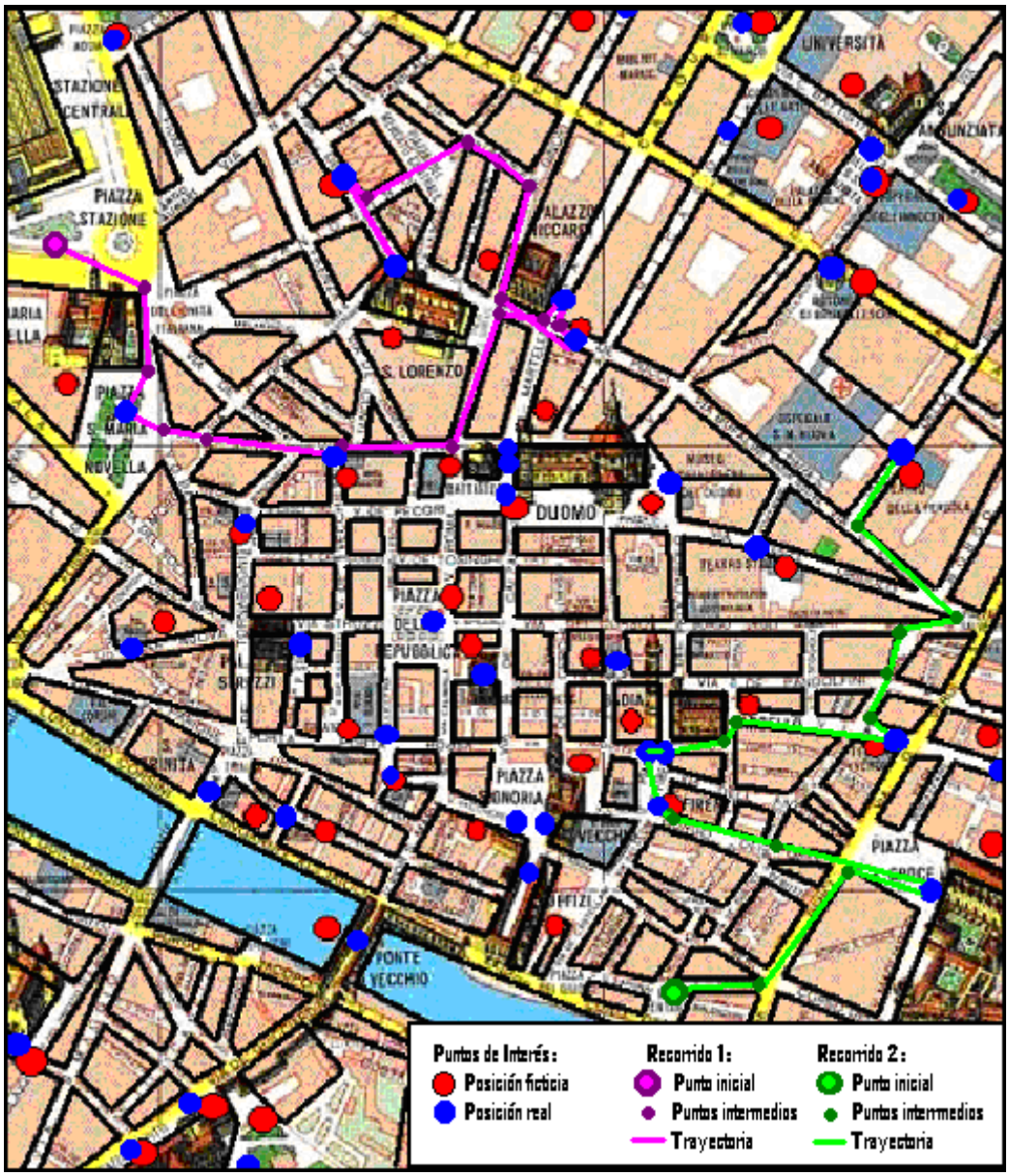



Figura 3.3: Mapa con los itinerarios seguidos por los nodos de las dos primeras pruebas.

Los puntos azules son las coordenadas que realmente representan a los Puntos de Interés, como bien se indica. Los puntos rojos que inicialmente los representaban en el mapa no están situados en posiciones alcanzables (están dentro de obstáculos), o bien no están colocados en la posición exacta en la que está situado el obstáculo.

Con la intención ya sabida de poder comprobar visualmente los resultados de las dos primeras pruebas, se quiere señalar que las dimensiones del mapa elegido son 515x 600 píxeles, y recordar que las coordenadas en el programa se numeran de izquierda a derecha (eje x), y de arriba abajo (eje y).

Además de para verificar la correcta colocación de los puntos, la principal razón de presentar el mapa con los itinerarios es hacer ver cómo el nodo busca su próximo destino con la mayor eficiencia posible, es decir, con el camino más corto. Esto es el fruto de haber elegido la línea recta hacia la destinación como referencia para bordear los obstáculos, como bien se indica en el apartado 2.2.3. Eso sí, podría resultar extraño que el nodo en ocasiones no se dirija hacia el Punto de Interés que tiene más cercano, pero esto es debido a la propia aleatoriedad que se propone en el Modelo. Una vez que el nodo llega a una destinación final, se escoge aleatoriamente de entre todos los Puntos de Interés situados en un cierto rango de distancia, el próximo Punto de Interés al que dirigirse (ver en este caso apartado 2.2.2). Es lógico pensar que esa opción no es la más efectiva respecto al movimiento del nodo, y de hecho no lo es, pero se escoge por dos razones:

- Garantizar que todos los Puntos de Interés cercanos tienen la probabilidad de ser alcanzados por el nodo.
- Garantizar que se producen diferentes alternativas al llegar a un Punto de Interés. Esto es, cada nodo va a poder continuar con diferentes trayectorias pese a haber visitado en algún momento el mismo Punto de Interés. Esto da mayor riqueza al programa, ya que se permite la creación de miles de itinerarios diferentes.

Por otro lado, se quiere hacer notar que 6 es el número de Puntos de Interés visitados por cada uno de los nodos de que se ha estudiado su trayectoria. Considerando que el tiempo de simulación es de 2 horas y que cada nodo se para entre 5 minutos y media hora por Punto de Interés, significa una gran eficacia.

Para hacer más cuantiosa la comprobación de la movilidad, se muestran a continuación una serie de imágenes del área de trabajo seleccionada con diferentes itinerarios recorridos por los nodos. Son los resultados asociados a nuevas pruebas realizadas. En estas nuevas pruebas, el tiempo se ha visto aumentado de 7200 a 10800 segundos, para poder mostrar trayectorias más largas. No se vuelven a mostrar las imágenes del programa con los resultados porque ya se pudo comprobar anteriormente que coincidían con los mostrados en el mapa.

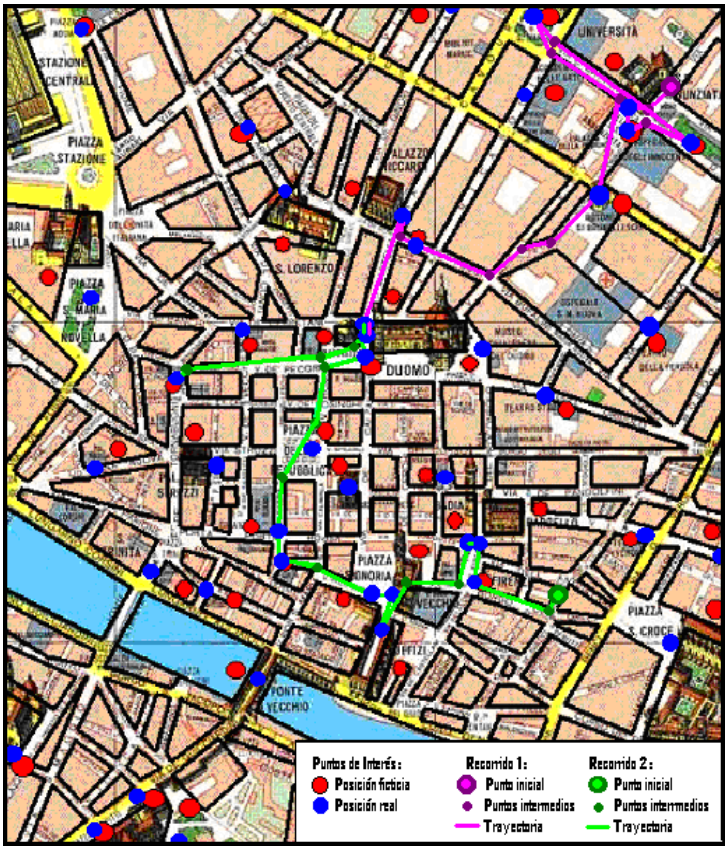



Figura 3.4: Mapa con los itinerarios seguidos por los nodos de la 3ª y 4ª prueba.

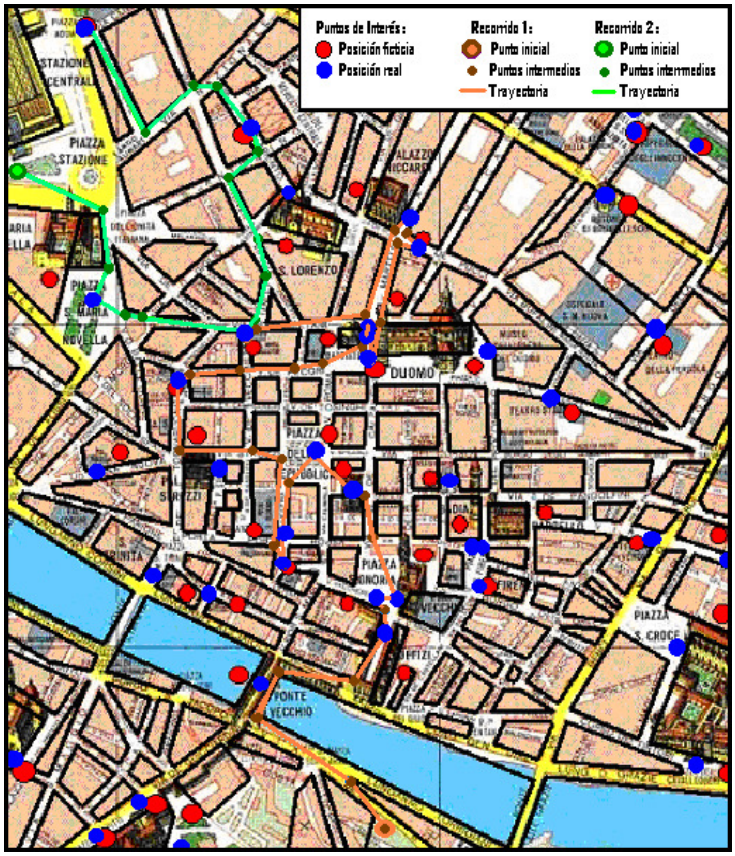



Figura 3.5: Mapa con los itinerarios seguidos por los nodos de la 5ª y 6ª prueba.

En las dos figuras anteriores se sigue observando la eficacia de las trayectorias, intentando buscar la línea recta en todo momento. De hecho, en cada uno de los recorridos se visitan respectivamente 12, 9 e incluso 14 Puntos de Interés. Eso sí, en el primer caso de la segunda figura se observan tan solo 4 Puntos de Interés, lo que se debe a la amplia distancia entre los mismos, y que el tiempo de permanencia en ellos ha sido elevado.

La figura que se muestra a continuación se presenta únicamente para acabar de ilustrar y corroborar el correcto funcionamiento del Modelo de Movilidad.

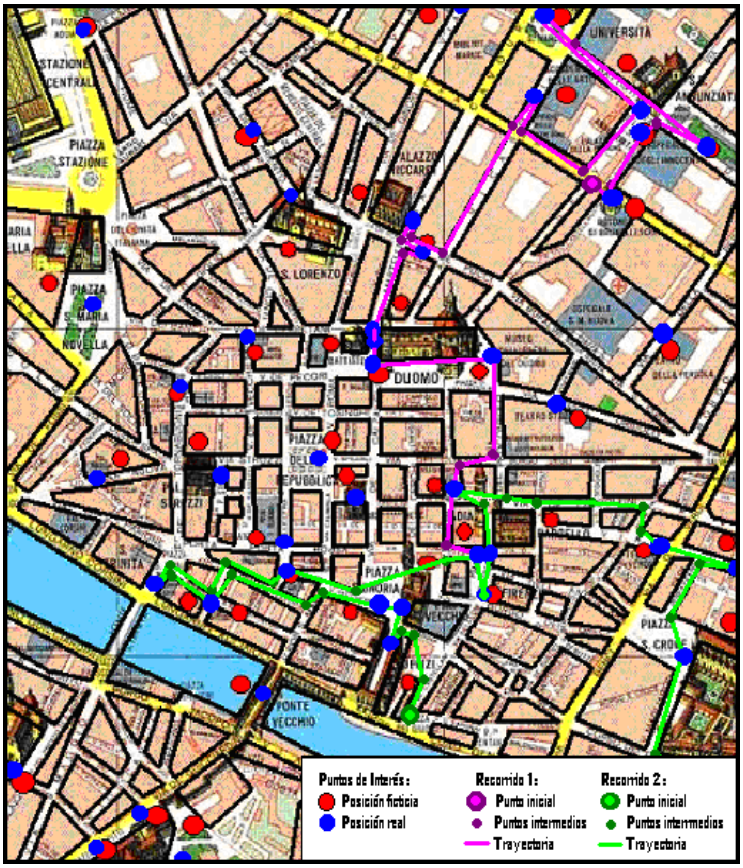



Figura 3.6: Mapa con los itinerarios seguidos por los nodos de las dos últimas pruebas.

## 3.3 Pruebas asociadas a la transmisión

En este caso se pretende demostrar cómo cada una de las peticiones de información de un nodo son atendidas cómo se describía en el apartado 2.3. Es decir, demostrar que el nodo llega a un Punto de Interés y solicita información sobre éste. Entonces, en el servidor se crean un número aleatorio de mensajes, definido entre unos márgenes, de cada tipo de mensajes. De esta forma son, entre 1 y un número máximo que se indica como parámetro, los mensajes creados de tipo Stream, y entre 0 y otro valor máximo, los mensajes creados de tipo Elastic. Se pretende probar, a su vez, que los mensajes los entrega el Punto de Acceso de forma ordenada (dando prioridad a los mensajes Stream), y corroborar que los paquetes que componen el mensaje se entregan en secuencia.

Por tanto, en cada una de las simulaciones de prueba que se realizarán a continuación, se expondrán tres tipos de líneas:

- Presentando la posición e identificador de un nodo que llega a un Punto de Interés, sobre todo para certificar que debe comenzar a recibir información.
- Indicando el número del Punto de Acceso que se dispone a enviar un mensaje. Esta información irá acompañada por el tipo de mensaje que se va a enviar, así como del identificador de mensaje y el número de paquetes que lo componen. Para finalizar, también se muestra el número de nodo al que se envía el mensaje.
- Líneas asociadas a la recepción de paquetes. Se presentan únicamente en los instantes en que se reciben el primer y/o el último paquete de un mensaje. Además, en caso de recibirse el último paquete se indica el número de paquetes recibidos del mensaje, para comprobar que se ha completado el mismo.

Las cuatro pruebas a realizar tendrán como características principales las que se muestran a continuación. Se han elegido así para poder mostrar todas las situaciones diferentes que se pueden mostrar en la transmisión:

- 1 solo nodo y 1 solo Punto de Acceso.
- 1 solo nodo y 64 Puntos de Acceso.
- 3 nodos y 1 solo Punto de Acceso.
- 3 nodos y 64 Puntos de Acceso.

El número máximo de mensajes Stream y Elastic que se reciben por petición, se ha fijado en 3 para las 2 primeras pruebas. Para las otras 2 pruebas se ha reducido a 1 solo mensaje. Además, la longitud del paquete se ha establecido en 1500 B, y las longitudes medias de los mensajes Stream y Elastic se ha establecido en 316 y 14136 B respectivamente.

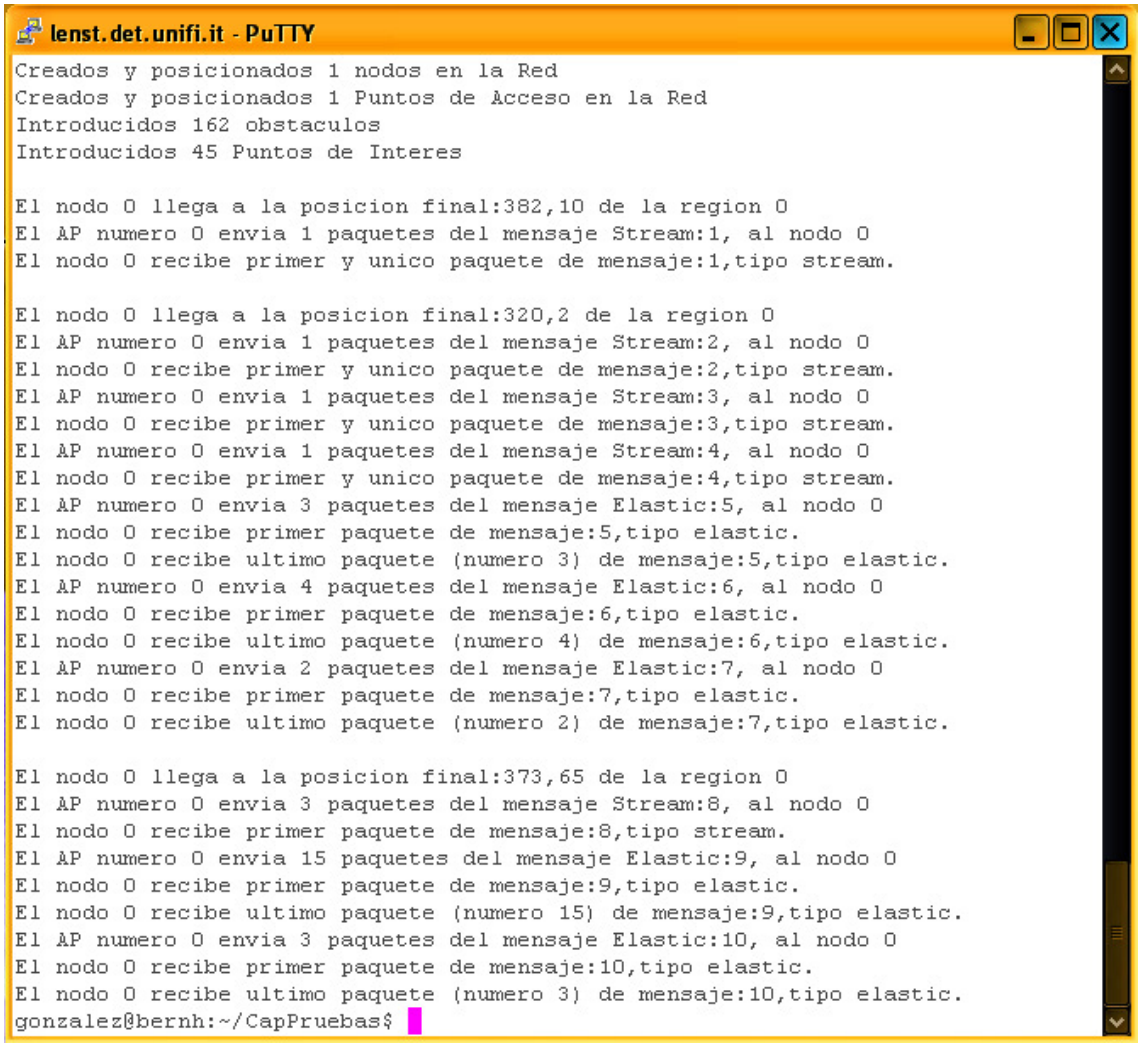



Figura 3.7: Resultados asociados a la prueba con 1 nodo y 1 Punto de Acceso.

La figura deja muy claro el correcto funcionamiento del protocolo de comunicación. Solo comentar los mensajes se comienzan a enumerar desde el número 1 en cada Punto de Acceso, y que las aleatoriedades al elegir el número de mensajes a enviar provocan las grandes diferencias de flujo de información de la primera (1 mensaje Stream y 0 mensajes Elastic) a la segunda solicitud de información (3 mensajes Stream y 3 mensajes Elastic). Por lo demás, se comprueba el correcto orden de creación y recepción de paquetes, y que los mensajes Stream gozan de prioridad y menor tamaño.

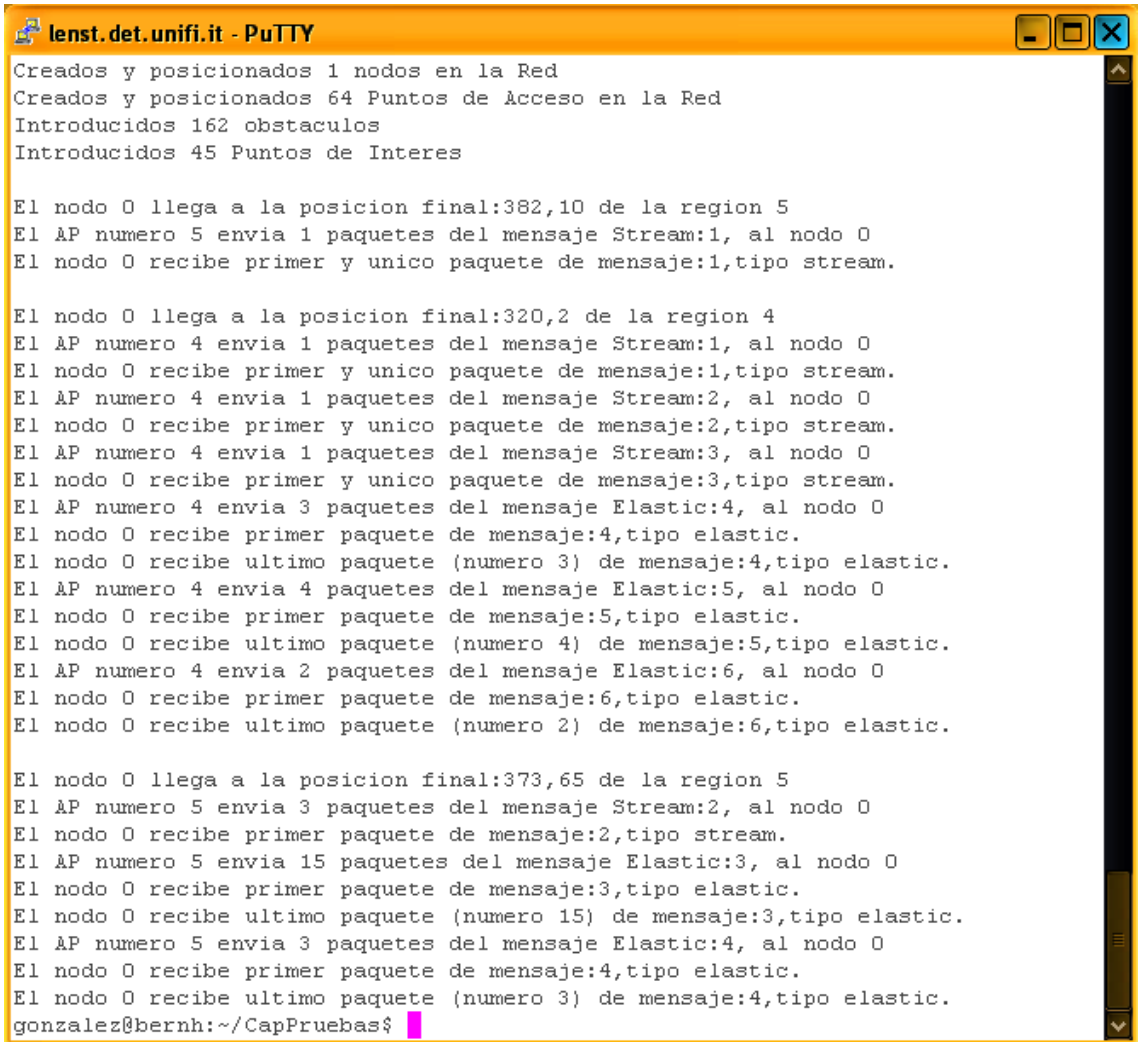



Figura 3.8: Resultados asociados a la prueba con 1 nodo y 64 Puntos de Acceso.

Respecto al caso anterior cabe destacar cómo ahora varía el Punto de Acceso desde el que se recibe la información cada vez que el nodo alcanza un destino, y cómo se reciben mensajes con el mismo identificador. Esto último es debido a que cada Punto de Acceso enumera los mensajes de forma independiente. Además, se comprueba como el nodo llega a los mismos destinos y se le envían el mismo número de paquetes y mensajes que en el caso anterior. Esto es debido a que la semilla que provoca la aleatoriedad no ha variado, y por tanto, para simulaciones simples, se producen los mismos resultados.

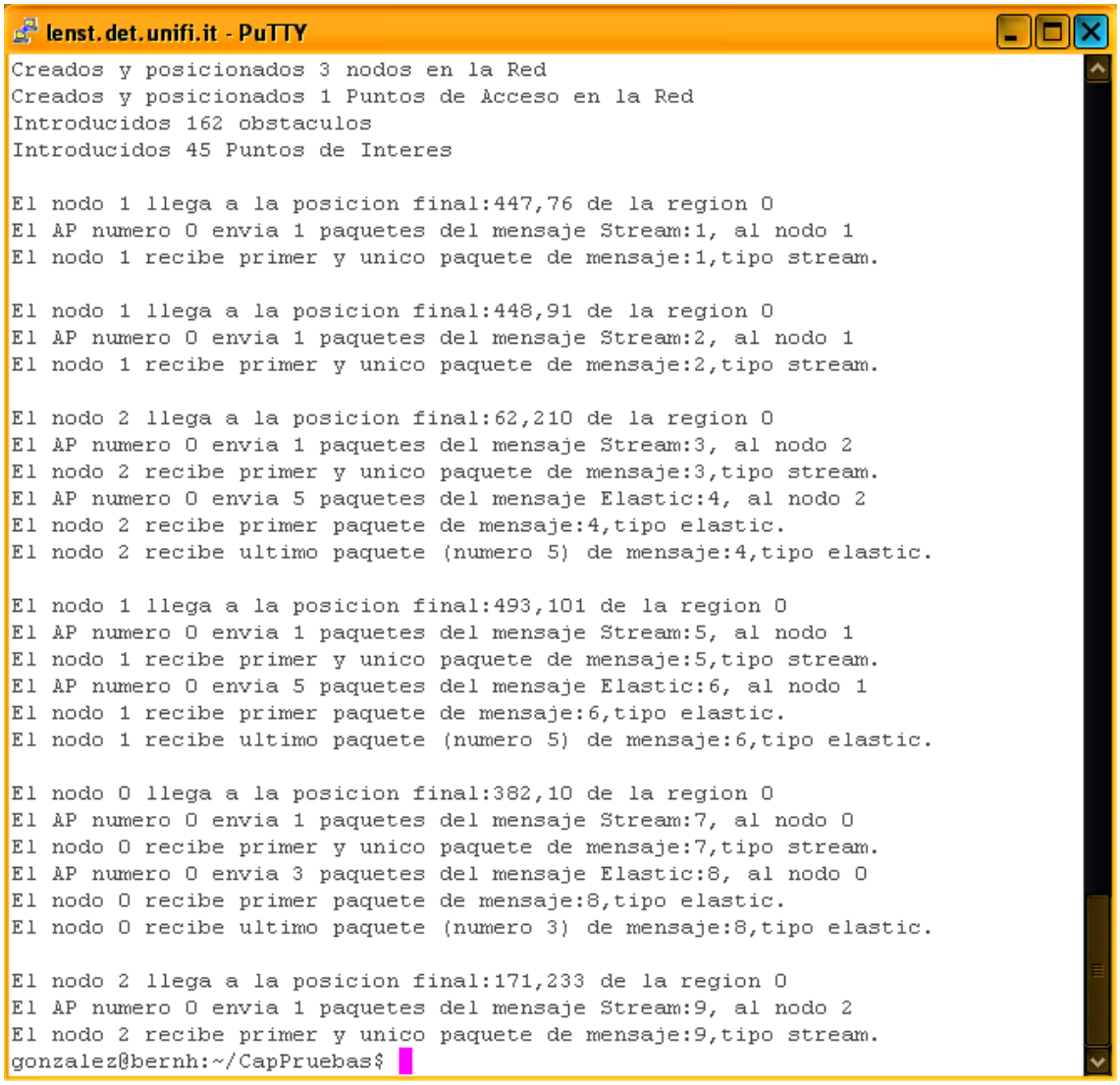



Figura 3.9: Resultados asociados a la prueba con 3 nodos y 1 Punto de Acceso.

En este caso se desea mostrar como el Punto de Acceso sigue enviando los paquetes y mensajes de forma ordenada a pesar de tener que atender a diferentes nodos. Se desea comentar a su vez, cómo durante el mismo tiempo de simulación cada nodo alcanza un número diferente de Puntos de Interés. La razón de esta diferencia hay que buscarla en las distancias a recorrer, así como en el tiempo de parada en cada Punto de Interés (que puede ir de 5 a 30 minutos).

Hay que indicar que tanto para esta prueba como para la siguiente el número máximo de mensajes Stream y Elastic que se pueden enviar se ha reducido a 1.

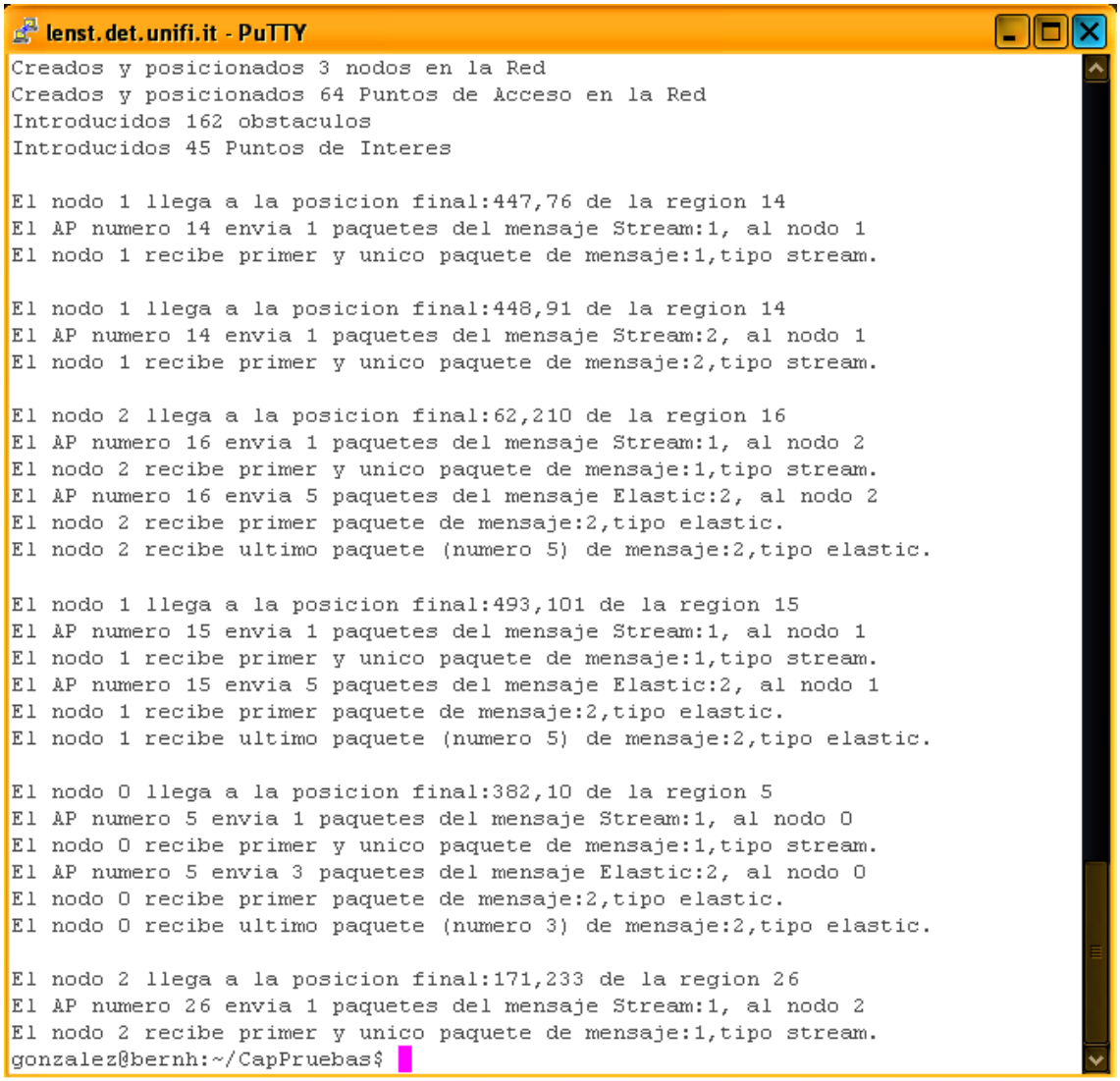



Figura 3.10: Resultados asociados a la prueba con 3 nodos y 64 Puntos de Acceso.

Esta última prueba solo pretende certificar los resultados mostrados en las pruebas anteriores. No se pierde la ordenación, cada nodo recibe información de diferentes Puntos de Acceso según el lugar en que se encuentre, los identificadores asociados a diferentes Puntos de Acceso pueden coincidir, los mensajes Stream tienen prioridad siempre...

# Capítulo 4

# Simulaciones

## 4.1 Área de trabajo

Puesto que este Proyecto Fin de Carrera toma su idea del proyecto SITI que se desarrolla en la ciudad de Florencia, e incluso nace con la idea de poder testar posibles evoluciones de dicho proyecto, el área de trabajo en el que se van a desarrollar las simulaciones no podía ser otro que el centro histórico de la ciudad de la región Toscana.

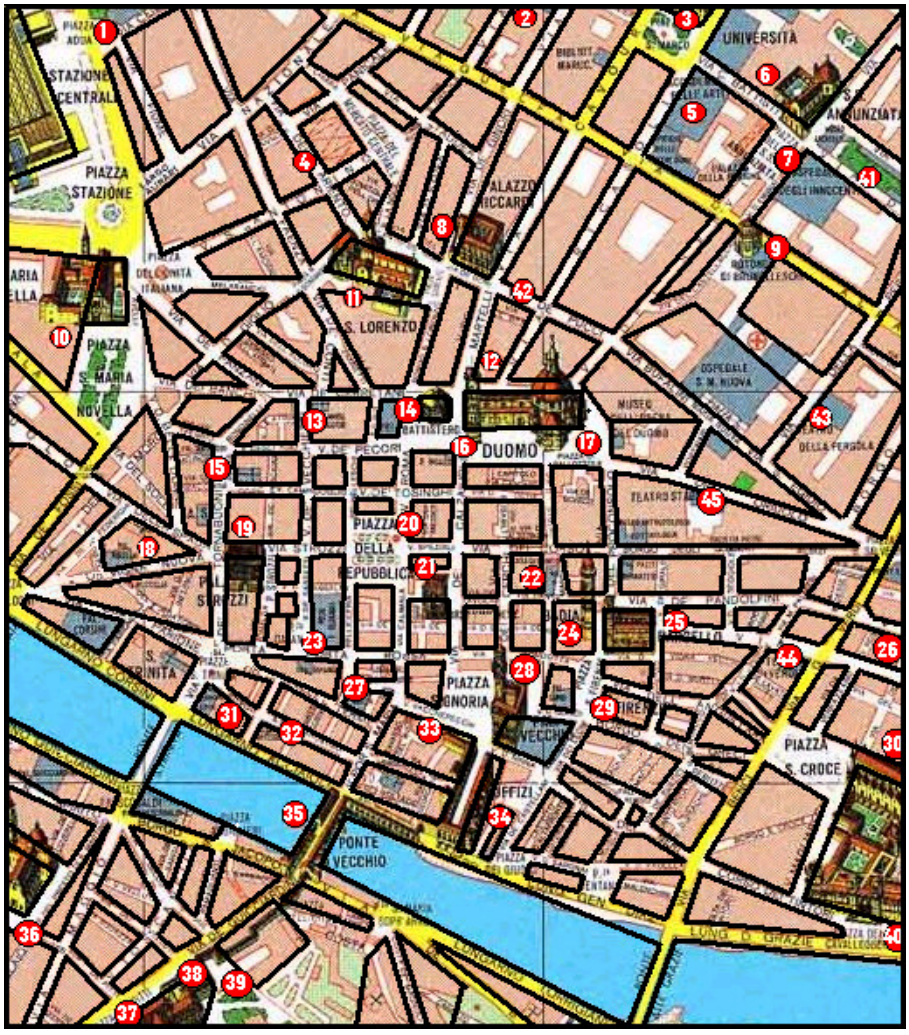



Figura 4.1: Mapa del centro histórico de Florencia.

Con esta idea, vamos a intentar asemejar lo máximo posible nuestras simulaciones a la realidad de dicho proyecto, sin olvidar que nosotros partimos de un modelo con ciertas limitaciones. No obstante, hay que tener en cuenta que el objetivo de las simulaciones es el de validar la robustez del modelo y no tanto buscar soluciones reales, con lo que habrá algunos aspectos o parámetros que varíen.

Uno de los parámetros a definir es el mapa, cuya imagen ha quedado reflejada en la figura 4.1. Las dimensiones reales del área del mapa elegido son 1220 x 1422 $m^2$. Se ha elegido de este tamaño porque así se da cabida a la mayoría de Puntos de Interés que están presentes en la ciudad. De hecho, se incluyen todos los Puntos de Interés del centro histórico, descartando solo aquellos localizados en la periferia. No merece la pena añadirlos al análisis puesto que dejarían muchas zonas "vacías" en el escenario, en las cuales deja de ser considerable el flujo de turistas.

Respecto a las dimensiones del mapa, comentar que se han traducido a píxeles para facilitar la tarea de dar valores a las coordenadas de los vértices de los obstáculos y de los Puntos de Interés. Sabiendo que las dimensiones del mapa son 515 x 600 píxeles, se concluye que 1 píxel = 2.37 metros. Esta igualdad hay que tenerla en cuenta, pues con ella habrá que traducir parámetros como la velocidad del turista, de m/s a píxel/s.

En la figura, los obstáculos son obviamente los cuadriláteros con borde negro. Todo obstáculo se ha creado con forma cuadrangular, como se indica en el apartado 2.2.1. Puede parecer contraproducente que se observen triángulos en la figura, pero es solo para su representación gráfica, pues en realidad se les asignan los cuatro vértices necesarios. Por otra parte, se observan obstáculos unidos para poder crear macroobstáculos con más de cuatro lados y reducir la limitación del número de lados.
De esta forma, el número de obstáculos que componen nuestro escenario es 162.

A su vez, en la figura los puntos rojos que se observan representan los Puntos de Interés que visitarán los terminales. Su número total, como se puede comprobar, es de 45 Puntos.

## 4.2 Presentación de los escenarios y parámetros de las simulaciones

Como es lógico, para las simulaciones habrá parámetros fijos así como parámetros variables, que de hecho serán los que diferencien entre los diferentes escenarios y simulaciones que se ejecutarán. Para reconocer los diferentes escenarios, se presentan a continuación:

- Escenario 1: Velocidad de transmisión 56 Kbps.
- Escenario 2: Velocidad de transmisión 384 Kbps.
- Escenario 3: Velocidad de transmisión 9.6 Kbps.

En los dos primeros escenarios se busca, no sólo comprobar que el modelo es capaz de soportar cargas medias y elevadas (2000 y 5000 nodos moviéndose por el centro histórico), sino que también se intentan optimizar parámetros como el número de Puntos de Acceso y la longitud de paquete, en función de la calidad del servicio y/o los costes. Las optimizaciones se realizan sobre todo para poder razonar el máximo número posible de incoherencias producidas en los resultados, más que con el objetivo propio de buscar la mejor solución. Es por ello por lo que se realiza una sola simulación para cada caso, sin variar en ningún momento parámetros como la semilla.

El objetivo del tercer escenario es, por su parte, validar el correcto funcionamiento del proceso de deambulación, comprobando que no afecta a la calidad del servicio. Por ello se usa una velocidad de transmisión menor, además de un gran número de Puntos de Acceso, buscando favorecer el cambio de región por parte de los nodos. Se hará una comparativa entre dos escenarios, uno con el parámetro '*DistHist*' fijado en unos 100 metros, y el otro con este parámetro fijado en 1 metro. Así, solo habrá deambulación en el segundo caso.

Como se puede comprobar, las velocidades de transmisión representan valores reales de sistemas celulares (GPRS, UMTS y GSM, respectivamente). Se escogen estos valores porque son valores reales y que de hecho pueden representar una situación real, no porque se esté buscando encontrar la solución óptima para cada caso, ya que de hecho, parámetros como la tasa FER, longitud de paquete... no se han escogido pensando en la

norma de estos sistemas. Para ello hay que recordar, que el objetivo de este proyecto no es el de buscar la solución, sino el de acabar de validar el modelo.

Sin más dilación, se presentan los valores dados a los parámetros de entrada más relevantes:

## CLASE System

- *SimulationTime*: Se fija intentando aproximar a tiempos reales de visitas turísticas, pero limitados por la sobrecarga a la que puede someterse la simulación. Así, se establece en 10800 seg. (3 horas) para las simulaciones de media carga. En las simulaciones de carga elevada se reduce a 7200 seg. Por último, en el tercer escenario, queda reducido a 1 hora.

- *WriteStatsInterval*: Parámetro que se fija en 50 segundos, valor de compromiso para tomar un número de muestras adecuado y obtener buenas gráficas.

- *RandomSeed*: La semilla de aleatoriedad no se variará en las simulaciones porque como ya se ha reiterado, en este capítulo no se trata de buscar una solución real, sino de hacer comparaciones, demostraciones ... que validen el simulador creado. Su valor será 1234567.

## CLASE Network region map

- *limitX, limitY*: Ancho y alto del área de trabajo, que como ya vimos en el apartado 4.1, son 515 y 600 píxeles respectivamente.

- *numnodi*: Serán 2000 o 5000 nodos en función de que queramos analizar la carga media o elevada a que se somete el modelo. Son valores reales, teniendo en cuenta las ocupaciones de los hoteles y añadiendo un amplio margen para los turistas que usan el servicio de forma independiente.

- *rangevertx*: Su valor se ha fijado en 4 píxeles para buscar el punto cercano al vértice a bordear en un amplio margen y no provocar congestiones innecesarias.

- *range_tra_poi*: Fijado en 300 píxeles (700 metros). Esta es una distancia apropiada puesto que se ha comprobado que en el área de trabajo todas las posiciones tienen situados en este radio como mínimo a dos Puntos de Interés a los que poder dirigir al nodo. Se ha buscado evitar con esto que el nodo se quedase sin Puntos de Interés cercanos a los que dirigirse.

- *BandaTx*: Su valor puede ser 9600, 56000 o 384000 bps, como ya se sabe.

- *DistanzaHisteresi*: Su valor se ha fijado en 4.2 píxeles (10 metros). Toma los valores 47 y 0.525 píxeles en el tercer escenario.

- *numostacoli, numpoi*: Parámetros fijos con los valores ya conocidos de 162 obstáculos y 45 Puntos de Interés respectivamente.

- *archiviOstacoli*: Parte del contenido de este archivo se presenta en la siguiente figura para mostrar su aspecto.

```
# Fichero para los vértices de los obstáculos
# v1x   v1y   v2x   v2y   v3x   v3y   v4x   v4y
0       0     7     0     41    87    0     104
57      30    63    27    90    90    80    97
68      19    101   0     128   61    101   85
112     0     177   0     182   8     136   56
194     10    216   22    190   43    174   32
199     0     235   0     228   15    200   1
```

Figura 4.2: Contenido parcial del archivo ficherOstac.txt.

- *archiviPoi*: El nombre elegido para el archivo que recoge las coordenadas es 'ficherPoi.txt', y parte de su contenido también se muestra a continuación:

```
#Coor.Poi
#x      y
58      16
320     2
382     10
174     90
373     65
447     76
...     ...
```

Figura 4.3: Contenido parcial del archivo ficherPoi.txt.

- *numzone_col*: El número de regiones, que se identifica con el número de Puntos de Acceso, es uno de los parámetros a optimizar en el sistema, con lo que será variable. Así, tomará valores que van desde 3 hasta 12 regiones por columna. Para dar estos valores se ha tenido en cuenta que un Punto de Acceso puede dar una cobertura en ciudad de radio entre 30 y 300 metros.

- *numzone_row*: Puesto que a efectos prácticos se va a considerar cuadrangular nuestra área de trabajo, poseerá siempre el mismo valor que '*numzone_col*'.

- *numAP*: Producto de los parámetros anteriores, por tanto oscilará entre 9 y 144.

## CLASE Node

- *velocitymax, velocitymin*: Por ser turistas los que usarán el servicio, se ha considerado que su velocidad máxima no superará los 0.47 píxeles/s (4 kms/h) y su mínima bajará hasta los 0.235 píxeles/s (2 kms/h).

- *HopTime*: Por estar el área de trabajo llena de obstáculos que impiden el desplazamiento continuado en línea recta, se ha reducido su valor a 1segundo.

- *MaxStopTime*: Se ha establecido en 30 minutos (1800 segundos) como valor medio de compromiso, por considerarse excesivo para la visita de algunos Puntos de Interés pero también escaso para otros. Se reduce a 0 segundos en el tercer escenario, como una de las condiciones del mismo.

- *MinStopTime*: Se ha estimado que en la realidad, menos de 5 minutos (300 s) no se dedican a ningún Punto de Interés. Se reduce a 0 en el tercer escenario.

## CLASE APoint

- *FER*: Parámetro variable que aumenta su valor si lo hace el número de Puntos de Acceso, por aumentar el radio de cobertura y el número de nodos a atender. Con esta restricción, tomará siempre uno de estos valores : 0.06, 0.08, 0.12, 0.15, 0.2.

- *maxttlxpck, maxttlxpckStream*: Valores que dependerán en realidad de muchos parámetros, pero sobretodo de la velocidad de transmisión, de la prioridad y de la calidad del servicio que queramos ofrecer. Es lógico por tanto, que varíen en cada escenario, y que los paquetes Stream tengan un valor menor:

    - Primer escenario: 20 y 3.5 segundos respectivamente.
    - Segundo escenario: 3 y 0.6 segundos respectivamente.
    - Tercer escenario: 100 y 15 segundos respectivamente.

- *maxLunPck*: Parámetro a optimizar. Puesto que la longitud media de los mensajes es 14136 B y 316 B según sean Elastic o Stream (se comenta a continuación), y realmente interesa que no se envíen paquetes con mucho relleno, se ha fijado el valor entorno a la media más pequeña (316 B). No obstante, hay que tener en cuenta que se puede afectar a los mensajes grandes ya que necesitarán muchos paquetes con sus correspondientes cabeceras (30 B por paquete). Con todo esto, se han escogido los valores de 500 B, 1000 B y 1500 B para intentar optimizar con ellos este parámetro.

- *Mean_Message_Length, Mean_Message_LengthStream*: A la hora de elaborar el presente Proyecto Fin de Carrera tan sólo se habían completado los elencos de información de texto e imágenes en la base de datos del proyecto SITI, con lo que se decidió que los mensajes con texto hiciesen la función de mensajes Stream, y los mensajes con imágenes la función de mensajes Elastic. Es cierto que la correspondencia no es válida, pero al menos se sigue respetando la prioridad. Tras analizar los tamaños de los mensajes con imágenes y con textos, se obtuvieron sus longitudes medias: 14163 B y 316 B respectivamente..

- *Minim_Message_Length, Minim_Message_LengthStream*: La imagen y el texto más pequeños de la base de datos estudiada tienen una dimensión de 2625 B y 61 B respectivamente.

- *maxMsg4Req, maxMsg4ReqStream*: Valores que también se obtienen de la base de datos. En ella se observa que el máximo número de mensajes con imagen que se mandan es 3, al igual que el número máximo de mensajes con texto.

## 4.3 Resultados asociados a las simulaciones del Escenario 1

### 4.3.1 Optimización de la longitud de paquete

Para optimizar la longitud de paquete se han hecho simulaciones con 2000 nodos y 25 Puntos de Acceso durante 3 horas, variando como único parámetro, lógicamente, la propia longitud de paquete. A ésta se le han dado los valores: 500, 1000 y 1500 B.

Notar que el número de Puntos de Acceso y nodos no influirá a la hora de decidir la longitud de paquete óptima, puesto que este parámetro es independiente. Por esta razón, se han elegido valores intermedios en ambos casos.

Los resultados a comparar serán los asociados al retraso de los mensajes, tanto Elastic como Stream, puesto que serán los mejores indicadores de la optimización de la longitud de paquete. De esta forma, el retraso medio acumulado (durante los 10800 segundos), y asociado a los diferentes tipos de mensajes es el siguiente:

| Longitud paquete | Retraso mensajes Stream | Retraso mensajes Elastic |
|---|---|---|
| **500 B** | 0.307104 | 8.06601 |
| **1000 B** | 0.566654 | 8.52449 |
| **1500 B** | 0.854672 | 9.25036 |

Tabla 4.1: Comparativa de los resultados asociados a las diferentes longitudes de paquete.

Se muestran a continuación en gráficas a lo largo del tiempo que dura la simulación:

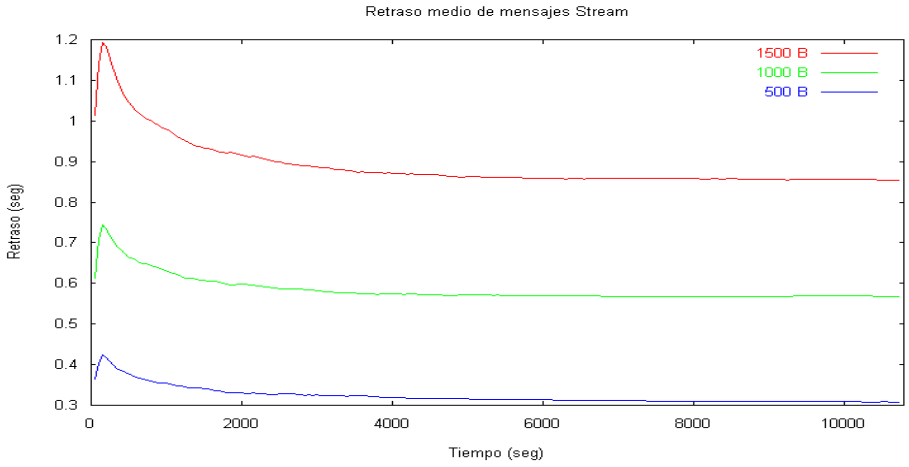



Figura 4.4: Gráfica con los retrasos medios asociados a los mensajes Stream.

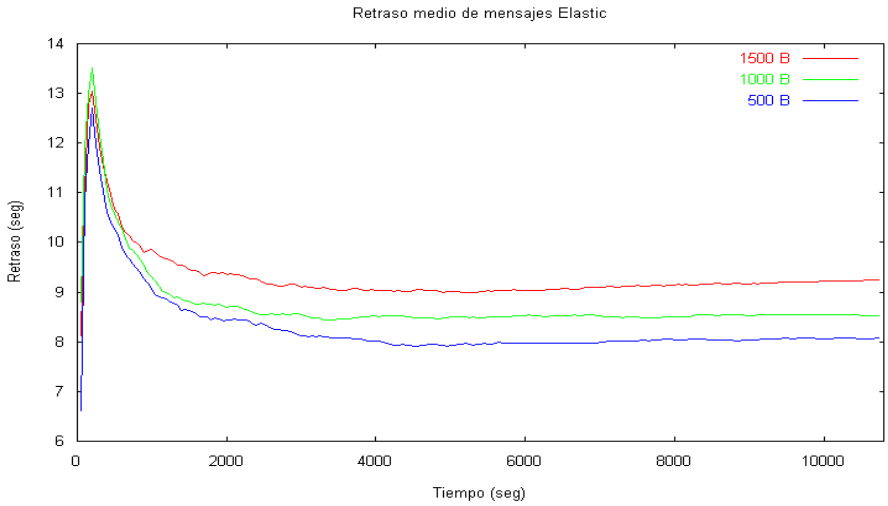



Figura 4.5: Gráfica con los retrasos medios asociados a los mensajes Elastic.

Se observa de las dos gráficas anteriores que al principio los mensajes sufren un mayor retraso. Esto es debido a que al inicio, los nodos se dirigen con gran probabilidad hacia un Punto de Interés cercano, y se acumulan múltiples peticiones de información en el servidor, las cuales se traducen en mayor tiempo de espera en las colas, que es lo que provoca este aumento del retraso.

La apreciación anterior habrá que tenerla en cuenta durante todo el capítulo que nos ocupa, puesto que es fundamental para explicar muchas de las gráficas que se presentarán.

A su vez, se observa claramente de las gráficas anteriores, como los retrasos asociados a los mensajes que están compuestos por paquetes de 500 B son significativamente menores a lo largo de las tres horas, tanto en el caso Stream como en el caso Elastic.

Los histogramas de los retrasos se muestran a continuación:

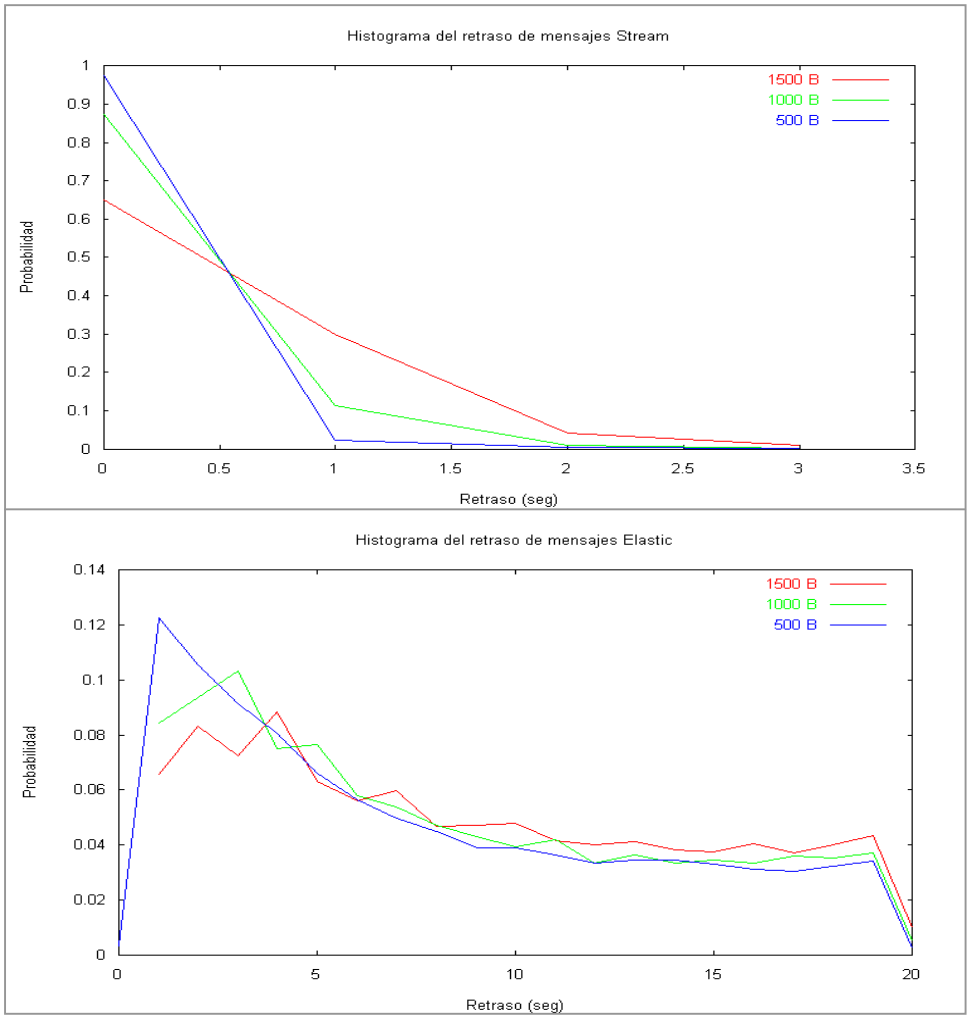



Figura 4.6: Histogramas con los retrasos de los mensajes Stream y Elastic.

Aquí se vuelve a observar, como efectivamente, los mensajes asociados a 500 B llegan en mayor medida con retrasos menores, y siempre con menor probabilidad cuando los retrasos son mayores.

Por tanto, la elección es clara. Para nuestro sistema, la longitud idónea de paquete es de 500 bytes.

### 4.3.2 Valoración del número de Puntos de Acceso óptimo para cargas medias

En este apartado se busca valorar el número de Puntos de Acceso óptimo para dar cobertura a los 2000 nodos que van a estar presentes en el área de trabajo. Para ello se dispondrán de simulaciones con un número variable de Puntos de Acceso, que irán desde 16 (4x4) hasta 49 (7x7). A su vez, la tasa FER decrecerá puesto que el radio del que se hará cargo el Punto de Acceso se hará menor. En estas simulaciones se ha hecho uso de una longitud de paquete de 1500 B, que aunque no se consideró la óptima, se usa por independizar las optimizaciones. De dichas simulaciones se confrontarán los resultados que se muestran a continuación, y que se han considerado los más apropiados para resolver la situación:

| **Nº Puntos Acceso** | **16** | **25** | **36** | **49** |
|---|---|---|---|---|
| **Nº Efectivo Puntos Acceso** | 15 | 19 | 26 | 29 |
| **Factor de carga** | 0.388249 | 0.317787 | 0.287236 | 0.211867 |
| **Nº medio paquetes buffer Stream** | 0.238453 | 0.169179 | 0.122942 | 0.664348 |
| **Nº medio paquetes buffer Elastic** | 22.818 | 16.7353 | 15.3091 | 10.806 |
| **Retraso medio mensajes Stream** | 0.988891 | 0.854672 | 0.844047 | 0.800829 |
| **Retraso medio mensajes Elastic** | 10.7554 | 9.25036 | 8.94632 | 8.25128 |
| **Nº medio paquetes perdidos** | 62.5127 | 52.8195 | 62.9094 | 53.7659 |
| **Eficiencia** | 0.60056 | 0.639981 | 0.644293 | 0.699052 |

Tabla 4.2: Comparativa de los resultados para cargas medias.

El número de Puntos de Acceso efectivos se obtiene de contabilizar los Puntos de Acceso inutilizados en la simulación y restarlos al total de los mismos. Estos se obtienen como resultado por pantalla. A continuación se muestra el resultado obtenido para estas simulaciones:


Figura 4.7: Resultado de la simulación con 16 Puntos de Acceso.

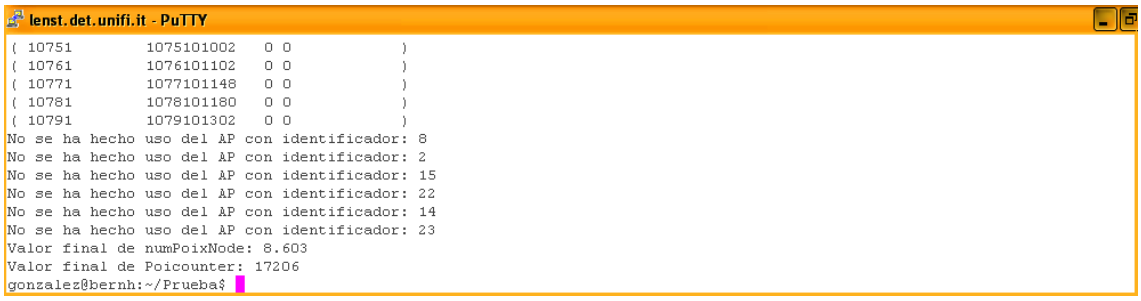



Figura 4.8: Resultado de la simulación con 25 Puntos de Acceso.

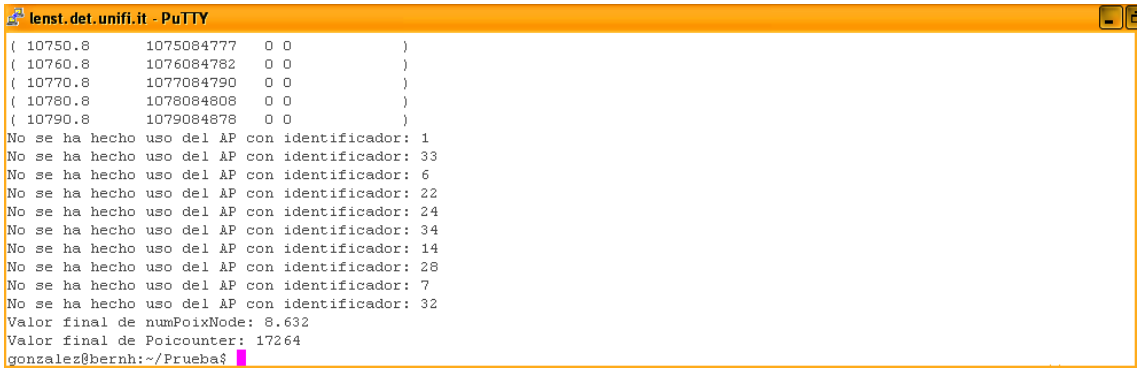



Figura 4.9: Resultado de la simulación con 36 Puntos de Acceso.

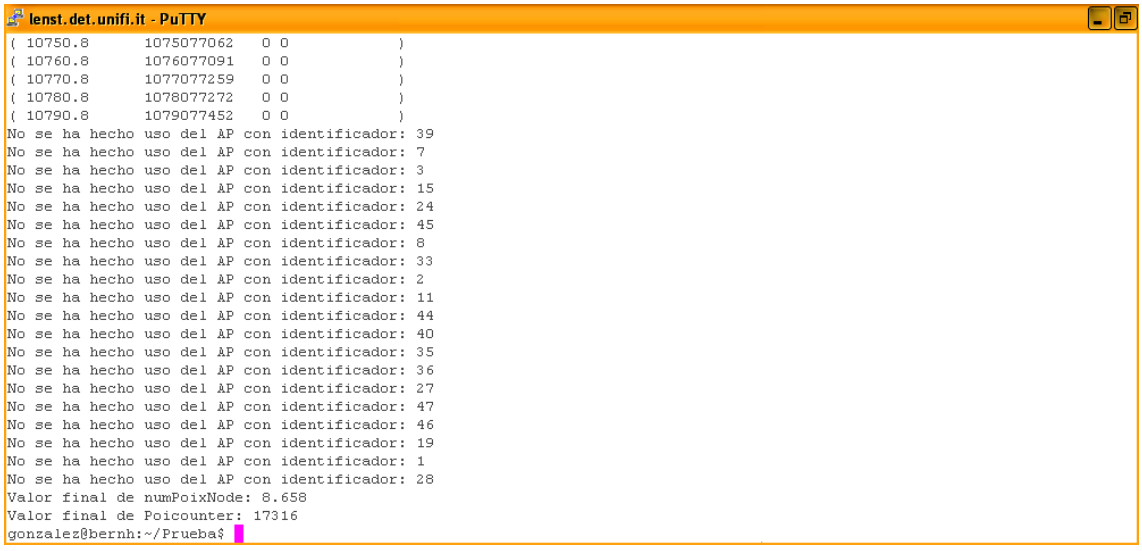



Figura 4.10: Resultado de la simulación con 49 Puntos de Acceso.

Efectivamente, se contabiliza un Punto de Acceso inutilizado para el primer caso (16-15), seis para el segundo caso, diez (36-26) para el tercer caso y veinte para el caso con 49 regiones. Con esto queda corroborada una situación que será clave en el resto de las simulaciones, ya que únicamente se tendrán en cuenta para las estadísticas los Puntos de Acceso que en algún momento estén activos.

La siguiente figura muestra una comprobación visual de los Puntos de Interés y regiones que quedan sin utilizar en el caso con 49 Puntos de Acceso.

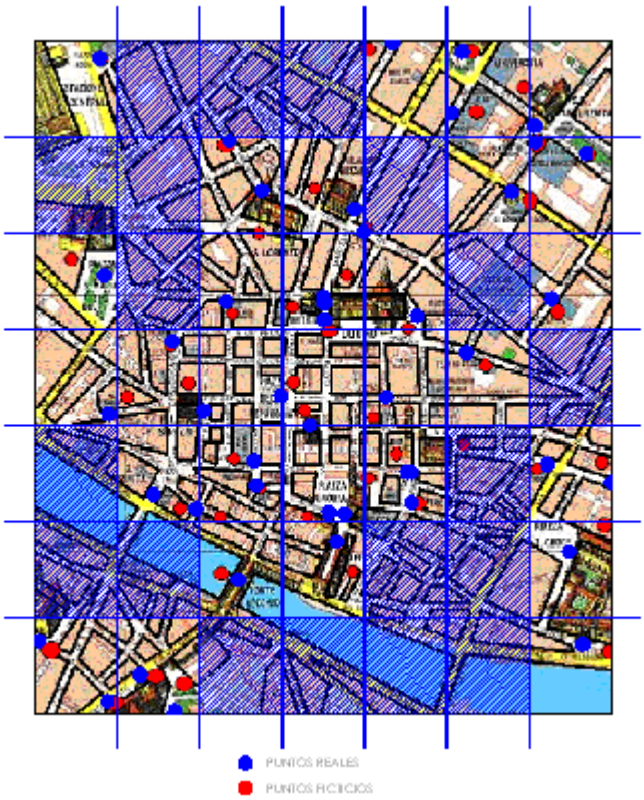



Figura 4.11: Regiones inutilizadas en la simulación con 49 Puntos de Acceso.

En la figura se ha dividido el mapa contenido en las 49 regiones que componen la situación. A su vez se han sombreado las regiones cuyos Puntos de Acceso han quedado inutilizados por la ausencia de Puntos de Interés en su zona de cobertura. A la vista del mismo, se puede comprobar como coinciden los Puntos de Acceso inutilizados con los identificadores mostrados en la figura 4.10. Para ello, tener en cuenta que las regiones se enumeran en el mapa empezando por el 0, de izquierda-derecha y de arriba-abajo.

Otro resultado que cabe comentar de las figuras 4.6 a 4.10, es el valor final del 'PoiCounter' que se observa. Representa el número total de Puntos de Interés vistos por todos los terminales. Debe ser similar para las cuatro simulaciones, ya que coinciden el número de nodos usados, la duración de las simulaciones y el valor de la semilla usado en las mismas. Además valida en mayor medida las comparaciones efectuadas. Efectivamente, mirando los valores (17326, 17206,17264, 17316 Puntos de Interés visitados respectivamente), se corrobora la suposición.

Respecto al siguiente resultado que se muestra en la tabla 4.2, el factor de carga, comentar que es un valor que no debe estar próximo a la unidad, pues eso significaría que los paquetes entran en el buffer del Punto de Acceso con la misma velocidad con la que salen. Teniendo en cuenta que en este caso los paquetes o mensajes entran en ráfagas por la estructura del servicio, es un factor importante a considerar puesto que puede aumentar en exceso el retraso de los propios paquetes. Por ello, para garantizar una buena calidad del servicio se procurará que el valor de este parámetro no supere e incluso no se aproxime a la unidad durante el tiempo que dura la simulación. La siguiente figura muestra la gráfica en la que se comparan, para cada instante de tiempo, los valores obtenidos para el factor de carga en las cuatro simulaciones de que se ha hecho uso en este apartado.

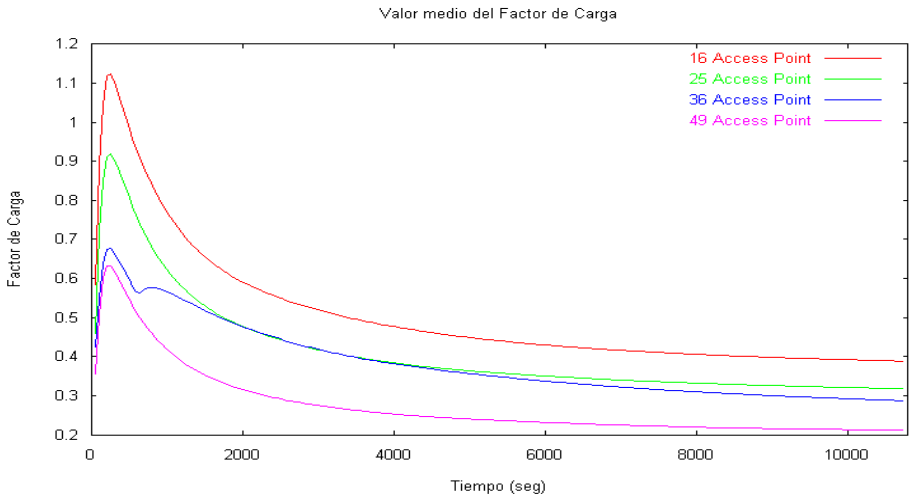



Figura 4.12: Gráfica de los valores del Factor de Carga para cada simulación.

A la vista de la gráfica, se observa que solo serían servibles los casos en que se dispone de 36 y 49 Puntos de Acceso, puesto que son los únicos que no se aproximan a la unidad ni a valores próximos en ningún momento de la simulación.

El pico del factor de carga se encuentra al inicio de las simulaciones porque es en esta fase de tiempo cuando la mayoría de los terminales alcanzan su primer destino, como ya se comentó en el apartado 4.3.1.

El siguiente parámetro a estudiar va a ser la ocupación media de los buffer Stream:

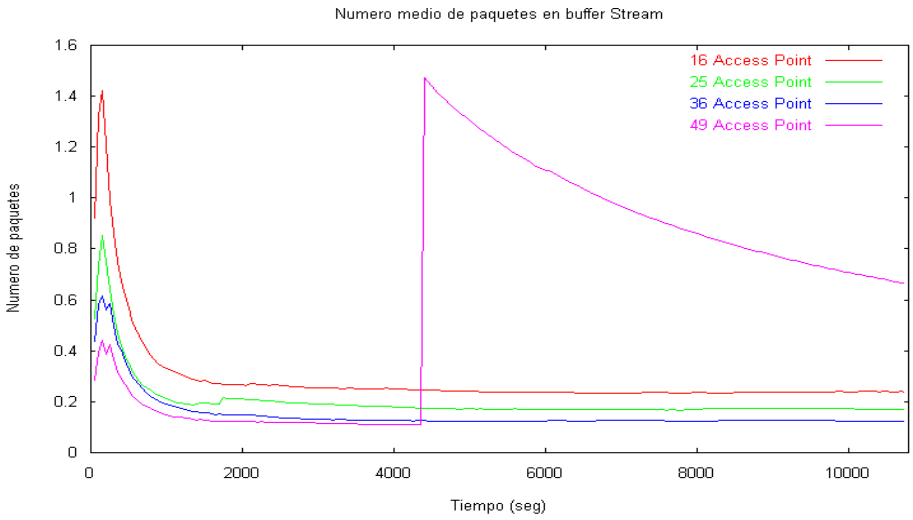



Figura 4.13: Gráfica del tamaño del buffer Stream para cada simulación.

En esta figura se observa una discontinuidad para el caso de 49 Puntos de Acceso. Ésta afecta a la correcta comparación de las gráficas, pero es debida a la modelación de las fuentes como una distribución de Pareto, que puede dar muestras infinitamente mayor a la media solicitada, como se observa a continuación.

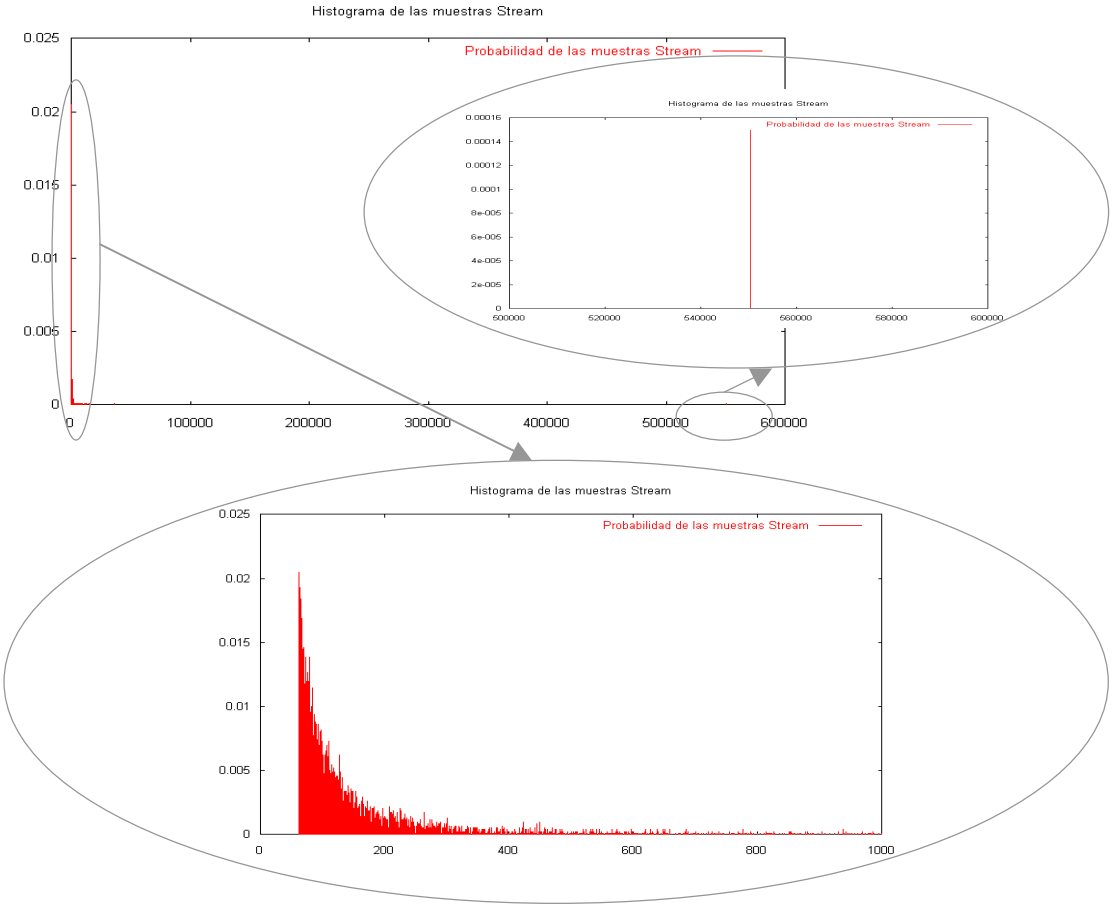



Figura 4.14: Histograma y detalles del tamaño de las muestras Stream en Bytes.

Se observa del detalle superior como se crean muestras incluso con un valor entorno a los 550000 B, habiendo creado la distribución de Pareto con una media de 316 B. Con el detalle inferior se pretende observar claramente la distribución, que como bien se indicó, tiene muestras a partir de los 61 B.

La justificación de la discontinuidad queda completamente clarificada con siguiente gráfica, en la que se representa el número de paquetes Stream creados para el caso con 49 Puntos de Acceso. Como se observa, existe también una discontinuidad y localizada en el mismo instante de tiempo, lo que certifica el razonamiento.

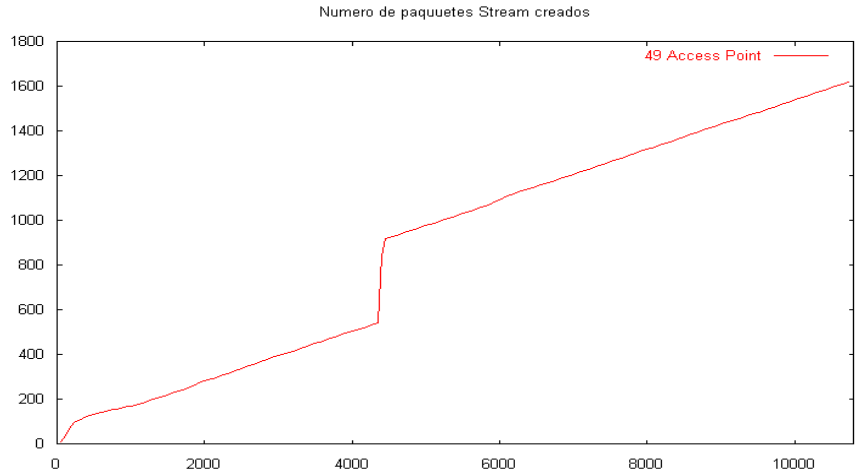



Figura 4.15: Gráfica de paquetes Stream creados en la simulación con 49 Puntos Acceso.

A la vista de la figura 4.13 se observa como la ocupación media es incluso menor de un paquete (1500 B) en los cuatro casos. Este valor, nos deja claro por tanto que la ocupación del buffer Stream no será un condicionante, puesto que es mínimo. Es muy lógico que los buffer Stream estén siempre casi vacíos debido a dos razones: los mensajes Stream son pequeños (media de 316 B que corresponde a un paquete), y además, los paquetes Stream tienen preferencia frente a los Elastic, con lo que están menos tiempo en el buffer.

En la gráfica inferior (figura 4.16) se comparan los valores correspondientes para las gráficas que analizan la ocupación media de los buffer Elastic. Como es lógico, la ocupación media aumenta, situándose entre los 30 y 40 paquetes durante la mayoría de la simulación, ascendiendo y ascendiendo incluso a los 120 paquetes en algún caso. No obstante, es un valor válido para un buffer real de memoria (175 Kbytes).

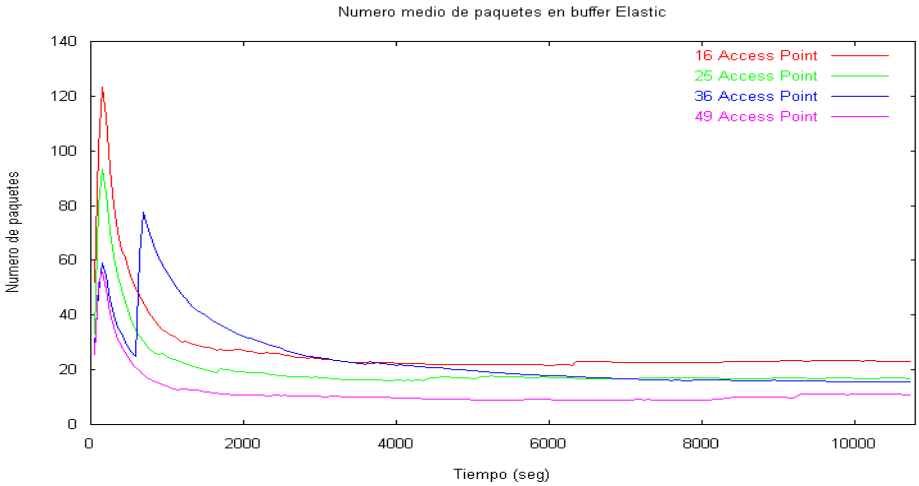



Figura 4.16: Gráfica del tamaño del buffer Elastic para cada simulación.

Los próximos resultados a analizar serán los relacionados con los retrasos, que se presuponen bastante útiles pues si se producen amplias diferencias pueden incluso resultar decisivos a la hora de nuestra elección final. Veamos así pues las gráficas:

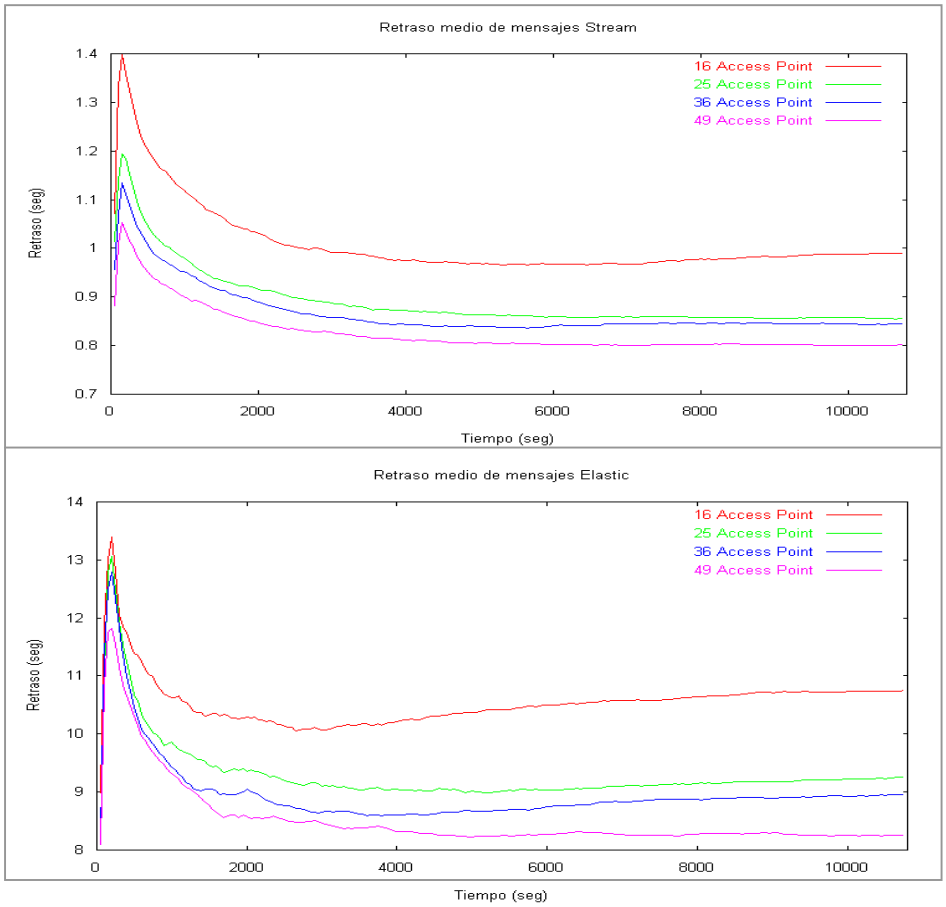



Figura 4.17: Gráficas de los retrasos de los mensajes Stream y Elastic.

Se observa que para ambos retrasos, el mejor caso es el de 49 Puntos de Acceso, que llega a mejorar hasta en casi 0.4 segundos el retraso Stream en instantes de máxima carga, mientras que para el caso Elastic llega a mejorar hasta casi 3 segundos al final de la simulación. Respecto al caso más similar, el de 36 Puntos de Acceso, mejora en 0.05 segundos el retraso Stream, y 0.7 segundos para el caso Elastic. Puesto que no aumenta en gran medida el retraso, el caso de 36 Puntos de Acceso también podría considerarse válido, volviendo a dejar como mejores situaciones las de 36 y 49 Puntos de Acceso.

Con el análisis de los dos últimos resultados se acabará de decidir entre los dos casos que hasta ahora hemos considerado óptimos (36 y 49 Puntos de Acceso). El primero contabiliza el número de paquetes perdidos por cualquier circunstancia. En realidad hace una media de los paquetes perdidos en el intervalo de tiempo que se está contabilizando, al ser un objeto 'ProbeSlice'. Veamos su gráfica:

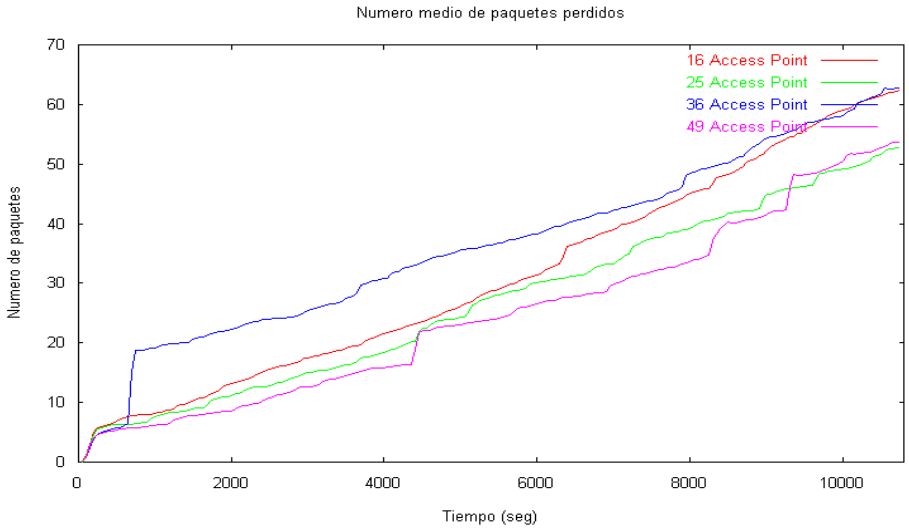



Figura 4.18: Gráfica del número de paquetes perdidos en cada simulación.

En este caso, la situación con 49 Puntos de Acceso deja de ser en algunos instantes la mejor respecto al parámetro, pero mirando la gráfica se concluye que es debido a la misma continuidad que ya se estudió anteriormente. De hecho, respecto a este parámetro es claramente mejor que en el caso con 36 Puntos de Acceso.

Para finalizar se analiza la eficiencia, que estudia el porcentaje de paquetes que ya han llegado a su destino en proporción con los paquetes que ya han sido creados en los buffer de un Punto de Acceso:

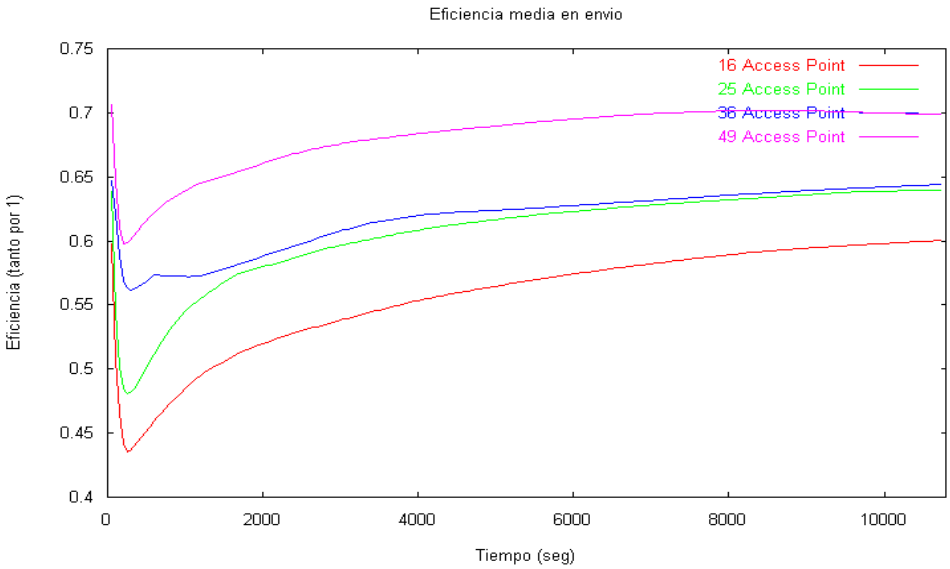



Figura 4.19: Gráfica de la eficiencia en cada simulación.

El caso con 49 Puntos de Acceso mejora la eficiencia en un 5.5 % respecto al caso con 36 Puntos de Acceso, lo que deja claro que resulta mejor la elección del caso con 49 Puntos de Acceso, sobretodo si se tiene en cuenta que en realidad solo produce un aumento efectivo de 3 Puntos de Acceso respecto al caso con 36 (26 frente a 29), y con ellos se consigue una considerable mejora en la mayoría de los parámetros.

Bien es cierto, que si tan solo se colocan los Puntos de Acceso en las regiones con Puntos de Interés, habría zonas en las que el turista se quedaría sin cobertura y no podría solicitar información de ningún tipo. Además, mientras más Puntos de Acceso sin dejan sin colocar, mayor área sin cobertura nos encontraremos. Se produce así un conflicto de intereses que tan solo habría que plantearse en el caso de que quisiéramos hacer el proyecto real. Para el caso que nos ocupa, basta quedarse con la opción que de forma más económica optimiza el sistema.

### 4.3.3 Valoración del número de Puntos de Acceso óptimo para cargas elevadas

En este apartado se busca optimizar el número de Puntos de Acceso para dar atender a 5000 nodos. Así, se dispondrán de simulaciones que irán desde los 81 (9x9) hasta los 144 (12x12) Puntos de Acceso. Se ha usado de la longitud de paquete óptima (500 B). Además, se ha reducido la simulación a 7200 seg., para no sobrecargar la simulación.

| Nº Puntos Acceso | 81 | 100 | 121 | 144 |
|---|---|---|---|---|
| Nº Efectivo Puntos Acceso | 33 | 34 | 37 | 35 |
| Factor de carga | 0.499889 | 0.492347 | 0.397958 | 0.550164 |
| Nº medio paquetes buffer Stream | 0.166801 | 0.178941 | 0.223684 | 2.17453 |
| Nº medio paquetes buffer Elastic | 84.9102 | 74.7331 | 60.0247 | 145.199 |
| Retraso medio mensajes Stream | 0.340661 | 0.340712 | 0.331065 | 0.344204 |
| Retraso medio mensajes Elastic | 9.22726 | 9.6113 | 8.60412 | 9.42283 |
| Nº medio paquetes perdidos | 134.679 | 121.975 | 109.057 | 176.927 |
| Eficiencia | 0.514134 | 0.540878 | 0.571866 | 0.511852 |

Tabla 4.3: Comparativa de los resultados para cargas elevadas en el escenario 1.

De la tabla cabe destacar, en primer lugar, que el número de Puntos de Acceso efectivo no aumenta siempre que lo hace el número de Puntos de Acceso total. La razón es que la diferente colocación de las regiones puede provocar que un número fijo y concreto de Puntos de Interés sean atendidos por un número diverso de Puntos de Acceso.

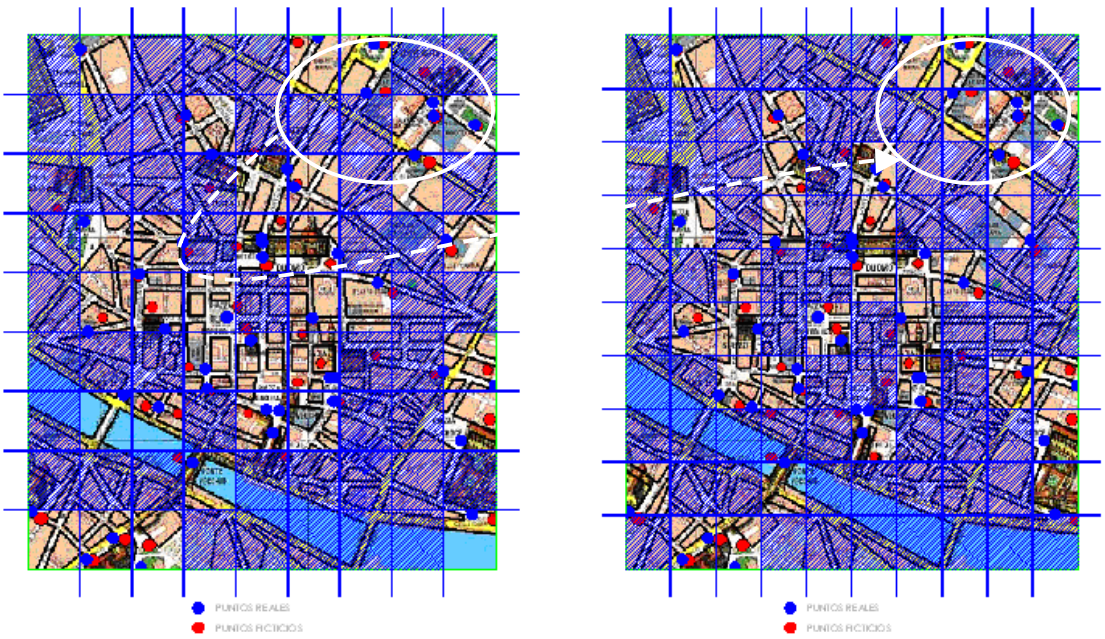



Figura 4.20: Regiones inutilizadas en los casos con 81 y 100 Puntos de Acceso.

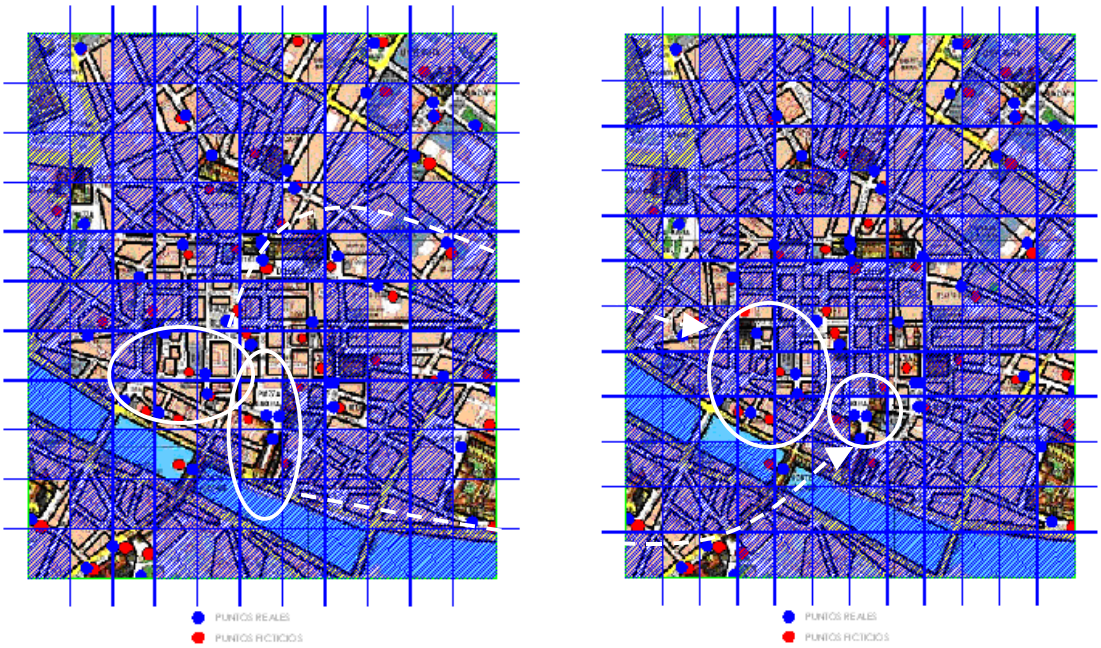



Figura 4.21: Regiones inutilizadas en los casos con 121 y 144 Puntos de Acceso.

Con las figuras 4.20 y 4.21 queda certificado el último comentario del párrafo anterior. Las situaciones redondeadas en cada figura intentan dejar claro cómo la colocación de las regiones provoca que se usen diferente número de Puntos de Acceso para satisfacer el mismo número de Puntos de Interés. En los dos mapas de la figura 4.20 se observa cómo pasan de 5 a 6 las regiones utilizadas para dar cobertura a los Puntos de Interés que se encuentran en esa zona. Para esa situación, aumenta el número de Puntos de Acceso efectivos cuando aumenta el número total de Puntos de Acceso (aunque solo sea en una unidad). Sin embargo, en los mapas de la figura 4.21, pasan de 4 a 3 y de 2 a 1 el número de Puntos de Acceso hábiles para atender a los Puntos de Interés situados en las zonas respectivas. En este caso, por su parte, se disminuye en 2 el número de Puntos de Acceso efectivos cuando se aumenta el número total de Puntos de Acceso.

Se observa a su vez de la tabla 4.3, que en el último caso se obtienen los peores resultados pese a ser la segunda situación que más Puntos de Acceso hace efectivos. Para explicar esta incongruencia se muestra la siguiente tabla:

| **Nº Puntos Acceso** | **81** | **100** | **121** | **144** |
|---|---|---|---|---|
| **Nº Efectivo Puntos Acceso** | 33 | 34 | 37 | 35 |
| **Nº paquetes creados por P.A.** | 39313.4 | 35866.3 | 31324.9 | 42883.1 |
| **Nº total paquetes creados** | **1297342.2** | **1219454.2** | **1159021.3** | **1500908.5** |

Tabla 4.4: Comparativa del número de paquetes creados para cargas elevadas en el escenario 1.

En ella, se multiplica la media del número de paquetes creados por Punto de Acceso por el número efectivo de Puntos de Acceso, para averiguar el número total de paquetes creados. Como se observa, el caso con 144 Puntos de Acceso es en el que más paquetes se crean, hasta un 31 % más respecto a la situación más favorecida.

Teniendo en cuenta esta circunstancia, se ilustran, analizan y comparan otros resultados:

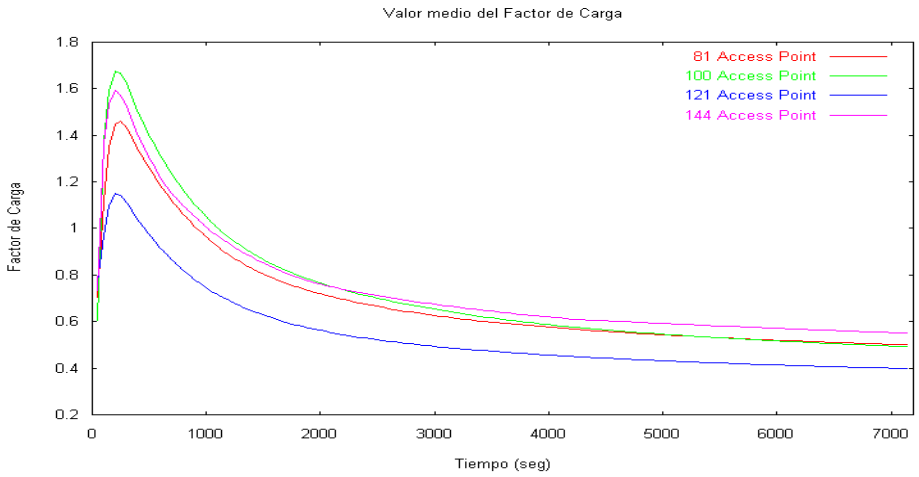



Figura 4.22: Gráfica de los valores del Factor de Carga para cada simulación.

El factor de carga supera la unidad en todas las simulaciones, por lo que podría pensarse en conseguir un aumento de Puntos de Acceso efectivos para suplir este inconveniente. Sin embargo, esto es difícilmente factible ya que algunos Puntos de Interés se encuentran muy próximos entre sí, y además provocaría regiones demasiado pequeñas. Por tanto, nos conformaremos con esta situación, e intentaremos valorarla.

Solo hay una situación que no supere en exceso la unidad, y esa es lógicamente la que más Puntos de Acceso efectivos utiliza y menos paquetes crea: 121 Puntos de Acceso. Además, al final de la simulación, su factor de carga es el único que se aleja lo suficiente de la unidad. Puesto que esta cualidad es muy importante para no provocar colapso, basta para elegir esta situación como la más adecuada. No obstante, se presentan algunos resultados para confirmar la elección y comentar irregularidades.

La figura 4.23 presenta las gráficas para el tamaño medio del buffer Stream. En ella se ha creado un detalle para mostrar correctamente todas las gráficas, lo cual lo impide la gráfica para 144 Puntos de Acceso. Esta posee valores muy superiores por la razón explicada con la tabla 4.4.

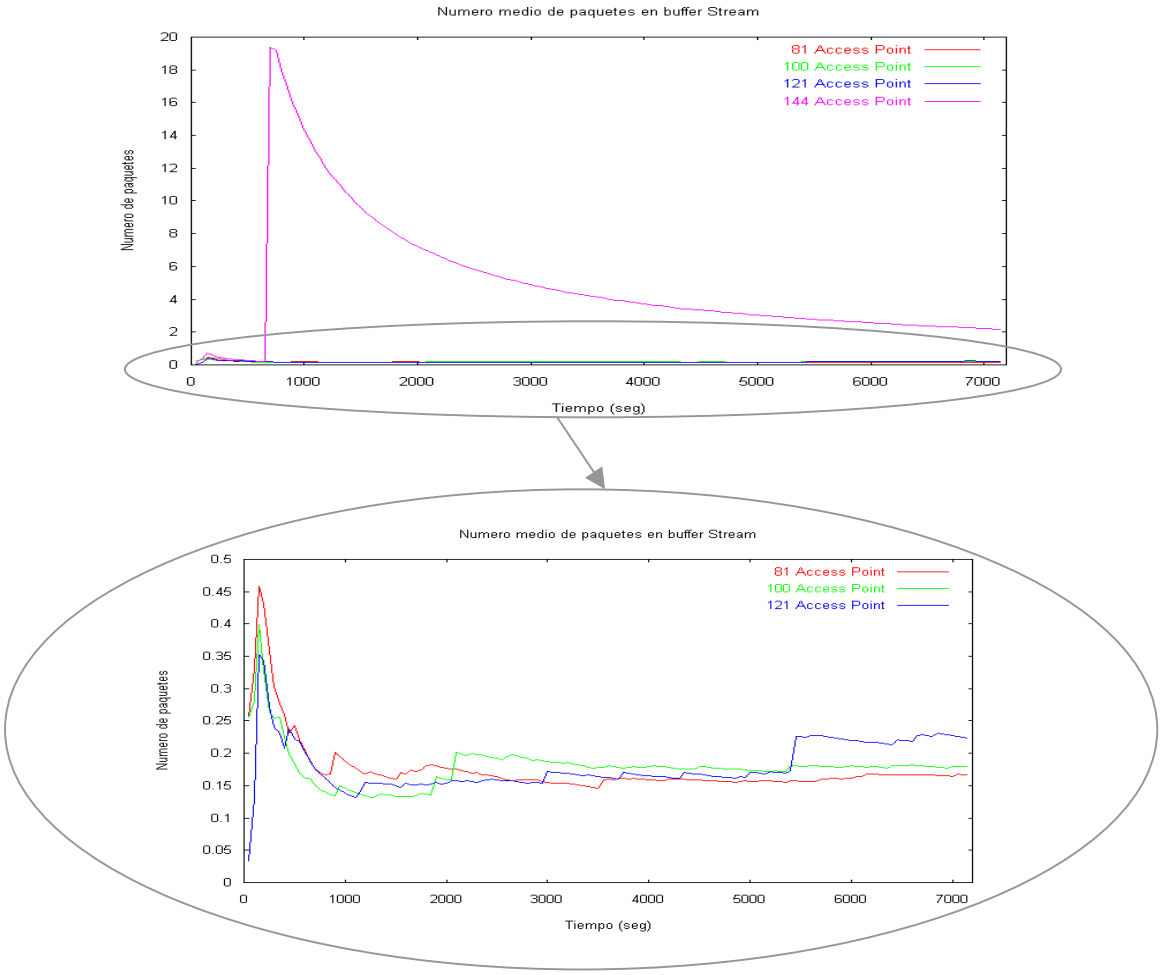



Figura 4.23: Gráfica y detalle del tamaño del buffer Stream para cada simulación.

El retraso medio obtenido para los mensajes Elastic con la situación en la que se ponen en liza 121 Puntos de Acceso, mejora en más de 0.5 segundos respecto al resto de retrasos en gran parte de la simulación. Se observa a continuación:

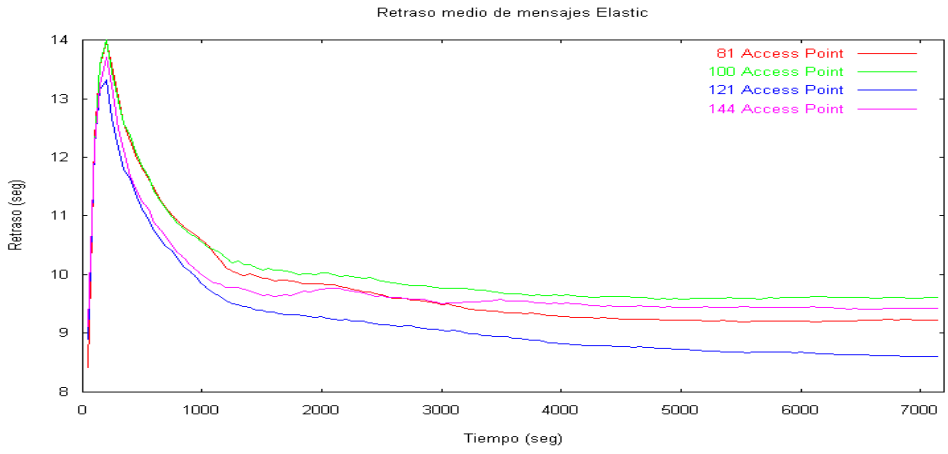



Figura 4.24: Gráficas de los retrasos de asociados a los mensajes Elastic.

Los dos siguientes resultados se muestran para examinar la problemática asociada debido a la elevada cantidad de nodos presentes en las simulaciones:

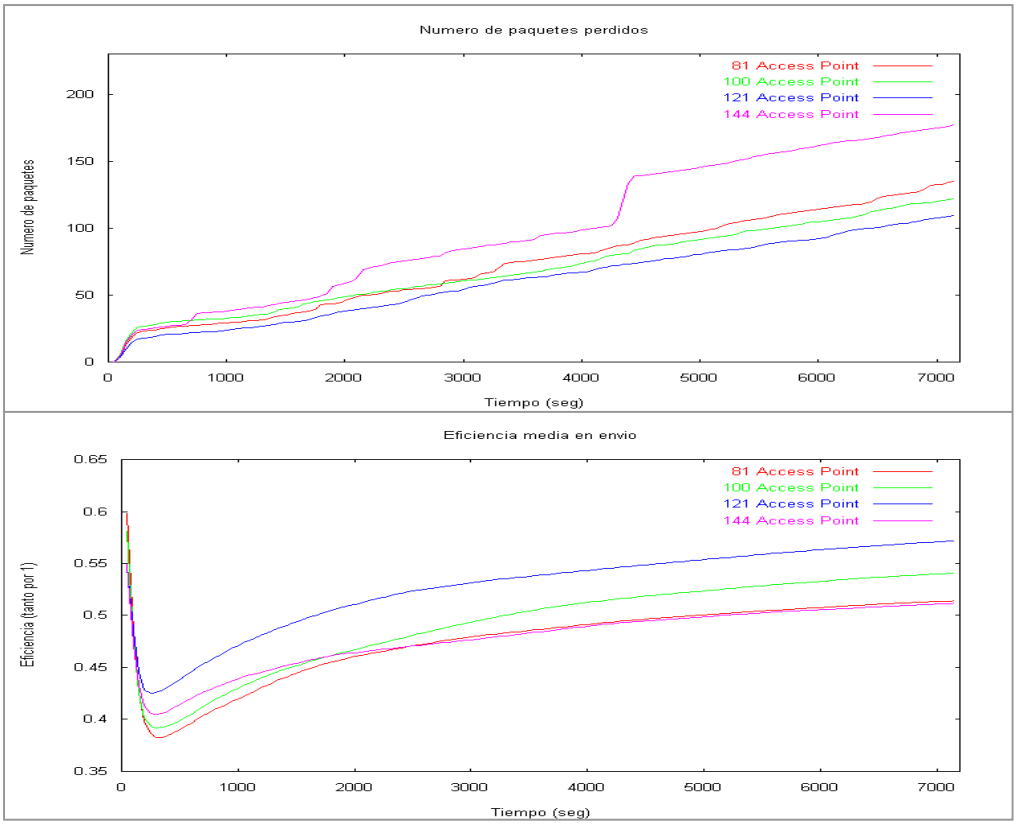



Figura 4.25: Gráficas de los paquetes perdidos y eficiencia en cada simulación.

Así pues, la segunda gráfica demuestra que los Puntos de Acceso se encuentran sometidos a una fuerte carga, puesto que del total de paquetes creados hasta un instante solo se han entregado a sus destinos aproximadamente el 50 %. En el mejor de los casos, llega a alcanzar el 57 %, que aún así se considera bastante escaso.

La primera gráfica de la figura 4.25 da valores alarmantes para los paquetes perdidos, que principalmente se producen por un retraso de los paquetes Elastic mayor a 20 seg., y que se sitúan en torno al 0.5 % del total de paquetes enviados.

## 4.4 Resultados asociados a las simulaciones del Escenario 2

### 4.4.1 Optimización de la longitud de paquete

Esta optimización se hace solo para asegurar que la longitud de paquete óptima es independiente de la velocidad de transmisión y de la situación emulada, por lo que no nos detendremos mucho en su desarrollo. Sólo se comentará que la simulación se ha hecho con 2000 nodos y 9 Puntos de Acceso (3x3). Se muestran así pues los resultados:

| Longitud paquete | Retraso mensajes Stream | Retraso mensajes Elastic |
|---|---|---|
| **500 B** | 0.0410537 | 0.831154 |
| **1000 B** | 0.072964 | 0.880876 |
| **1500 B** | 0.105525 | 0.965848 |

Tabla 4.5: Comparativa de los resultados asociados a las diferentes longitudes de paquete.

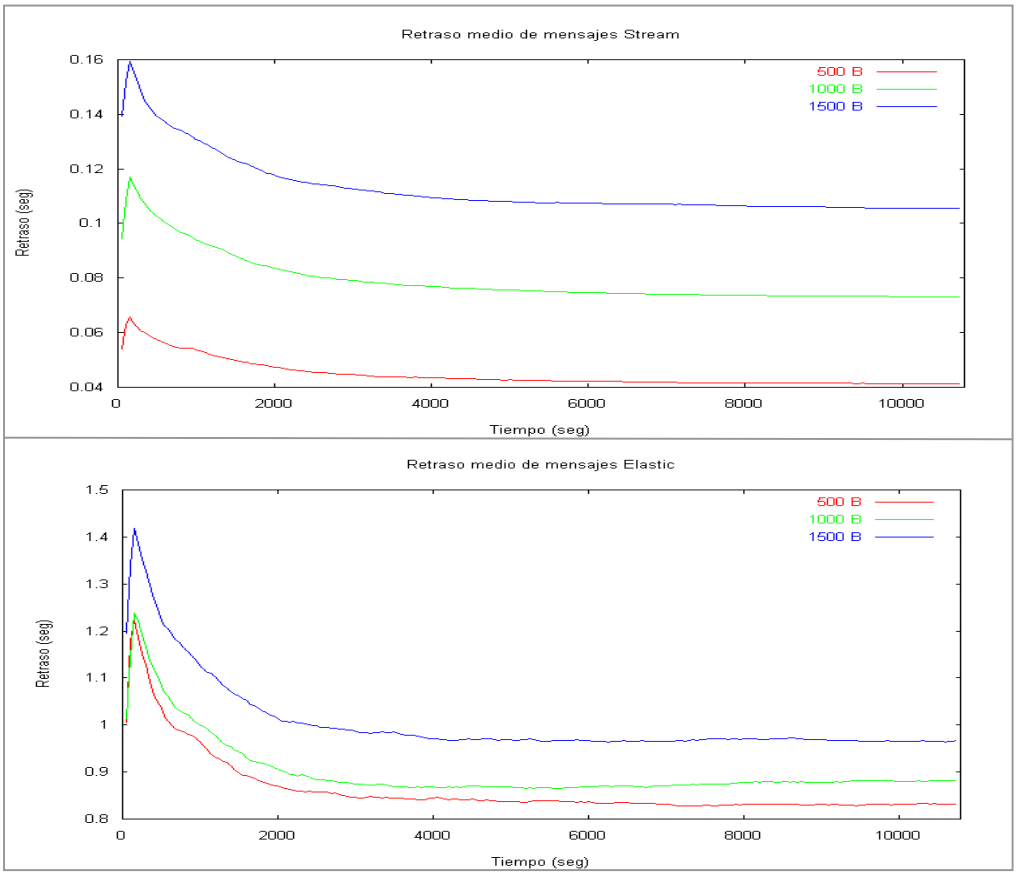



Figura 4.26: Gráficas de los retrasos medios de los mensajes Stream y Elastic.

Vuelve a quedar claro que la longitud óptima de paquete es de 500 B.

### 4.4.2 Valoración del número de Puntos de Acceso óptimo para cargas medias

En este caso se busca valorar el número de Puntos de Acceso óptimo para dar cobertura a 2000 nodos en el área de trabajo, teniendo una velocidad de transmisión de 384 Kbps. Para ello se dispondrán de simulaciones con un número variable de Puntos de Acceso, que irán desde 9 (3x3) hasta 36 (6x6). Se usan menos Puntos de Acceso para esta simulación respecto al apartado 4.3.2 puesto que la velocidad de transmisión es mucho mayor y se supone que bastará con estas cantidades.

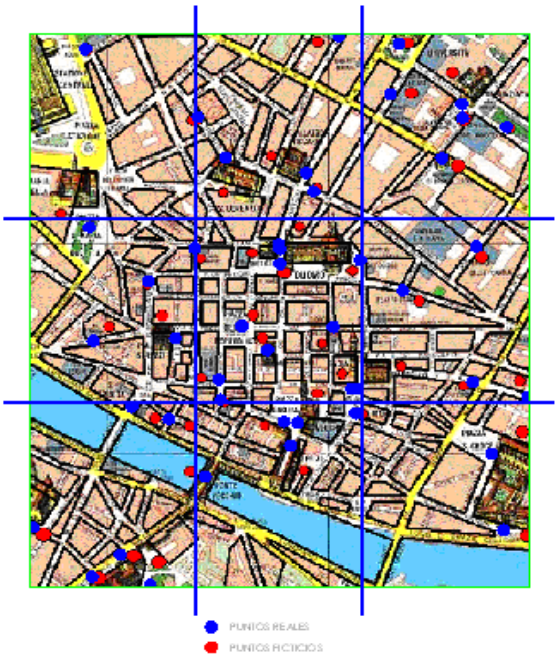
| **Nº Puntos Acceso** | **9** | **16** | **25** | **36** |
|---|---|---|---|---|
| **Nº Efectivo Puntos Acceso** | 9 | 15 | 19 | 26 |
| **Factor de carga** | 0.122323 | 0.0688147 | 0.047113 | 0.0464233 |
| **Nº medio paquetes buffer Stream** | 0.0352229 | 0.0209523 | 0.015562 | 0.0142235 |
| **Nº medio paquetes buffer Elastic** | 4.55741 | 3.66126 | 2.13364 | 1.77013 |
| **Retraso medio mensajes Stream** | 0.105525 | 0.102227 | 0.096288 | 0.096482 |
| **Retraso medio mensajes Elastic** | 0.965848 | 0.890307 | 0.827063 | 0.813792 |
| **Nº medio paquetes perdidos** | 47.2153 | 61.8657 | 41.6683 | 48.9662 |
| **Eficiencia** | 0.702597 | 0.717047 | 0.784741 | 0.727948 |

Tabla 4.6: Comparativa de los resultados para cargas medias en el escenario 2.

A simple vista, se puede observar de la tabla que la mayoría de los parámetros se han reducido en gran proporción frente al apartado 4.3.2 por el aumento ya comentado en la velocidad de transmisión. En ese sentido, los factores de carga se han reducido hasta dejarlos en valores en torno al 10 %, lo que da idea de la mejora en la situación.

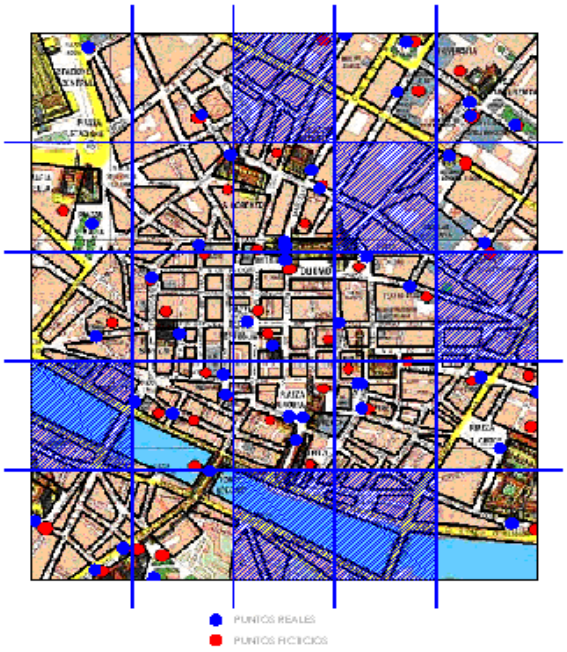
Se podría pensar que quizá habría que haber elegido casos con menos Puntos de Acceso puesto que el sistema no está ni mucho menos al máximo de sus exigencias. Sin embargo, la distancia máxima a la que puede emitir un Punto de Acceso en el centro de una ciudad descartó esta posibilidad.

Con la idea de seguir certificando el número de Puntos de Acceso efectivos, y además comprobar que se encuentran en las en las regiones correspondientes (ver apartado 4.3.2), se presentan las gráficas de la próxima página. Se expondrán los casos con 9 y 25 Puntos de Acceso:




lenst.det.unifi.it - PuTTY

```
( 10770.3       1077034969    0 0             )
( 10780.4       1078035007    0 0             )
( 10790.4       1079035120    0 0             )
Valor final de numPoixNode: 8.6295
Valor final de Poicounter: 17259
gonzalez@bernh:~/Prueba$
```




lenst.det.unifi.it - PuTTY

```
( 10770.2       1077016804    0 0             )
( 10780.2       1078016806    0 0             )
( 10790.2       1079016808    0 0             )
No se ha hecho uso del AP con identificador: 8
No se ha hecho uso del AP con identificador: 2
No se ha hecho uso del AP con identificador: 15
No se ha hecho uso del AP con identificador: 23
No se ha hecho uso del AP con identificador: 14
No se ha hecho uso del AP con identificador: 22
Valor final de numPoixNode: 8.6485
Valor final de Poicounter: 17297
gonzalez@bernh:~/Prueba$
```

Figura 4.27: Simulaciones y regiones inutilizadas para los casos con 9 y 25 Puntos de Acceso.

A continuación se muestran las gráficas de los resultados más relevantes para poder afinar en la optimización del número de Puntos de Acceso, así como los resultados incongruentes para poder explicarlos:

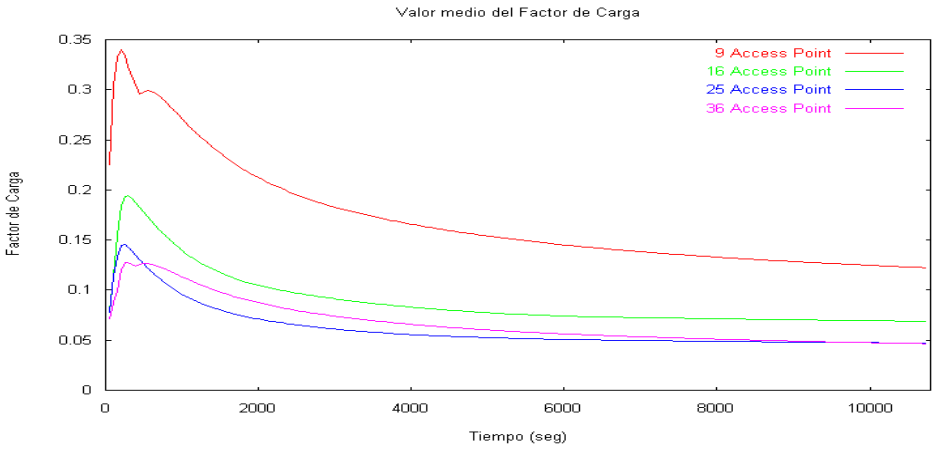



Figura 4.28: Gráfica del Factor de Carga en cada simulación.

Con la reducción del valor medio del factor de carga, el aprovechamiento de los Puntos de Acceso queda muy reducido. Debido a ello, se ha decidido descartar el caso con 36 Puntos de Acceso como posibilidad a valorar.

En la siguiente figura se representa el retraso de los mensajes Elastic. La situación con 36 Puntos de Acceso no mejora prácticamente el retraso respecto al caso con 25, lo que valida la decisión de descartar dicha situación. El caso con 16 Puntos de Acceso no se aleja mucho de estas dos mejores situaciones: 0.05 segundos. Puesto que se deben usar en realidad 4 Puntos de Acceso menos y que ya de por sí los retrasos conseguidos son buenos, se considera de momento caso óptimo el que pone en liza 16 Puntos de Acceso.

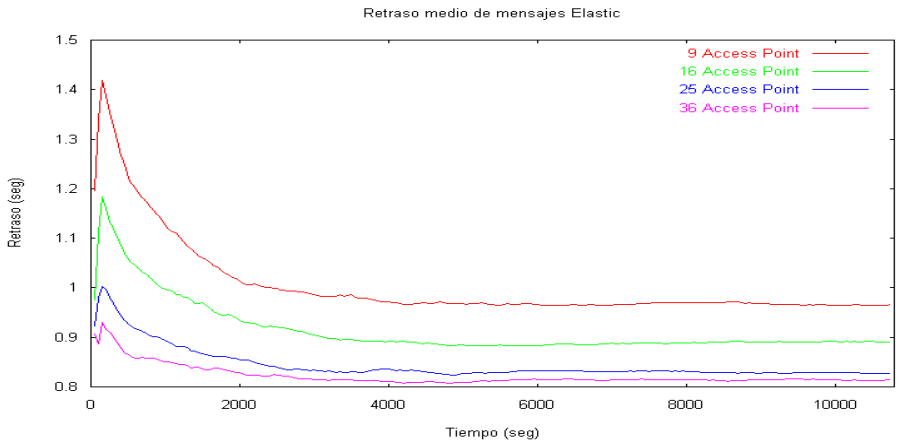



Figura 4.29: Gráficas del retraso de los mensajes Elastic.

A continuación se estudian las gráficas de los paquetes perdidos. Como ya sabemos, los paquetes se pierden cuando su tiempo de vida (TTL) es superado por el tiempo que han permanecido en el buffer de transmisión, o bien porque se han retransmitido más de un número de veces determinado. Si en un momento dado se aumenta el ritmo de creación de paquetes, y además el TTL posee un valor coherente, los paquetes perdidos crecerán en mayor medida. Es decir, si aumenta fuertemente el número de paquetes en el buffer, lo hará también el número de paquetes perdidos. Con esto se están intentando explicar las discontinuidades que se observan en la figura 4.30. Para clarificarlo aún más, se muestra debajo la gráfica del tamaño de buffer Elastic en su versión 'ProbeSlice', con el objetivo de mostrar la coincidencia de las discontinuidades en ambas gráficas.

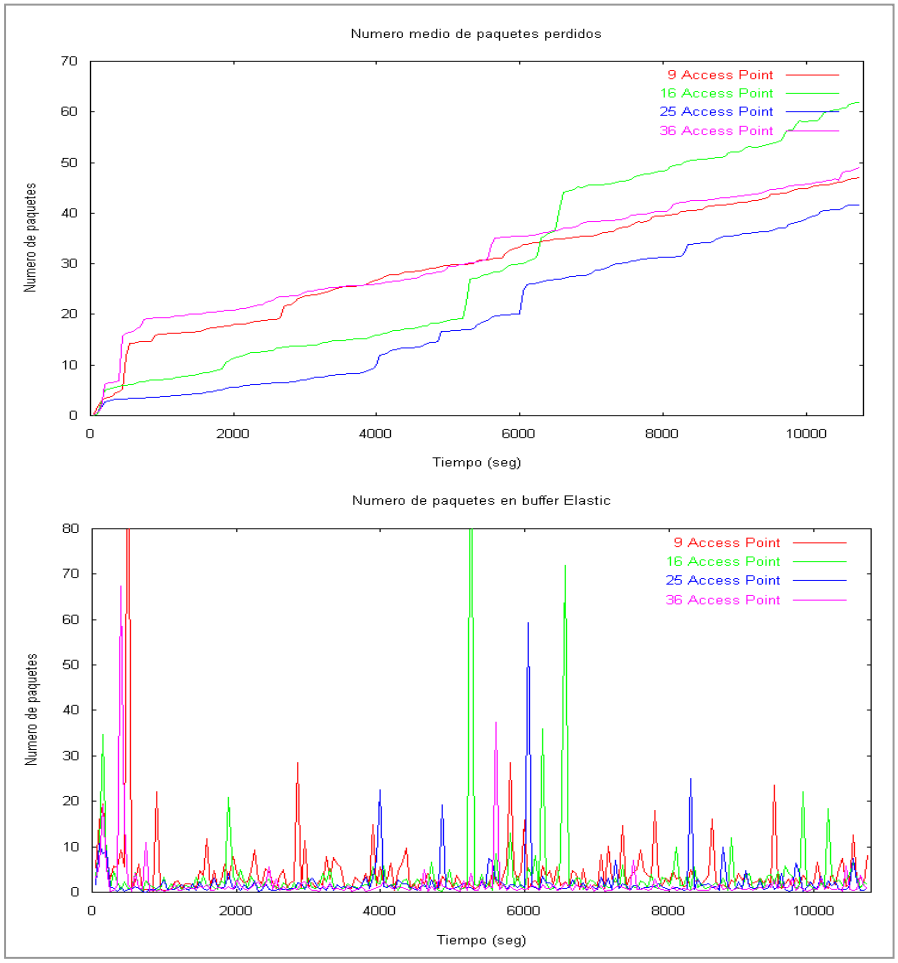



Figura 4.30: Gráficas de los paquetes perdidos y en buffer Elastic para cada simulación.

De esto se deduce que el número de paquetes perdidos no es un parámetro del todo fiable, puesto que depende en gran medida de las aleatoriedades. Por tanto, aunque se observe que con 16 Puntos de Acceso se está en este caso ante la peor situación, no hay que entenderlo como una antítesis.

El último resultado a analizar es la eficiencia media (figura 4.31). Resulta extraño que la asociada al caso con más Puntos de Acceso efectivos no sea la mejor, que sería lo lógico. Sin embargo, en este caso una gran parte de los paquetes se crean al inicio o mitad de la simulación (allí se encuentran las principales discontinuidades del buffer Elastic según la segunda gráfica de la figura 4.30). Esto afecta más a la media de la eficiencia que si se crean al final, por su definición y forma de medirla. De hecho, en el caso con mayor eficiencia (25 Puntos de Acceso) las principales discontinuidades en los paquetes creados se sitúan en la parte media y final de la simulación.

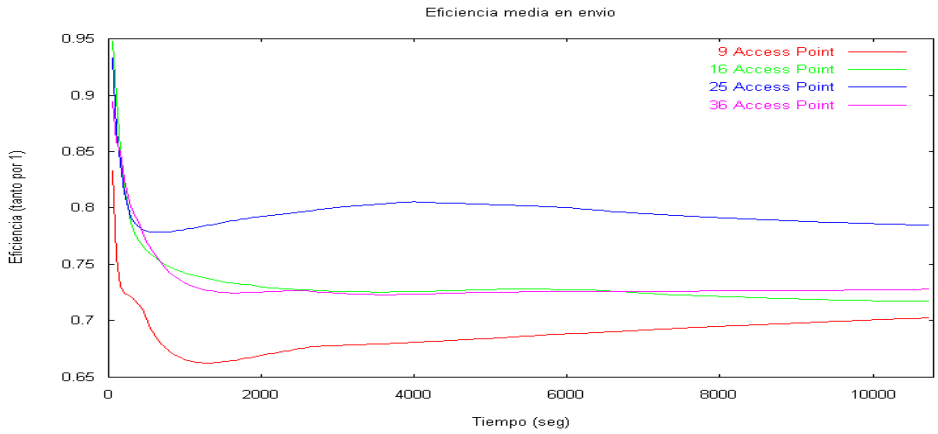



Figura 4.31: Gráfica de la eficiencia en cada simulación.

Llegados aquí, cabe pensar que las comparaciones a que hemos sometido los resultados podrían no ser válidas ya que el número de paquetes creados en cada caso no tiene porqué ser similar. Para rebatir esta idea se muestra una tabla como la tabla 4.4:

| **Nº Puntos Acceso** | **9** | **16** | **25** | **36** |
|---|---|---|---|---|
| **Nº Efectivo Puntos Acceso** | 9 | 15 | 19 | 26 |
| **Nº paquetes Stream creados P.A.** | 4029.29 | 2482.8 | 1907.56 | 1427.3 |
| **Nº paquetes Elastic creados P.A.** | 26776.9 | 18382.7 | 12520.3 | 9713.42 |
| **Nº total paquetes Stream creados** | 36263.61 | 37242 | 36243.64 | 37109.8 |
| **Nº total paquetes Elastic creados** | 240992.1 | 275740.5 | 237885.7 | 252548.92 |
| **Nº total paquetes creados** | **277255.71** | **312982.5** | **274129.34** | **289658.72** |

Tabla 4.7: Comparativa del número de paquetes creados para cargas medias en el escenario 2.

Se observan cantidades similares en los 4 casos, representando el menor valor el 87 % del valor máximo. Esta diferencia se puede considerar normal por las aleatoriedades. Por tanto, se concluye que las comparaciones son válidas, y más teniendo en cuenta que el caso más desfavorable coincide con el caso elegido.

### 4.4.3 Valoración del número de Puntos de Acceso óptimo para cargas elevadas

A continuación se muestran los principales resultados, añadiendo el total de paquetes creados, asociados a las simulaciones con velocidad de transmisión de 384 Kbps, con 5000 nodos y durante 7200 segundos (2horas). En dichas simulaciones variará el número de Puntos de Acceso, que irá desde 25 (5x5) hasta 64 (8x8).

| **Nº Puntos Acceso** | **25** | **36** | **49** | **64** |
|---|---|---|---|---|
| **Nº Efectivo Puntos Acceso** | 19 | 26 | 29 | 30 |
| **Nº paquetes creados por P.A.** | 26421.6 | 17924.8 | 20871.7 | 18128.8 |
| **Nº total paquetes creados** | 502010.4 | 466044.8 | 605279.3 | 543864 |
| **Factor de carga** | 0.147093 | 0.103966 | 0.0916375 | 0.0941351 |
| **Nº medio paquetes buffer Stream** | 0.0476752 | 0.0653526 | 0.0394013 | 0.0271856 |
| **Nº medio paquetes buffer Elastic** | 7.28712 | 4.09592 | 21.1722 | 6.17579 |
| **Retraso medio mensajes Stream** | 0.109523 | 0.107127 | 0.104745 | 0.103829 |
| **Retraso medio mensajes Elastic** | 0.975855 | 0.964722 | 0.915072 | 0.916649 |
| **Nº paquetes perdidos** | 38.4024 | 30.7605 | 57.2912 | 41.7986 |
| **Eficiencia media** | 0.683091 | 0.700248 | 0.717764 | 0.713731 |

Tabla 4.8: Comparativa de los resultados para cargas elevadas en el escenario 2.

A la vista de la tabla, se va a dilucidar ya la situación elegida. Puesto que los resultados son bastante interesantes para todos los casos: retraso de los mensajes Elastic menor a un segundo, retraso de los mensajes Stream de una décima, factor de carga mínimo, eficiencia del 70 % ..., se elige el caso que necesita menos Puntos de Acceso. La elección es lógica puesto que es la más económica. Además, esta opción da cobertura a más del 75 % del área total de trabajo haciendo uso únicamente de los Puntos de Acceso efectivos, mientras que las restantes darían, en orden de presentación: 72 %, 59 % y 46% del área de cobertura.

De esta forma, la situación elegida es la que hace uso de 25 Puntos de Acceso. No se acompaña ninguna gráfica porque no aporta nada nuevo a los resultados y decisiones ya expresados.

## 4.5 Resultados asociados a las simulaciones del Escenario 3

En estas simulaciones se busca valorar la influencia del proceso de deambulación en los resultados. Para ello se proponen dos situaciones de características idénticas en las que el único parámetro que variará será la distancia de Histéresis, el cual dictará si se produce el proceso. En la primera simulación se ha elevado a 112 metros (47 píxeles) para provocar que ni siquiera en el peor caso (en línea recta y a la velocidad máxima), el terminal pueda superar esta distancia antes de recibir los paquetes. En la segunda simulación reduce a 1.2 metros (0.52 píxeles), asegurando muchas transiciones.

En las dos situaciones, para favorecer las transiciones entre regiones, dispondrán de: 144 Puntos de Acceso, tiempo nulo de visita a los Puntos de Interés y velocidad de transmisión reducida a 9.6 Kbps. Del resto de parámetros, comentar que el tiempo de simulación se ha reducido a 3600 segundos y el número de nodos a 500, puesto que las simulaciones tienen que soportar más carga al no haber parada en los Puntos de Interés.

| | **Sin deambulación** | **Con deambulación** |
|---|---|---|
| **Nº efectivo Puntos de Acceso** | 35 | 77 |
| **Nº total paquetes creados** | 277506 | 348620 |
| **Nº paquetes con deambulación** | 0 | 3612.95 |
| **Nº medio paquetes buffer Stream** | 0.558218 | 0.332721 |
| **Nº medio paquetes buffer Elastic** | 193.355 | 129.984 |
| **Retraso medio mensajes Stream** | 2.39634 | 2.37478 |
| **Retraso medio mensajes Elastic** | 54.0104 | 49.3773 |
| **Factor de carga** | 1.05856 | 1.05059 |

Tabla 4.9: Comparativa de los resultados del escenario 3.

Como primera observación, notar como en la situación con deambulación, en la que se han creado incluso un 25 % de paquetes más que en la otra, se obtienen mejores resultados. Esto se debe a que en esta situación la carga se reparte entre 42 Puntos de Acceso más. Bien es verdad que en dichos Puntos de Acceso no se crearán paquetes, pero sí se dará servicio a los paquetes que lleguen mediante deambulación. Por tanto, en la situación con deambulación nos encontramos con Puntos de Acceso que únicamente se utilizarán cuando lleguen paquetes procedentes de un Punto de Acceso contiguo. Por ello la mejora que se produce en los resultados no es tan espectacular como debiera.

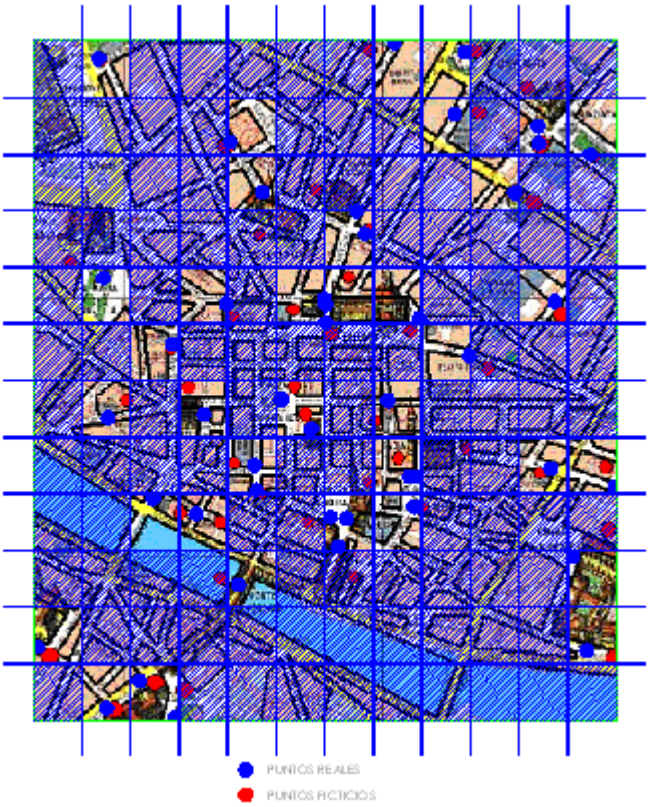



Figura 4.32: Regiones inutilizadas en la simulación sin deambulación.

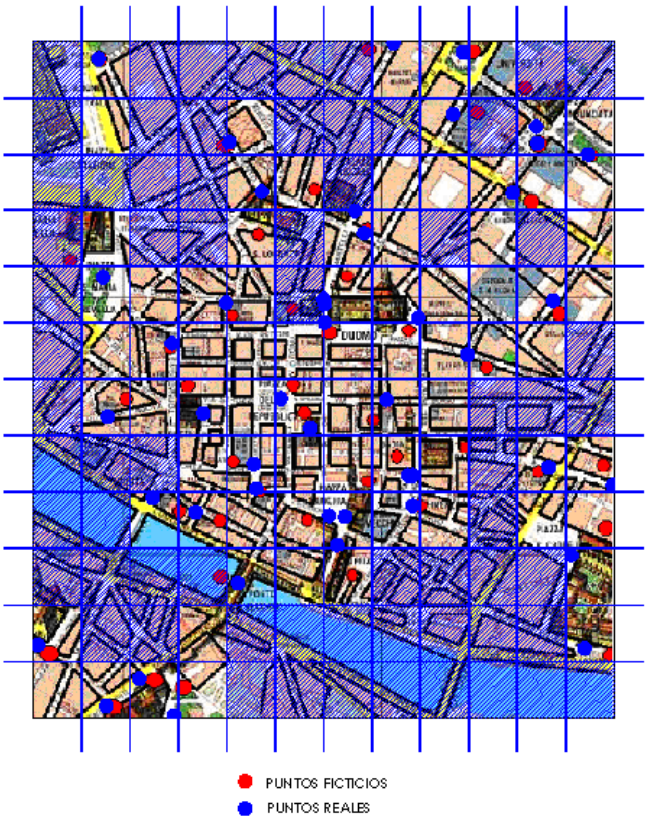



Figura 4.33: Regiones inutilizadas en la simulación con deambulación.

Como la principal idea de este apartado es certificar que se produce la deambulación, la página anterior se ha dedicado a ilustrar este hecho. Se muestran en ella dos imágenes del mapa divididas en las 144 regiones de que se hace uso en estas simulaciones. En la primera de ellas se analizan las regiones inutilizadas cuando no se produce deambulación, y en la segunda cuando sí se produce. Se puede observar claramente, no solo cómo aumentan las regiones usadas, sino también cómo la parte central queda exenta de regiones inútiles por estar poblada de Puntos de Interés cercanos.

La otra idea de este apartado es analizar cómo afecta la deambulación a los resultados. Por la tabla, hemos visto que se producían mejoras, pero vamos a corroborarlo con gráficas para cerciorarnos de que las mejoras se producen durante todo el tiempo de simulación.

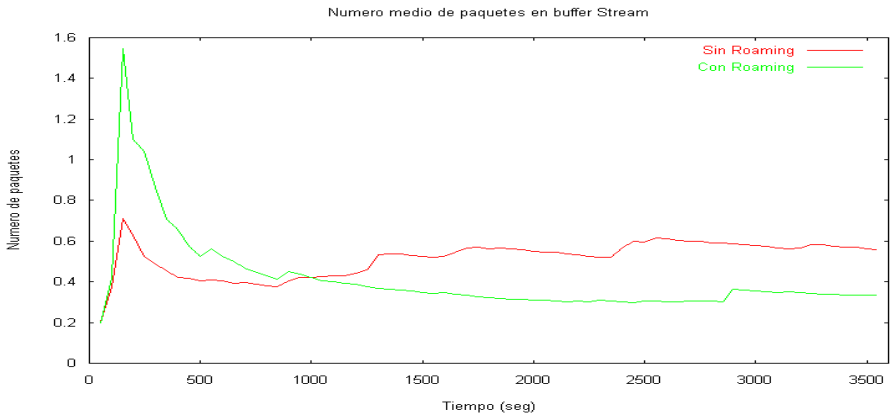



Figura 4.34: Gráfica de paquetes en buffer Stream para ambas simulaciones.

En este caso, al inicio no se produce tal mejora debido a que en realidad en la situación con deambulación se crean más paquetes y además aún no se ha dado tiempo a la mayoría de los terminales a que se desplacen de sus regiones para que intervenga la deambulación. Posteriormente, se observa claramente como se reduce la media, debido a que la deambulación entra en juego y hace que los paquetes se dividan entre más Puntos de Acceso.

No obstante, comentar que los paquetes Stream son los que menos sufrirán deambulación, puesto que por media, como se ve en la figura, tardan unos 2’5 segundos en ser enviados. Este tiempo suele ser insuficiente para que se produzca el proceso.

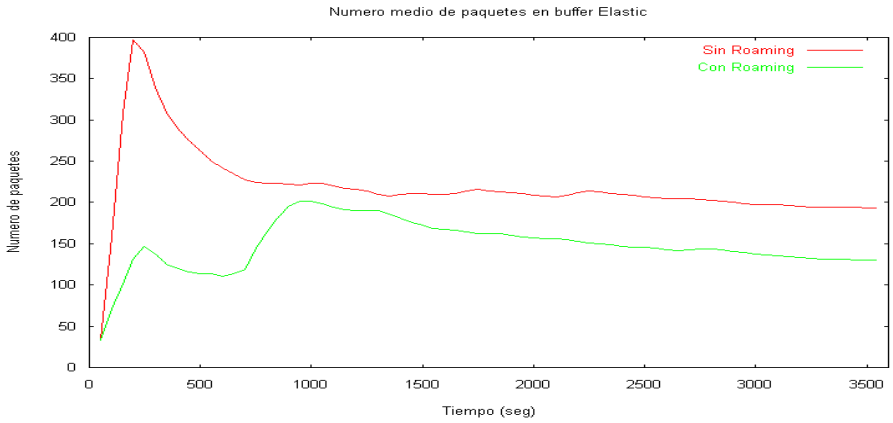



Figura 4.35: Gráfica de paquetes en buffer Elastic para ambas simulaciones.

El caso del buffer Elastic es diferente. Aquí sí es fácil que se produzca deambulación en todo momento, aunque sea al inicio, ya que como se observa en la tabla, el retraso medio ronda los 50 segundos y en este tiempo es fácil encontrar a terminales que se han cambiado de región. Así pues, pese a crearse más paquetes en el caso con deambulación, la presencia de este proceso provoca que en todo momento se encuentren los buffer de memoria Elastic más descargados.

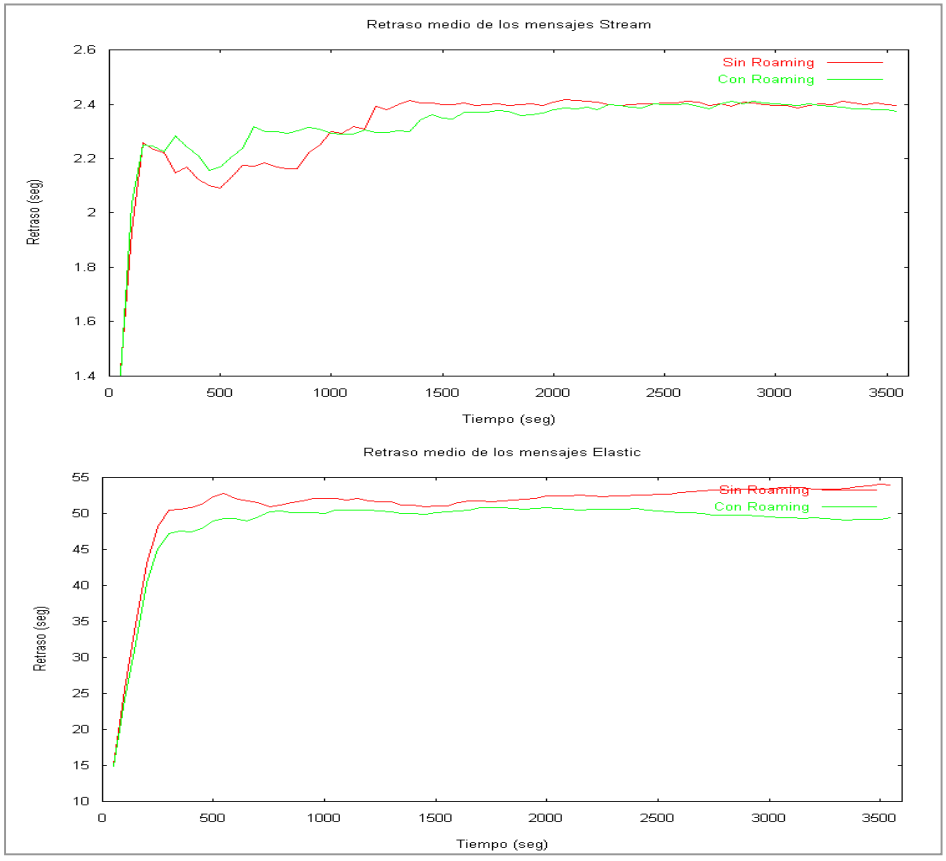



Figura 4.36: Gráfica del retraso en mensajes Stream y Elastic para cada simulación.

La gráfica superior de la figura 4.36 sirve para corroborar que el proceso de deambulación no lo sufren apenas los paquetes Stream puesto que se obtienen valores de retraso similares pese a disponer de muchos más Puntos de Acceso. De la gráfica inferior de la figura, destacar que en gran parte de la simulación el retraso de los paquetes Elastic mejora hasta en 5 segundos cuando se produce deambulación. Esta diferencia aporta una notable mejora en la calidad del servicio que certifica las ventajas de la deambulación.

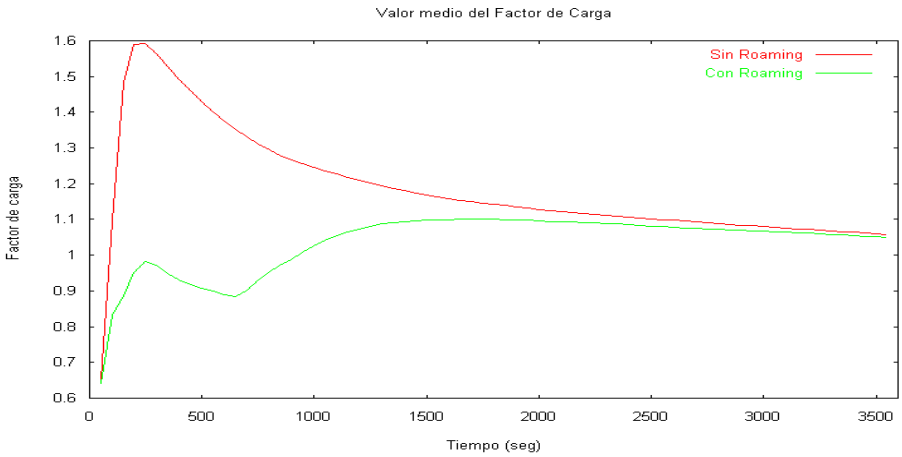



Figura 4.37: Gráfica del Factor de carga en ambas simulaciones.

En la gráfica superior se observa claramente la descongestión que se produce en gran parte de la simulación al usar la deambulación. La explicación de que al final se avecinen los resultados proviene del aumento de paquetes creados en la simulación con deambulación.

Como conclusión, destacar que el proceso de deambulación puede favorecer en gran medida a un sistema como el nuestro en el que la información se transmite a ráfagas, ya que provoca la descongestión de Puntos de Acceso probablemente saturados para pasar esta carga a Puntos de Acceso que generalmente van a estarlo menos. Los nuevos Puntos de Acceso se supone que estarán menos saturados por el hecho de que la información se crea a ráfagas, ya que es improbable que los dos Puntos de Acceso se encuentren saturados en el mismo instante. Por otro lado, es probable que el Punto de Acceso inicial esté saturado de carga ya que en esta situación el retraso de los paquetes aumenta, y por tanto, es más probable que el terminal pueda cambiar de región antes de recibir el paquete.

## 4.6 Resultados de movilidad asociados a las simulaciones

En el programa no solo se crean resultados asociados al tráfico de paquetes, también hay resultados de la movilidad que son los que se pretenden mostrar en este apartado.

Así pues, se procede a mostrar el resultado ProbeSlice que muestra la media de Puntos de Interés visitados por nodo durante una simulación. Al ser ProbeSlice se observarán muchas irregularidades en las gráficas porque se resetea la media en cada intervalo de tiempo. Por su parte, el valor medio dependerá del tiempo de simulación, con lo que se presenta una gráfica asociada a cada tiempo, y se comentan las relaciones entre ellas:

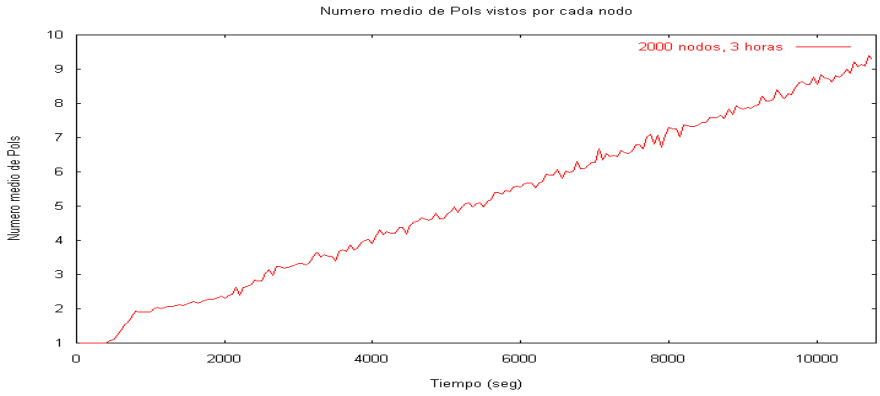



Figura 4.38: Gráfica con número medio de Puntos de Interés visitados durante 3 horas.

A la vista de la figura se observa que este resultado crece con una pendiente más o menos regular excepto al principio, donde en los primeros instantes no crece y justo después lo hace con mayor pendiente. Esto se debe a que los nodos inicialmente buscan su primer destino (no se visitan Puntos de Interés), y después los encuentran en instantes similares porque el primer Punto de Interés está siempre cercano (ver apartado 2.2.2).

Por otro lado, cada nodo visita una media de 9.27 Puntos de Interés. Si el tiempo medio de permanencia en un Punto de Interés es de 1050 segundos (media entre 300 y 1800 segundos), y la simulación dura 10800 segundos, se emplean únicamente 1066.5 segundos en búsquedas. Es decir, se dedica una media de 115 segundos para encontrar el próximo Punto. Pueden resultar muy escasos, pero se debe a una serie de razones:

- La mayoría de los Puntos de Interés se encuentran muy próximos entre sí.

- Nuestro Modelo de Movilidad busca los Puntos de Interés más cercanos en primer lugar, con lo que se acentúa la primera razón.
- La velocidad media de un terminal se sitúa en los 3 kms/h (valor medio entre 2 y 4 kms/h), quizá elevada si se considera que los usuarios del servicio serán turistas. Pero se escoge este valor ya que es preferible que esté en exceso.
- La cuarta razón proviene de que en realidad el centro de Florencia es bastante pequeño, y por tanto las distancias que aparecen en el mapa son pequeñas.

Con el objetivo de seguir analizando estos valores de tiempo, se muestran las siguientes gráficas con simulaciones de 7200 y 3600 segundos:

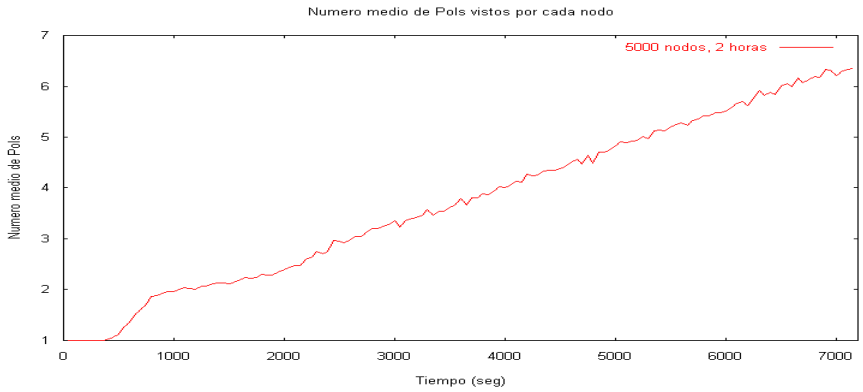



Figura 4.39: Gráfica con número medio de Puntos de Interés visitados durante 2 horas.

En esta situación se visitan 6.45 Puntos de Interés por cada nodo. Si se aplican los cálculos anteriores, se obtiene que se reduce aún más el tiempo empleado para localizar un Punto de Interés: 67 segundos. La razón que explica la reducción en el tiempo de búsqueda está en el Modelo de Movilidad; según este, cuanto menos Puntos de Interés haya visitado cada nodo, más posibilidades se tienen de encontrar uno cercano.

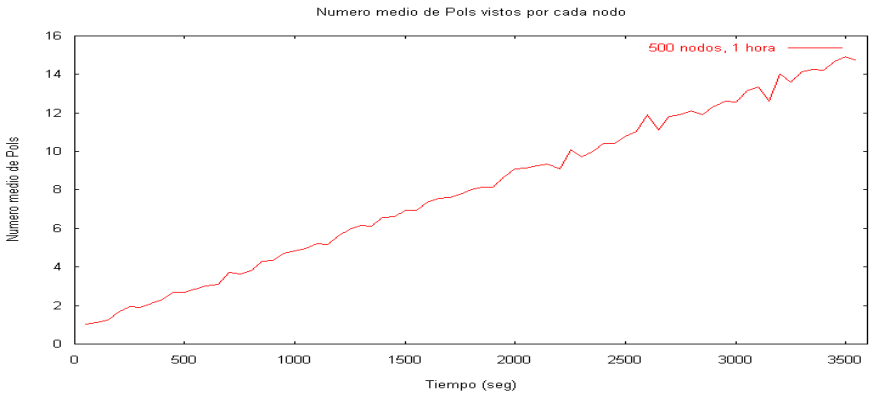



Figura 4.40: Gráfica con número medio de Puntos de Interés vistos durante 1 hora.

En este último caso se visitan 14.75 Puntos de Interés en tan solo 3600 segundos. Tiene sentido puesto que en esta simulación el tiempo medio de permanencia en el Punto de Interés es nulo. Haciendo cálculos, se dedican 245 segundos de media para buscar el próximo destino. La razón vuelve a estar en el Modelo de Movilidad. Cuanto más Puntos de Interés se visitan, menos probabilidad de que estén cercanos entre sí.

Para finalizar, se muestra un histograma en el que se representa la probabilidad con que se visita cada Punto de Interés, y junto a este el mapa de la simulación, para demostrar que los Puntos de Interés cercanos entre sí son los visitados con mayor probabilidad.

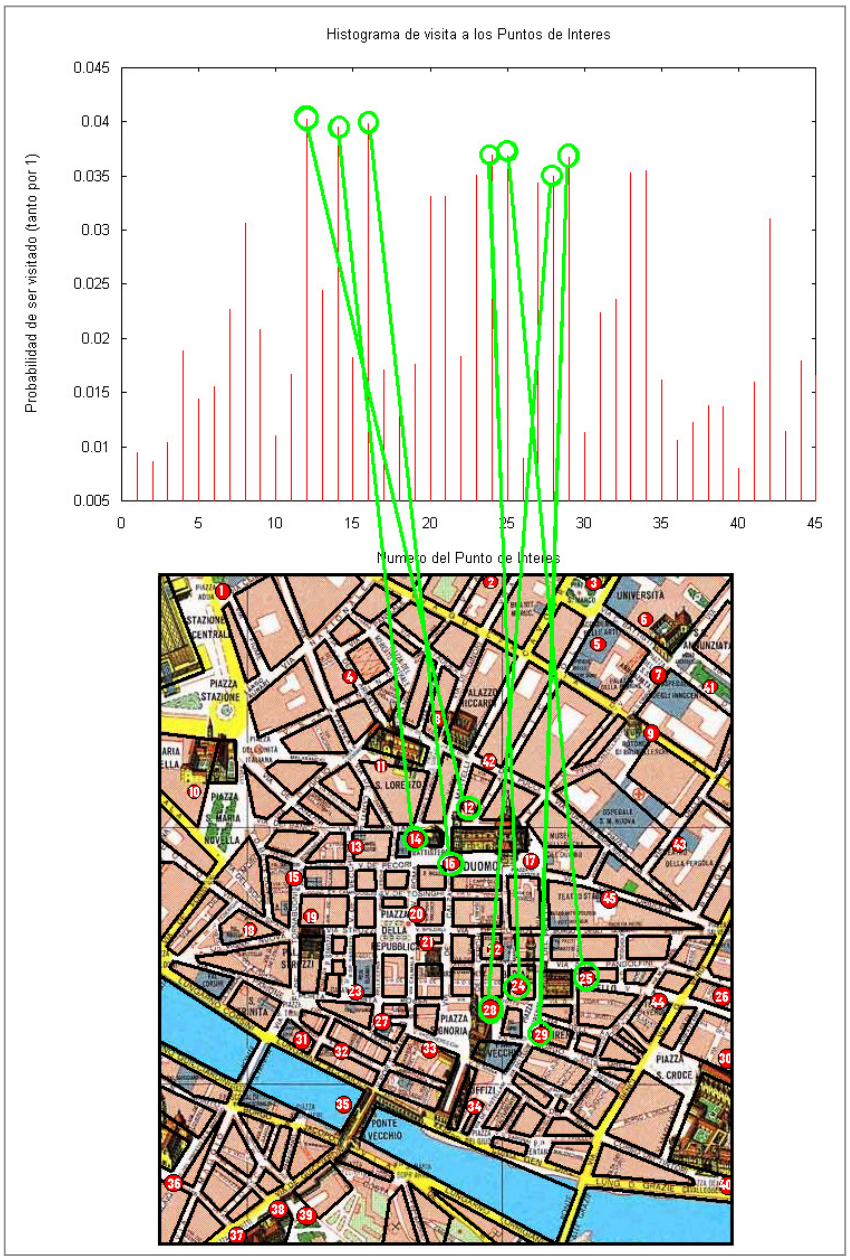



Figura 4.41: Histograma de visita a los Puntos de Interés,
y situación geográfica de los más visitados

# Capítulo 5
# Conclusiones y posibles mejoras

## *5.1 Conclusiones*

El presente Proyecto Fin de Carrera ha conseguido satisfacer plenamente los dos grandes propósitos para los que fue creado:

- Crear un simulador que modele una red de tráfico Streaming y Elastic en el centro histórico de cualquier ciudad, en la que los nodos son móviles y por tanto habrá que tener en cuenta su desplazamiento.
- Comprobar el correcto funcionamiento del simulador para situaciones reales de tráfico y de movilidad.

El primer objetivo queda adecuadamente explicado en el apartado 2, mientras que el segundo se satisface con los resultados de los apartados 3 y 4.

Sin embargo, hasta llegar a conseguir el correcto funcionamiento del programa y poder presentar las simulaciones adecuadas han surgido miles de dificultades, algunas de las cuales se presentan a continuación:

- La primera y mayor dificultad ha sido asegurar el correcto funcionamiento del modelo de Movilidad. Han sido muchas las variaciones las que se han tenido que hacer: para buscar las expresiones matemáticas y bucles que definen correctamente las funciones, encontrar el algoritmo que de forma más eficiente y bordease los obstáculos, … Además, no servía con un algoritmo medianamente bueno, puesto que iba a ser sometido a situaciones muy exigentes, como son las del centro histórico de una ciudad (miles de calles y muy pequeñas). Pero finalmente, tras las continuas modificaciones y pruebas posteriores, se ha podido crear un modelo al que de momento el alumno no le ha encontrado debilidad.

- La creación del propio programa y, sobre todo, su división en un número de clases adecuadas, fue un gran problema presentado al inicio. No se sabía si era necesario crear una clase Servidor (no se acabó creando) para independizarlo de la clase AccessPoint, tampoco si era necesaria la clase Regione por ser únicamente las zonas que contienen los Puntos de Acceso, costó imaginar una clase Common que fuera pública y compartida por el resto de clases,...

- Por otro lado, la elevada cantidad de tiempo dedicada al apartado de simulaciones. La razón se encuentra en que las simulaciones efectuadas eran muy duraderas y además era muy común tener que repetir algunas de ellas por error a la hora de elegir los parámetros.

- Finalmente, también fue complicado entender correctamente la plataforma NePSi. La variedad de clases y de variables que en éste se exponen, la forma de actuar de los eventos, cómo se crea el archivo de parámetros... fueron los responsables.

Tras explicar esta última dificultad podría parecer que el uso de NePSi ha dificultado, más que simplificar, la creación del programa. Todo lo contrario, ha sido el gran soporte para la creación del mismo. De hecho, habría sido una labor complicadísima crear el presente programa sin su utilización. Las características más difíciles de entender del mismo (explicadas arriba), son las que finalmente justifican su uso. Con NePSi, el usuario se despreocupa de la recogida de parámetros y resultados, de la correcta gestión de los eventos, e incluso la creación de distribuciones aleatorias; además, se heredan clases y variables que seguro serán muy útiles. Ha sido un gran aliado y el autor del presente Proyecto Fin de Carrera aconseja definitivamente su utilización.

La opinión final del autor sobre el presente Proyecto Fin de Carrera es que es un Proyecto demasiado extenso como para ser realizado en el supuesto período de 6 meses, de hecho se le dedicó casi un año; además, acaba presentando infinitas dificultades. Sin embargo, es un gran Proyecto para poder desarrollar las capacidades del alumno, sobre todo las relacionadas con la programación y la capacidad de invención. Nos encontramos con un conflicto de intereses tiempo-aprendizaje en el que el autor se siente afortunado por haberse decantado por la segunda opción.

## 5.2 Posibles mejoras

En esta sección se comentan algunas de las mejoras que se pueden realizar al programa desarrollado en este proyecto. Son las mejoras que el propio autor ha pensado tras su ejecución, y que se cree que darían un valor añadido al modelo que se desea representar.

En este sentido, el principal aspecto a mejorar se considera que es la parte de retrasmisión de los paquetes. La complicada tarea de retransmisión de paquetes queda reducida en el simulador a una probabilidad denominada FER (probabilidad de error de tramas), que ciertamente identifica en una red a la probabilidad con la que un paquete llega con error y hay que retransmitirlo. El problema reside en que este valor suele ser un valor que se toma a posteriori, es decir, después de hacer miles de pruebas con la red real, y no a priori incluso de hacer las pruebas con el simulador. Por tanto, las mejoras que se proponen en este apartado van a estar dirigidas a corregir esta limitación. Para ello, se habrían de considerar:

- La atenuación.
- Las interferencias.
- El ruido.

No se pretende en esta sección explicar cómo habrían de desarrollarse en el programa cada uno de los parámetros arriba expuestos; sin embargo, se muestra como ejemplo el desarrollo que se necesitaría para considerar la atenuación de forma adecuada.

Para considerar la atenuación habría que empezar por tener en cuenta que la mayoría de los obstáculos no son solo obstáculos físicos, sino que son también obstáculos electromagnéticos: suelen poseer paredes u otros elementos que la onda electromagnética debe atravesar. Por tanto, poseen un factor de atenuación que servirá para expresar la potencia perdida por atravesar el obstáculo. Eso sin considerar que el propio desplazamiento de la onda puede provocar pérdidas. De esta forma, cuando se envía un paquete habría que tener en cuenta la potencia de transmisión, la cantidad de potencia que pierde la onda por atravesar los obstáculos electromagnéticos que se encuentra en su recorrido, y el umbral de potencia en recepción. Si la potencia de la onda supera este umbral al llegar a su destino, el paquete, respecto a este parámetro, llega correctamente.

# Sección II

# Anexos

# Anexo A

# Listado del Programa

## *A.1 AccessPoint.cpp*

```
00001 #include "DevTerminal.h"
00002 #include "Node.h"
00003 #include "AccessPoint.h"
00004 #include "Mappa.h"
00005 #include "Regione.h"
00006 #include "Common.h"
00007 #include "Ev_TryToSend.h"
00008 #include "Ev_Msg_Request.h"
00009 #include "Ev_ObserveAP.h"
00010
00011 #define CABECERA 30
00012 #define BYTE 8
00013 //#############################################################################
00014 //-----------------------------------setup-------------------------------------
00015 //#############################################################################
00016 void AccessPoint::setup (ParamManager * Param, GlobalProbeManager * Results)
00017 {
00018   Param->addClass ("APoint", "Generic AccessPoint parameters");
00019   Param->addParameter ("APoint",new DoubleParameter ("FER",
00020                        "Frame Error Rate","", "(0,inf)"));
00021   Param->addParameter ("APoint",new IntParameter ("max_num_tx4pack",
00022                        "Numero maximo de transmisiones por paquete","", "(0,inf)"));
00023   Param->addParameter ("APoint",new DETimeParameter ("maxttlxpck",
00024                        "Time-To-Live de los paquetes Elastic","", "(0,inf)"));
00025   Param->addParameter ("APoint",new DETimeParameter ("maxttlxpckStream",
00026                        "Time-To-Live de los paquetes Stream","", "(0,inf)"));
00027   Param->addParameter ("APoint",new IntParameter ("maxLunPck",
00028                        "Tamaño de un apquete en bytes","", "(0,inf)"));
00029   Param->addParameter ("APoint",new DoubleParameter ("Mean_Message_Length",
00030                        "Tamaño medio en bytes de mensajes Elastic","", "(0,inf)"));
00031   Param->addParameter ("APoint",new DoubleParameter ("Mean_Message_LengthStream",
00032                        "Tamaño medio en bytes de mensajes Stream","", "(0,inf)"));
00033   Param->addParameter ("APoint",new DoubleParameter ("Minim_Message_Length",
00034                        "Numero minimo de bytes en un mensaje Elastic","","(0,inf)"));
00035   Param->addParameter ("APoint",new DoubleParameter ("Minim_Message_LengthStream",
00036                        "Numero minimo de bytes en un mensaje Stream","","(0,inf)"));
00037   Param->addParameter ("APoint",new IntParameter ("maxMsg4Req",
00038                        "Numero max de mensajes Elastic por peticion","","(0,inf)"));
00039   Param->addParameter ("APoint",new IntParameter ("maxMsg4ReqStream",
00040                        "Numero max de mensajes Stream por peticion","","(0,inf)"));
00041 }
00042
00043 //#############################################################################
00044 //-------------------------------Initialize------------------------------------
00045 //#############################################################################
00046 void AccessPoint::Initialize (void)
00047 {
00048   numnodi = get < IntParameter, int >(Param, "numnodi", "NwMap", "");
00049   HopTime = get < DETimeParameter, DETime > (Param, "HopTime", "Node", "");
```

```
00050   ObserveTime = get < DETimeParameter, DETime > (Param, "ObserveTime", "Node", "");
00051   maxLunPck = get < IntParameter, int >(Param, "maxLunPck", "APoint", "");
00052   max_num_tx4pack = get < IntParameter, int >(Param,"max_num_tx4pack","APoint","");
00053   Maxttlxpck = get < DETimeParameter, DETime > (Param, "maxttlxpck", "APoint", "");
00054   MaxttlxpckStream =
00055        get < DETimeParameter, DETime > (Param, "maxttlxpckStream", "APoint", "");
00056   meanMessageLength =
00057        get < DoubleParameter, double >(Param,"Mean_Message_Length", "APoint", "");
00058   meanMessageLengthStream =
00059        get <DoubleParameter, double>(Param,"Mean_Message_LengthStream","APoint","");
00060   location =
00061        get < DoubleParameter, double >(Param, "Minim_Message_Length", "APoint", "");
00062   locationStream =
00063        get <DoubleParameter,double>(Param,"Minim_Message_LengthStream","APoint","");
00064   maxMsg4Req = get < IntParameter, int >(Param, "maxMsg4Req", "APoint", "");
00065   maxMsg4ReqStream = get < IntParameter,int >(Param,"maxMsg4ReqStream","APoint","");
00066   TimeResolution =  get < IntParameter, int > (Param, "TimeResolution", "Test", "");
00067   BandaTx = get < DoubleParameter, double >(Param, "BandaTx", "NwMap", "");
00068   tempoSimulazione =
00069        get < DETimeParameter, DETime >(Param, "SimulationTime", "System", "");
00070
00071
00072   pckTrasmittingTime = DETime((BYTE*maxLunPck)/BandaTx);
00073   messaggiocorrente = 1;
00074   nodiAP = 0;
00075   NNumPckRoamingBorn = 0;
00076
00077   shape = meanMessageLength/(meanMessageLength - location);
00078   shapeStream = meanMessageLengthStream/(meanMessageLengthStream - locationStream);
00079   MessageLength = ParetoRndGen(location,shape);
00080   MessageLengthStream = ParetoRndGen(locationStream,shapeStream);
00081
00082   //--------------Probes-----------------------------------------------------------
00083   BufferElasticSize = Results->getGlobalProbeMeanSlice("BufferElastic");
00084   BufferStreamSize = Results->getGlobalProbeMeanSlice("BufferStream");
00085   NumPckRetrasmitted = Results->getGlobalProbeMeanSlice("PckRetrasmitted");
00086   NumPckLostTotal = Results->getGlobalProbeMeanSlice("PckLost");
00087   NumPckBorn = Results->getGlobalProbeMeanSlice("PckBorn");
00088   NumMSGStreamBorn = Results->getGlobalProbeMeanSlice("MsgStreamBorn");
00089   NumMSGElasticBorn = Results->getGlobalProbeMeanSlice("MsgElasticBorn");
00090   NumPCKBornElastic = Results->getGlobalProbeMeanSlice("PckBornElastic");
00091   NumPCKBornStream = Results->getGlobalProbeMeanSlice("PckBornStream");
00092   LoadFactor = Results->getGlobalProbeMeanSlice("LoadFactor");
00093   Eficiencia = Results->getGlobalProbeMeanSlice("Eficiencia");
00094   NumPckRoamingBorn = Results->getGlobalProbeMeanSlice("NumPckRoamingBorn");
00095
00096   //-----------Probes Mean-------------------------------------------------------
00097   MeanBufferElasticSize = Results->getGlobalProbeMean("MeanBufferElasticSize");
00098   MeanBufferStreamSize = Results->getGlobalProbeMean("MeanBufferStreamSize");
00099   MeanPckRitrasmessi = Results->getGlobalProbeMean("MeanPckRitrasmessi");
00100   MeanPckBornElastic = Results->getGlobalProbeMean("MeanPckBornElastic");
00101   MeanPckBornStream = Results->getGlobalProbeMean("MeanPckBornStream");
00102   MeanNodiPerAP = Results->getGlobalProbeMean("MeanNodiPerAP");
00103   MeanLoadFactor = Results->getGlobalProbeMean("MeanLoadFactor");
00104   MeanEficiencia = Results->getGlobalProbeMean("MeanEficiencia");
00105   MeanNumPckRoamingBorn = Results->getGlobalProbeMean("MeanNumPckRoamingBorn");
00106
00107
00108   NNumPCKElasticBorn = 0;
00109   NNumPCKStreamBorn = 0;
00110   NNumPCKRetrasmitted = 0;
00111   NNumMSGElasticBorn = 0;
00112   NNumMSGStreamBorn = 0;
00113   NNumPCKLost = 0;
00114   numACK = 0;
00115
00116   //-------------Inicializo los buffer de paquetes--------------------------------
00117   BufferPCK.clear();
```

```
00118   BufferPCKStream.clear();
00119
00120
00121   newEvent (new Ev_TryToSend(this),pckTrasmittingTime);
00122   newEvent (new Ev_ObserveAP (this), HopTime );
00123
00124 }
00125
00126 //#########################################################################################
00127 //---------------------------------Msg_Request-------------------------------------
00128 //#########################################################################################
00129 void AccessPoint::HEv_Msg_Request(int IDnodo)
00130 {
00131   int numTotMess,numElasticMess,numStreamMess,Iddest,idregionedest,i,numpck;
00132   double prob,numbytes;
00133   coordinates posdest;
00134   DETime ttlmassimo;
00135   bool type;
00136   double numPoixSecxNode,numMeanPck;
00137
00138   type = true;  //primero se crean los paquetes stream
00139   numElasticMess = rndUniformInt(0,maxMsg4Req);
00140   numStreamMess = rndUniformInt(1,maxMsg4ReqStream);
00141   numTotMess = numElasticMess+numStreamMess;
00142
00143   for(int j=0;j<numTotMess;j++) {
00144     if (j >= numStreamMess){ //cuando se termine con los stream se crean los elastic
00145         //Se elige el número de paquetes del mensaje imagen
00146         numbytes = MessageLength.get();
00147         //TrafficGeneratorElasticHistogram->Observe(numbytes);
00148         double pckdec= numbytes/(maxLunPck - CABECERA);
00149         numpck = int(ceil(pckdec));
00150         type = false;
00151         ttlmassimo = Maxttlxpck;
00152         NNumPCKElasticBorn = NNumPCKElasticBorn + numpck;
00153         NNumMSGElasticBorn = NNumMSGElasticBorn + 1;
00154     }
00155     else {
00156         //Se elige el número de paquetes del mensaje texto
00157         numbytes = MessageLengthStream.get();
00158         //TrafficGeneratorStreamHistogram->Observe(numbytes);
00159         double pckdec= numbytes/(maxLunPck - CABECERA);
00160         numpck = int(ceil(pckdec));
00161         ttlmassimo = MaxttlxpckStream;
00162         NNumPCKStreamBorn = NNumPCKStreamBorn + numpck;
00163         NNumMSGStreamBorn = NNumMSGStreamBorn + 1;
00164     }
00165     Iddest = IDnodo; // del nodo que hizo la peticion y que se pasa por parametro.
00166     posdest = pmappa->trovaposizionenodo(Iddest);
00167     idregionedest = pmappa->trovaregioneappartenenza(posdest);
00168
00169     //Generación de los paquetes del mensaje:
00170     for (int i = 0; i < numpck; i++) {
00171       packet pck;
00172       pck.IDMessage = messaggiocorrente;
00173       pck.IDpacket = i;
00174       pck.numpckinmsg = numpck;
00175       pck.lunghezza = maxLunPck;
00176       pck.IDNodosorgente = ID;//Id del AP
00177       pck.IDNododestinazione = Iddest;
00178       pck.posdestinazione = posdest;
00179       pck.max_num_tx = max_num_tx4pack;
00180       pck.tx_number = 0;
00181       pck.priority = type;
00182       pck.IDRegioneDestinazione = idregionedest;
00183       pck.TimeBorn = *T;
00184       pck.TTL = ttlmassimo;
00185
```

```
00186       if (type == false) {
00187         BufferPCK.push_back (pck);
00188       }
00189       else {
00190         BufferPCKStream.push_back (pck);
00191       };
00192     }
00193     messaggiocorrente = messaggiocorrente + 1;  //Incremento para dar otro
00194   }                                             //identificador al proximo mensaje
00195 };
00196

00197 //#########################################################################################
00198 //---------------------------------HEv_TryToSend-----------------------------------------
00199 //#########################################################################################
00200 void AccessPoint::HEv_TryToSend ()
00201 {
00202   packet pck;
00203   double FER;
00204   int IdRegDest,IDRegInic;
00205   DETime tempIfVuoto = DETime(double(pckTrasmittingTime)/10);
00206
00207   if ((BufferPCKStream.empty()) && (BufferPCK.empty()) ) {
00208         //Si no tengo ningun paquete en cola, no hago nada
00209         newEvent (new Ev_TryToSend(this),tempIfVuoto); //MENOR TIEMPO AQUI
00210         return;
00211   };
00212
00213   if ( !(BufferPCKStream.empty()) ) {
00214         //Si hay al menos un paquete en la cola Stream ,sirvo el primero de esa cola
00215         pck = BufferPCKStream.front ();
00216   }
00217    else {
00218         //Si no hay paquetes Stream , se procede al envio de los de la cola Elastic
00219         pck = BufferPCK.front ();
00220   }
00221   pmappa->spediscipacchetto (this,pck);
00222 }
00223
00224 //#########################################################################################
00225 //---------------------------------CopyPacket-----------------------------------------
00226 //#########################################################################################
00227 void AccessPoint::CopyPacket(vector <packet> vect,packet pck,int coda)
00228 {
00229   packet pck2,pck3;
00230
00231   int i = 0;
00232   NNumPckRoamingBorn = NNumPckRoamingBorn + vect.size();
00233
00234   if (coda==0){
00235       pck2 = BufferPCKStream.front();
00236       for(list < packet >::iterator itpck = BufferPCKStream.begin ();
00237           itpck != BufferPCKStream.end (); itpck++)
00238       {
00239             if((*itpck).IDMessage != (pck2.IDMessage)){
00240                while (i < vect.size()){
00241                    BufferPCKStream.insert(itpck,vect[i]);
00242                    i++;
00243                    itpck++;
00244                }
00245             }
00246             pck2 = *itpck;
00247       }
00248   }
00249   else {
00250     pck2 = BufferPCK.front();
00251     for (list < packet >::iterator itpck =  BufferPCK.begin ();
00252          itpck != BufferPCK.end (); itpck++)
```

```
00253     {
00254             if((*itpck).IDMessage != pck2.IDMessage){
00255                while (i < vect.size()){
00256                    BufferPCK.insert(itpck,vect[i]);
00257                    i++;
00258                    itpck++;
00259                }
00260             }
00261             pck2 = *itpck;
00262     }
00263   }
00264 };
00265
00266 //#########################################################################################
00267 //---------------------------------diminuirebuffer-----------------------------------------
00268 //#########################################################################################
00269 void AccessPoint::diminuirebuffer (packet pck)
00270 {
00271   if ( pck.priority ){
00272      if(BufferPCKStream.size() == 1){BufferPCKStream.clear();}
00273      else{BufferPCKStream.pop_front();}
00274   }
00275   else {
00276      if(BufferPCK.size() == 1){BufferPCK.clear();}
00277      else{BufferPCK.pop_front();}
00278   };
00279 };
00280
00281 //#########################################################################################
00282 //-------------------------incrementaNNumPCKRetrasmitted-----------------------------------
00283 //#########################################################################################
00284 void AccessPoint::incrementaNNumPCKRetrasmitted (void)
00285 {
00286   NNumPCKRetrasmitted = NNumPCKRetrasmitted + 1;
00287 };
00288
00289 //#########################################################################################
00290 //----------------------------incrementaNNumPCKLost------------------------------------
00291 //#########################################################################################
00292 void AccessPoint::incrementaNNumPCKLost (void)
00293 {
00294   NNumPCKLost = NNumPCKLost + 1;
00295 };
00296
00297 //#########################################################################################
00298 //-----------------------------------getid-----------------------------------------------
00299 //#########################################################################################
00300 int AccessPoint::getid ()
00301 {
00302   return (ID);
00303 };
00304
00305 //#########################################################################################
00306 //-------------------------------searchPackets-------------------------------------------
00307 //#########################################################################################
00308 vector <packet> AccessPoint::searchPackets(packet pck)
00309 {
00310   vector <packet> vectPCK;
00311   int idmess;
00312   packet pack2;
00313
00314   vectPCK.push_back(pck);
00315   if (pck.priority == false){
00316      pack2 = BufferPCK.front();
00317      while (pack2.IDMessage == pck.IDMessage){
00318          vectPCK.push_back(pack2);
00319          BufferPCK.pop_front();
00320          pack2 = BufferPCK.front();
```

```
00321       }
00322    }
00323    else{
00324       pack2 = BufferPCK.front();
00325       while (pack2.IDMessage == pck.IDMessage){
00326           vectPCK.push_back(pack2);
00327           BufferPCK.pop_front();
00328           pack2 = BufferPCK.front();
00329       }
00330    }
00331    return (vectPCK);
00332 };
00333
00334 //#########################################################################################
00335 //--------------------------------numACKpck-----------------------------------------
00336 //#########################################################################################
00337 void AccessPoint::numACKpck()
00338 {
00339         numACK++;
00340 }
00341
00342 //#########################################################################################
00343 //--------------------------------HEv_ObserveAP-------------------------------------
00344 //#########################################################################################
00345 void AccessPoint::HEv_ObserveAP()
00346 {
00347    double PcksInBuffer = NNumPCKStreamBorn + NNumPCKElasticBorn + NNumPckRoamingBorn;
00348
00349    if ((NNumPCKStreamBorn + NNumPCKElasticBorn + NNumPckRoamingBorn ) > 0){
00350        MeanEficiencia->Observe((numACK)/(NNumPCKStreamBorn + NNumPCKElasticBorn));
00351        Eficiencia->Observe((numACK)/(NNumPCKStreamBorn + NNumPCKElasticBorn));
00352        LoadFactor->Observe(PcksInBuffer*double(pckTrasmittingTime)/
00353                            (double(T->Tick()*pow(10,TimeResolution))));
00354        MeanLoadFactor->Observe(PcksInBuffer*double(pckTrasmittingTime)/
00355                                (double(T->Tick()*pow(10,TimeResolution))));
00356        MeanNodiPerAP->Observe(nodiAP);
00357        BufferElasticSize->Observe(BufferPCK.size());
00358        BufferStreamSize->Observe(BufferPCKStream.size());
00359        NumPckBorn->Observe(NNumPCKStreamBorn + NNumPCKElasticBorn);
00360        NumPCKBornStream->Observe(NNumPCKStreamBorn);
00361        NumPCKBornElastic->Observe(NNumPCKElasticBorn);
00362        MeanPckBornElastic->Observe(NNumPCKElasticBorn);
00363        MeanPckBornStream->Observe(NNumPCKStreamBorn);
00364        NumMSGStreamBorn->Observe(NNumMSGStreamBorn);
00365        NumMSGElasticBorn->Observe(NNumMSGElasticBorn);
00366        NumPckRetrasmitted->Observe(NNumPCKRetrasmitted);
00367        NumPckLostTotal->Observe(NNumPCKLost);
00368        NumPckRoamingBorn->Observe(NNumPckRoamingBorn);
00369        MeanBufferElasticSize->Observe(BufferPCK.size());
00370        MeanBufferStreamSize->Observe(BufferPCKStream.size());
00371        MeanPckRitrasmessi->Observe(NNumPCKRetrasmitted);
00372        MeanNumPckRoamingBorn->Observe(NNumPckRoamingBorn);
00373    }
00374    if((NNumPCKStreamBorn + NNumPCKElasticBorn + NNumPckRoamingBorn == 0)&&
00375      ((T->Tick()*pow(10,TimeResolution))>=int(double(tempoSimulazione-ObserveTime))))
00376    {
00377        cout<<"No se ha hecho uso del AP con identificador: "<<ID<<'\n';
00378    }
00379    newEvent (new Ev_ObserveAP (this), ObserveTime );
00380 };
```

## A.2 AccessPoint.h

```
00001 #ifndef __AccessPoint_h__
00002 #define __AccessPoint_h__
00003
00004 #include "Common.h"
00005 #include "probemean.h"
00006 #include "probehistogram.h"
00007 #include "derandom_time.h"
00008
00009 extern long int Poicounter;
00010 class Mappa;
00011 //#############################################################################
00012 //---------------------------- Clase AccessPoint ------------------------------
00013 //#############################################################################
00014 class AccessPoint:public DevTerminal
00015 {
00016 protected:
00017   //Puntero a objetos externos
00018   Mappa * pmappa;
00019   DESystem *System;
00020
00021   //Variables para la identificacion del AccessPoint
00022   int ID;                  //ID del AccessPoint.
00023   unsigned int IDregione; //ID de la region en que se encuentra el AccessPoint.
00024
00025   //Variables DETime
00026   DETime pckTrasmittingTime; //Tiempo en llegar un paquete a su nodo destino.
00027   DETime tempoSimulazione;   //Tiempo de duracion de la simulacion.
00028   DETime HopTime,ObserveTime;
00029
00030   //Variables para la creacion de las distribuciones aleatorias
00031   ParetoRndGen MessageLength;       //Objeto que almacena las muestras Elastic.
00032   ParetoRndGen MessageLengthStream; //Objeto que almacena las muestras Stream.
00033   double meanMessageLength,meanMessageLengthStream; //Variables que almacenan la
00034                                                     //media de cada distribucion.
00035   double location,locationStream,shape,shapeStream; //Variables para dar la forma
00036                                                     //a la distribucion.
00037
00038   //Variables para la generacion de paquetes y mensajes
00039   int maxLunPck,maxMsg4Req,maxMsg4ReqStream;
00040
00041   //Variables para la gestion del trafico generado.
00042   int max_num_tx4pack;//Transmisiones por paquete antes de que sea eliminado.
00043   list < packet >BufferPCK;       //Lista o buffer FIFO de paquetes Elastic.
00044   list < packet >BufferPCKStream; //Lista o buffer FIFO de paquetes Stream.
00045   double BandaTx;     //Ancho de banda del que se dipone (en bits/seg).
00046   int numnodi,nodiAP; //Numero de nodos y de Puntos de Acceso de la simulacion.
00047   DETime Maxttlxpck,MaxttlxpckStream;//Valores del TTL maximos.
00048   unsigned int messaggiocorrente; //En esta variable, que incrementa siempre que el
00049               //Punto de Acceso crea un nuevo mensaje, escribo el ID del nuevo
00050               //mensaje que se crea. Asi, cada nodo esta univocamente definido con
00051               //la dupla IDAccessPointsorgente - IDmessaggio.
00052
00053   //Variables para los Probes y la adquisicion de resultados
00054   double
NNumPCKElasticBorn,NNumPCKStreamBorn,numACK,NNumPCKRetrasmitted,NNumMSGElasticBorn;
00055   double NNumPCKLost,NNumMSGStreamBorn,NNumPckRoamingBorn,TimeResolution;
00056
00057   ProbeSlice *NumMSGStreamBorn;
00058   ProbeSlice *NumMSGElasticBorn;
00059   ProbeSlice *NumMSGPckBorn;
00060   ProbeSlice *BufferElasticSize;
```

```
00061   ProbeSlice *BufferStreamSize;
00062   ProbeSlice *NumPckRetrasmitted;
00063   ProbeSlice *EndToEndDelayElastic;
00064   ProbeSlice *EndToEndDelayStream;
00065   ProbeSlice *EndToEndDelayTotal;
00066   ProbeSlice *NumPckBorn;
00067   ProbeSlice *NumPCKBornElastic;
00068   ProbeSlice *NumPCKBornStream;
00069   ProbeSlice *NumPckLostTotal;
00070   ProbeSlice *LoadFactor;
00071   ProbeSlice *Eficiencia;
00072   ProbeSlice *NumPckRoamingBorn;
00073
00074   ProbeMean *MeanBufferElasticSize;
00075   ProbeMean *MeanBufferStreamSize;
00076   ProbeMean *MeanE2EStream;
00077   ProbeMean *MeanE2EElastic;
00078   ProbeMean *MeanPckRitrasmessi;
00079   ProbeMean *MeanPckBornElastic;
00080   ProbeMean *MeanPckBornStream;
00081   ProbeMean *MeanNodiPerAP;
00082   ProbeMean *MeanLoadFactor;
00083   ProbeMean *MeanEficiencia;
00084   ProbeMean *MeanNumPckRoamingBorn;
00085   ProbeHistogram *EndToEndDelayHistogramStream;
00086   ProbeHistogram *EndToEndDelayHistogramElastic;
00087
00088 public:
00089 //---------------------------------------constructor--------------------------------
00090     AccessPoint (DESystem * _System, int _ID, Mappa * _pmappa)
00091   : DevTerminal (_System, _ID, 0)
00092    , System (_System)
00093    , ID (_ID)
00094    , pmappa (_pmappa)
00095   {
00096   };
00097 //---------------------------------------destructor---------------------------------
00098   AccessPoint::~AccessPoint () {};
00099 //-----------------------------------------setup------------------------------------
00100   static void setup (ParamManager * Param, GlobalProbeManager * Results);
00101 //---------------------------------------initialize---------------------------------
00102   virtual void Initialize (void);
00103 //-----------------------------------------getid------------------------------------
00104   int getid ();
00105 //-------------------------------------Ev_ObserveAP---------------------------------
00106   void HEv_ObserveAP();
00107 //-------------------------------------Msg_Request----------------------------------
00108   void HEv_Msg_Request(int IDnodo);
00109 //-------------------------------------Ev_TryToSend---------------------------------
00110   void HEv_TryToSend();
00111 //-------------------------------------CopyPacket-----------------------------------
00112   void CopyPacket(vector <packet> vect,packet pck,int coda);
00113 //-----------------------------------diminuirebuffer--------------------------------
00114   void diminuirebuffer (packet pck);
00115 //--------------------------incrementaNNumPCKRetrasmitted-------------------------
00116   void incrementaNNumPCKRetrasmitted (void);
00117 //---------------------------------------numACKpck----------------------------------
00118   void numACKpck(void);
00119 //-------------------------------incrementaNNumPCKLost----------------------------
00120   void incrementaNNumPCKLost (void);
00121 //-----------------------------------setposition--------------------------------
00122   void setposition (coordinates pos, coordinates posfinale);
00123 //---------------------------------aggiornaIDregione--------------------------------
00124   void aggiornaIDregione ();
00125 //-------------------------------------searchPackets--------------------------------
00126 vector <packet> searchPackets(packet pck);
00127 };
00128 #endif
```

## A.3 Common.cpp

```
00001 #include "Common.h"
00002 #include "math.h"
00003 #include "Mappa.h"
00004 #define PI 3.1415926
00005 static gsl_rng *theGen;
00006
00007 //#########################################################################################
00008 //-----------------------------------spostadaa-----------------------------------------
00009 //#########################################################################################
00010 coordinates spostadaa (coordinates posizione, coordinates posizionefinale,
00011                        double maxvelocity)
00012 {
00013   coordinates pos;
00014   double angolo;
00015
00016   if (maxvelocity >
00017       sqrt ((posizione.x - posizionefinale.x) * (posizione.x - posizionefinale.x) +
00018             (posizione.y - posizionefinale.y) * (posizione.y - posizionefinale.y)))
00019   {
00020         return (posizionefinale);
00021   }
00022
00023   else {
00024       if (posizionefinale.x != posizione.x) {
00025           angolo =
00026              atan((posizionefinale.y-posizione.y)/(posizionefinale.x-posizione.x));
00027           if (posizionefinale.x < posizione.x) {
00028               angolo = angolo + PI;
00029           }
00030       }
00031       else {
00032           if (posizionefinale.y > posizione.y) {
00033               angolo = PI / 2;
00034           }
00035           else {
00036               angolo = -PI / 2;
00037           }
00038       }
00039       pos.x = (maxvelocity * cos (angolo)) + posizione.x;
00040       pos.y = (maxvelocity * sin (angolo)) + posizione.y;
00041       return (pos);
00042   };
00043 }
00044
00045 //#########################################################################################
00046 //-----------------------------------trovaintersezione-----------------------------------
00047 //#########################################################################################
00048 coordinates trovaintersezione (coordinates v1, coordinates v2, coordinates pos1,
00049                                coordinates pos2)
00050 {
00051 //Devuelve la interseccion de las dos rectas v1-v2 y pos1-pos2.
00052 //Si las rectas son paralelas, devuelve (0,0).
00053 //Si las rectas son coincidentes, devuelve (-1,-1).
00054   coordinates intersezione;
00055   double m1, m2, q1, q2;
00056
00057   //Si son rectas verticales coincidentes.
00058   if (v1.x == v2.x && pos1.x == pos2.x && v1.x == pos1.x) {
00059       intersezione.x = -1;
00060       intersezione.y = -1;
00061       return (intersezione);
```

```
00062   }
00063   //Si son rectas verticales paralelas.
00064   if (v1.x == v2.x && pos1.x == pos2.x && v1.x != pos1.x) {
00065       intersezione.x = 0;
00066       intersezione.y = 0;
00067       return (intersezione);
00068   }
00069   //Si la recta que pasa por v1-v2 es vertical.
00070   if (v1.x == v2.x) {
00071       intersezione.x = v1.x;
00072       m1 = (pos2.y-pos1.y)/(pos2.x-pos1.x);//coeficiente angular de recta pos1-pos2
00073       q1 = pos1.y - ((pos1.x * (pos2.y - pos1.y)) / (pos2.x - pos1.x));
00074       intersezione.y = m1 * v1.x + q1;
00075       return (intersezione);
00076   }
00077   //Si la recta que pasa por pos1-pos2 es vertical.
00078   if (pos1.x == pos2.x) {
00079       intersezione.x = pos1.x;
00080       m2 = (v2.y - v1.y) / (v2.x - v1.x);  //coeficiente angular de recta v1-v2.
00081       q2 = v1.y - ((v1.x * (v2.y - v1.y)) / (v2.x - v1.x));
00082       intersezione.y = m2 * pos1.x + q2;
00083       return (intersezione);
00084   }
00085   //Si son rectas horizontales paralelas.
00086   if (pos1.y == pos2.y && v1.y == v2.y && pos1.y != v1.y) {
00087       intersezione.y = 0;
00088       intersezione.x = 0;
00089       return (intersezione);
00090   }
00091   //Si son rectas horizontales coincidentes.
00092   if (pos1.y == pos2.y && v1.y == v2.y && pos1.y == v1.y) {
00093       intersezione.y = -1;
00094       intersezione.x = -1;
00095       return (intersezione);
00096   }
00097   //Si la recta que pasa por pos1-pos2 es horizontal.
00098   if (pos1.y == pos2.y) {
00099       intersezione.y = pos1.y;
00100       m2 = (v2.y - v1.y) / (v2.x - v1.x);  //coeficiente angular de recta v1-v2.
00101       q2 = v1.y - ((v1.x * (v2.y - v1.y)) / (v2.x - v1.x));
00102       intersezione.x = ((pos1.y - q2) / m2);
00103       return (intersezione);
00104   }
00105   //Si la recta que pasa por v1-v2 es horizontal.
00106   if (v1.y == v2.y) {
00107       intersezione.y = v1.y;
00108       m1 = (pos2.y-pos1.y)/(pos2.x-pos1.x);//coeficiente angular de recta pos1-pos2
00109       q1 = pos1.y - ((pos1.x * (pos2.y - pos1.y)) / (pos2.x - pos1.x));
00110       intersezione.x = ((v1.y - q1) / m1);
00111       return (intersezione);
00112   }
00113
00114   //Caso general de ambas rectas oblicuas respecto a los ejes ordenados.
00115   m1 = (pos2.y - pos1.y) / (pos2.x - pos1.x);//coeficiente angular de pos1-pos2
00116   m2 = (v2.y - v1.y) / (v2.x - v1.x);      //coeficiente angular de recta v1-v2.
00117   q1 = pos1.y - ((pos1.x * (pos2.y - pos1.y)) / (pos2.x - pos1.x));
00118   q2 = v1.y - ((v1.x * (v2.y - v1.y)) / (v2.x - v1.x));
00119
00120   if (m1 != m2) {
00121       intersezione.x = (m1 * q2 - m2 * q1) / (m1 * (m1 - m2)) - q1 / m1;
00122       intersezione.y = (m1 * q2 - m2 * q1) / (m1 - m2);
00123       return (intersezione);
00124   }
00125   else {
00126       if (q1 == q2) {           //Rectas coincidentes.
00127           intersezione.x = -1;
00128           intersezione.y = -1;
00129       }
```

```
00130       else {                    //Rectas paralelas.
00131           intersezione.x = 0;
00132           intersezione.y = 0;
00133       }
00134       return (intersezione);
00135   }
00136 }
00137
00138 //#############################################################################
00139 //-------------------------------trovadistanza-------------------------------
00140 //#############################################################################
00141 double trovadistanza (coordinates pos1, coordinates pos2)
00142 {
00143 //Devuelve la distancia entre dos puntos.
00144   return (sqrt((pos1.y - pos2.y) * (pos1.y - pos2.y) +
00145                (pos1.x - pos2.x) * (pos1.x - pos2.x)));
00146 }
00147
00148 //#############################################################################
00149 //-----------------------------------intersezionecompresa---------------------------
00150 //#############################################################################
00151 bool intersezionecompresa (coordinates intersezione, coordinates v3, coordinates v2)
00152 {
00153 //Funcion que devuelve true si el punto que se debe encontrar sobre la recta v2-v3,
00154 //cae entre v2 y v3.Notar que si cae sobre los puntos v2 o v3, restituye false.
00155   bool ris;
00156   ris = false;
00157
00158   //Caso de recta v1-v2 vertical.
00159   if (v3.x == v2.x) {
00160       if (v2.y < v3.y) {
00161           if (intersezione.y >= v2.y && intersezione.y <= v3.y) {
00162              return (true);
00163           }
00164           else {
00165              return (false);
00166           }
00167       }
00168       else {
00169           if (intersezione.y >= v3.y && intersezione.y <= v2.y) {
00170              return (true);
00171           }
00172           else {
00173              return (false);
00174           }
00175       }
00176   }
00177
00178   //Caso de recta v1-v2 horizontal.
00179   if (v2.y == v3.y) {
00180       if (v2.x < v3.x) {
00181           if (intersezione.x >= v2.x && intersezione.x <= v3.x) {
00182              return (true);
00183           }
00184           else {
00185              return (false);
00186           }
00187       }
00188       else {
00189           if (intersezione.x >= v3.x && intersezione.x <= v2.x) {
00190              return (true);
00191           }
00192           else {
00193              return (false);
00194           };
00195       }
00196   }
00197
```

```
00198   //Caso de recta v1-v2 oblicua.
00199   if (v3.x >= v2.x) {
00200       if (intersezione.x >= v2.x && intersezione.x <= v3.x) {
00201           return (true);
00202       }
00203       else {
00204           return (false);
00205       };
00206   }
00207   else {
00208       if (intersezione.x >= v3.x && intersezione.x <= v2.x) {
00209           return (true);
00210       }
00211       else {
00212           return (false);
00213       };
00214   }
00215 }
00216
00217 //#############################################################################
00218 //----------------------------------puntosuretta-------------------------------
00219 //#############################################################################
00220 bool puntosuretta (coordinates posizione, coordinates v1, coordinates v2)
00221 {
00222 //Funcion que devuelve true solo si el punto 'posizione' se encuentra sobre la recta
00223 //que une v1 con v2.
00224   double xmin, xmax, ymin, ymax;
00225   bool ris;
00226   ris = false;
00227
00228   if (v1.x <= v2.x) {
00229       xmin = v1.x;
00230       xmax = v2.x;
00231   }
00232   else {
00233       xmin = v2.x;
00234       xmax = v1.x;
00235   };
00236   if (v1.y <= v2.y) {
00237       ymin = v1.y;
00238       ymax = v2.y;
00239   }
00240   else {
00241       ymin = v2.y;
00242       ymax = v1.y;
00243   };
00244
00245   //Caso de recta v1-v2 horizontal.
00246   if (v1.y == v2.y) {
00247       if ((posizione.y == v1.y) && (posizione.x >= xmin)
00248          && (posizione.x <= xmax))
00249       {
00250           return (true);
00251       }
00252   };
00253
00254   //Caso de recta v1-v2 vertical.
00255   if (v1.x == v2.x) {
00256       if ((posizione.x == v1.x) && (posizione.y >= ymin)
00257          && (posizione.y <= ymax))
00258       {
00259           return (true);
00260       }
00261   };
00262
00263   //Caso de recta v1-v2 ni horizontal ni vertical.
00264   if ((v1.x != v2.x) && (v1.y != v2.y)) {
00265       if ((posizione.x - v1.x) / (v2.x - v1.x) ==
```

```
00266           (posicion.y - v1.y) / (v2.y - v1.y))
00267       {
00268           return (true);
00269       }
00270   };
00271
00272   return (false);
00273 };
00274
00275 //#############################################################################
00276 //------------------------------distanzapuntoretta-----------------------------
00277 //#############################################################################
00278 double distanzapuntoretta (coordinates vert1, coordinates vert2, coordinates punto)
00279 {
00280 //Funcion que devuelve la distancia del punto 'punto' a la recta que pasa por
00281 //vert1-vert2. Devuelve 0 si el punto se encuentra en la recta.
00282   double angolo, angolo2, angoloretta, distanza;
00283
00284   if (puntosuretta (punto, vert1, vert2)) {
00285       return (0);
00286   }
00287
00288   else {
00289       if (vert1.x != vert2.x) {
00290           angoloretta = atan ((vert1.y - vert2.y) / (vert1.x - vert2.x));
00291           if (vert2.x < vert1.x) {
00292              angoloretta = angoloretta + PI;
00293           }
00294       }
00295       else {
00296           if (vert2.y > vert1.y) {
00297              angoloretta = PI / 2;
00298           }
00299           else {
00300              angoloretta = -PI / 2;
00301           }
00302       }
00303       if (vert1.x != punto.x) {
00304           angoloretta = atan ((vert1.y - punto.y) / (vert1.x - punto.x));
00305           if (punto.x < vert1.x) {
00306              angoloretta = angoloretta + PI;
00307           }
00308       }
00309       else {
00310           if (punto.y > vert1.y) {
00311              angoloretta = PI / 2;
00312           }
00313           else {
00314              angoloretta = -PI / 2;
00315           }
00316       }
00317       //angolo es el angulo entre vert1-vert2 y vert1-punto.
00318       angolo = fabs (angoloretta - angolo2);
00319       distanza = trovadistanza (vert1, punto) * sin (angolo);
00320       return (distanza);
00321   };
00322 };
```

## A.4 Common.h

```
00001 #ifndef __Common_h__
00002 #define __Common_h__
00003 #include "deevent.h"
00004 #include "rndgen.h"
00005 #include "derandom_time.h"
00006 #include "DevTerminal.h"
00007 #include <vector>
00008 #include <list>
00009 #include <string>
00010 #include <cmath>
00011 #include <cstdlib>
00012 #include <ctime>
00013 #include "misc.h"
00014 #include <fstream>
00015 class Mappa;
00016 //#############################################################################
00017 //---------------------------------Clase Common--------------------------------
00018 //#############################################################################
00020 //-----------------------------estructura coordinates--------------------------
00021 typedef struct
00022 {
00023   double x;
00024   double y;
00025 } coordinates;
00026
00027 //-----------------------------estructura pacchetto----------------------------
00028 typedef struct
00029 {
00030   unsigned long int IDpacket;   //ID del paquete.
00031   unsigned int IDMessage;       //ID del mensaje al que pertenece el paquete.
00032   int lunghezza;                //Tamaño del paquete,se crean todos del mismo tamaño
00033   int IDNodosorgente;           //ID del nodo que ha generado el paquete.
00034   int IDNododestinazione;       //ID del nodo destinatario.
00035   int IDRegioneDestinazione;    //ID de la region en la que se encuentra el
00036                                 //destinatario en el momento de la generacion.
00037   coordinates posdestinazione;  //Posicion del destinatario en el momento de la
00038                                 //generacion del paquete.
00039   DETime TTL;                   //Maximo tiempo permitido para la consigna.
00040   DETime TimeBorn;              //Instante de nacimiento.
00041   bool priority;                //Si es true, el paquete es de tipo Stream;
00042                                 //de otro modo es un paquete Elastic
00043   unsigned int numpckinmsg;     //Numeo de paquetes del mensaje con id = IDMessage.
00044   int tx_number;                //Numero de transmisiones efectuadas para el paquete
00045   int max_num_tx;               //Numero maximo de transmisiones permitidas.
00046 } packet;
00047
00048 //---------------------------------spostadaa---------------------------------
00049 coordinates spostadaa (coordinates posizione, coordinates posizionefinale,
00050                        double maxvelocity);
00051 //-------------------------------trovaintersezione---------------------------------
00052 coordinates trovaintersezione (coordinates v1, coordinates v2,
00053                                coordinates pos1,coordinates pos2);
00054 //---------------------------------trovadistanza-----------------------------------
00055 double trovadistanza (coordinates pos1, coordinates pos2);
00056 //-----------------------------intersezionecompresa----------------------------
00057 bool intersezionecompresa (coordinates intersezione,coordinates v3,coordinates v2);
00058 //---------------------------------puntosuretta------------------------------------
00059 bool puntosuretta (coordinates posizione, coordinates v1, coordinates v2);
00060 //-----------------------------distanzapuntoretta------------------------------
00061 double distanzapuntoretta (coordinates vert1,coordinates vert2,coordinates punto);
00062 #endif
```

## A.5 Ev_Msg_Request.h

```
00001 //#############################################################################
00002 //------------------------------Clase Ev_Msg_Request-----------------------------
00003 //#############################################################################
00004 class Ev_Msg_Request : public DEEvent
00005 {
00006
00007  protected:
00008         AccessPoint *theAccessPoint;
00009         int IDnodo;
00010
00011  public:
00012         Ev_Msg_Request(AccessPoint *_theAccessPoint, int _IDnodo)
00013               : DEEvent(),
00014                 theAccessPoint(_theAccessPoint),
00015                 IDnodo(_IDnodo)
00016         {};
00017
00018         virtual const char *Type() const {
00019                         return "Ev_Msg_Request";
00020         };
00021
00022   void Handler() { theAccessPoint->HEv_Msg_Request(IDnodo); };
00023
00024   virtual void print(ostream &os) const {
00025     os << Time << ": " << setw(DBG_EV_WIDTH) << Type() << " - " << *theAccessPoint;
00026   };
00027   virtual ~Ev_Msg_Request (){};
```

## A.6 Ev_Muovinodo.h

```
00001 //#############################################################################
00002 //------------------------------Clase Ev_Muovinodo-------------------------------
00003 //#############################################################################
00004 class Ev_Muovinodo : public DEEvent
00005 {
00006  protected:
00007   Node *theNode;
00008
00009  public:
00010   Ev_Muovinodo( Node *_theNode)
00011     : DEEvent(),
00012       theNode(_theNode)
00013   {};
00014   virtual const char *Type() const {
00015     return "Ev_Muovinodo";
00016   };
00017
00018   void Handler() { theNode->HEv_Muovinodo(); };
00019   virtual void print(ostream &os) const {
00020     os << Time << ": " << setw(DBG_EV_WIDTH) << Type() << " - " << *theNode;
00021   };
00022   virtual ~Ev_Muovinodo (){};
```

## A.7 Ev_ObserveAP.h

```
00001 //#############################################################################
00002 //------------------------------Clase Ev_ObserveAP-------------------------------
00003 //#############################################################################
00004 class Ev_ObserveAP : public DEEvent
00005 {
00006
00007  protected:
00008         AccessPoint *theAccessPoint;
00009
00010  public:
00011         Ev_ObserveAP (AccessPoint *_theAccessPoint)
00012                 : DEEvent(),
00013                 theAccessPoint(_theAccessPoint){};
00014
00015         virtual const char *Type() const {
00016                         return "Ev_ObserveAP";
00017         };
00018
00019   void Handler() { theAccessPoint->HEv_ObserveAP(); };
00020
00021   virtual void print(ostream &os) const {
00022     os << Time << ": " << setw(DBG_EV_WIDTH) << Type() << " - " << *theAccessPoint;
00023   };
00024 virtual ~Ev_ObserveAP (){};
```

## A.8 Ev_ObserveNode.h

```
00001 //#############################################################################
00002 //------------------------------Clase Ev_ObserveNode-----------------------------
00003 //#############################################################################
00004 class Ev_ObserveNode : public DEEvent
00005 {
00006
00007  protected:
00008         Node *theNode;
00009
00010  public:
00011         Ev_ObserveNode (Node *_theNode)
00012                 : DEEvent(),
00013                 theNode(_theNode){};
00014
00015         virtual const char *Type() const {
00016                         return "Ev_ObserveNode";
00017         };
00018
00019   void Handler() { theNode->HEv_ObserveNode(); };
00020
00021   virtual void print(ostream &os) const {
00022     os << Time << ": " << setw(DBG_EV_WIDTH) << Type() << " - " << *theNode;
00023   };
00024   virtual ~Ev_ObserveNode (){};
00025 };
```

## A.9 Ev_ReadPacket.h

```
00001 //#############################################################################
00002 //------------------------------Clase Ev_ReadPacket------------------------------
00003 //#############################################################################
00004 class Ev_ReadPacket: public DEEvent
00005 {
00006  protected:
00007   Node *theNode;
00008
00009  public:
00010   Ev_ReadPacket( Node *_theNode)
00011     : DEEvent(),
00012       theNode(_theNode){};
00013
00014   virtual const char *Type() const {
00015     return "Ev_ReadPacket";
00016   };
00017
00018   void Handler() { theNode->HEv_ReadPacket(); };
00019
00020   virtual void print(ostream &os) const {
00021     os << Time << ": " << setw(DBG_EV_WIDTH) << Type() << " - " << *theNode;
00022   };
00023   virtual ~Ev_ReadPacket (){};
00024 };
```

## A.10 Ev_ReceivePacket.h

```
00001 //#############################################################################
00002 //----------------------------Clase Ev_ReceivePacket---------------------------
00003 //#############################################################################
00004 class Ev_ReceivePacket : public DEEvent
00005 {
00006  protected:
00007   Node *theNode;
00008   packet pck;
00009   AccessPoint *pAPoint;
00010
00011  public:
00012   Ev_ReceivePacket( Node *_theNode,packet _pck,AccessPoint *_pAPoint)
00013     : DEEvent(),
00014       theNode(_theNode),
00015       pck (_pck),
00016       pAPoint (_pAPoint)
00017   {};
00018
00019   virtual const char *Type() const {
00020     return "Ev_ReceivePacket";
00021   };
00022
00023   void Handler() { theNode->HEv_ReceivePacket(pck,pAPoint); };
00024
00025   virtual void print(ostream &os) const {
00026     os << Time << ": " << setw(DBG_EV_WIDTH) << Type() << " - " << *theNode;
00027   };
00028   virtual ~Ev_ReceivePacket (){};
00029 };
```

## A.11 Ev_TryToSend.h

```
00001 //#############################################################################
00002 //-------------------------------Clase Ev_TryToSend-----------------------------
00003 //#############################################################################
00004 class Ev_TryToSend : public DEEvent
00005 {
00006  protected:
00007   AccessPoint *theAccessPoint;
00008
00009  public:
00010   Ev_TryToSend( AccessPoint *_theAccessPoint)
00011     : DEEvent(),
00012       theAccessPoint(_theAccessPoint)
00013   {};
00014
00015   virtual const char *Type() const {
00016     return "Ev_TryToSend";
00017   };
00018
00019   void Handler() { theAccessPoint->HEv_TryToSend(); };
00020
00021   virtual void print(ostream &os) const {
00022     os << Time << ": " << setw(DBG_EV_WIDTH) << Type() << " - " << *theAccessPoint;
00023   };
00024     virtual ~Ev_TryToSend (){};
00025 };
```

## A.12 Mappa.cpp

```
00001 #include "Mappa.h"
00002 #include "Ostacolo.h"
00003 #include "Common.h"
00004 #include "Node.h"
00005 #include "AccessPoint.h"
00006 #include "Regione.h"
00007 #include "Ev_TryToSend.h"
00008 #include "Ev_Msg_Request.h"
00009 #include "Ev_ReceivePacket.h"
00010 #include <fstream>
00011 #include <sstream>
00012
00013 #include <list>
00014 #define PI 3.1415926
00015 #define MAX_LINE_LENGTH 256
00016 #define BYTE 8
00017
00018 //#############################################################################
00019 //-----------------------------------SETUP-------------------------------------
00020 //#############################################################################
00021 void Mappa::setup (ParamManager * Param, GlobalProbeManager * Results)
00022 {
00023   Param->addClass ("NwMap", "Network region map");
00024   Param->addParameter ("NwMap",new DoubleParameter ("limitX",
00025                        "ancho del area de trabajo(coordenada x)",
00026                        "", "(0,inf)"));
00027   Param->addParameter ("NwMap",new DoubleParameter ("limitY",
```

```
00028                       "alto del area de trabajo (coordenada y)",
00029                       "", "(0,inf)"));
00030   Param->addParameter ("NwMap",new IntParameter ("numnodi",
00031                       "numero de nodos en la red","", "(0,inf)"));
00032   Param->addParameter ("NwMap",new DoubleParameter ("rangevertex",
00033                       "rango en el que se busca una posicion intermedia"
00034                       " cercana a los vertices del obstaculo","", "(0,inf)"));
00035   Param->addParameter ("NwMap",new DoubleParameter ("range_tra_poi",
00036                       "rango en el que se busca un poi cercano al ultimo alcanzado",
00037                       "", "(0,inf)"));
00038   Param->addParameter ("NwMap",new DoubleParameter ("BandaTx",
00039                       "Banda de transmision expresada en Hz","", "(0,inf)"));
00040   Param->addParameter ("NwMap",new DoubleParameter ("DistanzaHisteresi",
00041                       "distancia que se necesita para cambiar de region al nodo",
00042                       "", "(0,inf)"));
00043 //---------------------------------------Obstaculos-------------------------------
00044   Param->addParameter ("NwMap",new IntParameter("numostacoli",
00045                       "numero de obstaculos","", "(0,inf)"));
00046   Param->addParameter ("NwMap",new StringParameter ("archiviOstacoli",
00047                       "fichero para los vertices de los obstaculos", "","(0,inf)"));
00048   Param->addParameter ("NwMap",new IntParameter ("numpoi",
00049                       "numero de Puntos de Interes", "","(0,inf)"));
00050   Param->addParameter ("NwMap",new StringParameter ("archiviPoi",
00051                       "fichero para las coordenadas de los Poi", "","(0,inf)"));
00052 //---------------------------------------Regiones---------------------------------
00053   Param->addParameter ("NwMap",new IntParameter ("numzone_col",
00054                       "numero de columnas de regiones que dividen al area",
00055                                           "", "(0,inf)"));
00056   Param->addParameter ("NwMap",new IntParameter ("numzone_row",
00057                       "numero de filas en que se divide el area","", "(0,inf)"));
00058   Param->addParameter ("NwMap",new IntParameter ("numAP",
00059                       "numero de Puntos de Acceso en la red","", "(0,inf)"));
00060 };
00061
00062 //############################################################################
00063 //-----------------------------------INITIALIZE-----------------------------------
00064 //############################################################################
00065 void Mappa::Initialize (void)
00066 {
00067   unsigned int IDregione;
00068   int numzonecol, numzonerow, numeroregioni;
00069
00070   lunghezza = get < DoubleParameter, double >(Param, "limitX", "NwMap", "");
00071   altezza = get < DoubleParameter, double >(Param, "limitY", "NwMap", "");
00072   numzonecol = get < IntParameter, int >(Param, "numzone_col", "NwMap", "");
00073   numzonerow = get < IntParameter, int >(Param, "numzone_row", "NwMap", "");
00074   numnodi = get < IntParameter, int >(Param, "numnodi", "NwMap", "");
00075   FER = get < DoubleParameter, double >(Param, "FER", "APoint", "");
00076   distmaxpoi = get < DoubleParameter, double >(Param, "range_tra_poi", "NwMap", "");
00077   distHist = get < DoubleParameter,double >(Param,"DistanzaHisteresi", "NwMap", "");
00078   BandaTx = get < DoubleParameter, double >(Param, "BandaTx", "NwMap", "");
00079   maxLunPck = get < IntParameter, int >(Param, "maxLunPck", "APoint", "");
00080   max_num_tx4pack = get < IntParameter, int >(Param,"max_num_tx4pack","APoint", "");
00081   Maxttlxpck = get < DETimeParameter, DETime > (Param, "maxttlxpck", "APoint", "");
00082   MaxttlxpckStream = get < DETimeParameter, DETime >
00083                          (Param,"maxttlxpckStream", "APoint", "");
00084   rangeposintermedia = get <DoubleParameter,double>(Param,"rangevertex","NwMap","");
00085   numostacoli = get < IntParameter, int >(Param, "numostacoli", "NwMap", "");
00086   string archiviOstacoli = get < StringParameter, string >
00087                                (Param,"archiviOstacoli","NwMap","");
00088   numpoi = get < IntParameter, int >(Param, "numpoi", "NwMap", "");
00089   string archiviPoi = get<StringParameter,string>(Param, "archiviPoi", "NwMap", "");
00090
00091   pckTrasmittingTime = DETime((BYTE*maxLunPck)/BandaTx);
00092   doublpckTrasmittingTime = DETime(double(pckTrasmittingTime)*2);
00093
00094   Ostacolo *tempO;
00095   string name;
```

```
00096   coordinates tempPoi;
00097   int i, j, numostacoli;
00098
00099   coordinates vert1, vert2, vert3, vert4;
00100   char *strpos;
00101   Regione *tempRegione;
00102   AccessPoint *tempAccessPoint;
00103   regioni.clear ();
00104
00105   //Divido el area en regiones de forma rectangular.
00106   //La numeracion de las regiones va de arriba-izquierda a abajo-derecha:
00107   //     0       1       2
00108   //     3       4       5
00109   //     6       7       8
00110   numeroregioni = numzonecol * numzonerow;
00111   for (j = 0; j < numzonerow; j++) {
00112     for (i = 0; i < numzonecol; i++) {
00113       vert1.x = (i * largregioni);
00114       vert1.y = ((j + 1) * altregioni);
00115       vert3.x = (vert1.x) + largregioni;
00116       vert3.y = (vert1.y) - altregioni;
00117       IDregione = (j * numzonecol) + i;
00118       tempRegione = new Regione (vert1, vert3, IDregione);
00119       regioni.push_back (tempRegione);
00120     }
00121   }
00122
00123   //-------------------Creacion de los obstaculos-----------------------------------
00124   ifstream* InputFile = new ifstream(archiviOstacoli.c_str(), ios::in);
00125   if ( InputFile->fail() ) {
00126     cout << "file \"" << archiviOstacoli << "\" does not exists.\n";
00127     abort();
00128   }
00129
00130   j = 1;
00131
00132   string buf, testbuf, trueline;
00133   char line[MAX_LINE_LENGTH];
00134   double v1x,v1y,v2x,v2y,v3x,v3y,v4x,v4y;
00135
00136   while( !InputFile->eof() ) {
00137     testbuf = "";
00138     do {
00139        InputFile->getline(line, MAX_LINE_LENGTH);
00140        buf = string(line);
00141        buf = buf.substr(0, buf.find('#'));
00142        trueline = buf;
00143        for(i = buf.find_first_of(" \t");i != buf.npos;i =buf.find_first_of(" \t",i))
00144          buf.erase(i, 1);
00145        testbuf.append(buf, 0, buf.find("\\"));
00146     } while(!InputFile->eof() && buf[buf.size()-1] == '\\');
00147
00148     if( testbuf != "" ) {
00149       istringstream m_stream(trueline);
00150       m_stream >> v1x >> v1y >> v2x >> v2y >> v3x >> v3y >> v4x >> v4y  ;
00151       vert1.x = v1x;
00152         vert1.y = v1y;
00153       vert2.x = v2x;
00154         vert2.y = v2y;
00155       vert3.x = v3x;
00156         vert3.y = v3y;
00157       vert4.x = v4x;
00158         vert4.y = v4y;
00159         tempO = new Ostacolo (vert1,vert2,vert3,vert4);
00160         tempO -> assegnaID ((j-1));
00161         tempO -> assegnarangevertex (rangeposintermedia);
00162         ostacoli.push_back (tempO);
00163         j = j + 1;
```

```
00164     }
00165   }
00166   cout<<"Introducidos "<<(j-1)<<" obstaculos"<<'\n';
00167 
00168 
00169   //-------------------Creacion de los Puntos de Interes----------------------------
00170   InputFile = new ifstream(archiviPoi.c_str(), ios::in);
00171   if ( InputFile->fail() ) {
00172     cout << "file \"" << archiviPoi << "\" does not exists.\n";
00173     abort();
00174   }
00175   j = 1;
00176   double px,py;
00177   while( !InputFile->eof() ) {
00178     testbuf = "";
00179     do {
00180       InputFile->getline(line, MAX_LINE_LENGTH);
00181       buf = string(line);
00182       buf = buf.substr(0, buf.find('#'));
00183       trueline = buf;
00184       for (i = buf.find_first_of(" \t");i != buf.npos;i =buf.find_first_of(" \t",i))
00185         buf.erase(i, 1);
00186       testbuf.append(buf, 0, buf.find("\\"));
00187     } while(!InputFile->eof() && buf[buf.size()-1] == '\\');
00188 
00189     if( testbuf != "" ) {
00190       istringstream m_stream(trueline);
00191       m_stream >> px >> py;
00192       tempPoi.x = px;
00193         tempPoi.y = py;
00194         poi.push_back (tempPoi);
00195         j++;
00196     }
00197   }
00198 
00199    cout<<"Introducidos "<<(j-1)<<" Puntos de Interes"<<'\n';
00200 
00201    vector <int> vectinit;
00202    vectinit.clear();
00203    vectinit.push_back(-1);
00204    for (int z = 0 ; z < numnodi ; z++){
00205         ostacoliadiacenti.push_back(vectinit);
00206    };
00207 
00208    ostacoliadiacenti.clear();
00209    creaadiacenzeostacoli();
00210 };
00211 
00212 //###########################################################################
00213 //----------------------------------getpoicoord--------------------------------
00214 //###########################################################################
00215 coordinates Mappa::getpoicoord (int indice)
00216 {
00217 
00218 return (poi[indice]);
00219 
00220 };
00221 
00222 //###########################################################################
00223 //---------------------------posizioneraggiungileblobal-----------------------
00224 //###########################################################################
00225 bool
00226 Mappa::posizioneraggiungileblobal (coordinates posizione)
00227 {
00228 //Esta funcion sirve para saber si un punto de coordinadas 'posizione' cae o no
00229 //dentro de un obstaculo fisico en el interior del mapa. Devuelve TRUE si el punto
00230 //esta fuera del obstaculo.
00231   bool risultato;
```

```
00232   risultato = true;
00233
00234   //Primero controla que posizione caiga dentro del area de trabajo
00235   if (posizione.x < (0) || posizione.x > (lunghezza) || posizione.y < (0)
00236       || posizione.y > (altezza)) {
00237     return (false);
00238   };
00239
00240   //Despues controla si cae dentro de algun obstaculo
00241   for (int i = 0; i < ostacoli.size (); i++) {
00242     risultato = (risultato) && (ostacoli[i]->puntoraggiungibile (posizione));
00243     if(risultato == false){
00244       return (risultato);
00245     }
00246   }
00247   return (risultato);
00248 };
00249
00250 //########################################################################################
00251 //---------------------------traiettoriafattibileglobal-----------------------------
00252 //########################################################################################
00253 bool Mappa::traiettoriafattibileglobal (coordinates posizionepartenza,
00254                                         coordinates posizionearrivo){
00255 //Esta funcion sirve para determinar si la linea recta que une 'posizionepartenza' y
00256 //'posizionearrivo' se cruza con cualquier obstaculo fisico del interior del mapa.
00257 //Devuelve TRUE si no corta ningun obstaculo, de otro modo FALSE.
00258   bool risultato;
00259   risultato = true;
00260
00261   for (int i = 0; i < ostacoli.size (); i++) {
00262     risultato = risultato
00263       && (ostacoli[i]->
00264           traiettoriafattibile (posizionepartenza, posizionearrivo));
00265   };
00266
00267   return (risultato);
00268 };
00269
00270 //########################################################################################
00271 //--------------------------------inseriscinodo-----------------------------------
00272 //########################################################################################
00273 void Mappa::inseriscinodo (Node * pnodo)
00274 {
00275 //Funcion que introduce el nodo apuntado por pnodo en el elenco de nodos.
00276   elenconodi.push_back (pnodo);
00277 };
00278
00279 //########################################################################################
00280 //-------------------------------trovadestintermedia--------------------------------
00281 //########################################################################################
00282 coordinates Mappa::trovadestintermedia (coordinates posizione,coordinates
00283                                         posizionefinale,int Idostacoloaggiramento,
00284                                         int idverticeaggiramento, Node* nodo){
00285 //Esta funcion sirve para determinar una posicion intermedia concorde al modelo de
00286 //movilidad usado.Se invoca cuando un nodo debe desplazarse hacia una destinacion
00287 //fisica que no es alcanzable en linea recta,pues hay obstaculos en medio.
00288   int i, indice, indice1, cont, idostacolovicino,numostacoliadiacenti;
00289   int indiceok,verticeindexok;
00290   coordinates vertice, nuovadest,verticetest,verticeok,verticeprecedente;
00291   double dmax, dist, distmin;
00292   unsigned int numeroostacolo;
00293   bool pippo;
00294
00295   distmin = lunghezza + altezza;
00296   dmax = sqrt((lunghezza * lunghezza) + (altezza * altezza));
00297   indice = 0;
00298   verticeprecedente.x =
00299             ostacoli[Idostacoloaggiramento]->getvindice(idverticeaggiramento).x;
```

```
00300   verticeprecedente.y =
00301             ostacoli[Idostacoloaggiramento]->getvindice(idverticeaggiramento).y;
00302
00303   //Primero se encuentra el obstaculo mas vecino al nodo y se bordea buscando
00304   //avecinarno a la destinacion
00305   for (i = 0; i < ostacoli.size (); i++) {
00306
00307     vertice = ostacoli[i]->getv1 ();    //vertice v1
00308     dist = trovadistanza (vertice, posizione);
00309     if((dist<dmax)&&(!(ostacoli[i]-
>traiettoriafattibile(posizione,posizionefinale)))
00310       &&((ostacoli[i]->getvindice (1)).x > rangeposintermedia)
00311       &&((ostacoli[i]->getvindice (1)).y > rangeposintermedia)
00312       &&((ostacoli[i]->getvindice (1)).x < (lunghezza -rangeposintermedia))
00313       &&((ostacoli[i]->getvindice (1)).y < (altezza - rangeposintermedia)))
00314     {
00315          dmax = dist;
00316          numeroostacolo = ostacoli[i]->getID ();
00317          indice = i;
00318     };
00319
00320     vertice = ostacoli[i]->getv2 ();    //vertice v2
00321     dist = trovadistanza (vertice, posizione);
00322     if((dist<dmax)&&(!(ostacoli[i]-
>traiettoriafattibile(posizione,posizionefinale)))
00323       &&((ostacoli[i]->getvindice (2)).x > rangeposintermedia)
00324       &&((ostacoli[i]->getvindice (2)).y > rangeposintermedia)
00325       &&((ostacoli[i]->getvindice (2)).x < (lunghezza -rangeposintermedia))
00326       &&((ostacoli[i]->getvindice (2)).y < (altezza - rangeposintermedia)))
00327     {
00328          dmax = dist;
00329          numeroostacolo = ostacoli[i]->getID ();
00330          indice = i;
00331     };
00332
00333     vertice = ostacoli[i]->getv3 ();    //vertice v3
00334     dist = trovadistanza (vertice, posizione);
00335     if((dist<dmax)&&(!(ostacoli[i]-
>traiettoriafattibile(posizione,posizionefinale)))
00336       &&((ostacoli[i]->getvindice (3)).x > rangeposintermedia)
00337       &&((ostacoli[i]->getvindice (3)).y > rangeposintermedia)
00338       &&((ostacoli[i]->getvindice (3)).x < (lunghezza -rangeposintermedia))
00339       &&((ostacoli[i]->getvindice (3)).y < (altezza - rangeposintermedia)))
00340     {
00341          dmax = dist;
00342          numeroostacolo = ostacoli[i]->getID ();
00343          indice = i;
00344     };
00345
00346     vertice = ostacoli[i]->getv4 ();    //vertice v4
00347     dist = trovadistanza (vertice, posizione);
00348     if((dist<dmax)&&(!(ostacoli[i]-
>traiettoriafattibile(posizione,posizionefinale)))
00349       &&((ostacoli[i]->getvindice (4)).x > rangeposintermedia)
00350       &&((ostacoli[i]->getvindice (4)).y > rangeposintermedia)
00351       &&((ostacoli[i]->getvindice (4)).x < (lunghezza -rangeposintermedia))
00352       &&((ostacoli[i]->getvindice (4)).y < (altezza - rangeposintermedia)))
00353     {
00354          dmax = dist;
00355          numeroostacolo = ostacoli[i]->getID ();
00356          indice = i;
00357     };
00358   }
00359
00360   //Se cual es el obstaculo mas cercano al nodo que esta en medio de su trayectoria
00361   numostacoliadiacenti = 0;
00362   idostacolovicino = ostacoli[indice]->getID();
00363   if( ostacoliadiacenti[idostacolovicino].size() > 1 ){
```

```
00364       numostacoliadiacenti = ostacoliadiacenti[idostacolovicino].size() - 1;
00365       for(vector<coordinates>::iterator itcoord = verticiostacoli[indice].begin() ;
00366           itcoord != verticiostacoli[indice].end();itcoord++)
00367           {
00368              verticetest.x = (*itcoord).x;
00369              verticetest.y = (*itcoord).y;
00370              if((verticetest.x != -1) && (verticetest.y != -1) &&
00371                ((verticetest.x != verticeprecedente.x) ||
00372                ((verticetest.y != verticeprecedente.y))) && (verticetest.x !=0))
00373              {
00374                   if(verticevisibileglobal(posizione,verticetest)&&
00375                     (verticetest.x > rangeposintermedia)&&
00376                     (verticetest.y > rangeposintermedia)&&
00377                     (verticetest.x < (lunghezza -rangeposintermedia))&&
00378                     (verticetest.y < (altezza - rangeposintermedia)))
00379                   {
00380                         dist = trovadistanza(posizionefinale,verticetest);
00381                         if(dist <= distmin){
00382                            distmin = dist;
00383                            verticeok.x = verticetest.x;
00384                            verticeok.y = verticetest.y;
00385                         }
00386                   }
00387              }
00388           };
00389
00390           // En verticeok estan las coordenadas del vertice
00391           //elegido como destinacion intermedia
00392           cont = ostacoli[indice]->determinaidvertice(verticeok);
00393           if(cont != (-1)){
00394              indiceok = indice;
00395              verticeindexok = cont;
00396              pippo = true;
00397              while(pippo){
00398                 nuovadest = ostacoli[indice]->puntovicinovertice (verticeindexok);
00399                 if (posizioneraggiungileblobal(nuovadest)) {
00400                    pippo = false;
00401                 };
00402              }
00403           }
00404           else{
00405              vector<int>::iterator itindici;
00406              for(itindici = ostacoliadiacenti[indice].begin() ;
00407                  itindici != ostacoliadiacenti[indice].end() ; itindici++)
00408              {
00409                    if(((*itindici) != (-1)) && ((*itindici) != indice)){
00410                        cont = ostacoli[(*itindici)]->determinaidvertice(verticeok);
00411                        if(cont != (-1)){
00412                            indiceok = (*itindici);
00413                            verticeindexok = cont;
00414                            pippo = true;
00415                            while(pippo){
00416                               nuovadest =
00417                                ostacoli[indiceok]->puntovicinovertice(verticeindexok);
00418                               if (posizioneraggiungileblobal(nuovadest)) {
00419                                   pippo = false;
00420                               };
00421                            }
00422                        }
00423                    }
00424              }
00425           }
00426           nodo->setAggiramentoostacolo(indiceok,verticeindexok);
00427           return(nuovadest);
00428    }
00429
00430
00431    for (cont = 1; cont < 5; cont++) {
```

```
00432     if ((ostacoli[indice]->verticevisibile (posizione, cont))&&
00433        ((ostacoli[indice]->getvindice (cont)).x > rangeposintermedia)&&
00434        ((ostacoli[indice]->getvindice (cont)).y > rangeposintermedia)&&
00435        ((ostacoli[indice]->getvindice (cont)).x < (lunghezza -rangeposintermedia))&&
00436        ((ostacoli[indice]->getvindice (cont)).y < (altezza - rangeposintermedia)))
00437     {
00438           dist = trovadistanza(posizionefinale,ostacoli[indice]->getvindice (cont));
00439           if (dist <= distmin) {
00440              indice1 = cont;
00441              distmin = dist;
00442           }
00443     }
00444   }
00445
00446   //A la salida de este 'for' tengo el indice del vertice del obstaculo que es,
00447   // entre los vertices visibles desde 'posizione', el mas vecino a la destinacion.
00448   pippo = true;
00449   while(pippo){
00450      nuovadest = ostacoli[indice]->puntovicinovertice (indice1);
00451      if (posizioneraggiungileblobal(nuovadest)) {
00452         pippo = false;
00453      };
00454   }
00455   nodo->setAggiramentoostacolo(indice,indice1);
00456   return (nuovadest);
00457 };
00458
00459 //###########################################################################
00460 //-----------------------------trovaregioneappartenenza----------------------------
00461 //###########################################################################
00462 int Mappa::trovaregioneappartenenza (coordinates position)
00463 {
00464 //Esta funcion devuelve el ID de la region que contiene la posicion 'position'
00465   int index = -1;
00466   int i;
00467
00468   for (i = 0; i < regioni.size (); i++) {
00469      if (regioni[i]->posappartenente (position)) {
00470         index = i;
00471         return(regioni[i]->getid());
00472      };
00473   };
00474
00475   if (index >= 0) {
00476     return (regioni[i]->getid());
00477   }
00478   else {
00479     return (-1);
00480   }
00481 };
00482
00483 //###########################################################################
00484 //-------------------------------trovaposizionenodo-----------------------------
00485 //###########################################################################
00486 coordinates Mappa::trovaposizionenodo (int Iddest)
00487 {
00488 //Funcion que devuelve la posicion en el instante actual del nodo con ID = Iddest
00489   coordinates pos;
00490   bool trovato;
00491   trovato = false;
00492
00493   for (vector < Node * >::iterator itnodi = elenconodi.begin ();
00494        itnodi != elenconodi.end (); itnodi++) {
00495     if ((*itnodi)->getid () == Iddest) {
00496       pos = (*itnodi)->getposition ();
00497       trovato = true;
00498       return(pos);
00499     }
```

```
00500   }
00501
00502   if(trovato == true){
00503      return (pos);
00504   }
00505   else{
00506      cout<<"Error,el nodo "<<Iddest<<" no se encuentra en la clase Mappa"<<'\n';
00507      pos.x=0;
00508      pos.y=0;
00509      return(pos);
00510   };
00511 };
00512
00513 //#########################################################################################
00514 //------------------------------inserisciAccessPoint-----------------------------
00515 //#########################################################################################
00516 void Mappa::inserisciAccessPoint (AccessPoint * tempAP)
00517 {
00518   elencoAccessPoint.push_back(tempAP);
00519 };
00520
00521 //#########################################################################################
00522 //-----------------------------------getNumNodi-------------------------------------
00523 //#########################################################################################
00524 int Mappa::getNumNodi()
00525 {
00526 return(elenconodi.size());
00527 };
00528
00529 //#########################################################################################
00530 //----------------------------creaadiacenzeostacoli-----------------------------
00531 //#########################################################################################
00532 void Mappa::creaadiacenzeostacoli()
00533 {
00534   if(ostacoli.size() == 0){return;};
00535   vector <int> vectinit;
00536   vectinit.clear();
00537   verticiostacoli.clear();
00538   vectinit.push_back(-1);
00539   int idost,idostad;
00540   coordinates vertex;
00541   vector < coordinates> pippo;
00542   vertex.x=-1;
00543   vertex.y=-1;
00544   pippo.push_back(vertex);
00545
00546   for (int z = 0 ; z < ostacoli.size() ; z++){
00547         ostacoliadiacenti.push_back(vectinit);
00548         verticiostacoli.push_back(pippo);
00549   };
00550
00551   for(int x = 0 ; x < ostacoli.size() ; x++){
00552      idost = ostacoli[x]->getID();
00553      for(int y = 0 ; y < ostacoli.size() ; y++){
00554         if(x != y){
00555            idostad = ostacoli[y]->getID();
00556            if((comparecoord(ostacoli[x]->getv1(),ostacoli[y]->getv1()))||
00557               (comparecoord(ostacoli[x]->getv1(),ostacoli[y]->getv2()))||
00558               (comparecoord(ostacoli[x]->getv1(),ostacoli[y]->getv3()))||
00559               (comparecoord(ostacoli[x]->getv1(),ostacoli[y]->getv4()))||
00560               (comparecoord(ostacoli[x]->getv2(),ostacoli[y]->getv1()))||
00561               (comparecoord(ostacoli[x]->getv2(),ostacoli[y]->getv2()))||
00562               (comparecoord(ostacoli[x]->getv2(),ostacoli[y]->getv3()))||
00563               (comparecoord(ostacoli[x]->getv2(),ostacoli[y]->getv4()))||
00564               (comparecoord(ostacoli[x]->getv3(),ostacoli[y]->getv1()))||
00565               (comparecoord(ostacoli[x]->getv3(),ostacoli[y]->getv2()))||
00566               (comparecoord(ostacoli[x]->getv3(),ostacoli[y]->getv3()))||
00567               (comparecoord(ostacoli[x]->getv3(),ostacoli[y]->getv4()))||
```

```
00568                (comparecoord(ostacoli[x]->getv4(),ostacoli[y]->getv1()))||
00569                (comparecoord(ostacoli[x]->getv4(),ostacoli[y]->getv2()))||
00570                (comparecoord(ostacoli[x]->getv4(),ostacoli[y]->getv3()))||
00571                (comparecoord(ostacoli[x]->getv4(),ostacoli[y]->getv4())))
00572             {
00573                   ostacoliadiacenti[idost].push_back(idostad);
00574             }
00575          }
00576       }
00577    }
00578
00579    for(int w = 0 ; w < ostacoli.size() ; w++){
00580       if(ostacoliadiacenti[w].size() == 1){
00581          vectinit.clear();
00582          verticiostacoli[w].push_back(ostacoli[w]->getv1());
00583          verticiostacoli[w].push_back(ostacoli[w]->getv2());
00584          verticiostacoli[w].push_back(ostacoli[w]->getv3());
00585          verticiostacoli[w].push_back(ostacoli[w]->getv4());
00586       }
00587       else{
00588          verticiostacoli[w].push_back(ostacoli[w]->getv1());
00589          verticiostacoli[w].push_back(ostacoli[w]->getv2());
00590          verticiostacoli[w].push_back(ostacoli[w]->getv3());
00591          verticiostacoli[w].push_back(ostacoli[w]->getv4());
00592          idostad = -1;
00593          for(int r = 0 ; r < ostacoliadiacenti[w].size() ; r++){
00594             idostad = ostacoliadiacenti[w].at(r);
00595             if( idostad != (-1) ){
00596                if(!((comparecoord(ostacoli[w]->getv1(),ostacoli[idostad]->getv1()))
00597                   ||(comparecoord(ostacoli[w]->getv2(),ostacoli[idostad]->getv1()))
00598                   ||(comparecoord(ostacoli[w]->getv3(),ostacoli[idostad]->getv1()))
00599                   ||(comparecoord(ostacoli[w]->getv4(),ostacoli[idostad]->getv1()))))
00600                {
00601                      verticiostacoli[w].push_back(ostacoli[idostad]->getv1());
00602                };
00603                if(!((comparecoord(ostacoli[w]->getv1(),ostacoli[idostad]->getv2()))
00604                   ||(comparecoord(ostacoli[w]->getv2(),ostacoli[idostad]->getv2()))
00605                   ||(comparecoord(ostacoli[w]->getv3(),ostacoli[idostad]->getv2()))
00606                   ||(comparecoord(ostacoli[w]->getv4(),ostacoli[idostad]->getv2()))))
00607                {
00608                      verticiostacoli[w].push_back(ostacoli[idostad]->getv2());
00609                };
00610                if(!((comparecoord(ostacoli[w]->getv1(),ostacoli[idostad]->getv3()))
00611                   ||(comparecoord(ostacoli[w]->getv2(),ostacoli[idostad]->getv3()))
00612                   ||(comparecoord(ostacoli[w]->getv3(),ostacoli[idostad]->getv3()))
00613                   ||(comparecoord(ostacoli[w]->getv4(),ostacoli[idostad]->getv3()))))
00614                {
00615                      verticiostacoli[w].push_back(ostacoli[idostad]->getv3());
00616                };
00617                if(!((comparecoord(ostacoli[w]->getv1(),ostacoli[idostad]->getv4()))
00618                   ||(comparecoord(ostacoli[w]->getv2(),ostacoli[idostad]->getv4()))
00619                   ||(comparecoord(ostacoli[w]->getv3(),ostacoli[idostad]->getv4()))
00620                   ||(comparecoord(ostacoli[w]->getv4(),ostacoli[idostad]->getv4()))))
00621                {
00622                  verticiostacoli[w].push_back(ostacoli[idostad]->getv4());
00623                };
00624             }
00625          }
00626       }
00627    }
00628 };
00629
00630 //#########################################################################
00631 //---------------------------------verticeesterno--------------------------
00632 //#########################################################################
00633 bool Mappa::verticeesterno(int idostacolo, coordinates vertice)
00634 {
00635    for(vector<Ostacolo*>::iterator itostacolo = ostacoli.begin();
```

```
00636       itostacolo != ostacoli.end() ; itostacolo++)
00637   {
00638      if((*itostacolo)->getID() != idostacolo){
00639         if( (*itostacolo)->puntoraggiungibile(vertice) == false ){return(false);};
00640      }
00641   }
00642   return(true);
00643 };
00644
00645 //#############################################################################
00646 //-----------------------------verticevisibileglobal---------------------------------
00647 //#############################################################################
00648 bool Mappa::verticevisibileglobal(coordinates posizione,coordinates verticetest)
00649 {
00650   int idv;
00651   for(vector<Ostacolo*>::iterator itostacolo = ostacoli.begin();
00652       itostacolo != ostacoli.end() ; itostacolo++)
00653   {
00654         idv = (*itostacolo)->determinaidvertice(verticetest);
00655         if(idv != (-1)){
00656                 return((*itostacolo)->verticevisibile(posizione,idv));
00657         }
00658   }
00659 };
00660
00661 //#############################################################################
00662 //-----------------------------spediscipacchetto---------------------------------
00663 //#############################################################################
00664 void Mappa::spediscipacchetto (AccessPoint * pAPoint,packet pck)
00665 {
00666   int IdRegDest,IDRegInic;
00667   Node *tempNode;
00668   DETime tempoIfNoSend = DETime(double(pckTrasmittingTime)/10);
00669   vector < packet > vectorpack;
00670
00671   if ((pck.tx_number < pck.max_num_tx) && (((*T) - pck.TimeBorn) < pck.TTL)) {
00672     for (vector < Node*>::iterator itnodo = elenconodi.begin ();
00673        itnodo != elenconodi.end (); itnodo++)
00674     {
00675       if ( ((*itnodo)->getid()) == pck.IDNododestinazione ) {
00676         tempNode = *itnodo;
00677         IdRegDest = (*itnodo)->getIdregione();
00678         IDRegInic = pck.IDRegioneDestinazione;
00679         if (IdRegDest != IDRegInic){ //el nodo se encuentra en una region distinta
00680            for (vector < Regione* >::iterator itregione = regioni.begin ();
00681                 itregione != regioni.end (); itregione++)
00682            {
00683               if ( (*itregione)->getid() == IDRegInic ) {
00684                  if((*itregione)->distanzaconfine ((*itnodo)->getposition ())
00685                     > distHist)
00686                  {
00687                     vectorpack=pAPoint->searchPackets(pck);//coge los paq del msge
00688                     ChangePckToAP(pAPoint,vectorpack,pck,IdRegDest);
00689                     newEvent (new Ev_TryToSend(pAPoint),tempoIfNoSend);
00690                     return;
00691                  }
00692                  else {break;}
00693               }
00694            }
00695         }
00696         else {break;}
00697       };
00698     }
00699   }
00700
00701   if ((pck.tx_number < pck.max_num_tx) && (((*T) - pck.TimeBorn) < pck.TTL)) {
00702      pck.tx_number = pck.tx_number + 1;
00703      //Realiza la condicion del FER
```

```
00704       FER=0.1;
00705       if (rndUniformDouble (0, 1) < FER) { //El paquete no se llega a entregar.
00706          pAPoint->incrementaNNumPCKRetrasmitted();
00707          newEvent (new Ev_TryToSend(pAPoint),doublpckTrasmittingTime);
00708          return;
00709       }
00710       else{                                 //Si se llega a entregar.
00711          pAPoint->numACKpck();
00712          newEvent (new Ev_ReceivePacket (tempNode,pck,pAPoint),pckTrasmittingTime);
00713       }
00714       newEvent (new Ev_TryToSend(pAPoint),doublpckTrasmittingTime);
00715    }
00716    else{            //El paquete se pierde al no superar alguna condicion.
00717       pAPoint->incrementaNNumPCKLost();
00718       pAPoint->diminuirebuffer(pck);
00719       newEvent (new Ev_TryToSend(pAPoint),DETime(tempoIfNoSend));
00720    }
00721 }
00722
00723
00724 //#########################################################################################
00725 //-----------------------------------ChangePckToAP-----------------------------------
00726 //#########################################################################################
00727 void
00728 Mappa::ChangePckToAP (AccessPoint *pAPoint,vector <packet>vect,packet pck,int
IdRegDest)
00729 {
00730    int coda; //coda =0, stream, coda=1, elastic
00731    for (vector <AccessPoint *>::iterator itaccesspoint = elencoAccessPoint.begin ();
00732        itaccesspoint != elencoAccessPoint.end (); itaccesspoint++)
00733    {
00734     if ( (*itaccesspoint)->getid() == IdRegDest ) {
00735       if (pck.priority == true){
00736           coda = 0;       //se copia el paq en la cola Stream del nuevo Punto Acceso
00737           (*itaccesspoint)->CopyPacket(vect,pck,coda);
00738       }
00739
00740       else {
00741           coda = 1;       //se copia el paq en la cola Elastic del nuevo Punto Acceso
00742           (*itaccesspoint)->CopyPacket(vect,pck,coda);
00743       }
00744       break;
00745     }
00746    }
00747 }
00748
00749 //#########################################################################################
00750 //-----------------------------------richiediAPoint-----------------------------------
00751 //#########################################################################################
00752 void Mappa::richiediAPoint(Node *pnode,int IDregione,int ID)
00753 {
00754    bool pippo;
00755    DETime TimeTxReq;
00756    AccessPoint *tempAP;
00757    pippo=true;
00758
00759    while (pippo) {
00760      for (vector < AccessPoint * >::iterator iterAP = elencoAccessPoint.begin ();
00761           iterAP != elencoAccessPoint.end (); iterAP++) {
00762              if ((*iterAP)->getid() == IDregione) {
00763                  tempAP = *iterAP;
00764                  pippo=false;
00765                  break;
00766              }
00767      }
00768    }
00769    newEvent (new Ev_Msg_Request (tempAP,ID),pckTrasmittingTime);
00770 }
```

```
00771
00772 //#########################################################################################
00773 //-----------------------------------comparecoord-------------------------------------
00774 //#########################################################################################
00775 bool Mappa::comparecoord(coordinates c1,coordinates c2)
00776 {
00777   if((c1.x==c2.x)&&(c1.y==c2.y)){
00778         return (true);
00779   }
00780   else{
00781         return (false);
00782   }
00783 }
00784
00785 //#########################################################################################
00786 //-----------------------------------nodiperAP-------------------------------------
00787 //#########################################################################################
00788 int Mappa::nodiperAP(int IdReg)
00789 {
00790   int counter;
00791   counter = 0;
00792   for (vector < Node*>::iterator itnodo = elenconodi.begin ();
00793        itnodo != elenconodi.end (); itnodo++)
00794   {
00795             if ( ((*itnodo)->getIdregione ()) == IdReg ) {
00796                 counter++;
00797             };
00798   }
00799   return (counter);
00800 }
```

## A.13 Mappa.h

```
00001 #ifndef __Mappa_h__
00002 #define __Mappa_h__
00003
00004 #include "Common.h"
00005 #include <map>
00006 #include "Ostacolo.h"
00007
00008 class Regione;
00009 class Node;
00010 class AccessPoint;
00011 //#########################################################################################
00012 //------------------------------ Classe AccessPoint ----------------------------
00013 //#########################################################################################
00014 class Mappa:public DEDevice
00015 {
00016 protected:
00018   //Punteros a objetos externos.
00019   DESystem *System;
00020
00021   //Variables para la geometria del mapa y movilidad de  los terminales
00022   double lunghezza, altezza; //Anchura y altura del area de trabajo.
00023   double rangeposintermedia; //Usado en el calculo de la posicion intermedia.
00024   double distmaxpoi; //Distancia maxima a un PoI para que sea considerado.
00025   double distHist; //Distancia que se ha de separar un nodo para considerar roaming.
00026   vector < vector<int> > ostacoliadiacenti;//Para saber si hay obstaculos adyacentes
00027   vector < vector<coordinates> >verticiostacoli;//Sirve en el caso de que hayan
00028                               //obstaculos adyacentes. En tal caso se considera el
00029                               //macroobstaculo constituido por los vertices externos.
00030
```

```
00031   //Variables para hacer referencia a los objetos en el mapa.
00032   vector < Ostacolo * >ostacoli;
00033   vector < Regione * >regioni;
00034   vector < Node * >elenconodi;
00035   vector <coordinates>poi;
00036   vector < AccessPoint * >elencoAccessPoint;
00037 
00038   //Variables para la gestion del trafico.
00039   double FER;     //Representa la probabilidad de error de trama.
00040   double BandaTx; //Ancho de banda disponible (en bits/seg).
00041   DETime pckTrasmittingTime; //Tiempo que tarda en llegar un paquete a su destino.
00042   DETime doublpckTrasmittingTime; //Cada cuanto se envia un paquete.
00043   int maxLunPck;  //Maxima longitud en bytes de un paquete.
00044   int numnodi,numpoi;  //Numero de nodos y de PoI de la simulacion.
00045   int max_num_tx4pack; //Transmisiones por paquete antes de su eliminacion.
00046   DETime Maxttlxpck,MaxttlxpckStream; //Valores de TTL maximos.
00047 
00048 public:
00049 //-----------------------------------constructor-----------------------------------
00050   Mappa (DESystem * _System):DEDevice (_System){};
00051 //-----------------------------------destructor------------------------------------
00052   ~Mappa () {};
00053 //---------------------------------------setup-------------------------------------
00054   static void setup (ParamManager * Param, GlobalProbeManager * Results);
00055 //----------------------------------Initialize-------------------------------------
00056   virtual void Initialize (void);
00057 //--------------------------posizioneraggiungileblobal-----------------------------
00058   bool posizioneraggiungileblobal (coordinates posizione);
00059   bool traiettoriafattibileglobal (coordinates posizionepartenza,
00060                                    coordinates posizionearrivo);
00061 //-----------------------------------getpoicoord-----------------------------------
00062   coordinates getpoicoord (int indice);
00063 //---------------------------------trovadestintermedia-------------------------------
00064   coordinates trovadestintermedia(coordinates posizione,coordinates posizionefinale,
00065                                   int Idostacoloaggiramento,
00066                                   int idverticeaggiramento, Node* nodo);
00067 //-----------------------------------inseriscinodo---------------------------------
00068   void inseriscinodo (Node * pnodo);
00069 //-------------------------------trovaregioneappartenenza----------------------------
00070   int trovaregioneappartenenza (coordinates position);
00071 //---------------------------------trovaposizionenodo--------------------------------
00072   coordinates trovaposizionenodo (int Iddest);
00073 //-------------------------------inserisciAccessPoint------------------------------
00074   void inserisciAccessPoint (AccessPoint * tempAccessPoint);
00075 //-----------------------------------getNumNodi------------------------------------
00076   int getNumNodi();
00077 //-----------------------------creaadiacenzeostacoli-------------------------------
00078   void creaadiacenzeostacoli();
00079 //---------------------------------verticeesterno--------------------------------
00080   bool verticeesterno(int idostacolo, coordinates vertice);
00081 //-----------------------------verticevisibileglobal-----------------------------
00082   bool verticevisibileglobal(coordinates posizione,coordinates verticetest);
00083 //---------------------------------spediscipacchetto-----------------------------
00084   void spediscipacchetto (AccessPoint * pAPoint,packet pck);
00085 //-----------------------------------ChangePckToAp---------------------------------
00086   void ChangePckToAP (AccessPoint * pAPoint,vector <packet> vectpack,
00087                       packet pck, int IdRegDest);
00088 //-----------------------------------richiediAPoint--------------------------------
00089   void richiediAPoint(Node *pnode,int IDregione,int ID);
00090 //-------------------------------------getposAP------------------------------------
00091   coordinates getposAP(int IDregione);
00092 //-----------------------------------comparecoord----------------------------------
00093   bool comparecoord(coordinates c1,coordinates c2);
00094 //-----------------------------------nodiperAP-------------------------------------
00095   int nodiperAP(int IdReg);
00096 };
00097 #endif
```

## A.14 Node.cpp

```
00001 #include "DevTerminal.h"
00002 #include "Node.h"
00003 #include "Mappa.h"
00004 #include "AccessPoint.h"
00005 #include "Common.h"
00006 #include "Ev_ReceivePacket.h"
00007 #include "Ev_Muovinodo.h"
00008 #include "Ev_ObserveNode.h"
00009
00010 //###############################################################################
00011 //-----------------------------------setup-----------------------------------------
00012 //###############################################################################
00013 void Node::setup (ParamManager * Param, GlobalProbeManager * Results)
00014 {
00015   Param->addClass ("Node", "Generic ad hoc Node parameters");
00016   Param->addParameter ("Node",new DoubleParameter ("velocitymax",
00017                        "Maxima velocidad en pix/s con que se mueve","","(0,inf)"));
00018   Param->addParameter ("Node",new DoubleParameter ("velocitymin",
00019                        "Minima velocidad con que se mueve, en pix/s","", "(0,inf)"));
00020   Param->addParameter ("Node",new DETimeParameter ("HopTime",
00021                        "Tiempo entre un desplazamiento y el proximo","","(0,inf)"));
00022   Param->addParameter ("Node",new DETimeParameter ("ObserveTime",
00023                        "Tiempo entre observaciones, en seg","", "(0,inf)"));
00024   Param->addParameter ("Node",new DETimeParameter ("MaxStopTime",
00025                        "Tiempo maximo que se puede parar un nodo","", "(0,inf)"));
00026   Param->addParameter ("Node",new DETimeParameter ("MinStopTime",
00027                        "Tiempo minimo que puede estar parado","", "(0,inf)"));
00028 }
00029
00030 //###############################################################################
00031 //---------------------------------Initialize--------------------------------------
00032 //###############################################################################
00033 void Node::Initialize (void)
00034 {
00035   int countnodi,poidestino;
00036   double lunghezza, altezza, distmaxpoi;
00037   bool pippo;
00038   DETime inizioreset;
00039
00040   numerototnodi = get < IntParameter, int >(Param, "numnodi", "NwMap", "");
00041   distmaxpoi = get < DoubleParameter, double > (Param,"range_tra_poi","NwMap", "");
00042   lunghezza = get < DoubleParameter, double >(Param, "limitX", "NwMap", "");
00043   altezza = get < DoubleParameter, double >(Param, "limitY", "NwMap", "");
00044   numpoi = get < IntParameter, int >(Param, "numpoi", "NwMap", "");
00045   distmaxpoi = get < DoubleParameter, double > (Param,"range_tra_poi","NwMap","");
00046   HopTime = get < DETimeParameter, DETime > (Param, "HopTime", "Node", "");
00047   ObserveTime = get < DETimeParameter, DETime > (Param, "ObserveTime", "Node", "");
00048   MaxStopTime = get < DETimeParameter, DETime > (Param, "MaxStopTime", "Node", "");
00049   MinStopTime = get < DETimeParameter, DETime > (Param, "MinStopTime", "Node", "");
00050   velocitymax = get < DoubleParameter, double >(Param, "velocitymax", "Node", "");
00051   velocitymin = get < DoubleParameter, double >(Param, "velocitymin", "Node", "");
00052   TimeResolution =  get < IntParameter, int > (Param, "TimeResolution", "Test", "");
00053   tempoSimulazione =get<DETimeParameter,DETime>(Param,"SimulationTime","System","");
00054
00055   velocitynode = rndUniformDouble(velocitymin,velocitymax);
00056   DETime moveNode = DETime(double(HopTime)/100);
00057
00058   Idostacoloaggiramento = -1;
00059   idverticeaggiramento = -1;
00060   pippo = true;
00061   poiCounterNode = 0;
```

```
00062
00063   while (pippo) {
00064     posizione.x = rndUniformDouble (0, lunghezza);
00065     posizione.y = rndUniformDouble (0, altezza);
00066     if (pmappa->posizioneraggiungileblobal (posizione)) {
00067         pippo = false;
00068     };
00069   }
00070   aggiornaIDregione();
00071
00072   poideinodi.clear();
00073
00074   //---------------------------Probes-----------------------------------------------
00075   NumPoiVisitedNode = Results->getGlobalProbeMeanSlice("NumPoiVisitedNode");
00076   NumPckArrivedTotal = Results->getGlobalProbeMeanSlice("PckArrivedTotal");
00077   NumPckArrivedStream = Results->getGlobalProbeMeanSlice("PckArrivedStream");
00078   NumPckArrivedElastic = Results->getGlobalProbeMeanSlice("PckArrivedElastic");
00079   NumPckLostTotal = Results->getGlobalProbeMeanSlice("PckLost");
00080   EndToEndDelayElastic = Results->getGlobalProbeMeanSlice("EndToEndDelayElastic");
00081   EndToEndDelayStreamMsg =
00082         Results->getGlobalProbeMeanSlice("EndToEndDelayStreamMsg");
00083   EndToEndDelayElasticMsg =
00084         Results->getGlobalProbeMeanSlice("EndToEndDelayElasticMsg");
00085   EndToEndDelayStream = Results->getGlobalProbeMeanSlice("EndToEndDelayStream");
00086   EndToEndDelayTotal = Results->getGlobalProbeMeanSlice("EndToEndDelayTotal");
00087   EndToEndDelayHistogramStream =
00088         Results->getGlobalProbeHistogram("EndToEndDealyStreamHistogram");
00089   EndToEndDelayHistogramElastic =
00090         Results->getGlobalProbeHistogram("EndToEndDealyElasticHistogram");
00091   EndToEndDelayMessageHistogramStream =
00092         Results->getGlobalProbeHistogram("EndToEndDelayMessageHistogramStream");
00093   EndToEndDelayMessageHistogramElastic =
00094         Results->getGlobalProbeHistogram("EndToEndDelayMessageHistogramElastic");
00095   PoiVisitedHistogram = Results->getGlobalProbeHistogram("PoiVisitedHistogram");
00096
00097   //-----------Probes Mean-------------------------------------------------------------
00098   MeanE2EStream = Results->getGlobalProbeMean("MeanE2EStream");
00099   MeanE2EElastic = Results->getGlobalProbeMean("MeanE2EElastic");
00100   MeanE2EMessageStream = Results->getGlobalProbeMean("MeanE2EMessageStream");
00101   MeanE2EMessageElastic = Results->getGlobalProbeMean("MeanE2EMessageElastic");
00102   MeanPckArrivedStream = Results->getGlobalProbeMean("MeanPckArrivedStream");
00103   MeanPckArrivedElastic = Results->getGlobalProbeMean("MeanPckArrivedElastic");
00104
00105   //-----------------------------------------------------------------------------------
00106   NNumPCKLost = 0;
00107   numpacchettiricevuti = 0;
00108   numpckarrivedStream = 0;
00109   numpckarrivedElastic = 0;
00110
00111   posizionefinale = creanuovadest();
00112   posizioneintermedia = posizionefinale;
00113
00114 //Se llama a los eventos necesarios para el correcto funcionamiento----------------
00115   newEvent (new Ev_Muovinodo (this),moveNode);
00116   newEvent (new Ev_ObserveNode(this),ObserveTime);
00117 }
00118
00119 //#####################################################################################
00120 //-----------------------------------HEv_ReceivePacket-------------------------------
00121 //#####################################################################################
00122 void Node::HEv_ReceivePacket (packet pck, AccessPoint *pAPoint)
00123 {
00124   int Idsorgente, IDmsg, numtotpck;
00125   bool trovato;
00126   DETime Delay,DelayMsg;
00127
00128   trovato = false;
00129   Idsorgente = pck.IDNodosorgente;
```

```
00130   IDmsg = pck.IDMessage;
00131
00132   if (pck.priority) {
00133       numpckarrivedStream = numpckarrivedStream + 1;
00134       Delay = (T->Tick() - pck.TimeBorn.Tick());
00135       EndToEndDelayStream->Observe(Delay);
00136       EndToEndDelayHistogramStream->Observe(Delay);
00137       EndToEndDelayTotal->Observe(Delay);
00138       MeanE2EStream->Observe(Delay);
00139       if(pck.IDpacket== 0){
00140           bornMsgStream = pck.TimeBorn.Tick();
00141           IdMsgStream = pck.IDMessage;
00142       }
00143       if((pck.IDpacket== (pck.numpckinmsg - 1))&&(pck.IDMessage == IdMsgStream)){
00144           DelayMsg = (T->Tick() - bornMsgStream);
00145           MeanE2EMessageStream->Observe(DelayMsg);
00146           EndToEndDelayStreamMsg->Observe(DelayMsg);
00147           EndToEndDelayMessageHistogramStream->Observe(DelayMsg);
00148       }
00149
00150   }
00151   else {
00152       numpckarrivedElastic = numpckarrivedElastic + 1;
00153       Delay = (T->Tick() - pck.TimeBorn.Tick());
00154       EndToEndDelayElastic->Observe(Delay);
00155       EndToEndDelayHistogramElastic->Observe(Delay);
00156       EndToEndDelayTotal->Observe(Delay);
00157       MeanE2EElastic->Observe(Delay);
00158       if(pck.IDpacket== 0){
00159           bornMsgElastic = pck.TimeBorn.Tick();
00160           IdMsgElastic = pck.IDMessage;
00161       }
00162       if((pck.IDpacket== (pck.numpckinmsg - 1))&& (pck.IDMessage == IdMsgElastic)){
00163           DelayMsg = (T->Tick() - bornMsgElastic);
00164           MeanE2EMessageElastic->Observe(DelayMsg);
00165           EndToEndDelayElasticMsg->Observe(DelayMsg);
00166           EndToEndDelayMessageHistogramElastic->Observe(DelayMsg);
00167       }
00168   }
00169   numpacchettiricevuti = numpacchettiricevuti + 1;
00170   pAPoint->diminuirebuffer(pck);
00171 };
00172
00173 //////////////////////////////////////////////////////////////////////////////
00174 //-------------------------------getposition-----------------------------------
00175 //////////////////////////////////////////////////////////////////////////////
00176 coordinates Node::getposition ()
00177 {
00178   return (posizione);
00179 }
00180
00181 //////////////////////////////////////////////////////////////////////////////
00182 //----------------------------setAggiramentoostacolo-------------------------------
00183 //////////////////////////////////////////////////////////////////////////////
00184 void Node::setAggiramentoostacolo(int idostacolo, int idvertice)
00185 {
00186   Idostacoloaggiramento = idostacolo;
00187   idverticeaggiramento = idvertice;
00188 };
00189
00190 //////////////////////////////////////////////////////////////////////////////
00191 //-----------------------------------muovinodo-----------------------------------
00192 //////////////////////////////////////////////////////////////////////////////
00193 void
00194 Node::HEv_Muovinodo ()
00195 {
00196   coordinates pos;
00197   bool pippo;
```

```
00198   DETime TimeTxReq,stoptime;
00199   DETime SpostaTime = DETime(double(HopTime)/100);
00200   int aux;
00201
00202   if (posizione.x == posizionefinale.x && posizione.y == posizionefinale.y) {
00203      //El nodo esta en su destino: pide informacion al AP,
00204      //se elige un nuevo de destino y se llama al evento después de StopTime.
00205      Idostacoloaggiramento = -1;
00206      idverticeaggiramento = -1;
00207      aggiornaIDregione ();
00208      poiCounterNode ++;
00209      NumPoiVisitedNode->Observe(poiCounterNode);
00210      Poicounter++;
00211      posizionefinale = creanuovadest ();
00212      pmappa->richiediAPoint(this,IDregione,ID);
00213      stoptime = DETime(rndUniformDouble(MinStopTime,MaxStopTime));
00214      newEvent (new Ev_Muovinodo (this), stoptime);
00215      return;
00216   }
00217
00218   else {//Si el nodo no ha llegado aun a posicion final
00219      if (pmappa->traiettoriafattibileglobal (posizione, posizionefinale)) {
00220          //Si el nodo puede moverse en linea recta hacia su destinacion
00221          //porque no tiene obstaculos en medio.
00222          posizioneintermedia.x = posizionefinale.x;
00223          posizioneintermedia.y = posizionefinale.y;
00224          pos = spostadaa (posizione, posizionefinale, velocitynode*double(HopTime));
00225          posizione.x = pos.x;
00226          posizione.y = pos.y;
00227      }
00228      else {// Si el nodo tiene obstaculos intermedios para llegar a su destino:
00229          if ((posizione.x == posizioneintermedia.x)
00230             &&(posizione.y == posizioneintermedia.y))
00231          {
00232                //Si ya ha llegado a la posicion intermedia,
00233                //se calcula la posicion intermedia sucesiva.
00234                pos = pmappa->trovadestintermedia (posizione, posizionefinale,
00235                                    Idostacoloaggiramento,idverticeaggiramento,this);
00236                posizioneintermedia = pos;
00237          };
00238          //Si la posicion intermedia tiene tambien obstaculos en medio, se recalcula
00239          if (!(pmappa->traiettoriafattibileglobal(posizione,posizioneintermedia))) {
00240                pos = pmappa->trovadestintermedia (posizione, posizioneintermedia,
00241                                    Idostacoloaggiramento,idverticeaggiramento,this);
00242                posizioneintermedia.x = pos.x;
00243                posizioneintermedia.y = pos.y;
00244                newEvent (new Ev_Muovinodo (this), SpostaTime);
00245                return;
00246          }
00247          else{
00248                pos = spostadaa (posizione, posizioneintermedia,
00249                                velocitynode*double(HopTime));
00250                posizione.x = pos.x;
00251                posizione.y = pos.y;
00252          }
00253      };
00254   };
00255
00256   //Actualiza la region, pues puede darse que al desplazarse haya cambiado a otra.
00257   aggiornaIDregione ();
00258
00259   //Para hacer que se vuelva a mover el nodo después de HopTime.
00260   newEvent (new Ev_Muovinodo (this), HopTime);
00261 };
00263 //#########################################################################
00264 //-----------------------------aggiornaIDregione-----------------------------
00265 //#########################################################################
00266 void Node::aggiornaIDregione ()
```

```
00267 {
00268   IDregione = pmappa->trovaregioneappartenenza(posizione);
00269 };
00270 
00271 //###########################################################################################
00272 //-----------------------------------getid-----------------------------------------
00273 //###########################################################################################
00274 int Node::getid ()
00275 {
00276   return (ID);
00277 };
00278 
00279 //###########################################################################################
00280 //-----------------------------------getIdregione-----------------------------------------
00281 //###########################################################################################
00282 int Node::getIdregione ()
00283 {
00284   return (this->IDregione);
00285 };
00286 
00287 //###########################################################################################
00288 //-----------------------------------HEv_ObserveNode-----------------------------------------
00289 //###########################################################################################
00290 void Node::HEv_ObserveNode()
00291 {
00292   NumPckArrivedTotal->Observe( (numpckarrivedStream + numpckarrivedElastic) );
00293   NumPckArrivedStream->Observe( numpckarrivedStream );
00294   NumPckArrivedElastic->Observe( numpckarrivedElastic );
00295   NumPckLostTotal->Observe(NNumPCKLost);
00296   MeanPckArrivedStream->Observe(numpckarrivedStream);
00297   MeanPckArrivedElastic->Observe(numpckarrivedElastic);
00298   newEvent (new Ev_ObserveNode(this), ObserveTime);
00299   if ((poiCounterNode == 0)&&((T->Tick()*pow(10,TimeResolution)) >=
00300      (int(double(tempoSimulazione-ObserveTime)))))
00301   {
00302         cout<<"No llega a destino el Nodo: "<<ID<<", con pos: "<<posizione.x<<","
00303         <<posizione.y<<"buscando:"<<posizionefinale.x<<","<<posizionefinale.y<<'\n';
00304   }
00305 };
00306 
00307 //###########################################################################################
00308 //-----------------------------------creanuovadest-----------------------------------------
00309 //###########################################################################################
00310 coordinates Node::creanuovadest ()
00311 {
00312   coordinates pos, posinic,posfin;
00313   int poidestino, idpoi, poipartenza, npoi, i,j;
00314   vector <int> posiblepoi;
00316   i = 0;
00317 
00318   if (poideinodi.size() > 0){
00319       npoi = poideinodi.size();
00320       idpoi = poideinodi[(npoi-1)];
00321       posinic = pmappa->getpoicoord (idpoi);
00322   }
00323   else{
00324       posinic = posizione;
00325   }
00326 
00327   for(int i = 0 ; i < numpoi ; i++){
00328       pos=pmappa->getpoicoord(i);
00329       if ((trovadistanza (posinic,pos) < distmaxpoi/6) && (!(isSeen(i)))){
00330           posiblepoi.push_back(i);
00331       }
00332   }
00333 
00334   if (posiblepoi.size()>0){
00335       poidestino = posiblepoi[rndUniformInt(0,(posiblepoi.size()-1))];
```

```
00336       poideinodi.push_back(poidestino);
00337       PoiVisitedHistogram->Observe(poidestino+1);
00338       posfin = pmappa->getpoicoord(poidestino);
00339       return (posfin);
00340   }
00341   posiblepoi.clear();
00342
00343   for(int i = 0 ; i < numpoi ; i++){
00344       pos=pmappa->getpoicoord(i);
00345       if ((trovadistanza (posinic,pos) < distmaxpoi/2)&& (!(isSeen(i)))){
00346           posiblepoi.push_back(i);
00347       }
00348   }
00349   if (posiblepoi.size()>0){
00350       poidestino = posiblepoi[rndUniformInt(0,(posiblepoi.size()-1))];
00351       poideinodi.push_back(poidestino);
00352       PoiVisitedHistogram->Observe(poidestino+1);
00353       posfin = pmappa->getpoicoord(poidestino);
00354       return (posfin);
00355   }
00356   posiblepoi.clear();
00357
00358   for(int i = 0 ; i < numpoi ; i++){
00359       pos=pmappa->getpoicoord(i);
00360       if ((trovadistanza (posinic,pos) < distmaxpoi)&& (!(isSeen(i)))){
00361           posiblepoi.push_back(i);
00362       }
00363   }
00364   if (posiblepoi.size()>0){
00365       poidestino = posiblepoi[rndUniformInt(0,(posiblepoi.size()-1))];
00366       poideinodi.push_back(poidestino);
00367       PoiVisitedHistogram->Observe(poidestino+1);
00368       posfin = pmappa->getpoicoord(poidestino);
00369       return (posfin);
00370   }
00371   posiblepoi.clear();
00372
00373   for(int i = 0 ; i < numpoi ; i++){
00374       pos=pmappa->getpoicoord(i);
00375       if (!isSeen(i)){
00376           posiblepoi.push_back(i);
00377       }
00378   }
00379   if (posiblepoi.size()>0){
00380       poidestino = posiblepoi[rndUniformInt(0,(posiblepoi.size()-1))];
00381       poideinodi.push_back(poidestino);
00382       PoiVisitedHistogram->Observe(poidestino+1);
00383       return (pmappa->getpoicoord(poidestino));
00384   }
00385   else  {
00386       poideinodi.clear();
00387       pos=creanuovadest();
00388       return(pos);
00389   };
00390 };
00392 //#########################################################################################
00393 //-----------------------------------isSeen----------------------------------------
00394 //#########################################################################################
00395 bool Node::isSeen(int index)
00396 {
00397   for(int i = 0 ; i < poideinodi.size() ; i++){
00398       if (poideinodi[i]==index){
00399           return (true);
00400       }
00401   }
00402   return (false);
00403 };
```

## A.15 Node.h

```
00001 #ifndef __Node_h__
00002 #define __Node_h__
00003
00004 #include "Common.h"
00005 #include "probemean.h"
00006 #include "probehistogram.h"
00007 #include "derandom_time.h"
00008
00009 extern long int Poicounter;
00010 class Mappa;
00011 class AccessPoint;
00012 //#########################################################################################
00013 //---------------------------------- Clase Node -----------------------------------------
00014 //#########################################################################################
00015 class Node:public DevTerminal
00016 {
00017 protected:
00018   //Punteros a los objetos externos
00019   Mappa * pmappa;
00020   DESystem *System;
00021
00022   //Variables DETime
00023   DETime HopTime;     //Cada cuanto tiempo un nodo llama al evento 'muovinodo'
00024   DETime ObserveTime; //Cada cuanto tiempo un nodo llama al evento 'ObserveNode'
00025   DETime MaxStopTime; //Tiempo maximo que esta parado un nodo en la destinacion.
00026   DETime MinStopTime; //Tiempo minimo que esta parado un nodo en la destinacion.
00027   DETime tempoSimulazione; //Tiempo que va a durar la simulacion.
00028   DETime bornMsgStream,bornMsgElastic; //Almacena los tiempos de nacimiento del
00029                                        //mensaje Stream o Elastic que se esta recibiendo.
00030
00031   //Variables para identificacion del nodo en el mapa y gestion de la movilidad
00032   int ID;                 //ID identificador del nodo.
00033   coordinates posizione; //x,y de la posicion en la que se encuentra en el instante.
00034   coordinates posizionefinale; //x,y della posicion final que se quiere alcanzar.
00035   coordinates posizioneintermedia; //x,y de la posicion intermedia hacia la que va.
00036   double velocitymax; //Maxima velocidad en pix/s con que el terminal se puede mover
00037   double velocitymin; //Minima velocidad en pix/s con que el terminal se puede mover
00038   double velocitynode;//Velocidad aleatoria con la que se mueve cada nodo.
00039   unsigned int IDregione; //ID de la region en la que se encuentra el nodo.
00040   double distmaxpoi;  //Distancia maxima a que puede estar un PoI del nodo
00041                       //para considerarlo posible destino.
00042   int Idostacoloaggiramento,idverticeaggiramento; //Almacenan el ultimo vertice
00043                                                   //sorteado, para evitar entrar en bucle.
00044   vector<int> poideinodi; //Sirve para saber los PoI que ha visitado ya el nodo.
00045
00046   //Variables para los Probes y adquisicion de resultados.
00047   double NNumPCKLost,numpacchettiricevuti;
00048   double numpckarrivedStream,numpckarrivedElastic;
00049   int numpoi,poiCounterNode,numerototnodi;
00050   unsigned int IdMsgStream, IdMsgElastic;
00051   double TimeResolution;
00052
00053   ProbeSlice *NumPckArrivedTotal;
00054   ProbeSlice *NumPckArrivedStream;
00055   ProbeSlice *NumPckArrivedElastic;
00056   ProbeSlice *NumPckLostTotal;
00057   ProbeSlice *EndToEndDelayElastic;
00058   ProbeSlice *EndToEndDelayStream;
00059   ProbeSlice *EndToEndDelayElasticMsg;
00060   ProbeSlice *EndToEndDelayStreamMsg;
00061   ProbeSlice *EndToEndDelayTotal;
```

```
00062   ProbeSlice *NumPoiVisitedNode;
00063
00064   ProbeMean *MeanE2EStream;
00065   ProbeMean *MeanE2EElastic;
00066   ProbeMean *MeanE2EMessageStream;
00067   ProbeMean *MeanE2EMessageElastic;
00068   ProbeMean *MeanPckArrivedStream;
00069   ProbeMean *MeanPckArrivedElastic;
00070
00071   ProbeHistogram *EndToEndDelayHistogramStream;
00072   ProbeHistogram *EndToEndDelayHistogramElastic;
00073   ProbeHistogram *EndToEndDelayMessageHistogramStream;
00074   ProbeHistogram *EndToEndDelayMessageHistogramElastic;
00075   ProbeHistogram *PoiVisitedHistogram;
00076
00077 public:
00078 //-----------------------------------constructor-----------------------------------
00079     Node (DESystem * _System, int _ID, Mappa * _pmappa)
00080   : DevTerminal (_System, _ID, 0)
00081    , System (_System)
00082    , ID (_ID)
00083    , pmappa (_pmappa)
00084   {
00085   };
00086 //-----------------------------------destructor-----------------------------------
00087   Node::~Node () {};
00088 //-------------------------------------setup-------------------------------------
00089   static void setup (ParamManager * Param, GlobalProbeManager * Results);
00090 //-----------------------------------initialize-----------------------------------
00091   virtual void Initialize (void);
00092 //-----------------------------------getposition-----------------------------------
00093   coordinates getposition ();
00094 //-----------------------------------getIdregione-----------------------------------
00095   int getIdregione ();
00096 //-------------------------------------getid-------------------------------------
00097   int getid ();
00098 //-----------------------------------muovinodo-----------------------------------
00099   void HEv_Muovinodo ();
00100 //---------------------------------HEv_ReceivePacket---------------------------------
00101   void HEv_ReceivePacket (packet pck, AccessPoint *pAPoint);
00102 //---------------------------------HEv_ObserveNode---------------------------------
00103   void HEv_ObserveNode ();
00104 //---------------------------------aggiornaIDregione---------------------------------
00105   void aggiornaIDregione ();
00106 //-----------------------------------creanuovadest-----------------------------------
00107   coordinates creanuovadest ();
00108 //-------------------------------setAggiramentoostacolo-------------------------------
00109   void setAggiramentoostacolo(int idostacolo, int idvertice);
00110 //-------------------------------------isSeen-------------------------------------
00111   bool isSeen(int index);
00112 };
00113
00114 #endif
```

## A.16 Ostacolo.cpp

```
00001 #include "Ostacolo.h"
00002 #include "math.h"
00003 #define PI 3.1415926
00004
00005 //#########################################################################################
00006 //-----------------------------------assegnaID-----------------------------------------
00007 //#########################################################################################
00008 void
00009 Ostacolo::assegnaID (unsigned int id)
00010 {
00011   IDOstacolo = id;
00012 };
00013
00014 //#########################################################################################
00015 //-----------------------------assegnarangevertx-----------------------------------
00016 //#########################################################################################
00017 void Ostacolo::assegnarangevertex (double range)
00018 {
00019   Rangevertex = range;
00020 };
00021
00022 //#########################################################################################
00023 //-----------------------------------getID-----------------------------------------
00024 //#########################################################################################
00025 unsigned int Ostacolo::getID ()
00026 {
00027   return (IDOstacolo);
00028 };
00029
00030 //#########################################################################################
00031 //-----------------------------------getv1-----------------------------------------
00032 //#########################################################################################
00033 coordinates Ostacolo::getv1 ()
00034 {
00035   return (v1);
00036 };
00037
00038 //#########################################################################################
00039 //-----------------------------------getv2-----------------------------------------
00040 //#########################################################################################
00041 coordinates Ostacolo::getv2 ()
00042 {
00043   return (v2);
00044 };
00045
00046 //#########################################################################################
00047 //-----------------------------------getv3-----------------------------------------
00048 //#########################################################################################
00049 coordinates Ostacolo::getv3 ()
00050 {
00051   return (v3);
00052 };
00053
00054 //#########################################################################################
00055 //-----------------------------------getv4-----------------------------------------
00056 //#########################################################################################
00057 coordinates Ostacolo::getv4 ()
00058 {
00059   return (v4);
00060 };
00061
```

```
00062 //#########################################################################################
00063 //-------------------------------getvindice-----------------------------------------
00064 //#########################################################################################
00065 coordinates Ostacolo::getvindice (int i)
00066 {
00067   if (i == 1) {
00068       return (v1);
00069   };
00070   if (i == 2) {
00071       return (v2);
00072   };
00073   if (i == 3) {
00074       return (v3);
00075   };
00076   if (i == 4) {
00077       return (v4);
00078   };
00079 };
00080
00081 //#########################################################################################
00082 //-------------------------------Constructores-----------------------------------------
00083 //#########################################################################################
00084 Ostacolo::Ostacolo (coordinates v1p, coordinates v2p, coordinates v3p,
00085                     coordinates v4p)
00086 {
00087   v1.x = v1p.x;
00088   v1.y = v1p.y;
00089   v2.x = v2p.x;
00090   v2.y = v2p.y;
00091   v3.x = v3p.x;
00092   v3.y = v3p.y;
00093   v4.x = v4p.x;
00094   v4.y = v4p.y;
00095 }
00096
00097 //#########################################################################################
00098 //-------------------------------puntoraggiungibile-----------------------------------------
00099 //#########################################################################################
00100 bool Ostacolo::puntoraggiungibile (coordinates posizione)
00101 {
00102 //Esta funcion recibe como entrada una posicion y restituye true o false en funcion
00103 //de que dicha posicion caiga dentro del obstaculo o no.
00104 //Nota: Ecuacion de la recta para 2 puntos -> (y-y1)/(y2-y1) = (x-x1)/(x2-x1)
00105
00106   int centrale = 0;
00107   if (v1.y == v2.y && v3.y == v4.y && v1.x == v4.x && v2.x == v3.x) {
00108   //Caso de obstaculo rectangular con lados horizontales y verticales.
00109   //En realidad el siguiente caso contiene tambien a este.
00110      if ((v1.x <= posizione.x) && (posizione.x <= v2.x) && (v1.y <= posizione.y)
00111         && (v4.y >= posizione.y))
00112      {
00113           return (false);
00114      }
00115      else {
00116           return (true);
00117     }
00118   }
00119   else {
00120   //Caso general de obstaculos con lados oblicuos:
00121   //Se calcula interseccion recta v1-v2 y recta v3-v4 con la recta vertical que pasa
00122   //por posizione y mira las intersecciones.Se observa si la ordenada de la posicion
00123   //se encuentra en medio de las 2 intersecciones.
00124   //Hago lo mismo con las rectas v2-v3 y v1-v4 considerando la abcisa de la posicion
00125      double intersezionex1, intersezioney1, intersezionex2, intersezioney2;
00126      int centrale = 0;
00127      //Interseccion con la recta v1-v2.
00128      if (v1.y != v2.y) {
```

```
00129           intersezioney1 =(v2.y - v1.y)*((posizione.x - v1.x)/(v2.x - v1.x)) + v1.y;
00130           intersezionex1 = v1.x +(v2.x - v1.x)*(intersezioney1 - v1.y)/(v2.y - v1.y);
00131       }
00132       else {
00133           intersezioney1 = v1.y;
00134           intersezionex1 = posizione.x;
00135       }
00136       //Interseccion con la recta v3-v4.
00137       if (v3.y != v4.y) {
00138           intersezioney2 =(v3.y - v4.y)*((posizione.x - v4.x)/(v3.x - v4.x)) + v4.y;
00139           intersezionex2 = v4.x +(v3.x - v4.x)*(intersezioney2 - v4.y)/(v3.y - v4.y);
00140       }
00141       else {
00142           intersezioney2 = v3.y;
00143           intersezionex2 = posizione.x;
00144       }
00145       if (intersezioney1 <= posizione.y && intersezioney2 >= posizione.y) {
00146           centrale++;
00147       }
00148
00149       //Interseccion con la recta v1-v4.
00150       if (v1.x != v4.x) {
00151           intersezionex1 = v1.x + (v4.x - v1.x)*(posizione.y - v1.y)/(v4.y - v1.y);
00152           intersezioney1 = v1.y +(v4.y - v1.y)*(intersezionex1 - v1.x)/(v4.x - v1.x);
00153       }
00154       else {
00155           intersezionex1 = v1.x;
00156           intersezioney1 = posizione.y;
00157       }
00158       //Interseccion con la recta v2-v3.
00159       if (v2.x != v3.x) {
00160           intersezionex2 = v2.x + (v3.x - v2.x)*(posizione.y - v2.y)/(v3.y - v2.y);
00161           intersezioney2 = v2.y +(v3.y - v2.y)*(intersezionex1 - v2.x)/(v3.x - v2.x);
00162       }
00163       else {
00164           intersezionex2 = v2.x;
00165           intersezioney2 = posizione.y;
00166       }
00167       if (intersezionex1 <= posizione.x && intersezionex2 >= posizione.x) {
00168           centrale++;
00169       }
00170       if (centrale == 2) {//Si la posizione se encuentra entre las 4 rectas (dentro).
00171           return (false);
00172       }
00173       else {
00174           return (true);
00175       }
00176   }
00177 }
00178
00179 //#############################################################################
00180 //----------------------------traiettoriafattibile-----------------------------
00181 //#############################################################################
00182 bool Ostacolo::traiettoriafattibile (coordinates pos1, coordinates pos2)
00183 {
00184   double m1, m2, q1, q2, posxmin, posxmax, posxminv, posxmaxv, posyminv;
00185   double posymaxv, posymin, posymax;
00186   int numintersezioni = 0; //Sirve para contar cuantos lados del obstaculo son
00187                            //interceptados por la trayectoria.
00188   coordinates intersez;
00189
00190   if (pos1.x <= pos2.x) {
00191       posxmin = pos1.x;      //En posxmin va la x de pos1 si pos1 se encuentra
00192       posxmax = pos2.x;      //a izquierda de pos2, o viceversa.
00193   }
00194   else {
00195       posxmin = pos2.x;
00196       posxmax = pos1.x;
```

```
00197   };
00198   if (pos1.y <= pos2.y) {
00199       posymin = pos1.y;
00200       posymax = pos2.y;
00201   }
00202   else {
00203       posymin = pos2.y;
00204       posymax = pos1.y;
00205   };
00206
00207 //Interseccion con el lado v1-v2.
00208   if (v1.x <= v2.x) {
00209       posxminv = v1.x;
00210       posxmaxv = v2.x;
00211   }
00212   else {
00213       posxminv = v2.x;
00214       posxmaxv = v1.x;
00215   };
00216   if (v1.y <= v2.y) {
00217       posyminv = v1.y;
00218       posymaxv = v2.y;
00219   }
00220   else {
00221       posyminv = v2.y;
00222       posymaxv = v1.y;
00223   };
00224   intersez = trovaintersezione (v1, v2, pos1, pos2);
00225   if (posxminv <= intersez.x && intersez.x <= posxmaxv
00226       && posxmin <= intersez.x && intersez.x <= posxmax
00227       && posyminv <= intersez.y && intersez.y <= posymaxv
00228       && posymin <= intersez.y && intersez.y <= posymax)
00229   {
00230         numintersezioni = numintersezioni + 1;
00231   }
00232
00233 //Interseccion con el lado v2-v3.
00234   if (v2.x <= v3.x) {
00235       posxminv = v2.x;
00236       posxmaxv = v3.x;
00237   }
00238   else {
00239       posxminv = v2.x;
00240       posxmaxv = v3.x;
00241   };
00242   if (v2.y <= v3.y) {
00243       posyminv = v2.y;
00244       posymaxv = v3.y;
00245   }
00246   else {
00247       posyminv = v3.y;
00248       posymaxv = v2.y;
00249   };
00250   intersez = trovaintersezione (v2, v3, pos1, pos2);
00251   if (posxminv <= intersez.x && intersez.x <= posxmaxv
00252       && posxmin <= intersez.x && intersez.x <= posxmax
00253       && posyminv <= intersez.y && intersez.y <= posymaxv
00254       && posymin <= intersez.y && intersez.y <= posymax)
00255   {
00256         numintersezioni = numintersezioni + 1;
00257   }
00258
00259 //Interseccion con el lado v3-v4.
00260   if (v3.x <= v4.x) {
00261       posxminv = v3.x;
00262       posxmaxv = v4.x;
00263   }
00264   else {
```

```
00265       posxminv = v4.x;
00266       posxmaxv = v3.x;
00267   };
00268   if (v3.y <= v4.y) {
00269       posyminv = v3.y;
00270       posymaxv = v4.y;
00271   }
00272   else {
00273       posyminv = v4.y;
00274       posymaxv = v3.y;
00275   };
00276   intersez = trovaintersezione (v3, v4, pos1, pos2);
00277   if (posxminv <= intersez.x && intersez.x <= posxmaxv
00278       && posxmin <= intersez.x && intersez.x <= posxmax
00279       && posyminv <= intersez.y && intersez.y <= posymaxv
00280       && posymin <= intersez.y && intersez.y <= posymax)
00281   {
00282         numintersezioni = numintersezioni + 1;
00283   }
00284
00285 //Interseccion con el lado v4-v1.
00286   if (v1.x <= v4.x) {
00287       posxminv = v1.x;
00288       posxmaxv = v4.x;
00289   }
00290   else {
00291       posxminv = v4.x;
00292       posxmaxv = v1.x;
00293   };
00294   if (v1.y <= v4.y) {
00295       posyminv = v1.y;
00296       posymaxv = v4.y;
00297   }
00298   else {
00299       posyminv = v4.y;
00300       posymaxv = v1.y;
00301   };
00302   intersez = trovaintersezione (v1, v4, pos1, pos2);
00303   if (posxminv <= intersez.x && intersez.x <= posxmaxv
00304       && posxmin <= intersez.x && intersez.x <= posxmax
00305       && posyminv <= intersez.y && intersez.y <= posymaxv
00306       && posymin <= intersez.y && intersez.y <= posymax)
00307   {
00308         numintersezioni = numintersezioni + 1;
00309   }
00310
00311 //Se comprueba si el numero de intersecciones es mayor que 0.
00312   if (numintersezioni > 0) {
00313       return (false);
00314   }
00315   else {
00316       return (true);
00317   };
00318
00319 }
00320
00321 //###########################################################################
00322 //-------------------------------verticevisibile-----------------------------
00323 //###########################################################################
00324 bool Ostacolo::verticevisibile (coordinates posizione, int numvertice)
00325 {
00326   coordinates intersezione;
00327   bool ris;
00328   ris = true;
00329
00330   if (numvertice == 1) {
00331       intersezione = trovaintersezione (posizione, v1, v3, v2);
00332       if (intersezionecompresa (intersezione, v3, v2)
```

```
00333          && intersezionecompresa (intersezione, v1, posizione))
00334       {
00335            ris = false;
00336       }
00337       intersezione = trovaintersezione (posizione, v1, v3, v4);
00338       if (intersezionecompresa (intersezione, v3, v4)
00339          && intersezionecompresa (intersezione, v1, posizione))
00340       {
00341            ris = false;
00342       }
00343
00344     //El if de debajo sirve porque la funcion intersezionecompresa devuelve false si
00345     //'intersezione' coincide con uno de los dos vertices v1,v3 (en este caso).Por
00346     //tanto, sirve para evitar ambigüedad si 'posizione' es externa al obstaculo
00347     //pero sobre la recta que une dos vertices opuestos.
00348       if ((puntosuretta (posizione, v1, v3))
00349          && (trovadistanza (posizione, v1) > trovadistanza (posizione, v3)))
00350       {
00351            ris = false;
00352       }
00353   }
00354   if (numvertice == 2) {
00355       intersezione = trovaintersezione (posizione, v2, v3, v4);
00356       if (intersezionecompresa (intersezione, v3, v4)
00357          && intersezionecompresa (intersezione, v2, posizione))
00358       {
00359            ris = false;
00360       }
00361       intersezione = trovaintersezione (posizione, v2, v1, v4);
00362       if (intersezionecompresa (intersezione, v1, v4)
00363          && intersezionecompresa (intersezione, v2, posizione))
00364       {
00365            ris = false;
00366       }
00367       if ((puntosuretta (posizione, v2, v4))
00368          && (trovadistanza (posizione, v2) > trovadistanza (posizione, v4)))
00369       {
00370            ris = false;
00371       }
00372   }
00373   if (numvertice == 3) {
00374       intersezione = trovaintersezione (posizione, v3, v1, v2);
00375       if (intersezionecompresa (intersezione, v1, v2)
00376          && intersezionecompresa (intersezione, v3, posizione))
00377       {
00378            ris = false;
00379       }
00380       intersezione = trovaintersezione (posizione, v3, v1, v4);
00381       if (intersezionecompresa (intersezione, v1, v4)
00382          && intersezionecompresa (intersezione, v3, posizione))
00383       {
00384            ris = false;
00385       }
00386       if ((puntosuretta (posizione, v1, v3))
00387          && (trovadistanza (posizione, v3) > trovadistanza (posizione, v1)))
00388       {
00389            ris = false;
00390       }
00391   }
00392   if (numvertice == 4) {
00393       intersezione = trovaintersezione (posizione, v4, v1, v2);
00394       if (intersezionecompresa (intersezione, v1, v2)
00395          && intersezionecompresa (intersezione, v4, posizione))
00396       {
00397            ris = false;
00398       }
00399       intersezione = trovaintersezione (posizione, v4, v3, v2);
00400       if (intersezionecompresa (intersezione, v3, v2)
```

```
00401          && intersezionecompresa (intersezione, v4, posizione))
00402       {
00403            ris = false;
00404       }
00405       if ((puntosuretta (posizione, v2, v4))
00406          && (trovadistanza (posizione, v4) > trovadistanza (posizione, v2)))
00407       {
00408       ris = false;
00409       }
00410   }
00411   return (ris);
00412 };
00413
00414 //#############################################################################
00415 //----------------------------Puntovicinovertice-------------------------------
00416 //#############################################################################
00417 coordinates Ostacolo::puntovicinovertice (int i)
00418 {
00419 //Funcion que devuelve las coordenadas de un punto exterior al obstaculo que este en
00420 //la cercanias del vertice con indice i. Si por ejemplo se pasa i=1, calcula la
00421 //recta que pasa por los vertices v1-v3 y coge un punto sobre tal recta que sea
00422 //externo al obstaculo y que diste de v1 como maximo el valor 'rangevertex'.
00423   double m, a, xrange,Rangemin,pmin;
00424   coordinates pos;
00425
00426   Rangemin = 0.2;
00427   pmin = 0;
00428
00429   if (i == 1) {
00430       if (v1.x == v3.x) {
00431           pos.x = v1.x;
00432           pos.y = (rndUniformDouble ( (v1.y - Rangevertex),(v1.y - Rangemin)));
00433       }
00434       else {
00435           m = (v3.y - v1.y) / (v3.x - v1.x);//coeficiente angular de la recta v1-v3
00436           a = atan (m);                     //angulo de inclinacion de la recta v1-v3
00437           xrange = Rangevertex * abs (cos (a));
00438           pmin = Rangemin * abs ( cos (a) );
00439           if (v1.x <= v3.x) {
00440              pos.x = rndUniformDouble ( (v1.x - xrange), (v1.x - pmin) );
00441           }
00442           else {
00443              pos.x = rndUniformDouble ( (v1.x + pmin) , (v1.x + xrange) );
00444           }
00445           pos.y = (v1.y + (v3.y - v1.y) * (pos.x - v1.x) / (v3.x - v1.x));
00446       }
00447   };
00448
00449   if (i == 3) {
00450       if (v1.x == v3.x) {
00451           pos.x = v1.x;
00452           pos.y = rndUniformDouble (  (v3.y + Rangemin),(v3.y + Rangevertex) );
00453       }
00454       else {
00455           m = (v3.y - v1.y) / (v3.x - v1.x);//coeficiente angular de la recta v1-v3
00456           a = atan (m);                     //angulo de inclinacion de la recta v1-v3
00457           xrange = Rangevertex * abs (cos (a));
00458           pmin = Rangemin * abs ( cos (a) );
00459           if (v1.x <= v3.x) {
00460              pos.x = rndUniformDouble ( (v3.x + pmin) , (v3.x + xrange) );
00461           }
00462           else {
00463              pos.x = rndUniformDouble ((v3.x - xrange), (v3.x - pmin) );
00464           }
00465           pos.y = (v1.y + (v3.y - v1.y) * (pos.x - v1.x) / (v3.x - v1.x));
00466       };
00467   };
00468
```

```
00469   if (i == 2) {
00470       if (v2.x == v4.x) {
00471           pos.x = v2.x;
00472           pos.y = rndUniformDouble ((v2.y - Rangevertex), (v2.y - Rangemin));
00473       }
00474       else {
00475           m = (v2.y - v4.y) / (v2.x - v4.x);//coeficiente angular de la recta v2-v4
00476           a = atan (m);                     //angulo de inclinacion de la recta v2-v4
00477           xrange = Rangevertex * abs (cos (a));
00478           pmin = Rangemin * abs ( cos (a) );
00479           if (v4.x <= v2.x) {
00480              pos.x = rndUniformDouble ( (v2.x + pmin) , (v2.x + xrange));
00481           }
00482           else {
00483              pos.x = rndUniformDouble ( (v2.x - xrange), (v2.x - pmin) );
00484           }
00485           pos.y = (v2.y + (v4.y - v2.y) * (pos.x - v2.x) / (v4.x - v2.x));
00486       };
00487   };
00488
00489   if (i == 4) {
00490       if (v2.x == v4.x) {
00491           pos.x = v2.x;
00492           pos.y = rndUniformDouble ( (v4.y + Rangemin), (v4.y + Rangevertex) );
00493       }
00494       else {
00495           m = (v2.y - v4.y) / (v2.x - v4.x);//coeficiente angular de la recta v2-v4
00496           a = atan (m);                     //angulo de inclinacion de la recta v2-v4
00497           xrange = Rangevertex * abs (cos (a));
00498           pmin = Rangemin * abs ( cos (a) );
00499           if (v4.x <= v2.x) {
00500              pos.x = rndUniformDouble ((v4.x - xrange), (v4.x - pmin) );
00501           }
00502           else {
00503              pos.x = rndUniformDouble ( (v4.x + pmin) , (v4.x + xrange));
00504           }
00505           pos.y = (v2.y + (v4.y - v2.y) * (pos.x - v2.x) / (v4.x - v2.x));
00506       };
00507   };
00508
00509   return (pos);
00510 };
00511
00512 //#############################################################################
00513 //----------------------------determinaidvertice---------------------------------
00514 //#############################################################################
00515 int Ostacolo::determinaidvertice(coordinates posizione)
00516 {
00517   int id;
00518   id = -1;
00519   if( (posizione.x == v1.x) && (posizione.y == v1.y)){id = 1;};
00520   if( (posizione.x == v2.x) && (posizione.y == v2.y)){id = 2;};
00521   if( (posizione.x == v3.x) && (posizione.y == v3.y)){id = 3;};
00522   if( (posizione.x == v4.x) && (posizione.y == v4.y)){id = 4;};
00523   return(id);
00524 };
```

## A.17 Ostacolo.h

```
00001 #ifndef __Ostacolo_h__
00002 #define __Ostacolo_h__
00003 #include "Common.h"
00004
00005 //////////////////////////////////////////////////////////////////////////////////
00006 //---------------------------------Clase Ostacolo---------------------------------
00007 //////////////////////////////////////////////////////////////////////////////////
00008 class Ostacolo
00009 {
00010 //Las coordenadas de los vertices del obstaculo se han tomado en este orden:
00011 //v1 : vertice alto izquierda.
00012 //v2 : vertice alto derecha.
00013 //v3 : vertice bajo derecha.
00014 //v4 : vertice bajo izquierda.
00015 private:
00016   coordinates v1, v2, v3, v4;
00017   double Rangevertex;          //Sirve para el algoritmo de movilidad(ver la
00018   unsigned int IDOstacolo;     //funcion puntovicinovertice).
00019
00020 public:
00021
00022 //Funciones para introducir y leer las variables privadas.
00023   void assegnaID (unsigned int id);
00024   void assegnarangevertex (double range);
00025   void assegnacoordinate (coordinates v1, coordinates v2,
00026                           coordinates v3, coordinates v4);
00027   unsigned int getID ();
00028   coordinates getv1 ();
00029   coordinates getv2 ();
00030   coordinates getv3 ();
00031   coordinates getv4 ();
00032   coordinates getvindice (int i);
00033
00034 //Funciones para el modelo de movilidad.
00035   bool puntoraggiungibile (coordinates posizione);
00036   bool traiettoriafattibile (coordinates pos1, coordinates pos2);
00037   bool verticevisibile (coordinates posizione, int numvertice);
00038   coordinates puntovicinovertice (int i);
00039   int determinaidvertice(coordinates posizione);
00040
00041 //Constructor.
00042   Ostacolo (coordinates v1, coordinates v2, coordinates v3, coordinates v4);
00043 };
00044
00045 #endif
```

## A.18 Regione.cpp

```
00001 #include "Regione.h"
00002
00003 //--------------------------------getv1-------------------------------------------
00004 coordinates Regione::getv1 ()
00005 {
00006   return (v1);
00007 };
00008
00009 //--------------------------------getv2-------------------------------------------
00010 coordinates Regione::getv2 ()
00011 {
00012   return (v2);
00013 };
00014
00015 //--------------------------------getv3-------------------------------------------
00016 coordinates Regione::getv3 ()
00017 {
00018   return (v3);
00019 };
00020
00021 //--------------------------------getv4-------------------------------------------
00022 coordinates Regione::getv4 ()
00023 {
00024   return (v4);
00025 };
00026
00027 //--------------------------------getid-------------------------------------------
00028 unsigned int Regione::getid ()
00029 {
00030   return (IDregione);
00031 };
00032
00033 //-------------------------------------setID--------------------------------------
00034 void Regione::setID (unsigned int ID)
00035 {
00036   IDregione = ID;
00037 };
00038
00039 //-------------------------------------setv1--------------------------------------
00040 void Regione::setv1 (coordinates v)
00041 {
00042   v1.x = v.x;
00043   v1.y = v.y;
00044 };
00045
00046 //-------------------------------------setv2--------------------------------------
00047 void Regione::setv2 (coordinates v)
00048 {
00049   v2.x = v.x;
00050   v2.y = v.y;
00051 };
00052
00053 //-------------------------------------setv3--------------------------------------
00054 void Regione::setv3 (coordinates v)
00055 {
00056   v3.x = v.x;
00057   v3.y = v.y;
00058 };
00059
00060 //-------------------------------------setv4--------------------------------------
00061 void Regione::setv4 (coordinates v)
```

```
00062 {
00063   v4.x = v.x;
00064   v4.y = v.y;
00065 };
00066
00067 //-----------------------------------posappartenente-------------------------------
00068 bool Regione::posappartenente (coordinates posizione)
00069 {
00070   bool res;
00071
00072   if (v1.x <= posizione.x && posizione.x <= v2.x && v1.y >= posizione.y
00073       && v4.y <= posizione.y)
00074   {
00075         res = true;
00076   }
00077   else {
00078         res = false;
00079   }
00080   return (res);
00081 };
00082
00083 //---------------------------------------Constructor-----------------------------------
00084 Regione::Regione (coordinates v1p, coordinates v3p, unsigned int ID)
00085 {
00086   v1.x = v1p.x;
00087   v1.y = v1p.y;
00088   v2.x = v3p.x;
00089   v2.y = v1p.y;
00090   v3.x = v3p.x;
00091   v3.y = v3p.y;
00092   v4.x = v1p.x;
00093   v4.y = v3p.y;
00094   IDregione = ID;
00095 };
00096
00097 //-----------------------------distanzaconfine(posizione)-------------------------
00098 double Regione::distanzaconfine (coordinates posizione)
00099 {
00100 //Esta funcion restituye la distancia de 'posizione' al lado mas cercano de la
00101 //region que se esta considerando.
00102   double distanza, distanzanew;
00103
00104   //Se inicializa distancia a un valor muy grande.
00105   distanza = 10 * trovadistanza (v1, v2);
00106
00107   distanzanew = distanzapuntoretta (v1, v2, posizione);
00108   if (distanzanew < distanza) {
00109       distanza = distanzanew;
00110   }
00111
00112   distanzanew = distanzapuntoretta (v3, v2, posizione);
00113   if (distanzanew < distanza) {
00114       distanza = distanzanew;
00115   }
00116
00117   distanzanew = distanzapuntoretta (v3, v4, posizione);
00118   if (distanzanew < distanza) {
00119       distanza = distanzanew;
00120   }
00121
00122   distanzanew = distanzapuntoretta (v1, v4, posizione);
00123   if (distanzanew < distanza) {
00124       distanza = distanzanew;
00125   }
00126   return (distanza);
00127 };
```

## A.19 Regione.h

```
00001 #ifndef Regione__h
00002 #define Regione__H
00003 #include "Common.h"
00004
00005 //#########################################################################################
00006 //----------------------------------Clase Regione----------------------------------------
00007 //#########################################################################################
00008 class Regione
00009 {
00010 //Los vertices v1,v2,v3,v4 van enumerados asi:
00011 // v1 : vertice alto izquierda.
00012 // v2 : vertice alto derecha.
00013 // v3 : vertice bajo derecha.
00014 // v4 : vertice bajo izquierda.
00015
00016 private:
00017   coordinates v1, v2, v3, v4;
00018   unsigned int IDregione;
00019
00020 public:
00021   Regione (coordinates v1, coordinates v3, unsigned int ID);
00022
00023   coordinates getv1 ();
00024   coordinates getv2 ();
00025   coordinates getv3 ();
00026   coordinates getv4 ();
00027   unsigned int getid ();
00028
00029   void setID (unsigned int ID);
00030   void setv1 (coordinates v);
00031   void setv2 (coordinates v);
00032   void setv3 (coordinates v);
00033   void setv4 (coordinates v);
00034
00035   bool posappartenente (coordinates pos);
00036   double distanzaconfine (coordinates posizione);
00037
00038 };
00039 #endif
```

## A.20 SysTest.cpp

```
00001 #include "SysTest.h"
00002 #include <cmath>
00003 //------------------------------------Constructor--------------------------------
00004 SysTest::SysTest (ParamManager * _Param, GlobalProbeManager * _Results,
00005                    ProbeManager * _ProbeMgr)
00006 :DESystem (_Param, _Results, _ProbeMgr, get < IntParameter, int >
00007 (_Param, "TimeResolution", "Test", ""), 1,1, 1)
00008           1, 1)
00009 {
00010   theNwMap = NULL;
00011   theNodes.clear ();
00012 }
00013
00014 //-----------------------------------------Setup---------------------------------
00015 void
00016 SysTest::setup (ParamManager * Param, GlobalProbeManager * Results)
00017 {
00018   Param->addClass ("Test", "Test system");
00019   Param->addParameter ("Test",new IntParameter ("TimeResolution",
00020                        "Risoluzione temporale", "-4", "(0,inf)"));
00021   Mappa::setup (Param, Results);
00022   Node::setup (Param, Results);
00023   AccessPoint::setup (Param,Results);
00024   Param->setDefault ("BaseFileName", "results/");
00025   Param->setDefault ("TimeResolution", "-4");
00026 }
00027
00028 //-------------------------------------Initialize---------------------------------
00029 void SysTest::Initialize ()
00030 {
00031   int i,j, numAP,numzonecol,numzonerow;
00032   coordinates pos, posf;
00033   Node *tempNode;
00034   AccessPoint *tempAP;
00035   string name;
00036   DESystem::Initialize ();
00037
00038   numnodi = get < IntParameter, int >(Param, "numnodi", "NwMap", "");
00039   tempoSimulazione =get<DETimeParameter,DETime>(Param,"SimulationTime","System","");
00040   numAP = get < IntParameter, int >(Param, "numAP", "NwMap", "");
00041   numzonecol = get < IntParameter, int >(Param, "numzone_col", "NwMap", "");
00042   numzonerow = get < IntParameter, int >(Param, "numzone_row", "NwMap", "");
00043   numAP = numzonecol * numzonerow;
00044   Poicounter = 0;
00045
00046   theNwMap = new Mappa (this);
00047   theNwMap->Initialize ();
00048
00049   for (i = 0; i < numAP; i++) {   //Se crean los AccessPoint.
00050       tempAP = new AccessPoint(this,i,theNwMap);
00051       tempAP->Initialize();
00052       theNwMap->inserisciAccessPoint(tempAP);
00053   };
00054
00055   for (j = 0; j < numnodi; j++) { //Se crean ahora los nodos en la red.
00056     tempNode = new Node (this, j, theNwMap);
00057     tempNode->Initialize ();
00058     theNwMap->inseriscinodo (tempNode);
00059   };
00060
00061   cout<<"Creados y posicionados "<<numnodi<<" nodos en la Red"<<'\n';
```

```
00062   cout<<"Creados y posicionados "<<numAP<<" Puntos de Acceso en la Red"<<'\n';
00063 };
00064
00065 //---------------------------------------Destructor---------------------------------
00066 SysTest::~SysTest ()
00067 {
00068   delete theNwMap;
00069   theNodes.clear ();
00070
00071   cout<<"Valor final de numPoixNode: "<<(double(Poicounter))/numnodi<<'\n';
00072   cout<<"Valor final de Poicounter: "<<Poicounter<<'\n';
00073 };
00074
00075 //---------------------------------------Main---------------------------------------
00076 void
00077 DEMain ()
00078 {
00079   RunSimul < SysTest > ();
00080 }
```

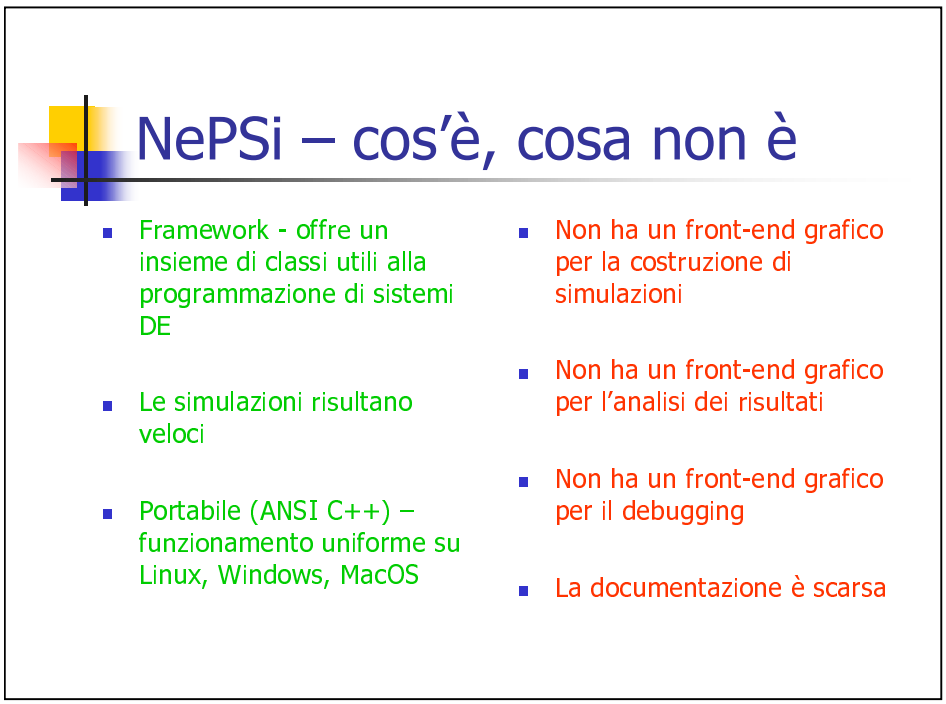
## A.21 SysTest.h

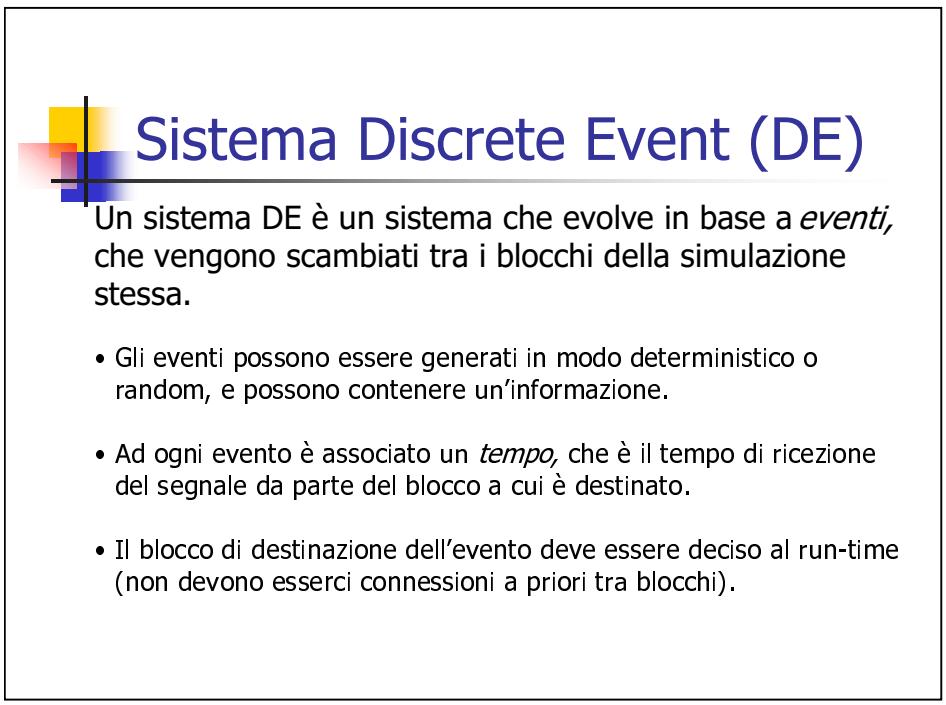
```
00001 #ifndef __Test_Routing_h__
00002 #define __Test_Routing_h__
00003
00004 #include "desystem.h"
00005 #include "dedevice.h"
00006 #include "dedebug.h"
00007 #include "demain.h"
00008 #include "Common.h"
00009 #include "Mappa.h"
00010 #include "Node.h"
00011 #include "AccessPoint.h"
00012
00013 long int Poicounter;
00014 //############################################################################
00015 //-----------------------------------Clase SysTest-----------------------------------
00016 //############################################################################
00017 class SysTest : public DESystem
00018 {
00019  protected:
00020   Mappa* theNwMap;
00021   vector <Node*> theNodes;
00022   int numnodi;
00023   DETime tempoSimulazione;
00024
00025  public:
00026   SysTest(ParamManager* _Param,GlobalProbeManager* _Results,ProbeManager* _ProbeMg);
00027   static void setup(ParamManager *Param, GlobalProbeManager *Results);
00028   void Initialize();
00029   virtual ~SysTest();
00030 };
00031
00032 #endif
```

# Anexo B

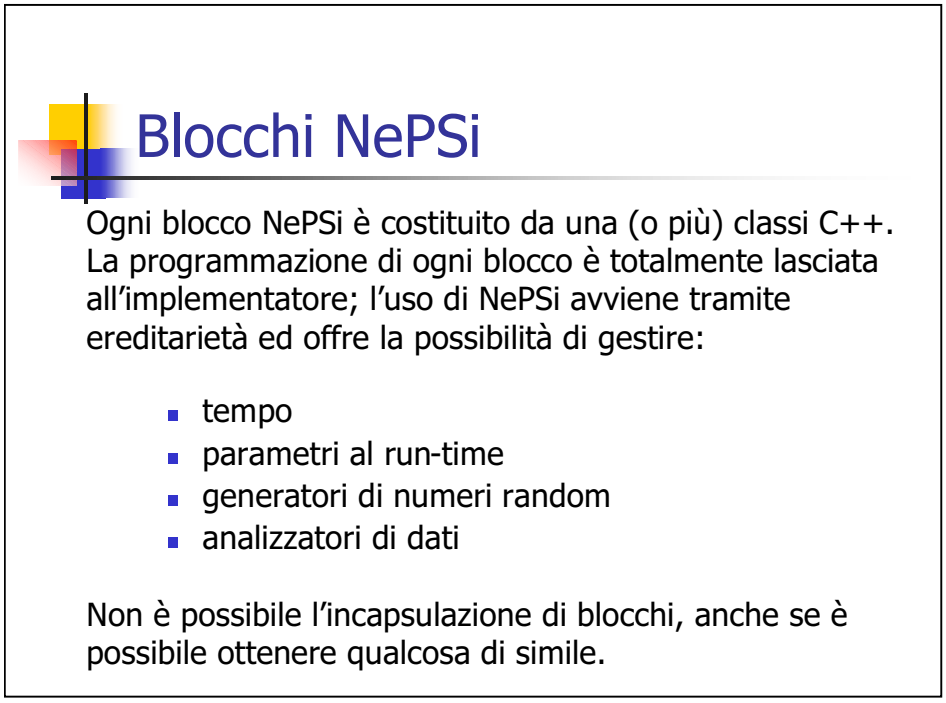
# Manual de NePSi


Il sistema INeSiS (Integrated Network Signal Processing Simulator) è una piattaforma di simulazione che comprende due sottosistemi:
MuDiSP(Multirate Digital Signal Processor): un simulatore che considera flussi di bit sincroni e collegamenti fra blocchi di tipo deterministico e stabilito a priori.
Network Protocol Simulator (NePSi): un simulatore a eventi discreti e asincroni in cui il flusso di dati fra i vari blocchi è stabilito in maniera dinamica nel corso del simulazione (in run-time).

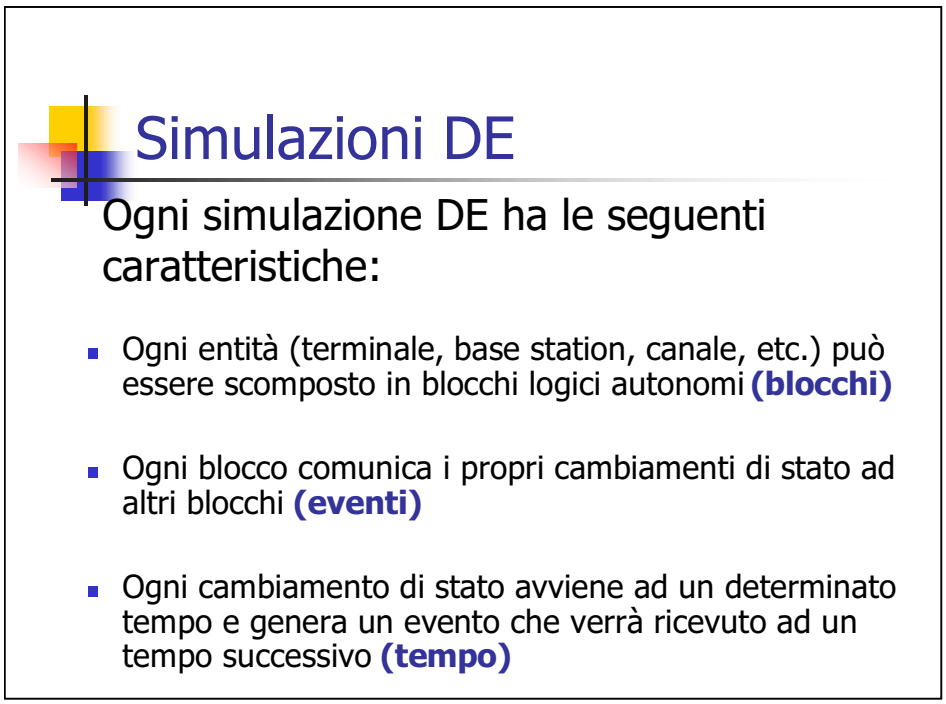

NePSi – cos'è, cosa non è
Framework - offre un insieme di classi utili alla programmazione di sistemi DE
Le simulazioni risultano veloci
Portabile (ANSI C++) – funzionamento uniforme su Linux, Windows, MacOS
Non ha un front-end grafico per la costruzione di simulazioni
Non ha un front-end grafico per l'analisi dei risultati
Non ha un front-end grafico per il debugging
La documentazione è scarsa


Sistema Discrete Event (DE)
Un sistema DE è un sistema che evolve in base a eventi, che vengono scambiati tra i blocchi della simulazione stessa.
• Gli eventi possono essere generati in modo deterministico o random, e possono contenere un'informazione.
• Ad ogni evento è associato un tempo, che è il tempo di ricezione del segnale da parte del blocco a cui è destinato.
• Il blocco di destinazione dell'evento deve essere deciso al run-time (non devono esserci connessioni a priori tra blocchi).

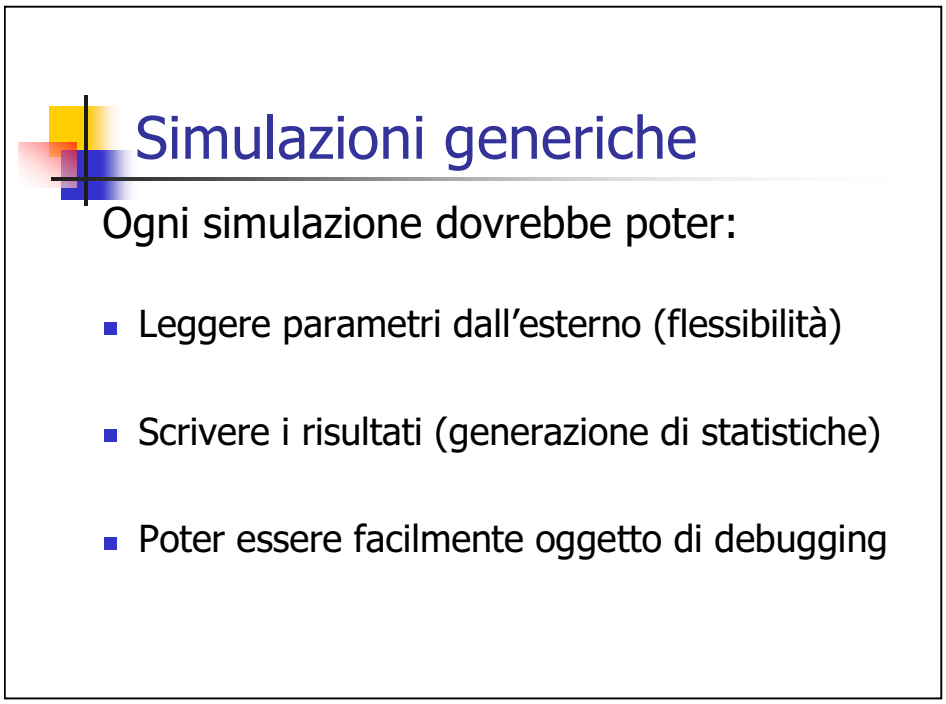
Gli eventi in generale sono scambi di messaggi fra blocchi del simulatore, un particolare tipo di evento può essere ad esempio il cambiamento di stato in una macchina a stati finiti.
Se l'evento è l'arrivo di un pacchetto, l'informazione associata all'evento sarà ad esempio il tipo di pacchetto.
Il tempo associato all'evento è sempre il tempo associato alla sua ricezione, che costituisce un elemento importante per la connotazione dell'evento in quanto è l'istante in cui verrà presumibilmente generato un altro evento.

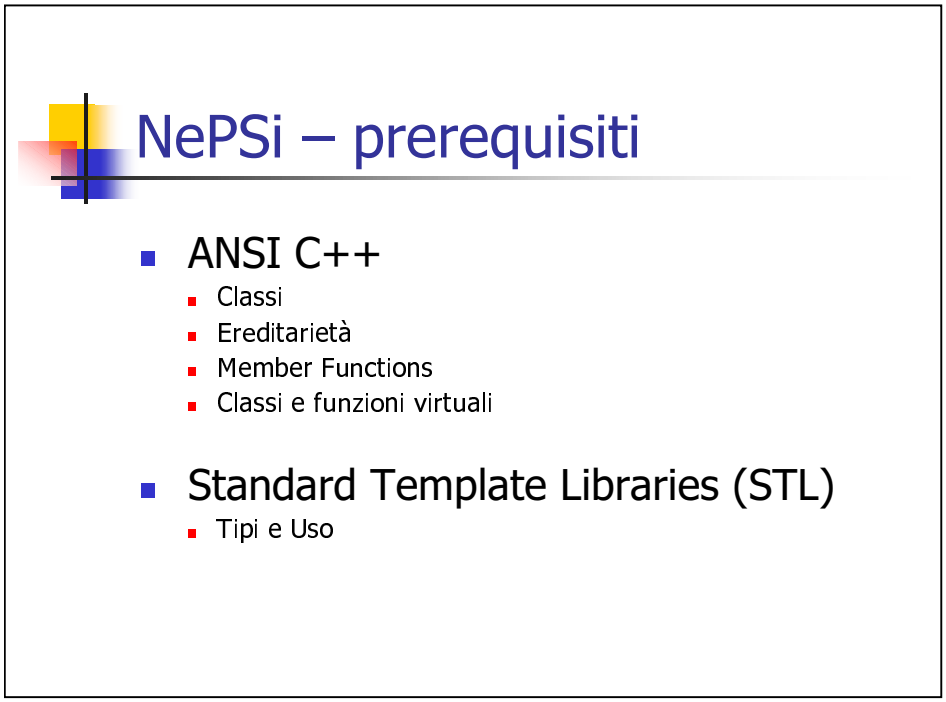



In generale un blocco NePSi corrisponde ad una classe C++, più eventuali sottoclassi di utility.
Ogni blocco ha come classi padri delle classi generiche, da cui eredita alcune funzioni fondamentali, come il concetto di tempo, la gestione dei parametri a run-time, i generatori random, etc.
Ogni blocco NePSi ha caratteristiche a sé stanti, non è cioè possibile generare un blocco "terminale mobile" e tanti sottoblocchi "generatore random", "ricezione ACK", e così via. Ogni blocco avrà caratteristiche peculiari, e anche se in realtà produrrà dati che verranno usati da un macro oggetto "terminale mobile" rimane un'entità autonoma. Ciò offre maggiore flessibilità al simulatore.

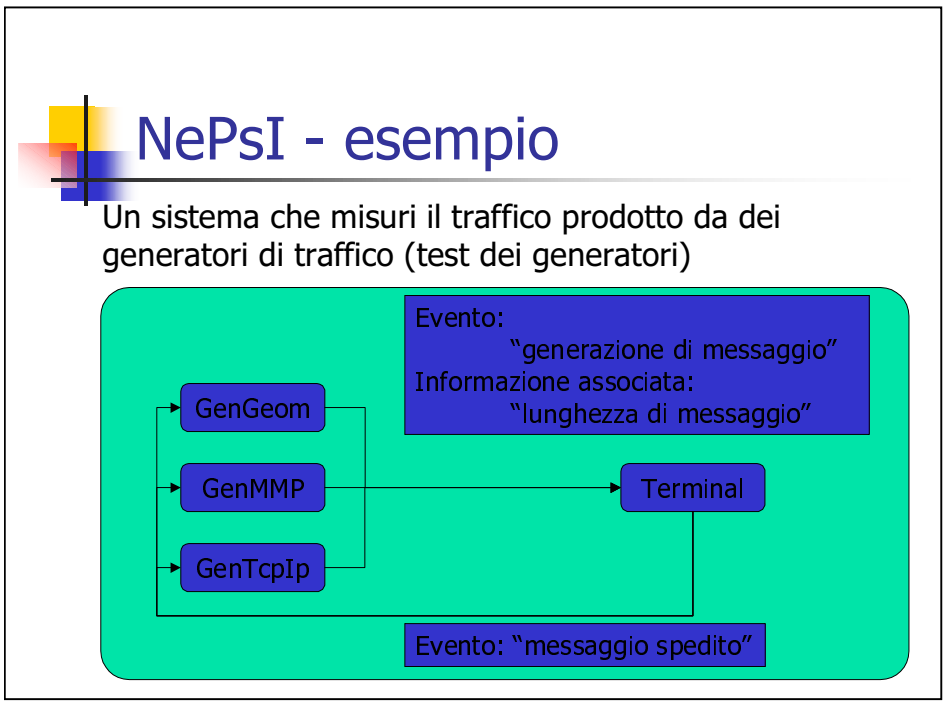



La dimensione di un blocco deve essere scelta accuratamente, scegliendo un blocco troppo piccolo (cioè che svolge poche funzioni) si rischia di renderlo inutile, scegliendo un blocco più grande è possibile che generi troppi eventi e che congestioni il sistema.

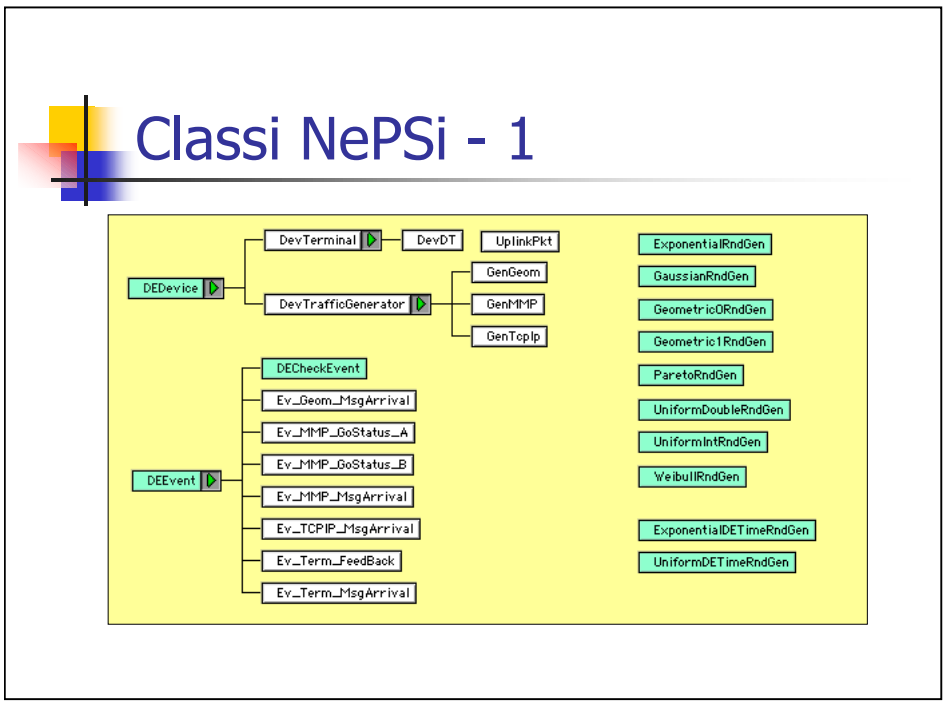




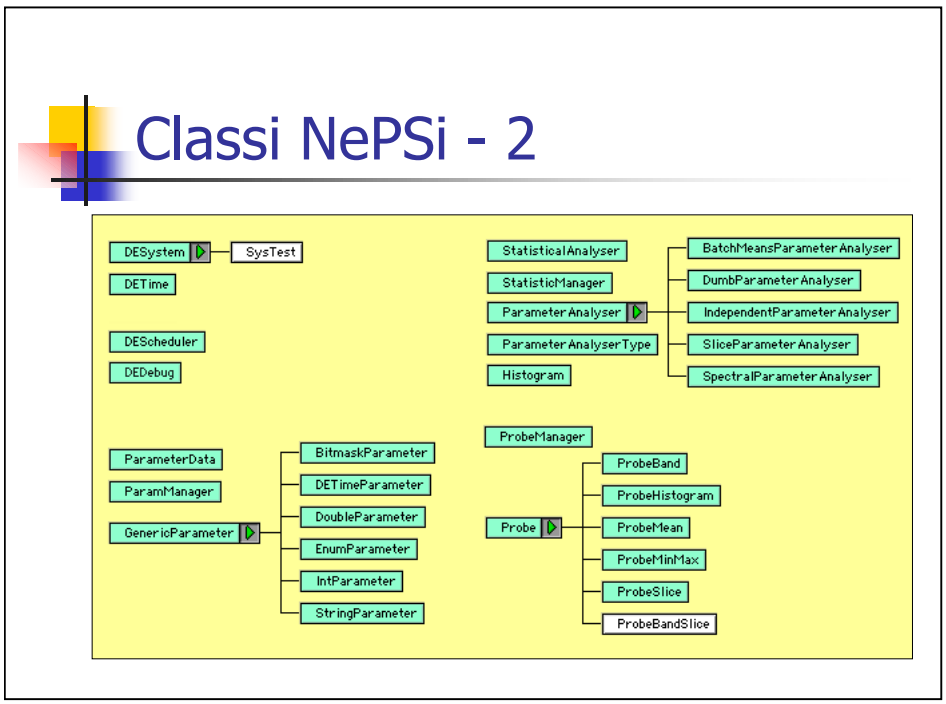

Conoscere le Standard Template Libraries permette di ottimizzare la scrittura del codice, evitando di dover perdere tempo a scrivere funzioni che già esistono. Un esempio evidente è la gestione dei vettori. Con la STL <vector> si possono facilmente gestire vettori di dimensioni dinamiche, senza doversi preoccupare di allocare e deallocare memoria.



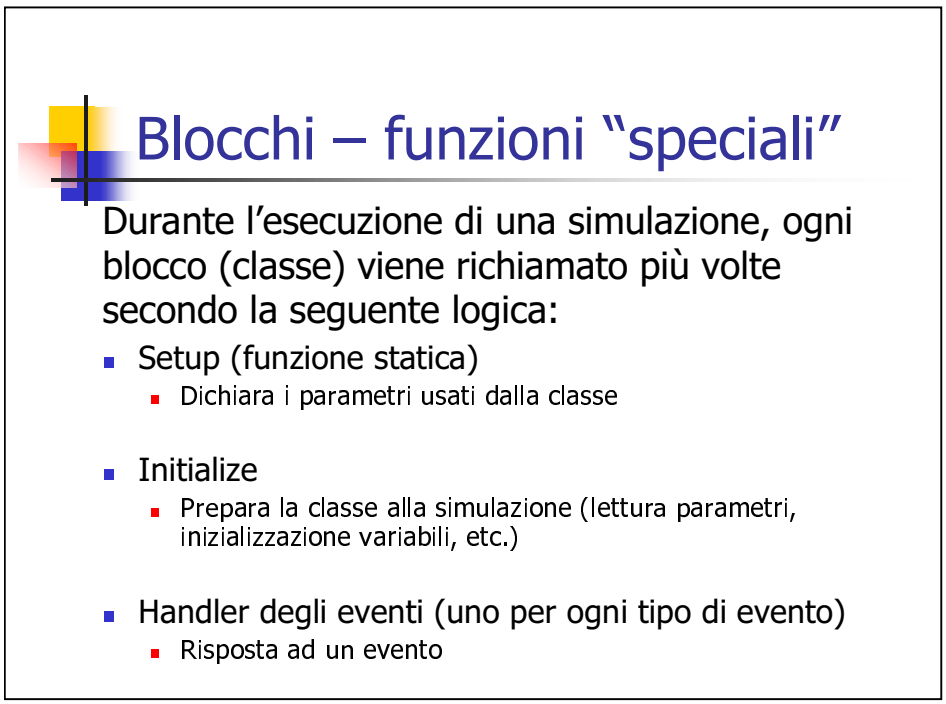

Il blocco GenGeom si riferirà al tempo mandando dei messaggi a sé stesso che hanno durata pari a quella voluta per scandire il tempo. Ad esempio, se GenGeom deve mandare dei messaggi (eventi) al Terminal ogni X secondi, dopo ogni messaggio a Terminal manderà a sé stesso un messaggio di durata X secondi, e solo dopo che lo avrà ricevuto genererà il nuovo evento per Terminal.
La scansione del tempo in NePSi avviene in sub-unità; in NePSi il tempo viene rappresentato con un numero di tipo ***unsigned long long int*** (indicato con ***time_int***).
La minima unità di tempo rappresentabile è denominata ***tick***; la frazione di secondo che un tick rappresenta viene decisa a priori dal programmatore. In genere si usa 1 tick = 1 μs.
La scelta di un unsigned long long int è motivata con la necessità di avere un grado di accuratezza nella determinazione del tempo anche dopo un numero elevato di eventi.




DEDevice è una classe virtuale da cui si derivano tutti i blocchi per terminali mobili; le classi generiche sono tutte di tipo ***virtual*** per evitare che vengano utilizzate, essendo esse implementate solo in qualità di classi padre in meccanismi di ereditarietà. Le classi DevTerminal e DevTrafficGenerator sono state dunque costruite dal programmatore, per questo particolare esempio applicativo, per ereditarietà da DEDevice.
UplinkPckt è un pacchetto molto semplice (ad esempio una struttura C) che serve al sistema per scambiare le informazioni. E' in sostanza l'informazione associata al messaggio di uplink.
DEEvent è la classe virtuale da cui derivano gli eventi. Un evento è in realtà una classe, e ciò per dare possibilità allo scheduler di sistema (l'oggetto che gestisce gli eventi e l'asse temporale del simulatore) di gestire più facilmente gli eventi. Se ad esempio si assegna ad ogni classe evento una variabile membro relativa al tempo, lo scheduler non si dovrà preoccupare di cosa quella classe esegua, ma potrà semplicemente leggere la variabile tempo di tutte le classi evento in coda e provvedere ad eseguirle in base a quale ha il tempo di arrivo minore.
DECheckEvent è una classe che serve al sistema per auto-controllarsi durante la simulazione.
Le classi indicate con colore chiaro derivate da DEEvent sono gli eventi relativi all'esempio in esame, e costituiscono gli eventi che i vari blocchi mandano a sé stessi o al Terminal. Ad esempio, Ev_Geom_MsgArrival è l'evento che il GenGeom manda a sé stesso per scandire l'intertempo di generazione eventi al Terminal, Ev_MMP_GoStatus_A è l'evento utile al generatore MMP per cambiare stato, e così via.
Altre classi utili, mostrate in figura, sono quelle relative ai generatori casuali, sia di numeri generici (blocco sopra) che di tempo (la coppia sottostante di classi relative a DETime).




Dalla classe DESystem si ottiene per derivazione SysTest, che definisce quali classi operano nel simulatore, e che contiene il "main".
DETime si occupa del tempo, dei tick, e della precisione.
DEScheduler si occupa di sapere qual è il prossimo evento, ed è l'istanza dell'oggetto Scheduler descritto in precedenza. Questa classe non deve di norma essere modificata dai programmatori utenti di NePSi.
In figura sono poi da notare tre blocchi principali di classi, quelle relative ai Parametri, che si occupano di lettura e scrittura dei parametri di simulazione, quelle dell'Analyser e quelle del Probe, che generano e gestiscono le statistiche da osservare. Fra questi due ultimi gruppi di classi, la principale differenza sta nel fatto che Analyser genera solo un numero limitato di parametri statistici e può decidere se interrompere o meno la simulazione analizzando tali dati, mentre Probe può generare più tipi di dati statistici ma non può agire sulla simulazione.




Sono le funzioni membro necessarie ad ogni blocco derivante dal DEDevice. La Setup deve essere "static", in quanto non appartiene a nessuna istanza della classe, ed ha effetti pertanto su tutte le istanze della classe stessa.
Viene usata per modificare i parametri, e viene chiamata una volta sola per simulazione, prima che vengano letti i parametri, e deve essere eseguita per ogni classe coinvolta. La Setup dentro Systest deve chiamare tutte le Setup delle classi dei vari blocchi definiti dall'utente (le Setup delle classi interne di NePSi sono chiamate in modo automatico).
Initialize viene chiamata più volte, ma sempre a tempo zero; ad esempio, se si vogliono fare simulazioni multi-run, in cui cioè si ripetono più prove di simulazione in sequenza, ogni volta che automaticamente si riavvia una nuova simulazione, tornando così al tempo zero, si effettua una nuova chiamata a Initialize, che rappresenta il costruttore degli oggetti.
Gli Handlers degli eventi sono poi le funzioni tramite cui una classe reagisce all'arrivo di un evento; Ogni classe avrà un Handler per ogni tipo di evento che gli può arrivare.

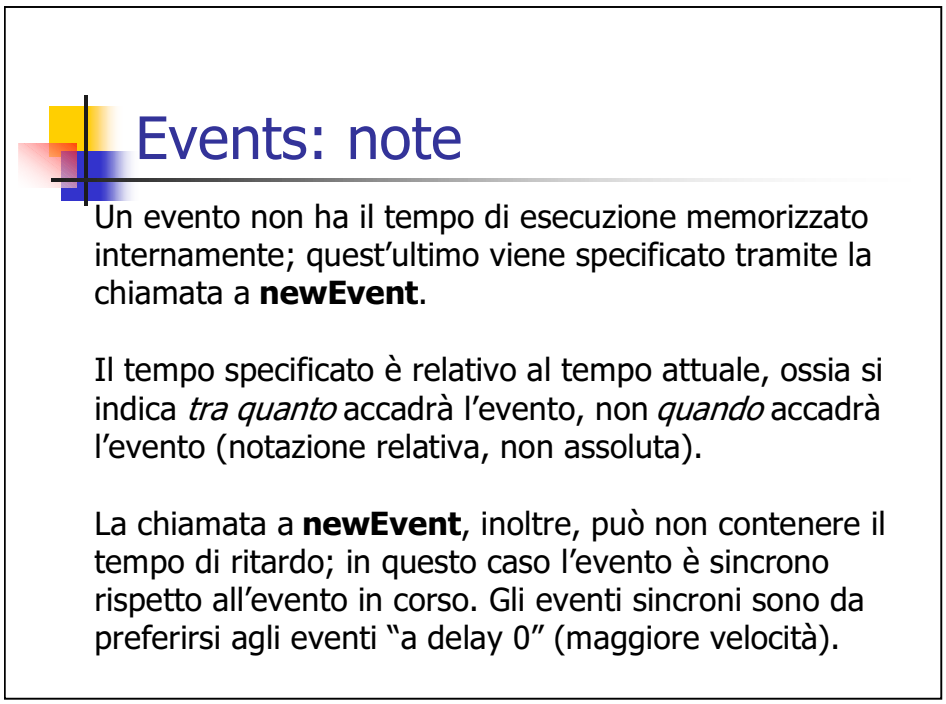
## Blocchi – funzioni "speciali" - 2

- Setup (funzione statica)
  - Viene eseguita UNA sola volta per esecuzione del programma e non deve contenere riferimenti non statici.
- Initialize
  - Viene eseguita più volte durante l'esecuzione, sempre all'inizio di un RUN (durante un'esecuzione si possono fare più RUN).
- Handler degli eventi (uno per ogni tipo di evento)
  - E' buona norma avere un handler per tipo di evento, ma nulla vieta di avere un handler che gestisce più tipi differenti.

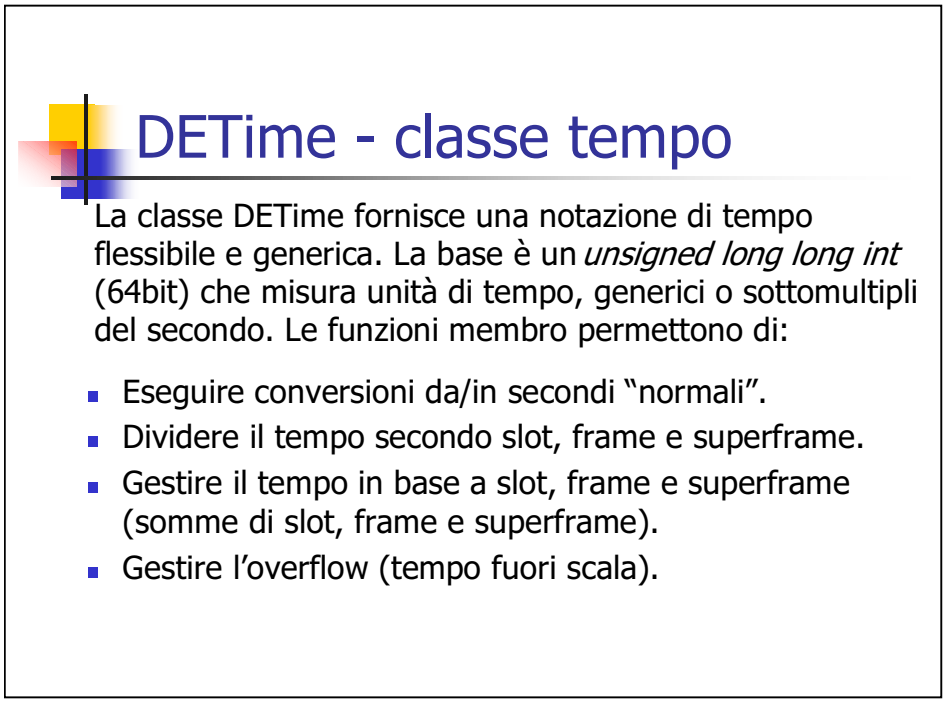
## Note e considerazioni:

Per costruire una nuova simulazione la cosa più semplice è copiare una simulazione già fatta e modificala. In questo modo si ha che:

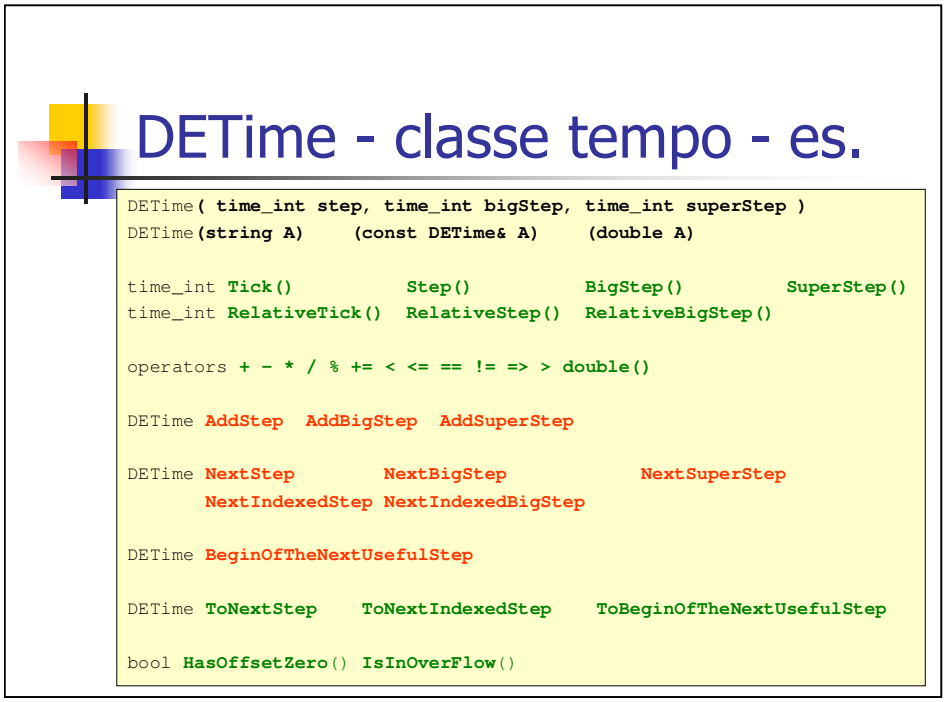
- Si ha uno stile di programmazione uniforme.
- Si evita di "dimenticarsi" alcune funzioni standard (in molte classi derivate si *devono* ridefinire alcune funzioni)
- Dagli esempi si *impara*



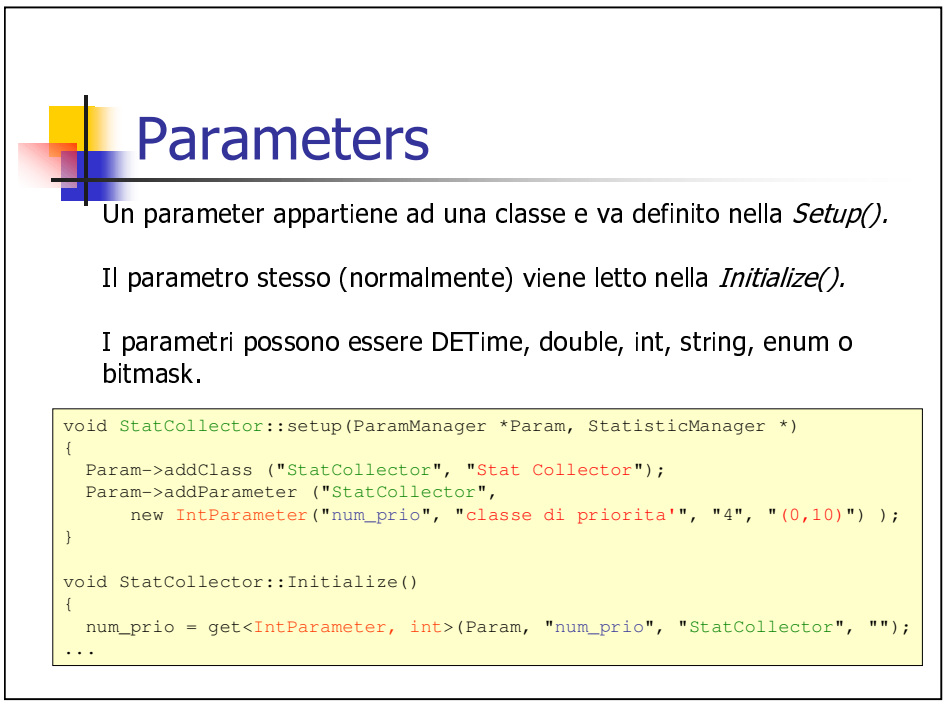

Qaundo si genera un evento (tipicamente all'interno dell'Handler di una altro evento), si comunicherà allo scheduler fra quanto tempo accadrà l'evento che si sta generando, tramite la chiamata alla funzione newEvent.
Il tempo è chiamato in DETime, e può essere espresso in tick, in secondi o in una forma utile alla rappresentazione del tempo in Slot/Frame/SuperFrame.



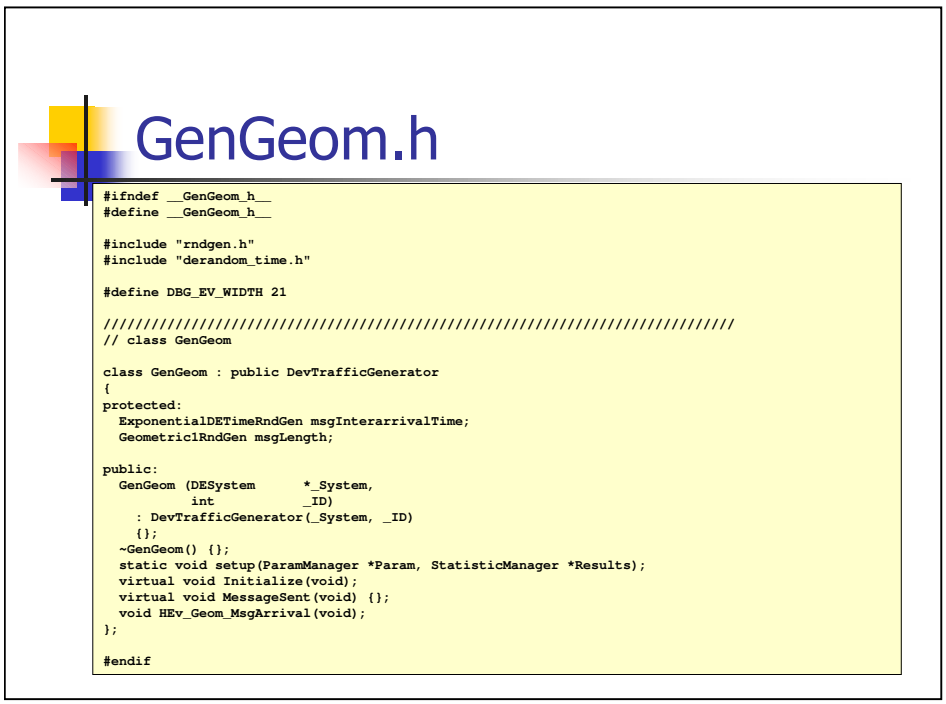
Il DETime può determinare quanti secondi compongano uno slot, quanti slot compongano un frame, e così via.
E' possibile chiedere a questo blocco in che slot/frame/superframe la simulazione si trova, oppure si può chiedere a che tempo corrisponde lo slot (o frame o superframe) successivo a quello attuale.



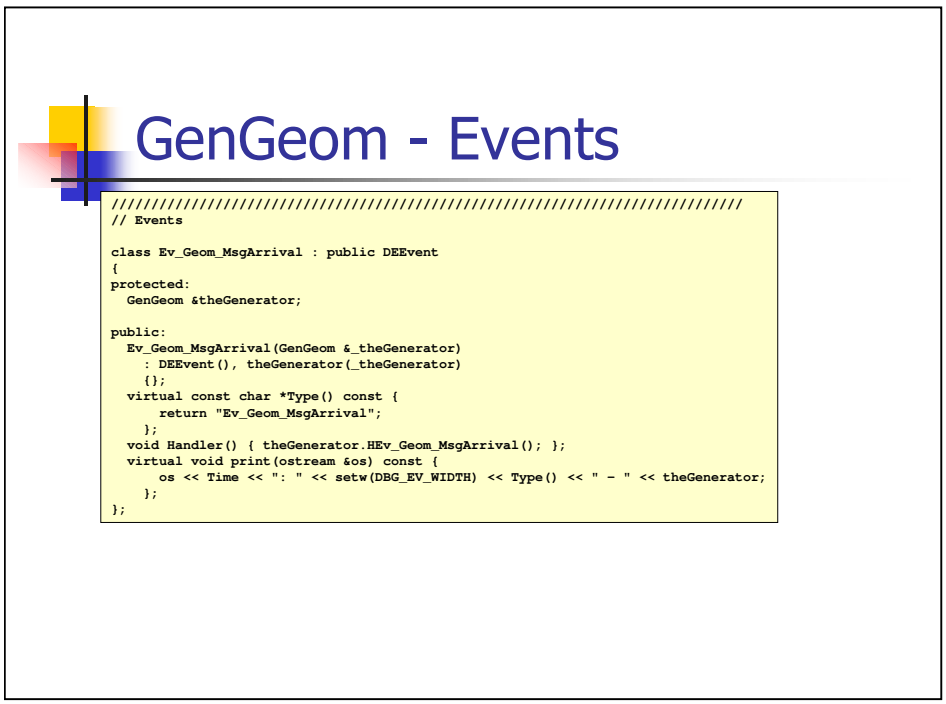

Nelle prime due righe del listato in figura si possono notare vari tipi di costruttori per la classe DETime, in cui a seconda dei parametri passati la classe DETime viene istanziata in modo diverso.
Ci sono poi gli operatori funzione, che ritornano valori di tick o secondi.
Gli operatori definiti sono poi tutti definiti tramite overloading.
Le funzioni in carattere rosso non vengono usate in quanto considerano valori assoluti di tempo. Ciò che interessa comunicare con lo scheduler è infatti "fra quanto" accadrà l'evento successivo, non "quando" in termini assoluti.
Le funzioni finali, che ritornano valori booleani, sono usate per sapere se si è all'inizio di uno slot oppure se si è in situazione di overflow, ossia se il tempo che è stato generato è troppo elevato; in tal caso, lo scheduler gestirà ugualmente questo evento in overflow, ma lo eseguirà a fine simulazione (o mai), marcandolo come "evento in overflow".



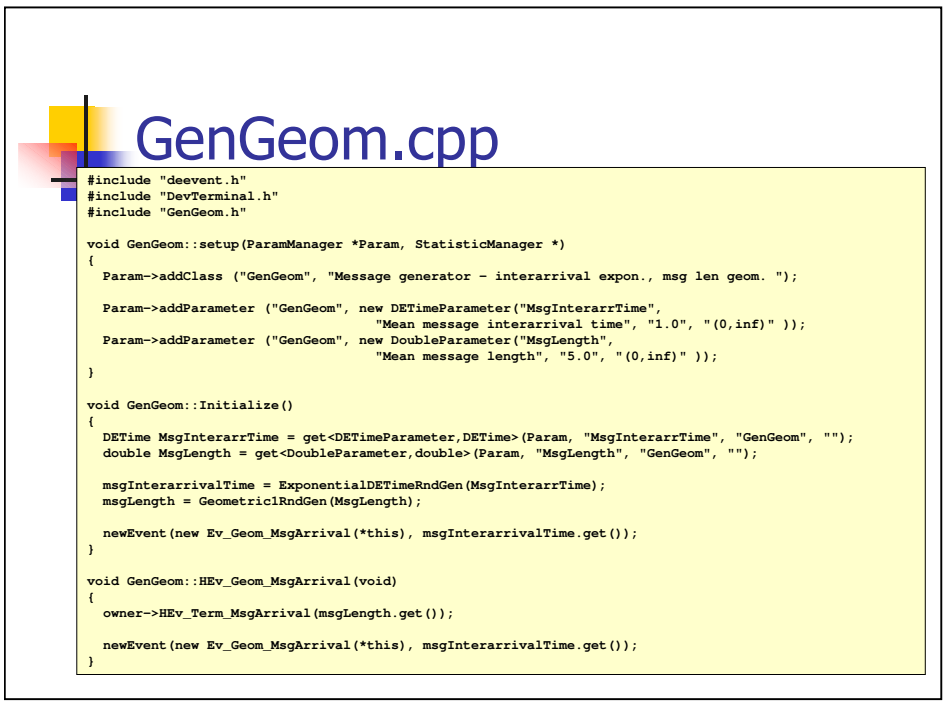

I parametri possono essere usati soltanto da classi derivanti da DEDevice, e ciò perché un evento non ha bisogno di parametri. “Stat collector” in carattere verde è il nome della classe, mentre quello in carattere blu è il commento che apparirà sul listato dei parametri.
“num_prio” in viola rappresenta il nome del parametro, a cui si associa ancora il commento in carattere blu; dentro Initialize si indica di che tipo è il parametro (Int, Double, etc.).
Dentro Initialize, “num_prio” in nero è una variabile, che viene inizializzata al valore letto dal fine dei parametri tramite il comando “get” predefinito.
IntParameter rappresenta il tipo di parametro, mentre int è la conversione che viene effettuata su quel tipo. C’è poi il nome del puntatore (Param), il il nome del parametro, il nome della classe (StatCollector) e infine un campo che nell’esempio è vuoto (non viene utilizzato).



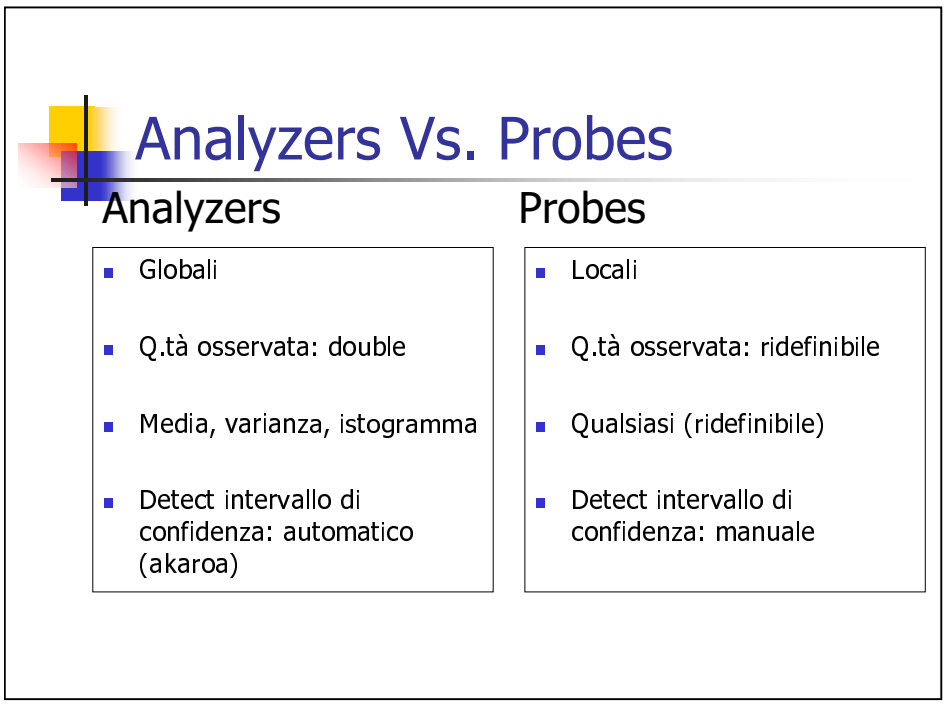

Il file mostrato in figura è un header di una classe Nepsi, deriva da DevTraffic Generator che a sua volta deriva dalla classe DeDevice. I comandi #ifndef, #define ed #endif servono per evitare la doppia inclusione. Vengono poi incluse le definizioni delle classi random. Si definisce poi la classe GenGeom, con due variabili interne, una che definisce una esponenziale, e l'altra una distribuzione geometrica.
Nella parte public si trovano le funzioni: il costruttore, che deve contenere un riferimento a DESystem (per costruire un oggetto GenGeom devo in realtà costruire un oggetto DEDevice che richiede DESystem). C'è poi il distruttore, la Setup, che deve essere dichiarata come static void, come richiesto ad ogni classe che deriva da DEDevice. Nelle classi derivate da DEEvent non servono invece né parametri né results né Setup. Ci sono poi le funzioni virtual void Inizialize e MessageSent, che gestisce l'evento ricezione Message_Sent da Terminal. L'handler è detto HEv_Geom_MsgArrival (void), che deve restituire un void. Occorre notare che un handler di evento deve restituire un void, ma che non è detto che venga invocato senza parametri (quindi con un void). La maggior parte degli handler di eventi sono della forma "void HEv_qualcosa(parametri).



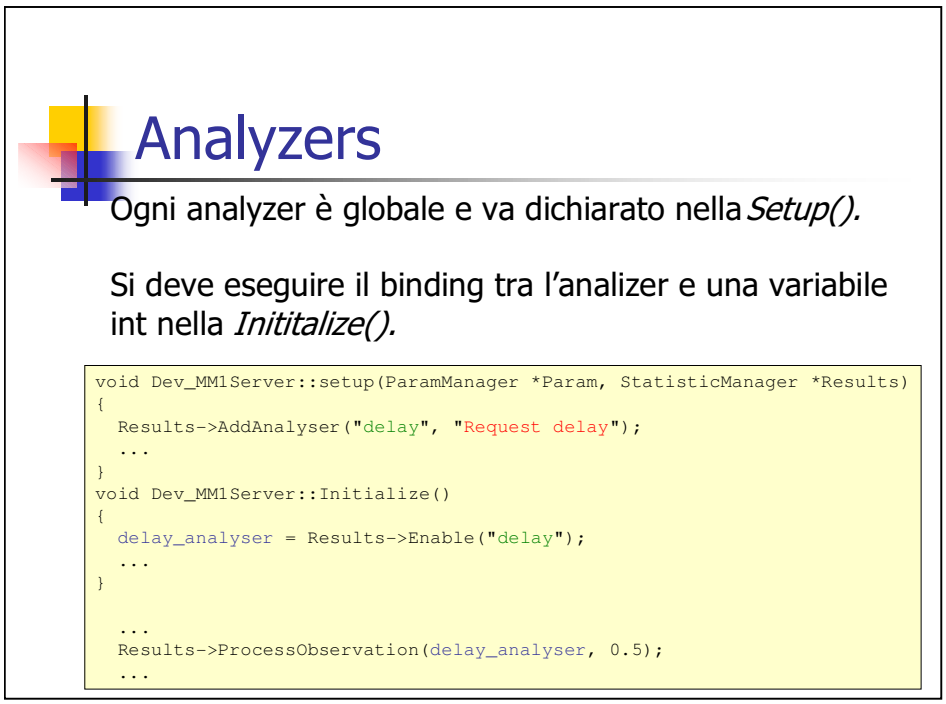

Questa è la seconda parte della schermata precedente (nel listato sta prima di "#endif".
L'evento che appartiene alla GenGeom viene detto Ev_Geom_etc.. E' l'evento mandato a questa classe, e viene definito nella relativa intestazione .h.
E' derivato da DEEvent, contiene un puntatore a GenGeom, cioè alla classe destinataria dell'evento (sé stessa).
Nel costruttore si indica il destinatario. C'è poi una funzione di servizio che stampa il nome della classe, e si chiama *Type.
C'è poi la funzione print che serve al Debugger per stampare l'evento.
Dopo << Type ( ) << ci metto quello che voglio verrà scritto nel debugger.
Poi, void Handler ( ) è la funzione chiamata dallo scheduler del Nepsi quando si accorge che quell'evento è il prossimo da eseguire, e che richiama la funzione HEV_Geom_MsgArrival ( ), che è definita nel .h come prototipo (vedi pag. precedente).



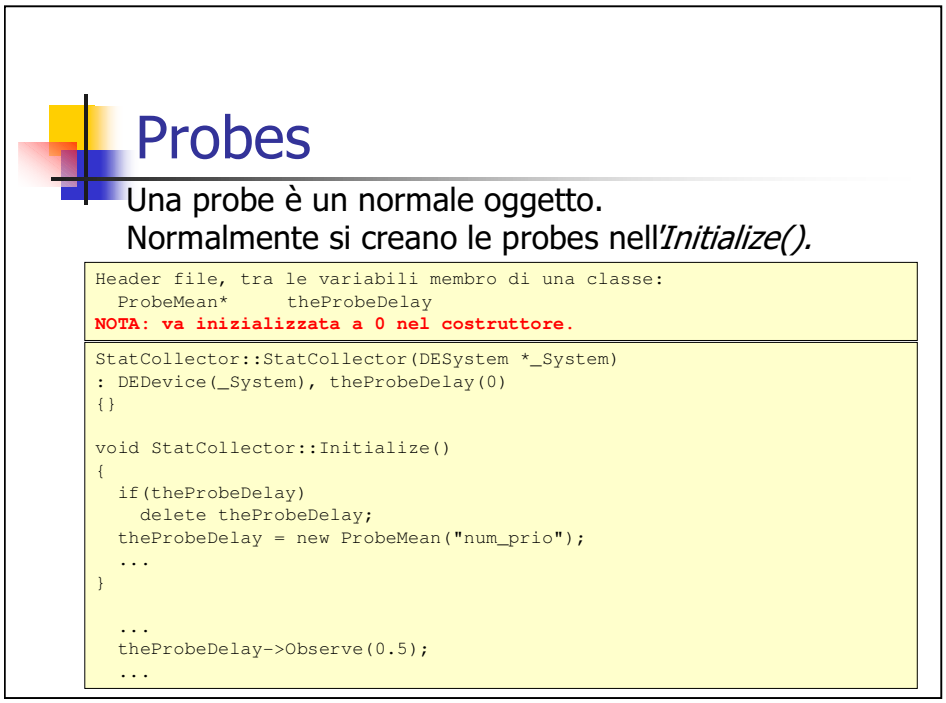

File GenGeom.cpp.
Nel Setup dichiaro due parametri, che nell'Initialize leggo e metto in due variabili. Poi mi costruisco due variabili random, e metto un newEvent al tempo del prossimo messaggio in arrivo, in cui comunico quale oggetto riceverà l'evento (lui stesso in questo caso) e fra quanto tempo (il tempo di interarrivo dei messaggi). Poi c'è l'handler dell'evento: prima uso owner (=proprietario del generatore), poi mi rischedulo e c'è dunque un'altra chiamata a newEvent.
Owner è il puntatore al DevTerminal a cui notifico l'arrivo del messaggio. Avrei potuto metterlo con un evento (cioè con un newEvent anziché usare owner). Un evento è utile per fare cose nel futuro. Se ho eventi sincroni posso evitare di usare eventi, chiamo cioè direttamente la funzione del destinatario. In realtà gli eventi vengono scritti nel log del debugger. Poi, se uso la newEvent, prima del prossimo evento questa funzione finirà.
Quindi si sarebbe potuto scrivere, anziché owner:
newEvent (new Ev_msgArrival (owner, msgLength.get()) )
oppure:
newEvent (new Ev_msgArrival (owner, msgLength.get()), DETime() ),




Un Analyzer lo definisco nella setup ed è comune a tutte le istanze della classe. Ogni istanza di quella classe lavora sullo stesso Analyzer. Automaticamente sa quando stoppare la simulazione. L'intervallo di confidenza è dunque gestito in automatico.
Le probes sono classi, sono locali, e se voglio una probe per trenta terminali la devo definire come statica.
Il metodo Akaroa definisce delle soglie per l'intervallo di confidenza della variabile osservata, e al momento in cui tale soglia viene superata l'Analyzer automaticamente interviene, eventualmente interrompendo la simulazione.




Un Analyzer lo definisco in modo simile ad un parametro. Devo aggiungere un unico Analyzer. Nell'Initialize Delay_Analyzer abilito l'analizzatore di Delay.




Le Probe sono oggetti, si creano nell'Initialize, se sono state definite come statiche nel setup. Vanno inizializzate a 0 nel costruttore per sicurezza.

## Probes - tipi

Tipi di Probe già presenti in NePSi; si può derivare per ereditarietà dalla classe *Probe.*

| | |
|---|---|
| ProbeMean | Time, E[x], E[$x^2$], E[$x^2$] - E[x]$^2$ |
| ProbeMinMax | Time, min(x), max(x) |
| ProbeHistogram | Istogramma(x) |
| ProbeBand | Time, sum(x)/Time |
| ProbeSlice | Come ProbeMean, ma le statistiche sono azzerate ad ogni output |

## Note di utilizzo:

NePSi è un framework in continua evoluzione, quindi non è perfetto e non risolve *tutti* i problemi di creazione di una simulazione. Si può usare in due modi:

☹ Modo cattivo:

- Si forza la struttura se NePSi impone limitazioni.
- Non si guardano gli header NePSi.
- Si fanno troppe domande ai gestori di NePSi ;-{))

☺ Modo buono:

- Se si trovano limiti imposti, ci si chiede perché. Eventualmente si chiedono chiarimenti e/o si suggeriscono modifiche.
- Si esplora NePSi (le parti "pubbliche").
- Si scrive la documentazione ;-{))