UNIVERSIDAD DE EXTREMADURA

CENTRO UNIVERSITARIO DE MÉRIDA

GRADO EN INGENIERÍA EN TELEMÁTICA

TRABAJO FIN DE GRADO

Diseño y desarrollo de un simulador de movilidad IPv6 para redes de próxima generación

Autor: Carlos Dalia Pacheco

Fdo.:

Director: Javier Carmona Murillo

Fdo.:

Mérida, Julio de 2015

AGRADECIMIENTOS

Quería dedicar este Trabajo Fin de Grado a mis padres, quienes con su esfuerzo realizado a lo largo de su vida han hecho posible que yo pudiera recibir la educación que tengo hoy en día y ser lo que hoy soy.

Me gustaría agradecer también a todos mis compañeros y amigos del Grado en Ingeniería en Telemática por aguantarme durante estos años de estudio y por todo el apoyo recibido tanto en los buenos como en los malos momentos.

Así mismo, quiero dar las gracias a mi profesor y director de este Trabajo Fin de Grado, Javier, por sus explicaciones y conocimientos durante el transcurso del mismo, por su paciencia y, sobre todo, por su ayuda incondicional sacando huecos en la agenda donde no los había.

***Gracias** a todos.*

RESUMEN

El tráfico en Internet está aumentando considerablemente en los últimos años e incluso se espera que el crecimiento sea aún mayor con las futuras redes de acceso LTE (Long Term Evolution). Por tanto, los mecanismos de gestión de la movilidad en IPv6 juegan un papel muy importante en estas arquitecturas, para gestionar la movilidad entre redes heterogéneas de forma eficiente. Distintas organizaciones de estandarización como el IETF (Internet Engineering Task Force) o el 3GPP (3rd Generation Partnership Project) están desarrollando nuevos protocolos capaces de gestionar esta movilidad en redes de próxima generación.

En el diseño de estos protocolos y arquitecturas, la estimación del rendimiento es fundamental, así como el análisis del comportamiento del sistema completo y sus componentes. Y debido a la propia naturaleza de las redes móviles, la simulación resulta de gran interés.

Por eso, se ha desarrollado este Trabajo Fin de Grado, cuyo objetivo principal es el estudio de los diferentes protocolos de gestión de la movilidad en IPv6 y la evaluación de su rendimiento. Para ello, se ha diseñado un simulador que permite analizar y medir el rendimiento de estos nuevos protocolos bajo diferentes circunstancias, como son la tipología de la red de acceso, los modelos de movimiento y los modelos de tráfico. Una vez simulados los diferentes eventos, se obtienen datos y gráficas muy útiles para el estudio que se nos presenta.

ABSTRACT

Mobile Internet data traffic is increasing significantly in recent years and it is expected to grow even higher with the future deployment of the LTE (Long Term Evolution) access networks. In this context, IPv6 mobility management protocols play an important role in these architectures to track efficiently the user's mobility through heterogeneous networks. Different mobility management protocols are being developed by several organizations such as the IETF (Internet Engineering Task Force) or the 3GPP (3rd Generation Partnership Project).

During the design of these protocols and architectures, the performance evaluation is a crucial step, as well as the understanding and visualization of the micro and macro behaviour of the systems and their components. And, due to the highly dynamic behaviour of mobile networks, simulation is of great interest.

So, the main goal of this work is the study of the different IPv6 management protocols and their performance evaluation. For this reason, a simulator has been designed in order to analyse and measure the performance of these new protocols under different circumstances, such as the type of the access network or mobility and traffic models. Once simulated all the events, useful data and graphs are obtained.

ÍNDICE

Índice de figuras

1. INTRODUCCIÓN

1.1. Motivación

Con la realización de este Trabajo Fin de Grado se tiene como propósito poner en práctica los conocimientos y habilidades adquiridos por el alumno Carlos Dalia Pacheco durante su formación académica en el Grado en Ingeniería en Telemática.

Corresponde a la asignatura Trabajo Fin de Grado, de 12 créditos ECTS. Es requisito haber obtenido el resto de créditos ECTS de los que se compone el Plan de Estudios para proceder a la defensa de este Trabajo.

1.2. Estructura del documento

En este Capítulo 1 se hace una breve descripción de esta memoria escrita. En el Capítulo 2 se engloba el marco teórico, explicando los dos protocolos objeto de estudio: MIPv6 y PMIPv6.

En el Capítulo 3 se hace una introducción y se explica el desarrollo del trabajo de simulación, cuyos resultados serán mostrados en el Capítulo 4. En el Capítulo 5 se extraen algunas conclusiones y líneas de trabajo futuro. Finalmente, el Capítulo 6, con la bibliografía utilizada, cierra este documento.

1.3. Metodología y objetivos

Para desarrollar el presente Trabajo Fin de Grado se ha utilizado el software Matlab R2014a sobre un sistema operativo Windows 8.1. También, hemos utilizado la herramienta *Trello* (https://trello.com/) para planificar y llevar a cabo las diferentes tareas.

Para realizar este trabajo, primero se procedió a un análisis y estudio exhaustivo de los principales protocolos de gestión de movilidad en IPv6, así como de las topologías, modelos de tráfico y modelos de movimiento posibles, para después pensar en el diseño del simulador e implementarlo. Una vez realizado todo lo anterior, se procedió a realizar distintas pruebas para la evaluación del rendimiento de los protocolos implementados sobre el simulador, a través de gráficas y datos obtenidos.

Este simulador de eventos permite analizar y medir el rendimiento de estos protocolos bajo distintas circunstancias, como son la topología de la red de acceso, los modelos de movimiento y los modelos de tráfico; ofreciendo un entorno controlado sobre el cual se puede entender el comportamiento de estos sistemas en mayor detalle.

2. MOVILIDAD EN INTERNET

En los últimos años se está produciendo un crecimiento en el tráfico en Internet debido principalmente a las nuevas tecnologías inalámbricas de banda ancha y a la cada vez mayor necesidad de conectividad debido a nuevos servicios disponibles a través de tecnologías móviles.

Este crecimiento se espera incluso que vaya a más con las futuras redes de acceso LTE (*Long Term Evolution*), que incrementarán de forma significativa el ancho de banda disponible. Algunos informes muestran que este aumento llegará a multiplicar por 11 el tráfico actual de Internet móvil en el año 2018.

Los operadores de red, junto con investigadores y el resto de la industria, están diseñando soluciones más eficientes que puedan gestionar este incremento.

Lo que se persigue principalmente es la provisión de conectividad, independientemente del lugar donde el móvil se encuentre.

Podemos entender "movilidad" de distintas formas: a nivel de red, de transporte y aplicación. Además, conviene diferenciar entre **movilidad** y **portabilidad**:

- **Movilidad:** la conexión no se interrumpe al cambiar de punto de acceso. Así mismo, la dirección IP no se cambia.

- **Portabilidad:** sólo garantiza que la comunicación se pueda ser establecida, pero no necesariamente usando la misma IP, lo que conllevaría a la interrupción de las comunicaciones en curso.

Uno de los principales retos que tienen que ser abordados es cómo manejar el tráfico y los usuarios en la red de transporte. Un aspecto fundamental durante el diseño de estos protocolos y arquitecturas es la estimación de su rendimiento, así como el análisis del comportamiento del sistema completo y de sus componentes. Generalmente, esta evaluación se realiza mediante alguna de las siguientes metodologías:

1. Experimentación con sistemas reales y prototipos.
2. Análisis matemático.
3. Simulación.

Por tanto, debido a la propia naturaleza de las redes móviles, donde distintas topologías, modelos de movilidad y modelos de tráfico deben ser analizados, la simulación resulta de gran interés.

En nuestro caso, nos centraremos en la movilidad ofrecida en la capa de red, usando unos protocolos desarrollados por el IETF (*Internet Engineering Task Force*) y conocido como Mobile IP y Proxy Mobile IP.

El soporte a la movilidad se desarrolló para IPv4 y también para IPv6. Sin embargo, desde el punto de vista del diseño, la situación de ambos protocolos no es la misma. Como IPv4 fue desarrollado mucho antes de que se concibieran escenarios de movilidad, los mecanismos de movilidad fueron incorporados a modo de extensiones al protocolo. Como consecuencia de esto, algunas extensiones, aunque presentan sustanciales ventajas desde el punto de vista técnico, son difíciles de desplegar a gran escala. Por otro lado, en IPv6 el soporte a la movilidad se consideró desde el primer momento del diseño.

2.1. El problema de la movilidad en Internet

2.1.1. Cuestiones básicas para el soporte de la movilidad

Principalmente se busca mantener abiertas (en funcionamiento) las comunicaciones cuando un dispositivo se mueva a otras redes diferentes, es decir, cuando cambie de punto de acceso.

Las cuestiones básicas que hay que tener en cuenta son:

- **Proceso de *handover* o *handoff* (paso de una celda de transmisión a otra):** la función más importante que se necesita para dar soporte a la movilidad es mantener abiertas las comunicaciones, mientras el nodo móvil se mueve y cambia de punto de acceso. Su principal objetivo es minimizar el tiempo de desconexión o pérdida del servicio durante el *handover*. Este proceso puede ser dos tipos: el

proceso de *handover* que ocurre cuando el traspaso se realiza entre redes homogéneas y el proceso de *handover* que ocurre cuando el traspaso se realiza entre redes heterogéneas (mucho más conflictivo).

- **Gestión de la localización:** tiene como objetivo identificar la ubicación actual del nodo móvil y mantener el camino seguido por él. Incluye dos tareas principales. La primera es el registro de ubicación o la actualización de la ubicación, en la que el terminal informa periódicamente al sistema para actualizar las bases de datos de ubicación con la información actualizada del lugar en que se encuentra. La segunda se denomina búsqueda automática, en la que el sistema determina la ubicación actual del terminal, cuando inicia una comunicación en base a la información disponible en las bases de datos del sistema. Esta búsqueda tiene dos partes principales: En primer lugar, determinar la base de datos del terminal llamado y la ubicación de subred. En segundo lugar, un proceso consistente en avisar al nodo a partir de la información obtenida en el proceso de localización anterior. por medio de mensajes de sondeo enviados a las subredes dentro del área de registro del terminal con quien se quiere comunicar.

- ***Multihoming:*** el nodo móvil puede acceder a Internet a través de múltiples enlaces simultáneamente, y seleccionar de forma dinámica el enlace que se desee en cada momento.

- **Aplicaciones:** el mecanismo de movilidad deberá ser trasparente, de este modo las aplicaciones o servicios que se estén usando en un momento determinado no sufrirán cambios.

2.1.2. Cuestiones de rendimiento para el soporte de la movilidad

También hay que prestar especial atención a las métricas de rendimiento que tienen que ver con el soporte de movilidad. Éstas son:

- ***Handover Latency:*** tiempo transcurrido desde que se recibe el último paquete de datos proveniente de la red antigua hasta que llega el primer paquete de datos proveniente de la red nueva durante el proceso de *handover*.

- ***Packet Delivery Cost***: coste del envío de tráfico de datos a o desde dispositivos móviles. Esta métrica es relevante, ya que el camino de los datos se modifica según indican los protocolos de gestión de movilidad.
- ***Tunnelling Cost***: sobrecarga provocada por el coste adicional de los mecanismos de *tunnelling* necesarios para el envío de tráfico.
- ***Packet Loss:*** número de paquetes perdidos durante el proceso de *handover*.
- ***Signaling Overhead:*** número de mensajes utilizados para los procesos de *handover* y de ubicación.
- ***Throughput:*** cantidad de datos transmitidos por la "Internet móvil" durante un periodo de tiempo determinado.

2.1.3. Limitaciones de TCP/IP para la movilidad en Internet

TCP/IP fue diseñado originalmente para redes de dispositivos fijos. Por tanto, cuando se quiso adaptar la pila de protocolos para dar soporte a la movilidad, surgieron una serie de limitaciones de TCP/IP tales como:

- Limitaciones de la capa de enlace.
- Limitaciones de las direcciones IP.
- Falta de capas o niveles de apoyo para la movilidad.
- Limitaciones en las aplicaciones.

2.2. Mobile IP

Mobile IP es un protocolo de capa de red concebido para proporcionar movilidad a terminales móviles. Fue diseñado por el IETF en dos versiones, basadas en IPv4 e IPv6.

El objetivo de ambos protocolos es permitir que usuarios que se mueven por diferentes áreas mantengan sus conexiones mientras cambian de punto de acceso.

2.2.1. Mobile IPv4 (RFC 5944)

En Mobile IPv4 se pueden distinguir las siguientes entidades funcionales:

- ***Mobile Node (MN):*** dispositivo móvil que puede cambiar su punto de conexión de una red a otra sin cambiar su dirección IP, y manteniendo activas las comunicaciones de niveles superiores.

- ***Home Agent (HA):*** router de la red propia que gestiona la localización del MN. Contiene una lista de nodos móviles (MNs) registrados. Responsable de interceptar y de hacer llegar al nodo móvil aquellos paquetes que son dirigidos a él mientras se encuentra fuera de la red propia.

- ***Foreign Agent (FA):*** router de la red visitada que coopera con el HA para proporcionar movilidad. Asiste al nodo móvil informando al HA de su actual *care-of address (CoA).*

- ***Care-of address (CoA):*** dirección IP local que identifica a un nodo móvil en su actual ubicación.

- ***Correspondant Node (CN):*** nodo fijo o móvil que se comunica con el MN.

- ***Home address:*** dirección IP permanente que se asigna al MN.

- ***Tunnel:*** camino que recorren los paquetes encapsulados desde el HA hasta el FA.

Mobile IP usa dos direcciones IP: la fija (*home address*) y la CoA usada para la movilidad del MN.

2.2.1.1. Arquitectura y operaciones de Mobile IPv4

El protocolo establece tres fases:

1. ***Agent Discovery:*** el MN debe ser capaz de detectar si está conectado a su red propia o a una visitada. Para ello el HA y el FA envían periódicamente mensajes (extensión de *ICMP Router Discovery*). Cuando el MN recibe este mensaje, es capaz de determinar en qué red esta conectado, y si es una red visitada obtiene una *Care-Of Address (CoA).*

2. ***Registration:*** el MN registra su CoA en el HA. El MN envía un *Registration Request* al FA, que, a su vez, lo reenvía al HA. El HA responde con un mensaje de *Registration Reply* aceptando la solicitud. A partir de este punto, el HA conoce

la localización del MN y la comunicación puede empezar, o continuar en caso de *handoff*. Este proceso se conoce también como ***binding update.***

3. ***Tunnelling:*** el CN se comunica con el MN (y viceversa). Cuando un CN envía un paquete IP al MN, la dirección de destino es la dirección asignada al MN en la red propia. Cuando este paquete llega a la red propia del MN, este es interceptado, encapsulado y reenviado por el HA al FA, que, a su vez, lo desencapsula y lo entrega al MN. Por otro lado, cuando el MN envía un paquete al CN, lo envía directamente utilizando su *home address* como dirección origen. Los paquetes enviado por el CN al MN son encaminados a través del HA. Este encaminamiento asimétrico, que no suele ser el óptimo, se conoce como *triangle routing.*

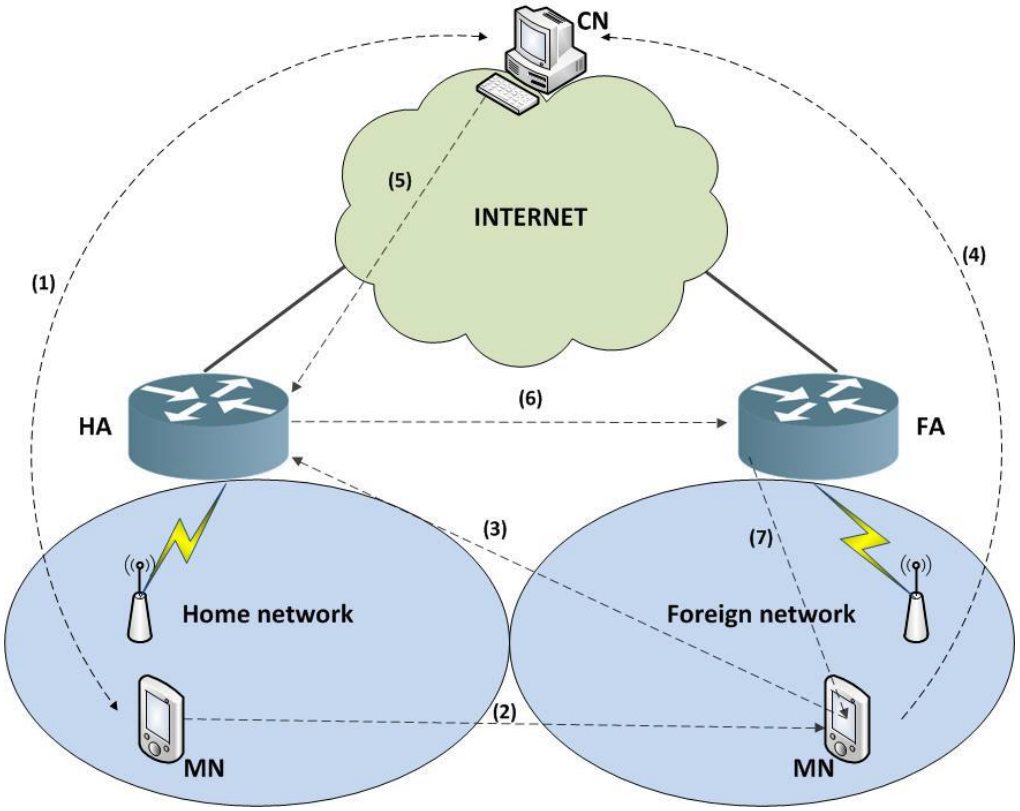


Figura 1: Esquema básico de MIPv4

1. Comunicación normal entre el MN y el CN.
1. El MN se mueve desde la red propia *(home network)* a otra red *(foreign network).*
2. El MN registra su *Care-Of address.*
3. El MN envía paquetes directamente al CN.
4. El CN envía paquetes a la red propia del MN.
5. HA tuneliza los paquetes hacia FA.
6. FA entrega los paquetes recibidos al MN.

2.2.2. Mobile IPv6 (RFC 6275)

Mobile IPv6 es muy similar a Mobile IPv4. Sin embargo, la movilidad en IPv4 no fue considerada durante la fase de diseño del protocolo, sino que es un añadido posterior.

En cambio en IPv6, la movilidad se tuvo en cuenta desde el diseño inicial y está totalmente integrada. Por este motivo. MIPv6 es más eficiente y evita algunos problemas sufridos por MIPv4. Entre otros, Mobile IPv6 no requiere FA, puesto que la autoconfiguración de direcciones de IPv6 proporciona la funcionalidad requerida en la fase de *Agent Advertisement.* La autoconfiguración de direcciones IPv6 permite a un nodo obtener una dirección IP de forma dinámica a través de la red. IPv6 incorpora dos métodos de autoconfiguración:

- El primero consiste en que el nodo se comunica con un router local utilizando un protocolo simple de solicitud-respuesta (*método stateless*).
- El segundo método consiste en que el nodo se comunica con un servidor de direcciones de la organización utilizando un protocolo de aplicación denominado DHCP (*Dynamic Host Configuration Protocol*).

En Mobile IPv6 se pueden distinguir las siguientes entidades funcionales:

- ***Mobile Node (MN):*** dispositivo móvil que puede cambiar su punto de conexión de una red a otra sin cambiar su dirección IP, y manteniendo activas las comunicaciones de niveles superiores.

- ***Home Agent (HA):*** router de la red propia que gestiona la localización del MN. Contiene una lista de nodos móviles (MNs) registrados. Responsable de interceptar y de hacer llegar al nodo móvil aquellos paquetes que son dirigidos a él mientras se encuentra fuera de la red propia.

- **Red origen o red propia *(home network):*** es la red cuyo prefijo coincide con el de la dirección permanente del nodo móvil. Los mecanismos de encaminamiento IP estándar entregarán los datagramas para la dirección permanente del nodo móvil en su red origen.

- **Red visitada *(foreign network):*** Cualquier otra red distinta de la red origen en la que se encuentra el nodo móvil y en la que ha adquirido una dirección auxiliar.

- ***Care-of Address (CoA):*** dirección IP local que identifica a un nodo móvil en su actual ubicación.

- ***Correspondant Node (CN):*** nodo fijo o móvil que se comunica con el MN.

- ***Home Address:*** dirección IP permanente que se asigna al MN.

- ***Binding Cache (BC):*** conjunto de entradas mantenidas por un HA o un CN, cada una de las cuales dispone de una asociación entre la dirección permanente de un nodo móvil y su CoA.

Mobile IPv6, a diferencia de MIPv4, utiliza una cabecera para los paquetes que incluye la dirección de la red propia (*home address*) y la CoA.

2.2.2.1. Arquitectura y operaciones de Mobile IPv6

Cabe destacar que Mobile IPv6 evita el *triangle routing*, porque cuando un CN envía un paquete a la *home address* del MN, el HA lo intercepta, encapsula y reenvía al MN. Sin embargo, el MN también puede enviar un mensaje de *Binding Update* al CN. Este mensaje incluye la CoA del MN, que se almacena en la *Binding Cache* del CN. Entonces, cualquier CN que quiera enviar un paquete, primero consulta su *Binding Cache* buscando la dirección IP destino. Si encuentra la entrada correspondiente en la cache, el paquete se envía directamente al MN usando la CoA asociada a la *home address* de éste. Esta funcionalidad es inherente a IPv6 y no requiere modificar los CN.

Según lo comentado anteriormente podemos distinguir dos tipos de funcionamientos: el básico y el optimizado:

- **Funcionamiento básico *(triangle routing)*:** Con esta forma de operar los datos enviados por el CN al nodo móvil llegarán a la red origen. El HA encapsula los datagramas y los envía a través de un túnel a la CoA del nodo móvil. Cuando éste tiene que enviar los datos al CN lo realiza también a través del túnel hasta el HA y una vez allí se encaminan de la forma estándar al CN. Este modo de

funcionamiento es totalmente transparente para el CN que no necesita ningún tipo de soporte adicional para comunicarse con un nodo móvil.

- **Funcionamiento optimizado:** Este modo de operación requiere que el CN disponga de soporte para MIPv6 ya que deberá realizar nuevas tareas y además dispondrá de una *binding cache.*

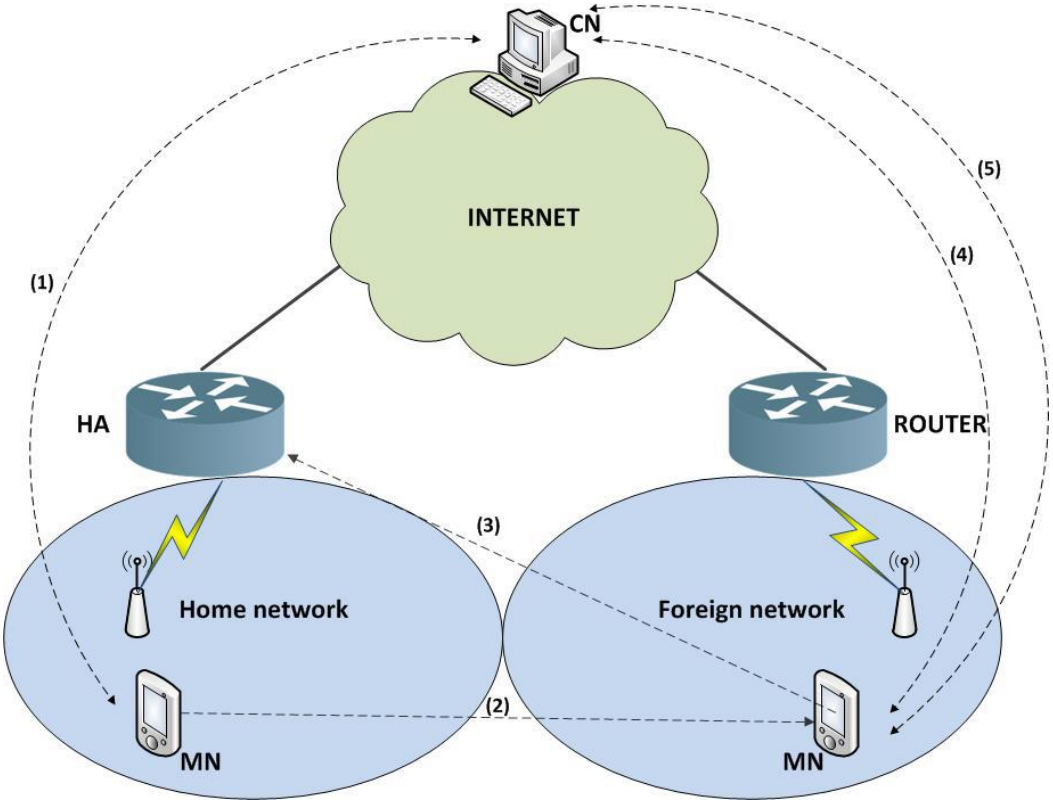


Figura 2: Esquema básico de MIPv6

1. MN comunica con CN.
2. MN se mueve a otra red.
3. MN registra su CoA en HA (mediante mensajes *binding update/acknowledgement*).
4. *Binding Update/acknowledgement.*
5. MN comunica directamente con CN.

El protocolo establece tres fases:

1. ***Agent discovery:*** cada nodo móvil con soporte MIPv6 tiene una dirección permanente que, como se ha comentado, es utilizada para enviar tráfico al nodo móvil independientemente de su punto de conexión a la red. Cuando un nodo móvil está conectado a su red origen funciona como cualquier otro nodo y las capacidades de MIPv6 no se utilizan. Sin embargo, cuando se mueve a otra red externa adquiere una CoA dentro del espacio de direcciones de esa red mediante los mecanismos convencionales de IPv6, ya sea autoconfiguración de IPv6 o DHCPv6.
2. ***Registration:*** cuando un nodo móvil está conectado a su red origen, funciona como cualquier otro nodo, y las capacidades de MIPv6 no se utilizan. Sin embargo, cuando se mueve a otra red externa, la información de encaminamiento referente a la dirección permanente del nodo móvil debe ser actualizada tanto en el HA como en cualquiera de los CN (solo si se está utilizado la optimización de rutas). MIPv6 proporciona esta funcionalidad añadiendo una estructura denominada *binding cache* en el HA y otra en CN (solo si se está utilizando la optimización de rutas). El procedimiento es el siguiente: el MN registra la CoA en el HA y también en el CN mediante mensajes *Binding Updates* (BU). El HA y el CN guardan la información recibida en su *binding cache*, y envían ambos hacia el nodo móvil mensajes de *Binding Acknowledgement* (BA).

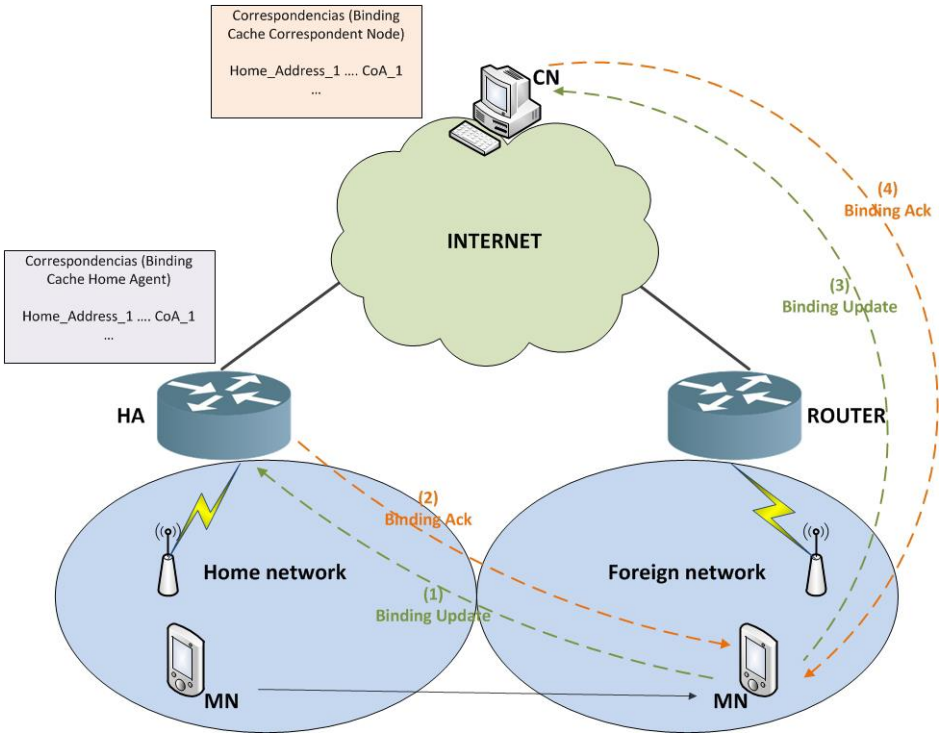


Figura 3: Fase de registro en MIPv6

3. ***Tunnelling:*** Una vez que se ha realizado el registro y que las *binding cache* están actualizadas con la nueva CoA, comenzará la transferencia de datos.

    - **Sin optimización de rutas:** Los paquetes desde el CN son encaminados al HA y desde allí se envían a través de un túnel al terminal móvil. Los paquetes que tienen como destino el CN son enviados por un túnel desde el nodo móvil al HA y desde allí son encaminados de forma normal hasta el CN. Para que esto funcione correctamente, el HA tiene que utilizar *Proxy Neighbor Discovery* para interceptar los paquetes direccionados a la *home address* del MN, y posteriormente tunelizarlos hacia la CoA del mismo.

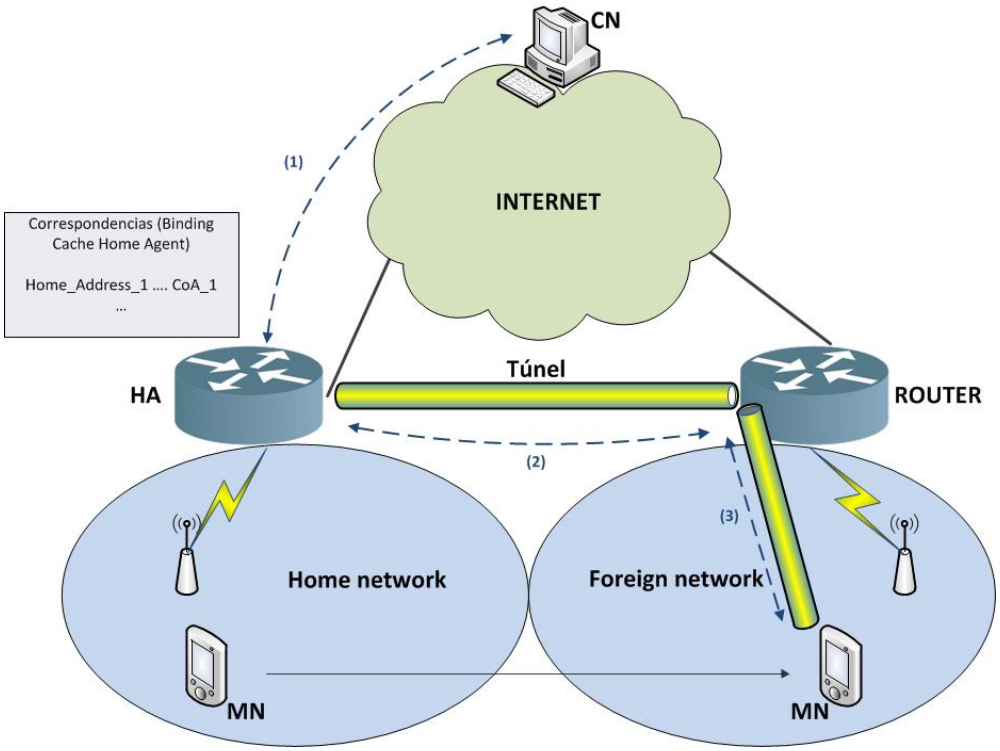


Figura 4: Tunnelling en MIPv6 sin optimización de rutas

- **Con optimización de rutas:** Los paquetes que se envían desde el CN pueden ser encaminados directamente a la CoA del MN. Cuando se envía un paquete a cualquier destino IPv6, el CN comprueba las entradas de su *binding cache* por si hay alguna correspondencia con la dirección destino del paquete. Si la encuentra, el nodo utiliza una nueva cabecera de encaminamiento (*Routing Header*) para hacer llegar el paquete al nodo móvil por medio de la CoA indicada.

  Encaminar los paquetes directamente a la CoA del nodo móvil supone utilizar el camino más corto para la comunicación. Además, elimina la congestión en el HA y, por tanto, el impacto de cualquier posible fallo del agente o en alguna de las redes que intervienen en el camino es sensiblemente menor.

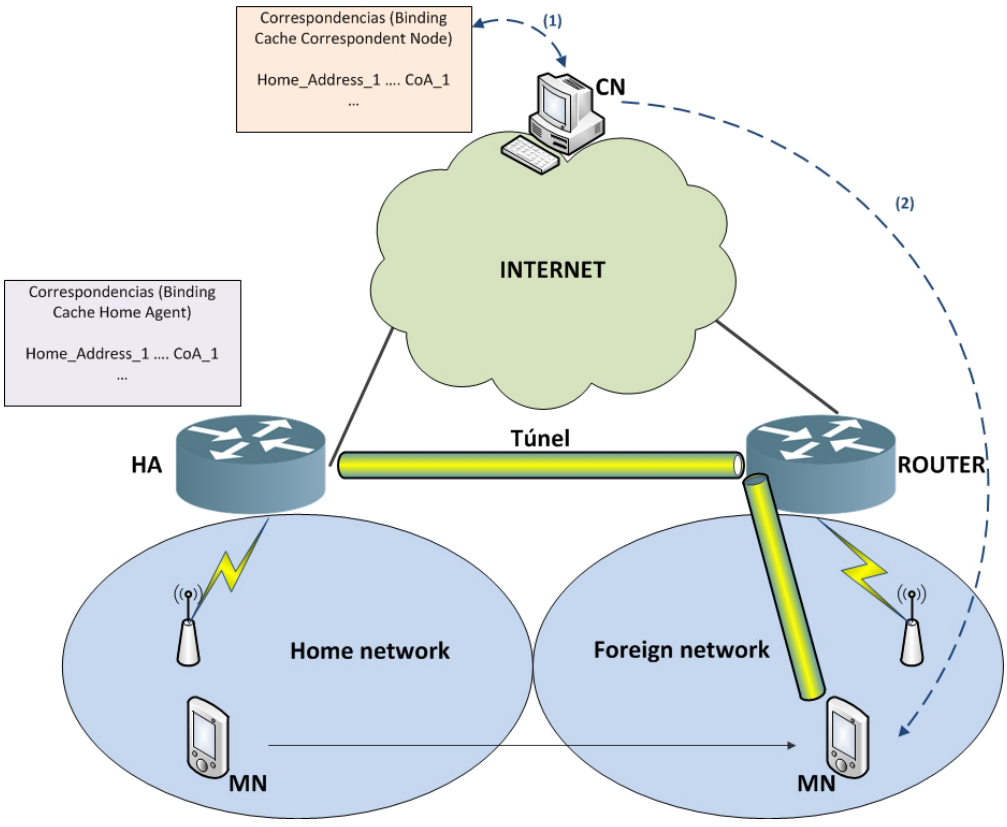


Figura 5: Tunnelling en MIPv6 con optimización de rutas

2.2.2.2. Intercambio de mensajes entre las entidades funcionales

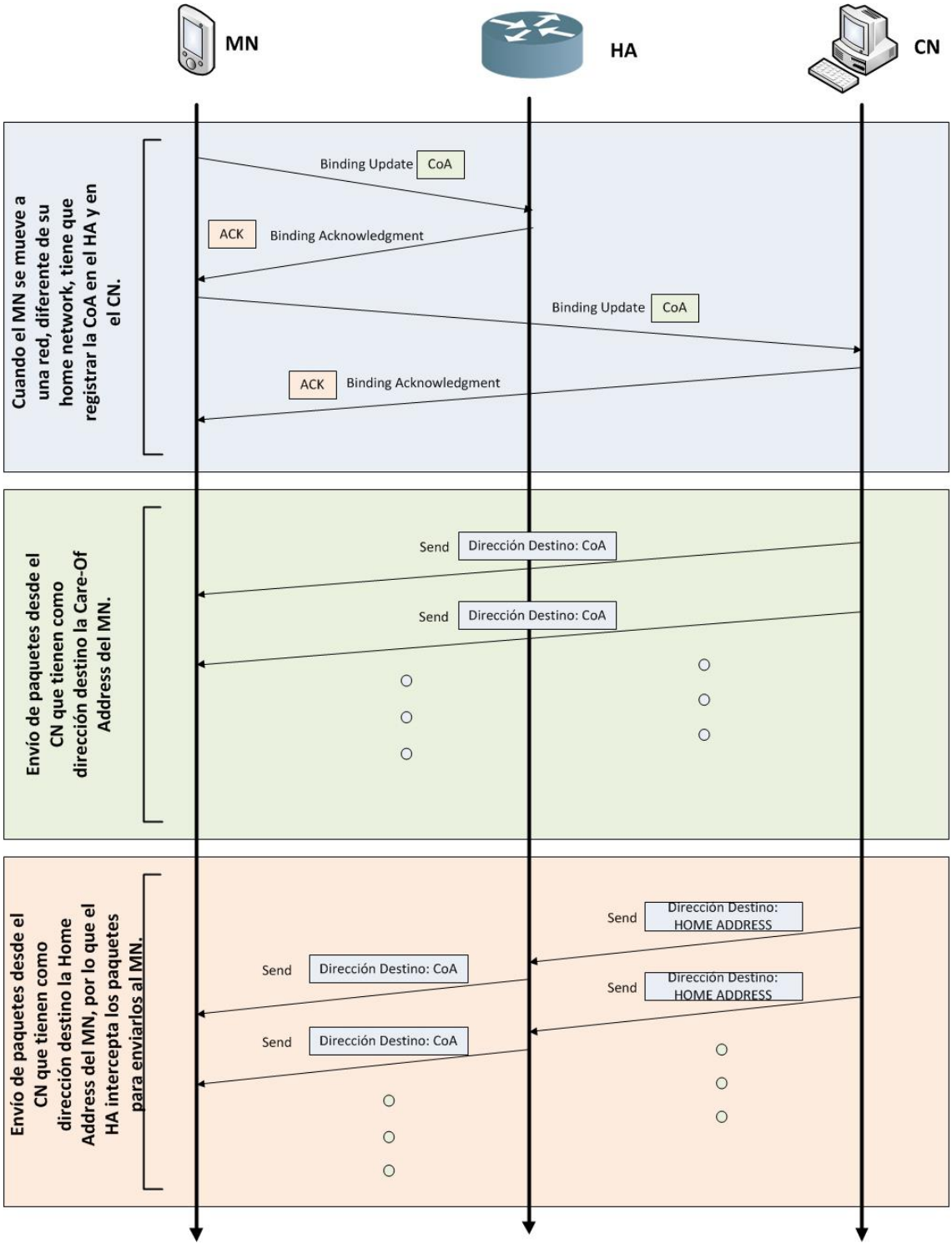


Figura 6: Intercambio de mensajes en MIPv6

2.2.2.3. Handover en Mobile IPv6

Cuando un nodo móvil cambia su punto de conexión a la red al realizar un movimiento, puede que durante un corto periodo de tiempo la comunicación se interrumpa y se aumente el retardo en la entrega de paquetes o, incluso, se pierdan los datagramas enviados al nodo móvil. Este proceso de movimiento de una red a otra se denomina *handover, handoff* o simplemente traspaso.

Muchos autores consideran que este proceso es una cuarta etapa en el funcionamiento de MIPv6 además del descubrimiento del agente, el registro y la transferencia de datos. Las etapas de las que consta este proceso en MIPv6 son las siguientes:

- Detección del movimiento.
- Descubrimiento del router.
- Configuración de la dirección.
- Detección de dirección duplicada.
- Autenticación y autorización (opcional).
- Registrar la nueva CoA.
- Completar el proceso de *Binding Update.*

El proceso es el siguiente: Un nodo móvil detecta que se ha movido a una nueva subred por medio del análisis periódico del mensaje *Router Advertisement* enviado por el router de acceso (AR). La información contenida en este mensaje permitirá al nodo móvil crear una nueva dirección auxiliar (CoA) en la red visitada. El nodo móvil en primer lugar necesita verificar que la dirección es única en su nuevo enlace. El nodo móvil realiza la detección de direcciones duplicadas (DAD) en su dirección de enlace local. Luego se realiza la autoconfiguración (*stateless o stateful*) para formar la nueva CoA. Con esta dirección ya se puede realizar el proceso DAD. En la actualidad, para realizar la detección de direcciones duplicadas, el nodo móvil tiene que enviar uno o varios mensajes *Neighbor Solicitation* a su nueva dirección y esperar por una respuesta durante, al menos, un segundo. Esto supone un tiempo adicional a la latencia del *handover.* Por esta razón, el nodo móvil debería realizar el DAD en paralelo con sus comunicaciones, o elegir no realizarlo.

Una vez que la nueva CoA se ha construido, el nodo móvil debe actualizar la caché de vínculos en su agente origen y en los CN con el envío de mensajes *Binding Update*. El nodo móvil puede solicitar asentimiento de este mensaje activando una opción en el mensaje (este bit normalmente se activa en el mensaje que se envía al agente origen).

2.3. Proxy Mobile IPv6 (RFC 5213)

El protocolo descrito anteriormente (MIPv6) se trata de un enfoque de gestión de movilidad basado en el host, ya que hay que modificar la pila de protocolos de los hosts que están comunicando para dar soporte a la movilidad. Sin embargo, hay otros protocolos como Proxy Mobile IPv6 que se basan en la red.

Debido al gran incremento de dispositivos móviles, parece menos costoso modificar la red en lugar de cada uno de los dispositivos.

A diferencia de la gestión de movilidad basada en el host, la gestión de movilidad basada en la red no requiere ninguna modificación del nodo móvil. Por este motivo se está acelerando el despliegue de PMIPv6 en la práctica.

2.3.1. Gestión de la movilidad basada en la red

Como se ha dicho anteriormente, en un enfoque basado en la red (PMIPv6) es el servicio de red el que se encarga de controlar la gestión de la movilidad. El nodo móvil no envía mensajes de señalización relativos a la movilidad.

Las características más importantes de los enfoques basados en la red son:

- Soporte para no tener que modificar el MN.
- Soporte para IPv4 e IPv6.
- Mejor rendimiento en el proceso de *handover*.

2.3.2. PMIPv6

Proxy Mobile IPv6 se basa en MIPv6. Utiliza conceptos/entidades del protocolo MIPv6 como el concepto HA. Este protocolo se diseñó para dar soporte a la gestión de movilidad de un MN localizado en un determinado dominio.

Una vez que un MN entra en un dominio PMIPv6 y realiza la autenticación, se va a asegurar que el MN está "siempre en su red origen". El servicio de red asigna un prefijo de red local único para cada MN, y conceptualmente este prefijo se mantiene siempre que el MN se mueva dentro de un dominio PMIPv6. Desde el punto de vista del MN, el dominio PMIPv6 se considera como la red origen (*home network*) del nodo móvil.

Este protocolo añade las siguientes entidades funcionalidades:

- ***Mobile Access Gateway* (MAG):** Esta entidad se ejecuta sobre los routers de acceso. La principal función es detectar el movimiento del nodo móvil y avisar al LMA de que el MN se ha movido mediantes mensajes de señalización. Se establece un túnel entre el MAG y el LMA para que el nodo móvil siga manteniendo su *home address.* Gracias a este túnel el MN va a poder emular que se encuentra en su *home network.*

- ***Local Mobility Anchor* (LMA):** Es similar al HA en MIPv6, aunque se han añadido capacidades necesarias para PMIPv6. La principal función del LMA es mantener accesible la dirección del nodo móvil, mientras este se mueve dentro de un determinado dominio PMIPv6. El LMA está provisto de una *binding cache*, en la cual almacena una entrada por cada MN registrado. Esta *caché* es más completa que la de MIPv6, contiene algunos campos extras. Algunos de ellos son: identificador del MN, prefijo del MN en su *home network*, una bandera que indica el registro de proxy y el identificador de la interfaz del túnel bidireccional entre el MAG y el LMA.

2.3.2.1. Arquitectura de PMIPv6

A continuación se muestra la arquitectura de PMIPv6, así como su funcionamiento dentro de un dominio determinado.

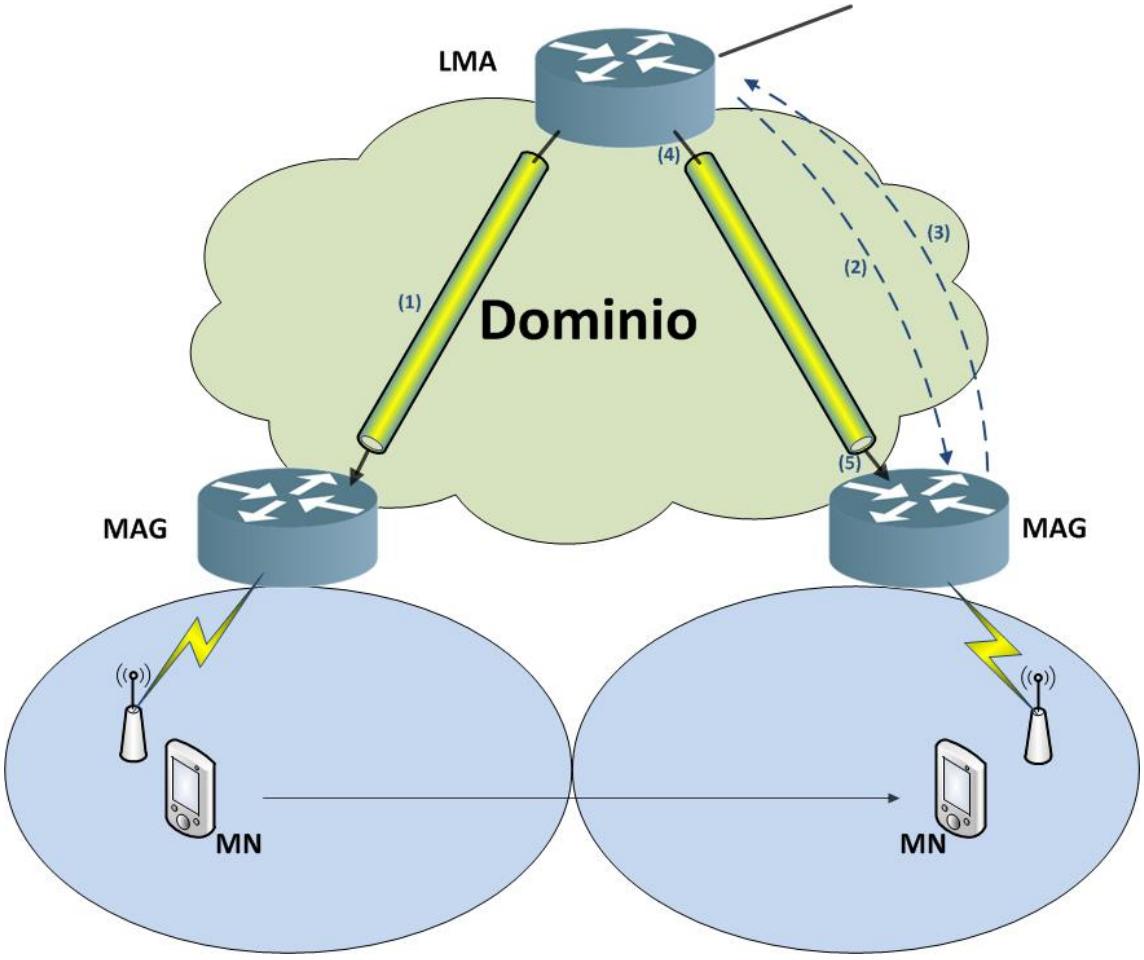


Figura 7: Esquema básico de PMIPv6

1. Túnel IP: túnel *IP-in-IP* entre el LMA y el MAG.
2. *Proxy Binding Acknowledgement* (PBA): Mensaje de control enviado desde el LMA al MAG.
3. *Proxy Binding Update* (PBU): Mensaje de control enviado desde el MAG hacia el LMA para establecer un vínculo entre la home address del MN y su Proxy-CoA.
4. *LMA address* (LMAA): Dirección LMA que será el punto de entrada del túnel.
5. *Proxy care-of-address* (Proxy-CoA): Dirección del MAG. Salida del túnel.

2.3.2.2. Intercambio de mensajes entre las entidades funcionales

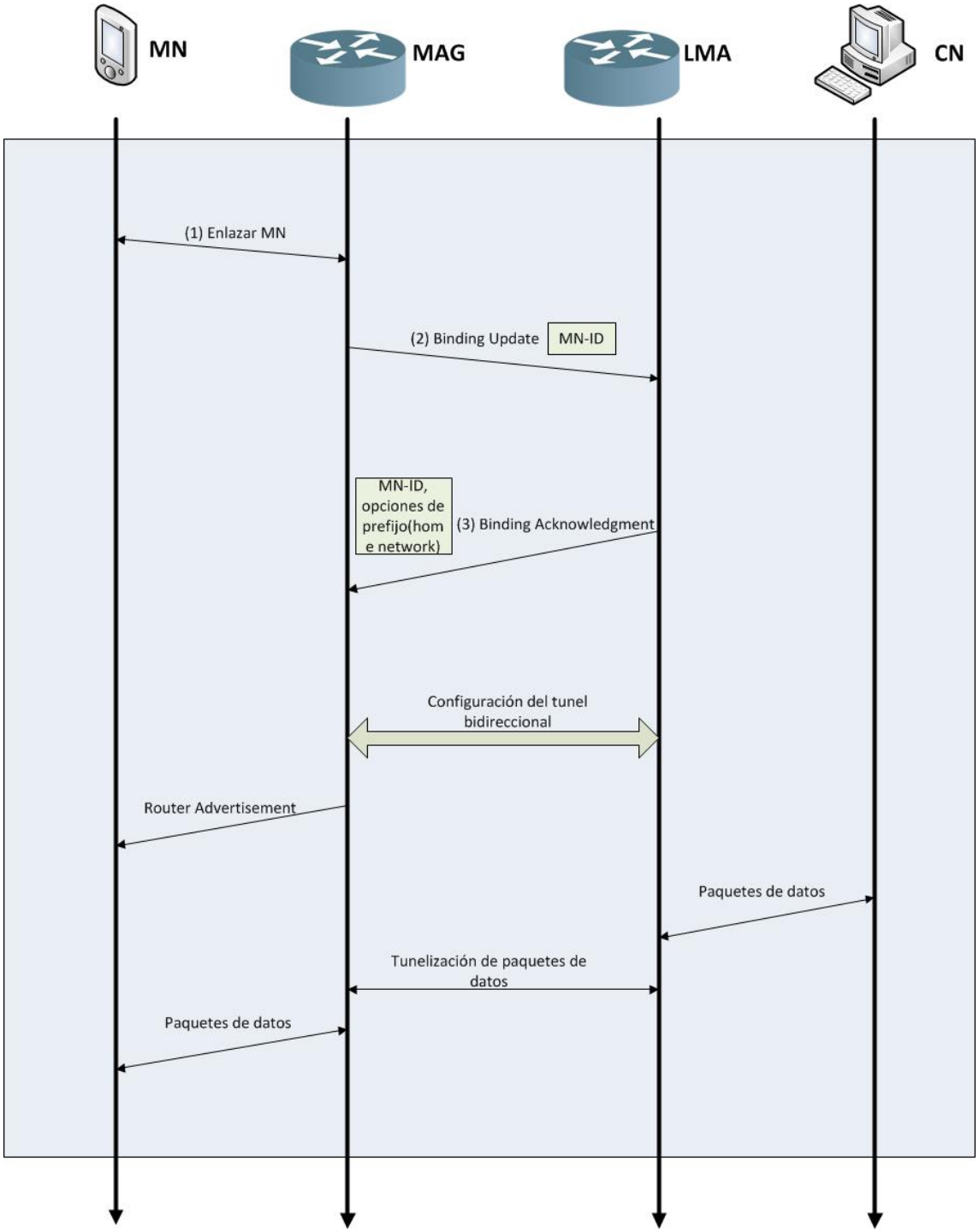


Figura 8: Intercambio de mensajes en PMIPv6

El flujo de los mensajes es el siguiente:

1. Enlazar el MN: Consiste en enlazar el MN con el MAG por primera vez.

2. *Proxy Binding Update* (PBU): El MAG envía un mensaje de *proxy binding update* incluyendo el identificador del LMA del MN en lugar del identificador del MN.

3. *Proxy Binding Acknowledgement* (PBA): El LMA envía un mensaje de *proxy binding acknowledgement* incluyendo las opciones del prefijo de la *home network* del nodo móvil, y establece una ruta para ese prefijo a través del túnel con el MAG.

A diferencia de MIPv6, el túnel en PMIPv6 se establece entre el LMA y el MAG, y no un MN. Una vez que el MAG recibe el mensaje *proxy binding acknowledgement* del LMA, se obtiene toda la información requerida para emular que el MN se encuentra en su *home network*. A partir de este momento se empiezan a enviar mensajes *Router Advertisement* al MN. Estos mensajes contienen el prefijo de la *home network* del MN.

Después de recibir el mensaje de RA, el MN configura su *home address* mediante la combinación del prefijo de la *home network* incluido en el mensaje RA y su dirección de interfaz.

Una vez que el túnel bidireccional se ha configurado correctamente, todo el tráfico enviado desde el MN va a llegar al LMA a través del túnel. El LMA recibe los paquetes de datos enviados por el CN al MN. El LMA, una vez que ha recibido los paquetes del CN, los envía al MAG a través del túnel.

Por último, el MAG, que se encuentra en el otro extremo del túnel, elimina la cabecera “exterior” y envía los paquetes hacia el nodo móvil.

2.4. Comparativa entre MIPv6 y PMIPv6

A continuación, se muestra una comparativa entre los dos protocolos examinados: Mobile IPv6 y Proxy Mobile IPv6.

ASPECTOS	MIPv6	PMIPv6
Gestión de la movilidad	Gestión de la movilidad basada en el host	Gestión de la movilidad basada en la red
Alcance de la movilidad	Movilidad global	Movilidad localizada
Entidades funcionales	HA	LMA
Entidades topológicas	AR	MAG
Modificación del MN	Sí	No
Mensajes de registro de la ubicación	*Binding Update*	*Proxy Binding Update*
Dirección del MN	*Home Address o CoA*	*Home Address* (siempre)
Relación entre el túnel y las entradas de la binding caché	1:1	1:n
Tipo de Router Advertisment	*Broadcast*	*Unicast*
Búsqueda de clave en binding caché	*Home Address*	Identificador del MN
Modelo de direccionamiento	Modelo *Shared-Prefix*	Modelo *Per-MN-Prefix*
Optimización de rutas	Soportado	No soportado
Detección de direcciones duplicadas	Realizado una vez que se mueva a una subred diferente	Realizado sólo una vez (inicialmente dentro del dominio)
Detección de movimiento	Requerido	No requerido

Figura 9: Tabla comparativa entre MIPv6 y PMIPv6

3. SIMULACIÓN DE LA MOVILIDAD EN IPv6

En este capítulo se hace una descripción del desarrollo realizado acerca de la simulación de la movilidad en IPv6.

3.1. Objetivos del Trabajo

Como ya se dijo al principio de este documento, los principales objetivos que se pretenderán alcanzar durante el desarrollo de este Trabajo Fin de Grado son los siguientes:

- Estudio y análisis de los protocolos Mobile IPv6 y Proxy Mobile IPv6 pensados para dar soporte a la movilidad en redes IPv6 de próxima generación, fijándonos en su arquitectura y mensajes intercambiados entre las entidades características de dichos protocolos.
- Desarrollo de simulador de eventos para analizar y medir el rendimiento de estos protocolos bajo distintas circunstancias, como la topología de la red de acceso, los modelos de movimiento y los modelo de tráfico.
- Evaluar los diferentes resultados obtenidos.

3.2. Planificación

Éstas son las tareas que se propusieron para llevar realizar este Trabajo:

	Detalle de la tarea	**Fecha inicio**	**Duración (días)**	**Fecha fin**
Tarea 1	Análisis y estudio de los protocolos de gestión de movilidad en IPv6	02/02/2015	12	13/02/2015
Tarea 2	Topologías, modelos de tráfico y modelos de movimiento	13/02/2015	16	28/02/2015
Tarea 3	Análisis y diseño del simulador a desarrollar	28/02/2015	17	16/03/2015
Tarea 4	Implementación del simulador	16/03/2015	67	21/05/2015
Tarea 5	Pruebas y evaluación del rendimiento de los protocolos implementados	21/05/2015	26	15/06/2015

Figura 10: Planificador de tareas

Las tareas anteriores se pueden ver mejor ordenadas en el tiempo en el siguiente Diagrama de Gantt:

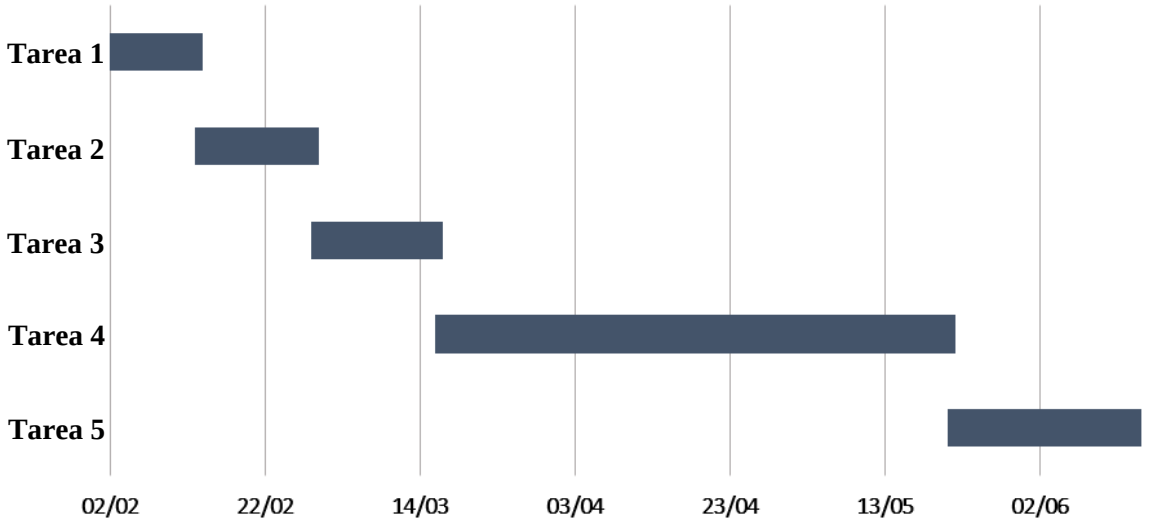


Figura 11: Diagrama de Gantt con las tareas repartidas en el tiempo

3.3. Desarrollo del simulador

Aquí se explicará cómo se ha planteado y desarrollado este Trabajo Fin de Grado cronológicamente en el tiempo.

Para estimar el rendimiento y comportamiento de arquitecturas, algoritmos y protocolos en sistemas de comunicaciones, se suelen utilizar tres metodologías diferentes:

1. Experimentos con sistemas reales y prototipos.
2. Modelo analítico (análisis matemático).
3. Simulación.

Durante la investigación y el desarrollo de sistemas de comunicaciones, los últimos dos métodos son los más importantes en la fase conceptual, ya que hacer prototipos de tales sistemas es sobre todo inviable debido a limitaciones económicas y técnicas.

La simulación se utiliza en particular para los sistemas que son muy dinámicos y cuyas propiedades son difíciles de capturar de forma matemática. A menudo, los métodos

analíticos muestran el comportamiento marginal de las características del sistema u ofrecen límites superiores e inferiores a algunas de las preguntas específicas de la investigación. Sin embargo, un análisis con profundidad conduce a una complejidad inaceptable de los modelos analíticos. Por el contrario, la simulación ofrece a los científicos e investigadores un ambiente controlado en el que un sistema puede ser investigado con más detalle. Diferentes conjuntos de parámetros y escenarios se pueden analizar con no mucho esfuerzo.

Por lo tanto, la simulación es una metodología potente y versátil para analizar y visualizar el comportamiento y el rendimiento de los sistemas de comunicación y redes. Y por eso, hemos elegido este tercer método para este Trabajo Fin de Grado.

3.3.1. Estudio de los protocolos MIPv6 y PMIPv6

Esta primera fase consistió en el estudio exhaustivo de los protocolos MIPv6 y PMIPv6, ya explicados anteriormente en el Capítulo 2, para la movilidad en redes IPv6 de próxima generación.

Una vez entendidos y comprendidos, llegó el momento de pensar en cómo implementar todas esas ideas anteriores en el simulador que se pretendía crear.

3.3.2. Lenguaje de programación

Valorando el lenguaje de programación a utilizar durante el desarrollo del simulador, nos encontramos con varias opciones:

- Java
- C/C++
- Matlab

Y finalmente, se eligió Matlab, ya que tiene mucha potencia, alta precisión y un amplio soporte matemático y de funciones ya desarrolladas, además de tener una comunidad muy extendida –y, por tanto, donde encontrar ayuda o recursos es muy fácil.

3.3.3. Consideraciones iniciales

Lo primero que se decidió es que el Trabajo Fin de Grado se desarrollaría íntegramente en inglés, para que no solamente pudiera ser utilizado por usuarios de habla hispana, y que se utilizaría la GUI de Matlab para crear un entorno amigable para el usuario donde se pudiera modificar cualquier parámetro con facilidad. Además, estaría desarrollado por módulos para futuros trabajos y ampliaciones sobre el simulador.

Una cosa era clara: había que simular un número *n* de nodos durante *x* segundos. Por tanto, esos dos parámetros tenían que estar desde el principio en nuestro simulador, al ser básicos. Así mismo, se decidió que todos los parámetros configurables se guardarían en un archivo de configuración con extensión "*.xml*".

Pero, además de eso, hacía falta la configuración con la que ejecutaríamos el simulador. Y eso era sólo posible añadiendo campos para la movilidad, topología de la red de acceso, modelo de tráfico y los protocolos a simular.

En los siguientes apartados, procedemos a ver cada uno de ellos.

3.3.4. Modelos de movilidad

Respecto a los modelos de movilidad, nos podemos encontrar bastantes a la hora de utilizarlos para simular. Los principales serían:

- *Random Waypoint (RWP)*
- *Real Mobility*
- *Fluid Flow*

Empezamos, por tanto, con el primero, *Random Waypoint*, que finalmente sería el único modelo de movilidad implementado en el simulador debido a los plazos que teníamos establecidos en el cronograma y que debíamos cumplir para terminar el Trabajo Fin de Grado en el plazo determinado.

3.3.4.1. Random Waypoint (RWP)

El modelo de movilidad *Random Waypoint* (RWP) ha sido ampliamente utilizado en las simulaciones de redes móviles. Este modelo de movilidad es un modelo estocástico simple y directo. En RWP, un nodo móvil se mueve en un plano continuo finito desde su posición actual a una nueva ubicación, eligiendo aleatoriamente sus coordenadas de destino, su velocidad de movimiento y la cantidad de tiempo que va a hacer una pausa cuando llega al destino.

Al alcanzar el destino, el nodo se detiene durante un tiempo de acuerdo con alguna variable aleatoria y el proceso se repite. Una vez transcurrido el tiempo de pausa, el nodo elige un nuevo destino, la velocidad y el tiempo de pausa. El movimiento de un nodo desde la posición de partida (*waypoint*) a su siguiente destino (*waypoint*) se define como período de movimiento o el tiempo de transición. La distancia recorrida entre los movimientos de un nodo de la *waypoint* de partida para su próximo *waypoint* se define como longitud de transición. Los puntos de destino ("puntos de referencia") están uniformemente escogidos al azar en el área del sistema.

En la siguiente imagen se puede observar el patrón de recorrido de un nodo móvil usando *Random Waypoint*:

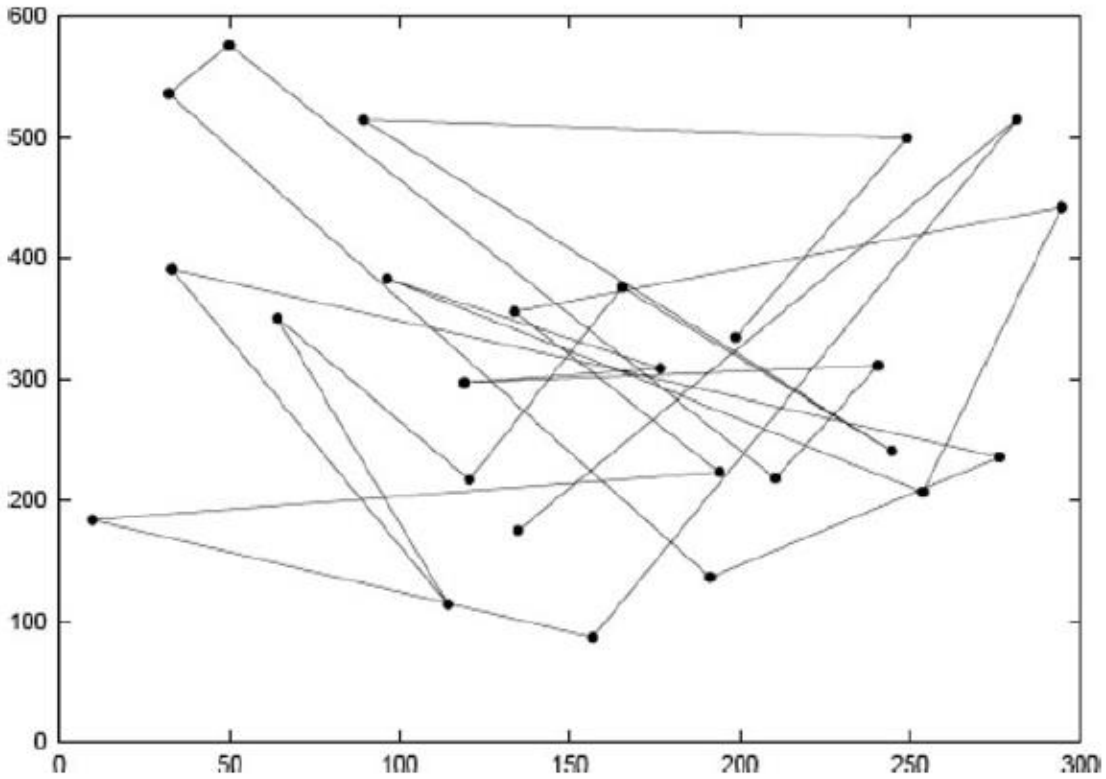


Figura 12: Patrón de recorrido de un nodo móvil usando RWP

Como decíamos, Matlab tiene una comunidad muy extendida y es posible encontrar mucha ayuda. Y, de hecho, fue posible encontrar el código implementado de *Random Waypoing*, el cual utilizamos para nuestro simulador. En la siguiente imagen se puede ver un fragmento del archivo "Generate_Mobility.m", que es, como su propio nombre indica, donde se genera la movilidad de RWP utilizando el número de nodos y el tiempo de simulación:

```
9     % Output:
10    %   - s_mobility:   mobility data generated.
11    %
12    function s_mobility = generate_Mobility(s_input)
13        global s_mobility_tmp;
14        global nodeIndex_tmp;
15
16        s_mobility.NB_NODES = s_input.NB_NODES;
17        s_mobility.SIMULATION_TIME = s_input.SIMULATION_TIME;
18        for nodeIndex_tmp = 1:s_mobility.NB_NODES
19            %%%%%%%%%%%%%%%%%%%%%%%%%%Initialize:
20            s_mobility_tmp.VS_NODE(nodeIndex_tmp).V_TIME = [];
21            s_mobility_tmp.VS_NODE(nodeIndex_tmp).V_POSITION_X = [];
22            s_mobility_tmp.VS_NODE(nodeIndex_tmp).V_POSITION_Y = [];
23            s_mobility_tmp.VS_NODE(nodeIndex_tmp).V_DIRECTION = [];
24            s_mobility_tmp.VS_NODE(nodeIndex_tmp).V_SPEED_MAGNITUDE = [];
25            s_mobility_tmp.VS_NODE(nodeIndex_tmp).V_IS_MOVING = [];
26            s_mobility_tmp.VS_NODE(nodeIndex_tmp).V_DURATION = [];
```

Figura 13: Fragmento de código de "Generate_Mobility.m" de RWP

Esta implementación de *Random Waypoint* también incluía un archivo llamado "Test_Animate.m", donde se realizaba una animación sobre la información generada en "Generate_Mobility.m". En la siguiente figura podemos ver un fragmento del código de este fichero:

```
v_t = 0:time_step:s_input.SIMULATION_TIME;

for nodeIndex = 1:s_mobility.NB_NODES
    %Simple interpolation (linear) to get the position, anytime.
    %Remember that "interp1" is the matlab function to use in order to
    %get nodes' position at any continuous time.
    vs_node(nodeIndex).v_x = interp1(s_mobility.VS_NODE(nodeIndex).V_TIME,s_mobil
    vs_node(nodeIndex).v_y = interp1(s_mobility.VS_NODE(nodeIndex).V_TIME,s_mobil
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;
hold on;
```

Figura 14: Fragmento de código de "Test_Animate.m" de RWP

Existen varias formas de división del terrero para la comunicación móviles, desde la tradicional hexagonal hasta otras formas más exactas como las de Voronoi, que es la utilizada en las redes 4G hoy en día. Se pueden observar en la siguiente imagen:

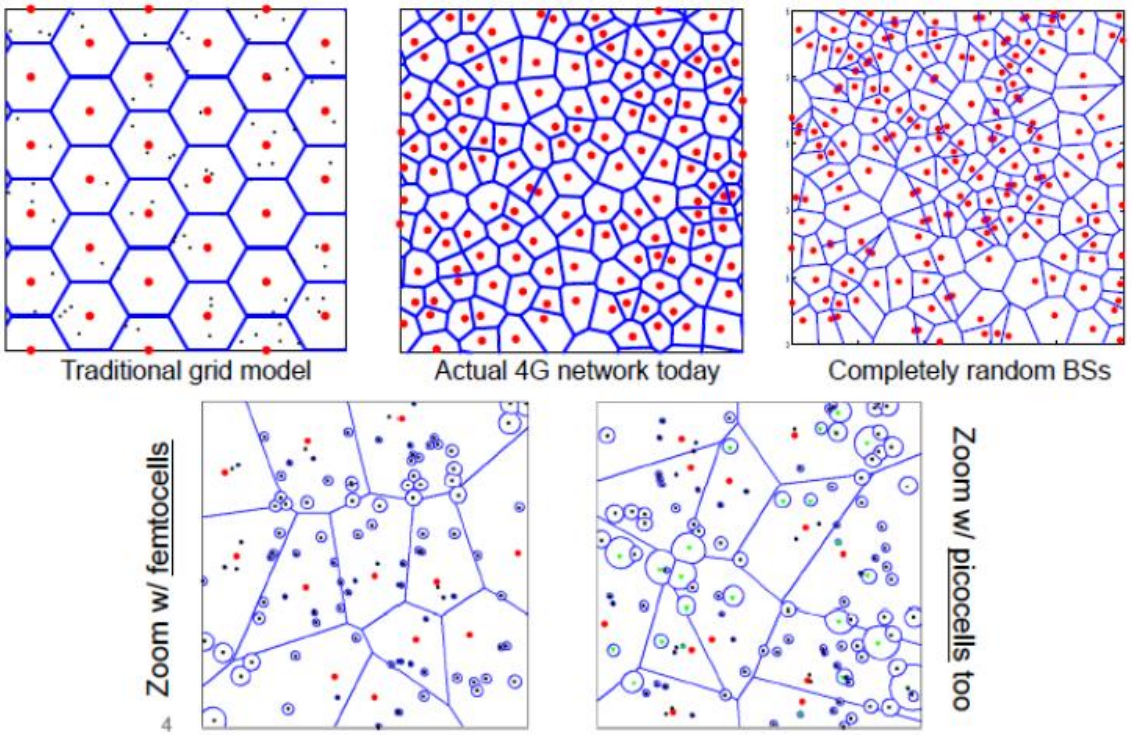


Figura 15: Distintos tipos de redes celulares

En este Trabajo Fin de Grado se trabajará con el modelo tradicional, trabajando con formas hexagonales:

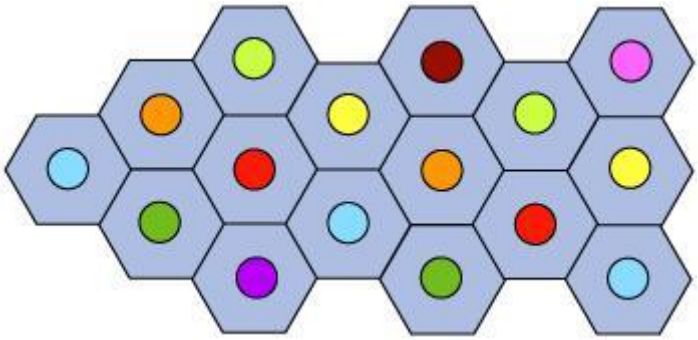


Figura 16: Modelo tradicional con celdas hexagonales

Así que creamos un nuevo fichero "hex.m" (para crear la estructura celular), al que se le debía pasar como parámetros los límites del dibujo y el radio de la celda –entre otras cosas– y se encargaría de dibujar las celdas en la animación anterior, así como de guardar un vector con las diferentes celdas creadas. En la siguiente imagen podemos ver un fragmento de este fichero:

```
% This function is executed when user wants to draw cells
% and see the simulation.
function [ hexArray ] = hexDraw( minx, maxx, miny, maxy, r,

axis ([minx maxx miny maxy])
c=r;
v=30:60:390;
cv=r*cosd(v);
sv=r*sind(v);
```

Figura 17: Fragmento de código de "hex.m"

Aparte de eso, *Random Waypoint* tiene más parámetros –como ya se ha visto– que podemos modificar, como la velocidad, tiempo de pausa, la dirección… Todo eso fue posible controlarlo gracias a la creación de la interfaz gráfica llamada "Principal" con los diferentes parámetros y añadiendo un botón de opciones avanzadas *Advanced* para modificar los parámetros exclusivos de *Random Waypoint*.

En la siguiente figura se puede observar una primera versión de la interfaz gráfica del simulador:

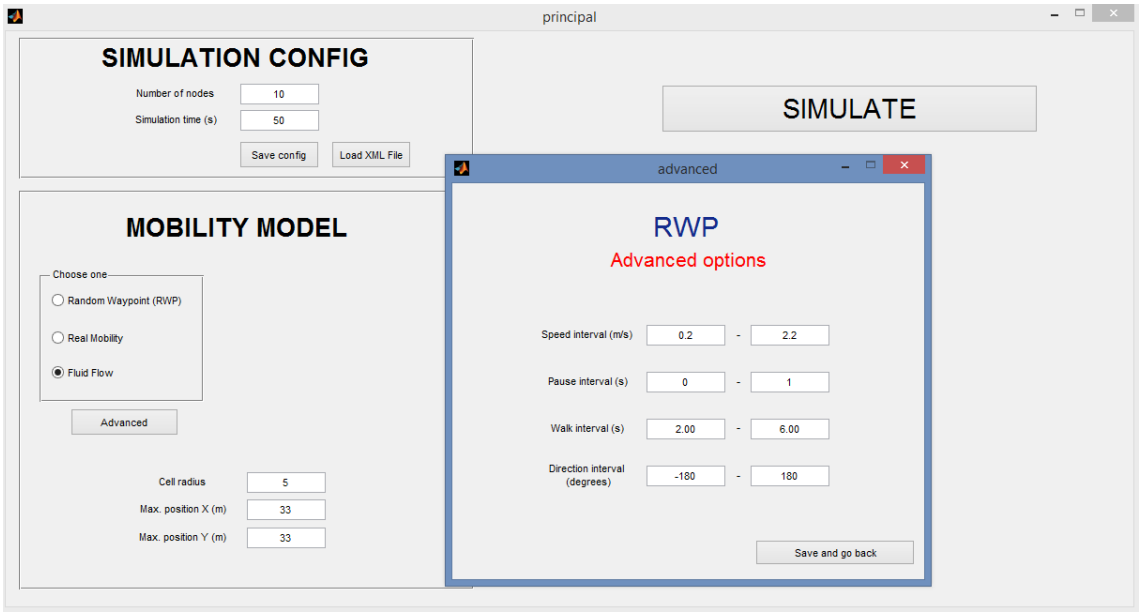


Figura 18: Primera versión de la interfaz gráfica del simulador

La parte de la movilidad estaba hecha, por tanto. A continuación, tocaba centrarse en la topología y los modelos de tráfico.

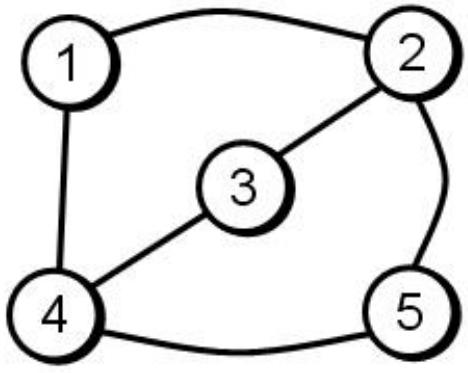
3.3.5. Topología

Respecto a la topología, dos parámetros son clave: el nivel de jerarquía y la conectividad:

- **Nivel de jerarquía**: niveles con los que cuenta la red.

- **Conectividad**: grado de conectividad de la red. En este simulador se ha distinguido entre:
  - *Low*: poca conectividad.
  - *Tree*: conectividad normal, con algunas conexiones no disponibles.
  - *Full*: todas las conexiones posibles.
  - *Trade-Off*: una parte de la red muy conectada y otra no.

Tomando estos dos parámetros como referencia, se creó el fichero "buildTopology.m", que se encarga de crear una matriz de conectividad o adyacencia dependiendo de las conexiones disponibles, así como de devolver también los routers de la última fila (que serán nuestros *routers de acceso*) y los de la penúltima fila.



M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Figura 19: Ejemplo de matriz de adyacencia

En la siguiente imagen se puede observar un fragmento de este fichero "buildTopology.m":

```
17    function [tree,v,ar] = buildTopology(n,connectivity)
18 -  nRows=n;
19 -  connect=connectivity;
20 -  cont=1;
21 -  leftSide=2;
22 -  rightSide=4;
23 -  for i=2:nRows
24 -      cont=cont+(i+1);
25 -  end
26 -  nNodes=cont;
27
28 -  cont=1;
29 -  tree=zeros(nNodes);
30 -  for i=1:nRows % Row
31 -      if (i==1) % Node 1 has 3 children (Nodes 2, 3 and 4)
32 -          tree(i,2:4)=1;
33 -          cont=1;
34 -      elseif (i==2)
35 -          if (connect==0)
36 -              tree(2,1)=1;
```

Figura 20: Fragmento de código de "buildTopology.m"

Gracias a la matriz de adyacencia obtenida y haciendo uso de la herramienta *Biograph* de Matlab para dibujar objetos, se crea "showTopology.m" con la finalidad de mostrar al usuario la topología de red que ha elegido dependiendo del nivel de jerarquía y la conectividad.

```
30 -  for i=1:nRows %Row
31 -      if (i==1) %Node 1 has 3 children
32 -          bg.nodes(i).Position= [inicioX inicioY];
33 -          cont=1;
34 -      elseif (i==2)
35 -          cont=4;
36 -          bg.nodes(2).Position=[inicioX-distance inicioY-distance];
37 -          bg.nodes(3).Position=[inicioX inicioY-distance];
38 -          bg.nodes(4).Position=[inicioX+distance inicioY-distance];
39 -      else
40 -          nodesThisRow=i+1;
41 -          if(mod(nodesThisRow,2)==0)
42 -              evenRow=1;
43 -          else evenRow=0;
44 -          end
45
46            %Side nodes...
47 -          leftSide=cont+1;
48 -          rightSide=cont+nodesThisRow;
49 -          for j=1:nodesThisRow
```

Figura 21: Fragmento de código de "showTopology.m"

Así, podemos mostrar al usuario la topología de su red. En las siguientes imágenes podemos ver las diferentes topologías de red dependiendo de la conexión y con un nivel de jerarquía 4:

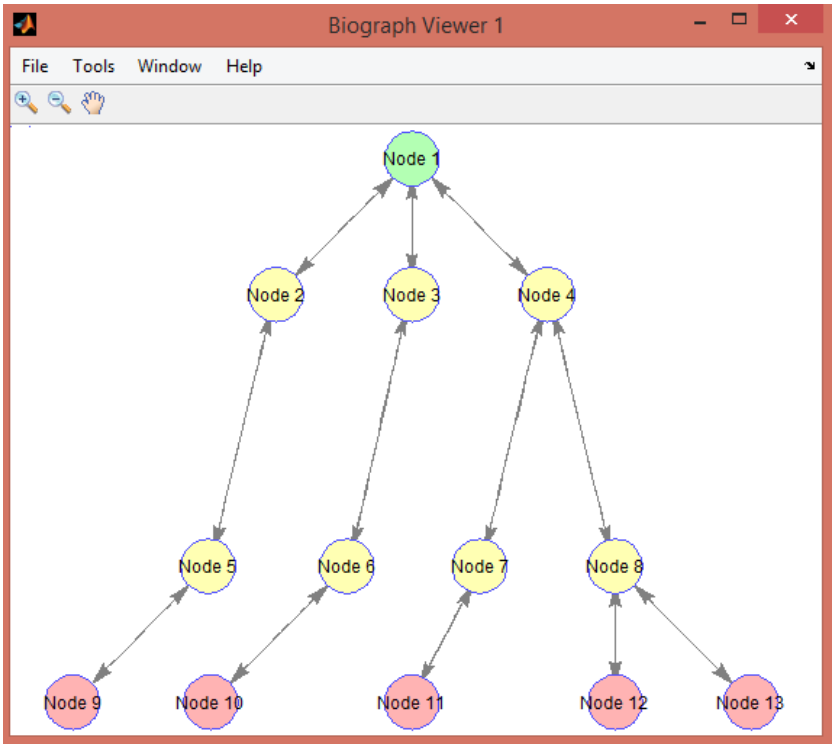


Figura 22: Ejemplo de conectividad "low"

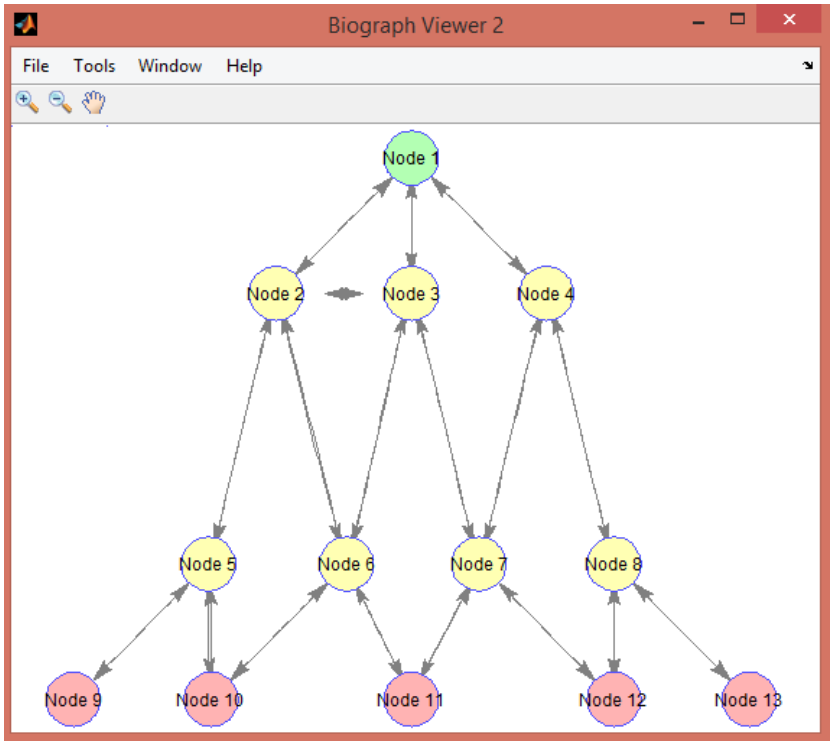


Figura 23: Ejemplo de conectividad "tree"

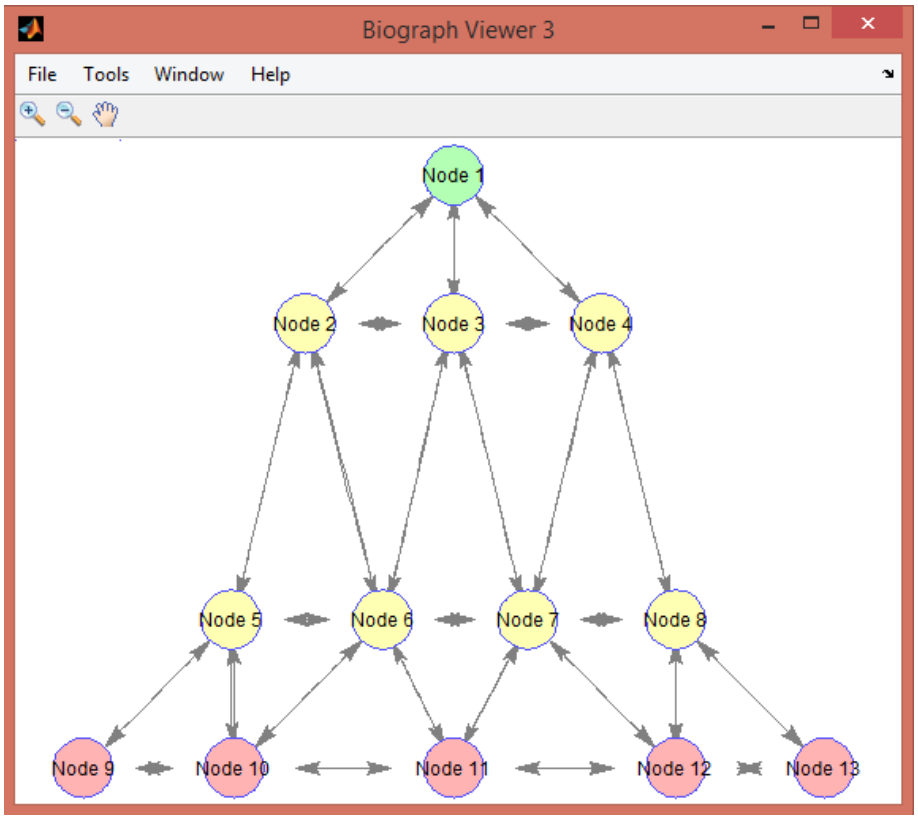


Figura 24: Ejemplo de conectividad "full"

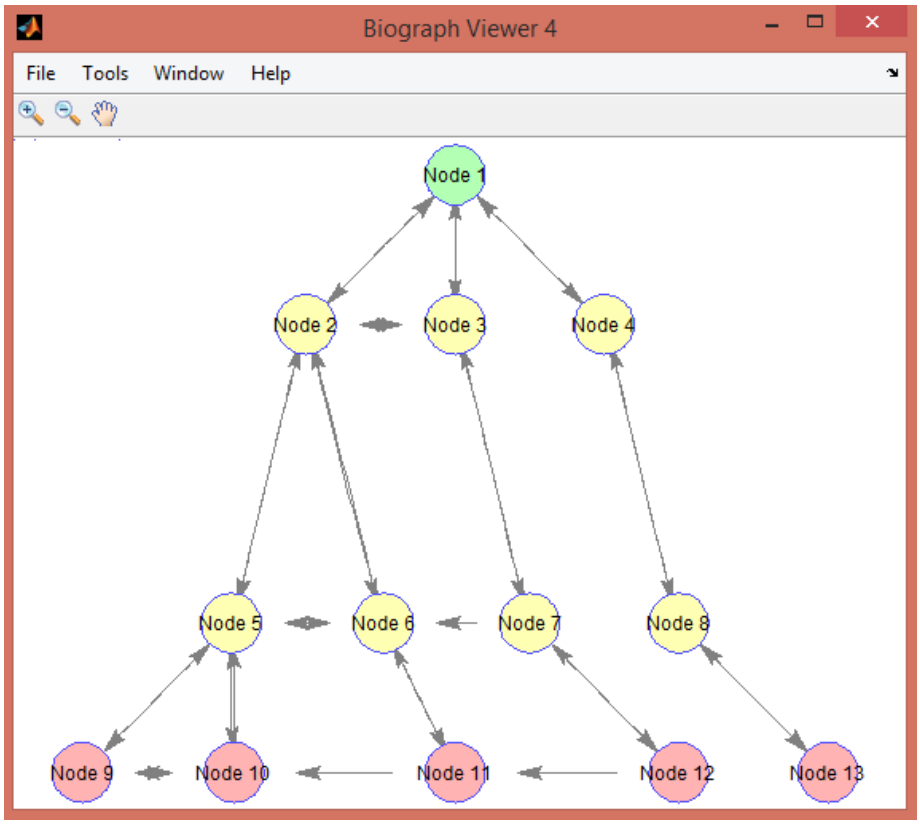


Figura 25: Ejemplo de conectividad "trade-off"

De esta manera, pudimos agregar el apartado *Topology* a la ventana principal. Obviamente, todos los datos que aparecen en esta ventana principal se van guardando en el fichero *xml* llamado "config.xml".

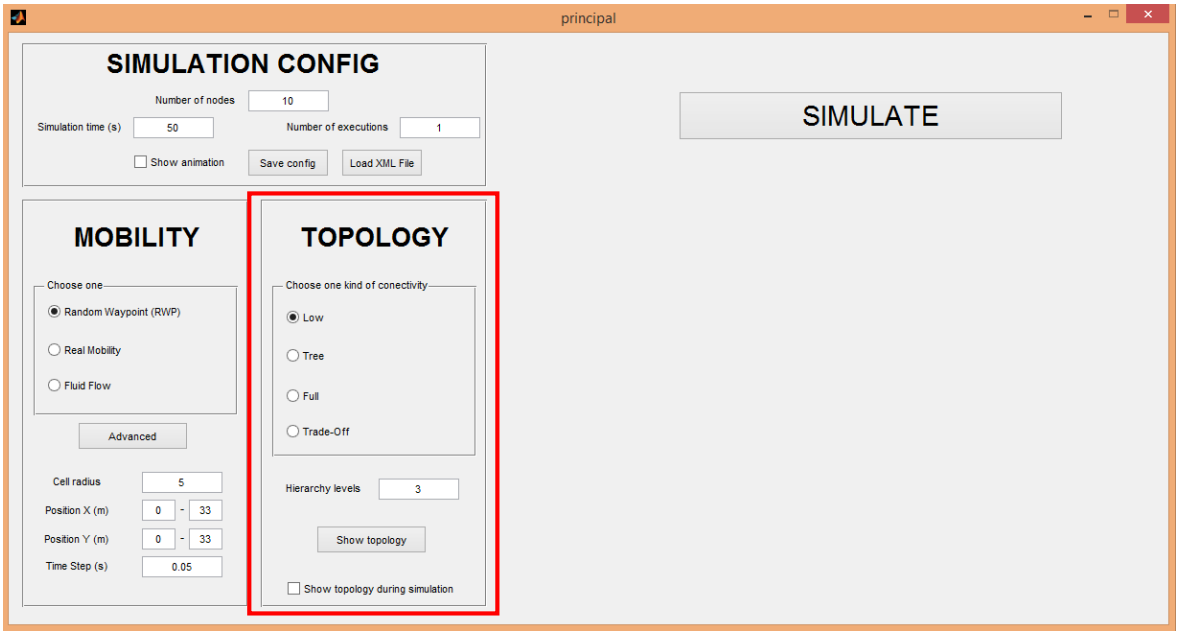


Figura 26: Ventana principal del simulador con el apartado "Topology"

3.3.6. Modelos de tráfico

Los dos modelos de tráfico que se pueden elegir son:

- Random
- Exponentially Distributed (Poisson)

Para ello, se creó un script nuevo llamado "trafficGeneration.m", donde se generarán las diferentes sesiones de demanda dependiendo de si el usuario elige "Random" o "Poisson".

```
12      % Output:
13      %   - str_mobility: s_mobility with traffic data added.
14      %
15      function [ str_mobility ] = trafficGeneration( type, demands, s_mobility, s_traffic)
16 -      if type == 1
17 -        str_mobility = sessionArrivalPoisson(s_mobility,s_traffic);
18 -      else
19 -        str_mobility = sessionArrivalRandom(demands,s_mobility,s_traffic);
20 -      end
21 -    end
```

Figura 27: Fragmento de código de "trafficGeneration.m"

En las siguientes imágenes podemos ver fragmentos de las funciones para simular las demandas al estilo “Random” o “Poisson”:

```
85    function str_mobility = sessionArrivalRandom(numDemands,s_mobility,s_traffic)
86
87 -  str_mobility = s_mobility;
88
89 -  for i = 1:s_mobility.NB_NODES % For each node
90 -      str_mobility.VS_NODE(i).V_SESSION_ARRIVALS=sort(rand(numDemands,1)*s_mobil
91 -      str_mobility.VS_NODE(i).V_SESSION_DURATION=rand(s_traffic.TRA_SESSION_DURA
92 -      str_mobility.VS_NODE(i).V_SESSION_DEMAND_VALUE=round((rand(numDemands,1)*(
93
94        %The origin of demand will be the node position at that time
95 -      for j=1:size(str_mobility.VS_NODE(i).V_SESSION_ARRIVALS,1)
96 -          t=1;
97 -          while str_mobility.VS_NODE(i).V_TIME(t)<=str_mobility.VS_NODE(i).V_SES
98 -              t=t+1;
99 -          end
100 -         str_mobility.VS_NODE(i).V_SESSION_DEMAND_SOURCE(j,1)=str_mobility.VS_N
101 -     end
102 - end
```

Figura 28: Fragmento de código de “trafficGeneration.m” correspondiente al modelo Random

```
33    function str_mobility = sessionArrivalPoisson(s_mobility,s_traffic)
34
35 -  str_mobility = s_mobility;
36
37 -  for i = 1:s_mobility.NB_NODES % For each node
38        %Generate the Poisson arrival process.
39 -      str_mobility.VS_NODE(i).V_SESSION_ARRIVALS=poisson(s_traffic.TRA_LAMBDA,s_mobility.SIMULATION_TIME);
40
41        %The duration of a typical session is exponentially distributed with mean rate mu_s
42 -      str_mobility.VS_NODE(i).V_SESSION_DURATION=exprnd(s_traffic.TRA_SESSION_DURATION,size(str_mobility.V
43 -      str_mobility.VS_NODE(i).V_SESSION_DEMAND_VALUE=round((rand(size(str_mobility.VS_NODE(i).V_SESSION_AR
44
45        %The origin of demand will be the node position at that time
46 -      for j=1:size(str_mobility.VS_NODE(i).V_SESSION_ARRIVALS,1)
47 -          t=1;
48 -          while s_mobility.VS_NODE(i).V_TIME(t)<=str_mobility.VS_NODE(i).V_SESSION_ARRIVALS(j)
49 -              t=t+1;
50 -          end
```

Figura 29: Fragmento de código de “trafficGeneration.m” correspondiente al modelo de Poisson

También, se creó un nuevo fichero llamado “mobility2topology.m” encargado de enlazar movilidad y topología, añadiendo nuevos campos como celda y router de acceso anterior y actual:

```
82 -            if (prevCell~=currCell) % If old and new cells are different, a movement has been done
83 -                structMob.VS_NODE(nodeIndex).V_MOVEMENT_DONE(indiceMov,1)=1;
84 -                structMob.VS_NODE(nodeIndex).V_MOVEMENT_PCELL(indiceMov,1)=prevCell;
85 -                structMob.VS_NODE(nodeIndex).V_MOVEMENT_NCELL(indiceMov,1)=currCell;
86 -            else
87 -                structMob.VS_NODE(nodeIndex).V_MOVEMENT_DONE(indiceMov,1)=0;
88 -                structMob.VS_NODE(nodeIndex).V_MOVEMENT_PCELL(indiceMov,1)=prevCell;
89 -                structMob.VS_NODE(nodeIndex).V_MOVEMENT_NCELL(indiceMov,1)=currCell;
90 -            end
91 -        end
92 -    end
93
94      % Adding new fields like the access router which controles the cell
95      % and the previous and new access routers for each movement.
```

Figura 30: Fragmento de código de "mobility2Topology.m"

Así mismo, se prescindió del "Test_Execute.m" incluido con el modelo de movilidad *Random Waypoint* para adecuarlo con los nuevos parámetros y ser quien inicie la simulación.

Este fichero se llamó "execution.m" y un fragmento de su código se puede observar en la siguiente figura.

```
4 -   global xml;
5
6 -   s_input = struct('V_POSITION_X_INTERVAL',[xml.mobility.minX xml.mobility.maxX],...%(m)
7                     'V_POSITION_Y_INTERVAL',[xml.mobility.minY xml.mobility.maxY],...%(m)
8                     'V_CELL_RADIUS_HEX',xml.mobility.radius,...%(m)
9                     'V_SPEED_INTERVAL',[xml.RWP.speed.minSpeed xml.RWP.speed.maxSpeed],...%(m/s)
.0                    'V_PAUSE_INTERVAL',[xml.RWP.pause.minPause xml.RWP.pause.maxPause],...%pause time
.1                    'V_WALK_INTERVAL',[xml.RWP.walk.minWalk xml.RWP.walk.maxWalk],...%walk time (s)
.2                    'V_DIRECTION_INTERVAL',[xml.RWP.direction.minDirection xml.RWP.direction.maxDirect
.3                    'SIMULATION_TIME',xml.simulation.time,...%(s)
.4                    'NB_NODES',xml.simulation.nodes);
.5
.6 -   s_topology = struct('TOP_CONECTIVITY',xml.topology.conectivity,...%(Low,Tree,Full or Trade-Off)
.7                    'TOP_HIERARCHY',xml.topology.hierarchy);...%(Hierarchy)
.8
.9 -   s_traffic = struct('TRA_LAMBDA',xml.traffic.lambda,...%(Low,Tree,Full or Trade-Off)
:0                    'TRA_SESSION_DURATION',xml.traffic.sessionDuration,...%(Hierarchy)
:1                    'TRA_DEMAND_INTERVAL',[xml.traffic.minDemand xml.traffic.maxDemand]);...%(s)
```

Figura 31: Fragmento de código de "exectution"

Por tanto, después de esto se añadió el apartado *Traffic model* a la ventana principal del simulador, indicando el modelo de tráfico a elegir y la tasa de llegada, duración de sesiones y el intervalo de valores de esas sesiones:

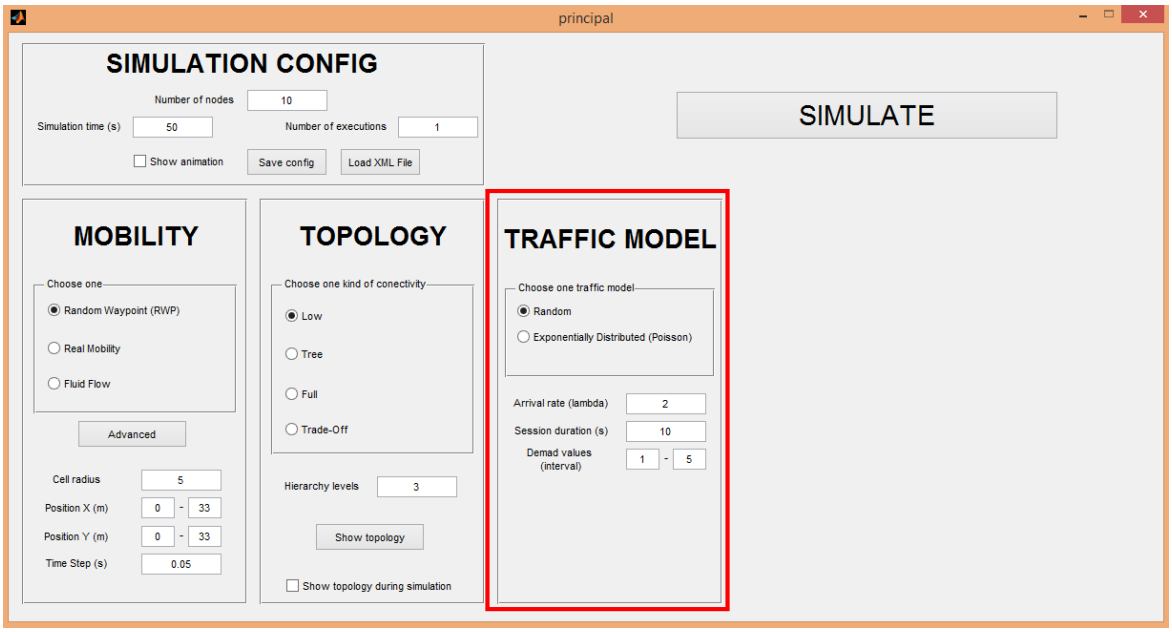


Figura 32: Ventana principal del simulador con el apartado de "Traffic model"

Ya estaban todos los "engranajes" del simulador preparados: parámetros generales de la simulación, movilidad, topología, tráfico…

Después de esto, tocaba implementar los protocolos MIPv6 y PMIPv6, que se verá a continuación.

3.3.7. Implementación de los protocolos MIPv6 y PMIPv6

Para implementar los protocolos MIPv6 y PMIPv6, lo primero que hicimos fue crear en el fichero de ejecución "execution.m" varias variables estructura "struct" para controlar los distintos parámetros.

Una de estas variables se creó para el valor del coste de los distintos tipos de mensajes entre entidades de la red.

Aunque no se utilicen todos, se deja creado para posibles trabajos futuros sobre este simulador:

```
% ...message cost.
s_message_cost = struct('RS',48,...
                        'RA',56,...
                        'NS',64,...
                        'BU',52,... %BU to HA
                        'BA',52,... %BA from HA
                        'PBU',52,... %PBU to LMA
                        'PBA',52,... %PBA from LMA
                        'MBU',52,... %MBU from MAR
                        'MBA',52,... %MBU to MAR
                        'MCReq',52,... %Req to Mobility DB
                        'MCRes',52,... %Res from Mobility DB
                        'HoTI',56,...
                        'CoTI',56,...
                        'HoT',64,...
                        'CoT',64,...
                        'IPIPtunnel',40,...
                        'MPLStunnel',4,...
                        'GRETunnel',44,...
                        'LSPsetup',64,...
                        'userData',120,...
                        'HandoverMessage',48);
```

Figura 33: Variable tipo estructura para el coste de los mensajes

Otra variable se implementó para controlar la configuración de los protocolos, con parámetros como la tasa de transmisión, la matriz de adyacencia y de capacidad, etc.

En la siguiente imagen podemos ver esta variable tipo estructura y sus diferentes parámetros:

```
%...protocol config.
s_config = struct('transmissionRate',64000,...  %Transmission rate in bps
                  'finishDemands',1,...         %Wheather or not to finish
                  'netMatrix',matrix,...
                  'capacityMatrix',capacityMatrix,...
                  'mobilityAnchors',[],...
                  'mobilityAnchorsActive',[],...
                  'd_ha_cn',5,...
                  'd_lma_cn',5,...
                  'd_iler_cn',5,...
                  'd_rootGw_cn',5,...
                  'd_sMAR_mobilityDB',3,...
                  'Bw',100000000,...  %bps
                  'Bwl',11000000,...  %bps
                  'Lwl',0.002,...   %seconds
                  'Lw',0.001,...    %seconds
                  'Pt',0.000001);   %seconds
```

Figura 34: Variable tipo estructura para la configuración del protocolo

Y por último, aunque no menos importante, otra variable tipo estructura donde almacenaremos los valores (como el número de *handovers* entre microceldas y macroceldas, los distintos tipos de coste, demandas bloqueadas, carga de la red…) de la simulación del protocolo para luego obtener resultados y hacer diferentes gráficas:

```
% ...to save the results.
s_result = struct ('numMicroHandovers',0,...
                   'numHandovers',0,...
                   'totalCost',zeros(1,xml.simulation.nodes),... %Cost per node
                   'signallingCostTotal',[0],...
                   'timeSignallingCostTotal',[0],...
                   'signallingCostPerNode',zeros(1,xml.simulation.nodes),... %Signalling cost per node
                   'deliveryCostTotal',[0],...
                   'timeDeliveryCostTotal',[0],...
                   'deliveryCostPerNode',zeros(1,xml.simulation.nodes),... %Delivery cost per node
                   'tunnellingCostTotal',[0],...
                   'timeTunnellingCostTotal',[0],...
                   'tunnellingCostPerNode',zeros(1,xml.simulation.nodes),... %Tunnelling cost per node
                   'handoverLatency',[0],...
                   'handoverLatencyTime',[0],...
                   'packetLoss',[0],...
                   'packetLossTime',[0],...
                   'MALoad',[0],...
                   'MALoadTime',[0],...
                   'MALoadInmediate',[0],...
                   'MALoadInmediateTime',[0],...
                   'mAvailable',capacityMatrix,...
                   'blockedDemands',0,...
                   'blockedDemandsDuringMov',0,...
                   'linkLoad',[0],...
                   'timeLinkLoad',[0]);
```

Figura 35: Variable tipo estructura almacenar los resultados

Para simular los diferentes eventos, lo correcto es ir haciéndolo tratando uno por uno desde el primero al último.

Para hacer eso, tras generar la movilidad y el tráfico, lo primero que hacemos es crear una función “sortEvents” para que creara un vector de eventos ordenados según el tipo, determinándose el evento tipo “0” si se trataba de un movimiento y tipo “1” si era una sesión (y también hemos creado otro de tipo “2” –aunque no ha sido utilizada– por si en trabajos futuros se quisiera utilizar una variable de monitorización que divide el tiempo de simulación entre 100).

Se hizo primero de esta manera para controlar que todo se ejecutase correctamente y las variables tuvieran todos los campos necesarios. En la siguiente imagen se puede observar un fragmento de esta función:

```
179     % This fuction is used to sort the different events
180     % the simulator needs to control.
181     function eventsArr = sortEvents (sMob,simTime)
182
183 -       monitoring = simTime/100;
184 -       arraySize = 0;
185
186 -       for i=1:sMob.NB_NODES
187 -           arraySize = arraySize + size(sMob.VS_NODE(i).V_TIME,1) + size(sMob.VS_NODE(i)
188 -       end
189
190 -       eventsArr(arraySize).time = 0;
191 -       eventsArr(arraySize).node = 0;
192 -       eventsArr(arraySize).movDemMon = 0;
193 -       eventsArr(arraySize).index = 0;
194 -       eventsArr(arraySize).mobManagement = 0;
195 -       index = 0;
196
197 -       for nodeIndex=1:sMob.NB_NODES
198 -           for i=1:size(sMob.VS_NODE(nodeIndex).V_TIME,1) % Each movement
199 -               index=index+1;
200 -               eventsArr(index).time = sMob.VS_NODE(nodeIndex).V_TIME(i);
201 -               eventsArr(index).node = nodeIndex;
```

Figura 36: Fragmento de código de la función "sortEvents" del script "execution.m" donde se ordenan los diferentes eventos

Una vez teniendo ese vector de eventos, ahora sí era conveniente ordenarlos según el tiempo. Y esto se consiguió gracias a la función "nestedSortStruct", disponible también gracias a la comunidad de Matlab en Matlab Central.

Los eventos irán ordenados de la siguiente manera:

1) Por tiempo
2) En caso de que el tiempo fuera el mismo (poco probable), por número de nodo.
3) En caso de que todo lo anterior fuera idéntico (muy poco probable), por tipo de evento (movimiento o demanda).

En la siguiente imagen podemos ver la llamada a "nestedSortStruct" desde "execution.m" para ordenar los eventos:

```
156         % Saving results...
157 -       eventsArray = sortEvents(s_mobility,s_input.SIMULATION_TIME);
158
159         % Sorting the different events...
160 -       inOrder = nestedSortStruct(eventsArray,{'time','node','movDemMon'});
```

Figura 37: Fragmento de código de "execution" donde se ordenan los eventos en función del tiempo

Por tanto, una vez con los eventos ordenados en función del tiempo, era hora de simularlos. Para hacer esto, se creó un nuevo fichero llamado "protocolSim." que devolviera el resultado de la simulación del protocolo, pasándole como parámetro los datos de movilidad, número de nodos, los eventos ordenados, la variable para resultados, la configuración del protocolo y el coste de los mensajes:

```
16    function [result] = protocolSim(s_mobility,nMN,eventsInOrder,s_result,s_config,s_message_cost)
17
18 -  s_mobility.NB_NODES=nMN;
19 -  s_mobility.VS_NODE=s_mobility.VS_NODE(1:nMN);
20
21 -  resultSim.MIP=s_result;
22 -  resultSim.PMIP=s_result;
23
24 -  s_config.mobilityAnchors=(1);
25
26 -  timeLastDemand=0;
27 -  firstTime=0;
28
29 -  for i=1:size(eventsInOrder,2) %i is the event's index
30 -      event=eventsInOrder(i); %In event is the event that is going to be processed
31 -      if event.movDemMon==1 % The event is a new session
32 -          for j=1:size(MN,2) %MN's index is j
33 -              k=1;
34 -              while k<=size(MN{j}.connection,2) %active sessions' index of each MN is k
35 -                  timeToFinish=MN{j}.connection{k}.timeToFinish;
36 -                  if (event.time>timeToFinish) && (MN{j}.connection{k}.free==0) %The demand has finished
```

Figura 38: Fragmento de código de "protocolSim.m"

Esta función trata básicamente de un bucle que recorre el vector de eventos ordenados por el tiempo y los va tratando según sean movimientos ("Tipo 0") o demandas ("Tipo 1").

Una vez conseguido simular MIPv6, PMIPv6 es muy similar ya que sólo hay que cambiar algunos parámetros como podría ser, por ejemplo, los límites de la comunicación cuando se hace el *tunnelling* (se pueden ver las diferencias entre MIPv6 y PMIPv6 en el capítulo 2 de este documento).

Por eso, se decidió simular los dos protocolos a la vez, dando la posibilidad al usuario a través de la interfaz gráfica de elegir si quiere ver los resultados de uno de los dos o de los dos a la vez. Así, se tendrían los resultados de la simulación de ambos protocolos (guardándolos en una variable llamada "resultSim" tipo celda) y se mostrarían sólo los resultados de los que eligiese el usuario.

En las imágenes que se mostrarán a continuación se podrá ver cómo se han implementado los protocolos en busca de los resultados para analizar su rendimiento en la red. Empezamos con el cálculo del coste de *routing*, también llamado **Delivery Cost,** total y para cada nodo, utilizando los parámetros que habíamos configurado previamente y siguiendo el funcionamiento de los protocolos estudiados:

```
%ROUTING-COST
%Calculate the cost of sending that information in the time between events
%Rounting cost in HB-DMM is expressed as the sum of the
%direct and the indirect routing
partialCostMIP=s_config.transmissionRate*tBetweenEvents;%These are the bit data sent in that time
partialCostMIP=(ceil(partialCostMIP/s_message_cost.userData)*s_message_cost.userData)*s_config.d_ha_cn+...  %
    (ceil(partialCostMIP/s_message_cost.userData)*s_message_cost.userData)*distanceMIPConnection+... %data bit
    ceil(partialCostMIP/s_message_cost.userData)*s_message_cost.IPIPtunnel*distanceMIPConnection; %overhead b

partialCostPMIP=s_config.transmissionRate*tBetweenEvents;%These are the bit data sent in that time
partialCostPMIP=(ceil(partialCostPMIP/s_message_cost.userData)*s_message_cost.userData)*s_config.d_lma_cn+...
    (ceil(partialCostPMIP/s_message_cost.userData)*s_message_cost.userData)*(distancePMIPConnection-1)+... %da
    ceil(partialCostPMIP/s_message_cost.userData)*s_message_cost.IPIPtunnel*(distancePMIPConnection-1); %overh

resultSim.MIP.totalCost(j)=resultSim.MIP.totalCost(j)+partialCostMIP;
lastCost=resultSim.MIP.deliveryCostTotal(end);
resultSim.MIP.deliveryCostTotal(end+1)=lastCost+partialCostMIP;
resultSim.MIP.timeDeliveryCostTotal(end+1)=currentTime;

resultSim.PMIP.totalCost(j)=resultSim.PMIP.totalCost(j)+partialCostPMIP;
lastCost=resultSim.PMIP.deliveryCostTotal(end);
resultSim.PMIP.deliveryCostTotal(end+1)=lastCost+partialCostPMIP;
resultSim.PMIP.timeDeliveryCostTotal(end+1)=currentTime;
```

Figura 39: Fragmento de código del cálculo del coste de routing ("Delivery Cost") en "protocolSim.m"

Total Cost será la suma de todos los costes por nodo. Y por otro lado, como se puede observar en la imagen anterior y en las posteriores, las diferencias entre MIPv6 y PMIPv6 son mínimas.

También, calculamos el coste de *tunnelling*, **Tunnelling Cost** y lo almacenamos, además de cada vez que ocurre un evento, también por cada nodo:

```
%TUNNELING-COST
%Calculate the cost of tunneling.
partialCostMIP=s_config.transmissionRate*tBetweenEvents;%These are the bit data sent in that time
partialCostMIP=ceil(partialCostMIP/s_message_cost.userData)*s_message_cost.IPIPtunnel*distanceMIPConnection; %overh
lastCost=resultSim.MIP.tunnellingCostTotal(end);
resultSim.MIP.tunnellingCostTotal(end+1)=lastCost+partialCostMIP;
resultSim.MIP.timeTunnellingCostTotal(end+1)=currentTime;

partialCostPMIP=s_config.transmissionRate*tBetweenEvents;%These are the bit data sent in that time
partialCostPMIP=ceil(partialCostPMIP/s_message_cost.userData)*s_message_cost.IPIPtunnel*(distanceMIPConnection-1);
lastCost=resultSim.PMIP.tunnellingCostTotal(end);
resultSim.PMIP.tunnellingCostTotal(end+1)=lastCost+partialCostPMIP;
resultSim.PMIP.timeTunnellingCostTotal(end+1)=currentTime;

%Per node
lastCost=resultSim.MIP.tunnellingCostPerNode(event.node);
resultSim.MIP.tunnellingCostPerNode(event.node)=lastCost+partialCostMIP;

lastCost=resultSim.PMIP.tunnellingCostPerNode(event.node);
resultSim.PMIP.tunnellingCostPerNode(event.node)=lastCost+partialCostPMIP;
```

Figura 40: Fragmento de código del cálculo del coste de tunnelling ("Tunneling Cost") en "protocolSim.m"

Así mismo, somos capaces de calcular la carga del *mobility* anchor, **Mobility Anchor Load**:

```
%MOBILITY-ANCHOR-LOAD
resultSim.MIP.MALoad(end+1)=resultSim.MIP.deliveryCostTotal(end)/size(s_config.mobilityAnchors,2);
resultSim.MIP.MALoadTime(end+1)=currentTime;

resultSim.PMIP.MALoad(end+1)=resultSim.PMIP.deliveryCostTotal(end)/size(s_config.mobilityAnchors,2);
resultSim.PMIP.MALoadTime(end+1)=currentTime;

%MOBILITY-ANCHOR-LOAD-INMEDIATE
%Number of mobility anchors in this moment
resultSim.MIP.MALoadInmediate(end+1)=resultSim.MIP.deliveryCostTotal(end)/size(s_config.mobilityAnchors,2);
resultSim.MIP.MALoadInmediateTime(end+1)=currentTime;

resultSim.PMIP.MALoadInmediate(end+1)=resultSim.PMIP.deliveryCostTotal(end)/size(s_config.mobilityAnchors,2);
resultSim.PMIP.MALoadInmediateTime(end+1)=currentTime;
```

Figura 41: Fragmento de código del cálculo de la carga del mobility anchor ("Mobility Anchor Load") en "protocolSim.m"

Calculamos el coste de señalización, **Signalling Cost** y lo almacenamos como el de Tunnelling Cost, por aparición y por nodo:

```
if firstTime == 0
    partialCostMIP=s_message_cost.BU*size(tmpPath,2)+s_message_cost.BA*size(tmpPath,2);

    resultSim.MIP.totalCost(event.node)=resultSim.MIP.totalCost(event.node)+partialCostMIP;
    lastCost=resultSim.MIP.signallingCostTotal(end); %Total
    resultSim.MIP.signallingCostTotal(end)=lastCost+partialCostMIP;
    resultSim.MIP.timeSignallingCostTotal(end)=event.time;

    lastCost=resultSim.MIP.signallingCostPerNode(event.node); %Per node
    resultSim.MIP.signallingCostPerNode(event.node)=lastCost+partialCostMIP;

    partialCostPMIP=s_message_cost.PBU*(size(tmpPath,2)-1)+s_message_cost.PBA*(size(tmpPath,2)-1);

    resultSim.PMIP.totalCost(event.node)=resultSim.PMIP.totalCost(event.node)+partialCostPMIP;
    lastCost=resultSim.PMIP.signallingCostTotal(end); %Total
    resultSim.PMIP.signallingCostTotal(end)=lastCost+partialCostPMIP;
    resultSim.PMIP.timeSignallingCostTotal(end)=event.time;

    lastCost=resultSim.PMIP.signallingCostPerNode(event.node); %Per node
    resultSim.PMIP.signallingCostPerNode(event.node)=lastCost+partialCostPMIP;
```

Figura 42: Fragmento de código del cálculo del coste de señalización ("Signalling Cost") en "protocolSim.m"

Con los datos que tenemos, también podemos calcular la latencia del *handover*, **Handover Latency**:

```
%3. HANDOVER-LATENCY
%2*t(s_u,h_MN-AMA)
%2*     (s/Bwl)+Lwl+h_x-y*((s/Bw)+Lw)+(h_x-y+1)*P_t
if firstTime == 0
    resultSim.MIP.handoverLatency(end)=2*(s_message_cost.HandoverMessage/s_config.Bwl)+s_config.Lwl+...
        size(MN{event.node}.path,2)*((s_message_cost.HandoverMessage/s_config.Bw)+s_config.Lw)+...
        (size(MN{event.node}.path,2)+1)*s_config.Pt;
    resultSim.MIP.handoverLatencyTime(end)=event.time;

    resultSim.PMIP.handoverLatency(end)=2*(s_message_cost.HandoverMessage/s_config.Bwl)+s_config.Lwl+...
        (size(MN{event.node}.path,2)-1)*((s_message_cost.HandoverMessage/s_config.Bw)+s_config.Lw)+...
        size(MN{event.node}.path,2)*s_config.Pt;
    resultSim.PMIP.handoverLatencyTime(end)=event.time;
else
    resultSim.MIP.handoverLatency(end+1)=2*(s_message_cost.HandoverMessage/s_config.Bwl)+s_config.Lwl+...
        size(MN{event.node}.path,2)*((s_message_cost.HandoverMessage/s_config.Bw)+s_config.Lw)+...
        (size(MN{event.node}.path,2)+1)*s_config.Pt;
    resultSim.MIP.handoverLatencyTime(end+1)=event.time;

    resultSim.PMIP.handoverLatency(end+1)=2*(s_message_cost.HandoverMessage/s_config.Bwl)+s_config.Lwl+...
        (size(MN{event.node}.path,2)-1)*((s_message_cost.HandoverMessage/s_config.Bw)+s_config.Lw)+...
        size(MN{event.node}.path,2)*s_config.Pt;
    resultSim.PMIP.handoverLatencyTime(end+1)=event.time;
end
```

Figura 43: Fragmento de código del cálculo de la latencia de handover ("Handover Latency") en "protocolSim.m"

Así mismo, también podemos calcular la cantidad de paquetes perdidos, **Packet Loss**:

```
%4. PACKET-LOSS
%Each session that needs to be updated has packet losses
if firstTime == 0
    partialCostMIP=resultSim.MIP.handoverLatency(end)*s_config.transmissionRate;
    lastCost=resultSim.MIP.packetLoss(end);
    resultSim.MIP.packetLoss(end)=lastCost+partialCostMIP;
    resultSim.MIP.packetLossTime(end)=event.time;

    resultSim.MIP.numHandovers=resultSim.MIP.numHandovers+1;

    partialCostPMIP=resultSim.PMIP.handoverLatency(end)*s_config.transmissionRate;
    lastCost=resultSim.PMIP.packetLoss(end);
    resultSim.PMIP.packetLoss(end)=lastCost+partialCostPMIP;
    resultSim.PMIP.packetLossTime(end)=event.time;

    resultSim.PMIP.numHandovers=resultSim.PMIP.numHandovers+1;
```

Figura 44: Fragmento de código del cálculo de la pérdida de paquetes ("Packet Loss") en "protocolSim.m"

También, somos capaces de calcular la carga de la red, **Link Load**, gracias a la matriz de capacidad disponible que tenemos y al uso de la red:

```
%For each event, links instantaneous load is calculated.
loadCapacity=0;
loadmAvailableMIP=0;
loadmAvailablePMIP=0;
for k=1:size(s_config.capacityMatrix,2)
    for p=1:size(s_config.capacityMatrix,2)
        if s_config.capacityMatrix (k,p)~=0
            loadCapacity=loadCapacity+s_config.capacityMatrix(k,p);
            loadmAvailableMIP=loadmAvailableMIP+resultSim.MIP.mAvailable(k,p);
            loadmAvailablePMIP=loadmAvailablePMIP+resultSim.PMIP.mAvailable(k,p);
        end
    end
end
loadUsedMIP=loadCapacity-loadmAvailableMIP;
loadUsedPMIP=loadCapacity-loadmAvailablePMIP;

resultSim.MIP.linkLoad(i)=(loadUsedMIP/loadCapacity)*100;
resultSim.MIP.timeLinkLoad(i)=event.time;

resultSim.PMIP.linkLoad(i)=(loadUsedPMIP/loadCapacity)*100;
resultSim.PMIP.timeLinkLoad(i)=event.time;
```

Figura 45: Fragmento de código del cálculo de la carga de la red ("Link Load") en "protocolSim.m"

Todo esto anterior es posible gracias a unas funciones como "findRoute" y "updateRoute", que se encargan de ver si existe un camino que permita la conexión y de actualizar los paths existentes, respectivamente.

```
function  [path,possible,newmAvailable] = findRoute(connectivityMatrix,source,destiny,demandValue,resultSim)
%cost=Inf;
%possible=1; %It shows if there is path or not
tmpConnectivity=zeros(size(connectivityMatrix,1),size(connectivityMatrix,1));
%Deleting links that can't afford the request
for i=1:size(connectivityMatrix,1)
    for j=1:size(connectivityMatrix,1)
        if (resultSim.MIP.mAvailable(i,j)<demandValue)
            tmpConnectivity(i,j)=0;
        else
            tmpConnectivity(i,j)=connectivityMatrix(i,j);
        end
    end
end

%tmpConnectivity is the connectivity matrix that could serve the demand
[e, L] = dijkstra (tmpConnectivity,source,destiny);
cost=e;
```

Figura 46: Fragmento de código del cálculo de rutas en "protocolSim.m"

Una vez todo calculado, desde "execution.m" sólo tenemos que llamar a la función "protocolSim", pasándole los parámetros adecuados, para que se encargue de realizar la simulación de los protocolos y lo guarde en una variable "results" tipo celda que tendrá *i* posiciones dependiendo de las *i* simulaciones que hagamos:

```
% Simulating protocol...
fprintf ('Simulating protocol(s)...\n');
results{i} = protocolSim(s_mobility,s_input.NB_NODES,inOrder,s_result,s_config,s_message_cost);
```

Figura 47: Llamada a "protocolSim.m" desde "execution.m"

Por tanto, en la ventana principal del simulador ya se pudo añadir otro apartado más, **Protocol**, para seleccionar el protocolo (MIPv6/ PMIPv6) y mostrar sus resultados.

En esta primera versión con el nuevo apartado en la ventana, sólo se podía escoger uno de los dos protocolos. En versiones posteriores esto se cambiará para elegir los dos si se desea.

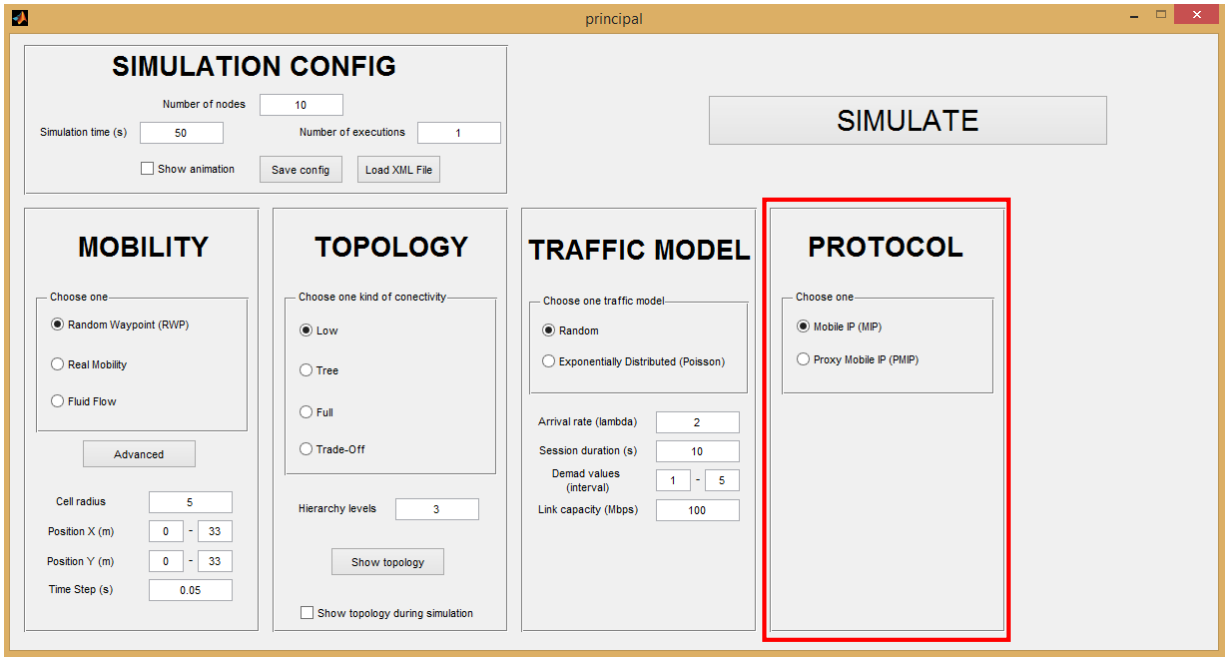


Figura 48: Ventana principal del simulador con el campo "Protocol" añadido

3.3.8. Otras mejoras incluidas

Antes de mostrar resultados, se decidieron hacer algunas mejoras en el simulador:

- En la animación, colorear las celdas según pertenecieran a un router de acceso o a otro:

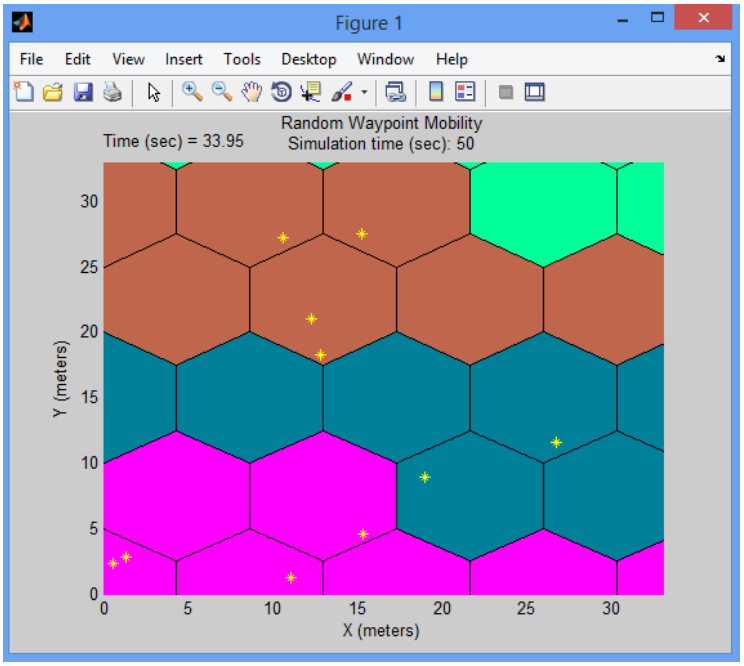


Figura 49: Ventana de animación con celdas coloreadas

- Desactivar campos en **Mobility**, como los dos tipos de modelos de movimientos no utilizados (para que no los pudiese seleccionar el usuario al no estar implementados) y la posición inicial de los ejes X e Y, ya que se comportaba de una manera extraña si tenían valores negativos.

3.3.9. Resultados y gráficas

Para mostrar los resultados y gráficas, se decidió crear un nuevo apartado en la ventana principal, **Results**, para que el usuario indique qué gráficas desea ver, así como si desea un fichero de log, como se puede observar en la siguiente imagen:

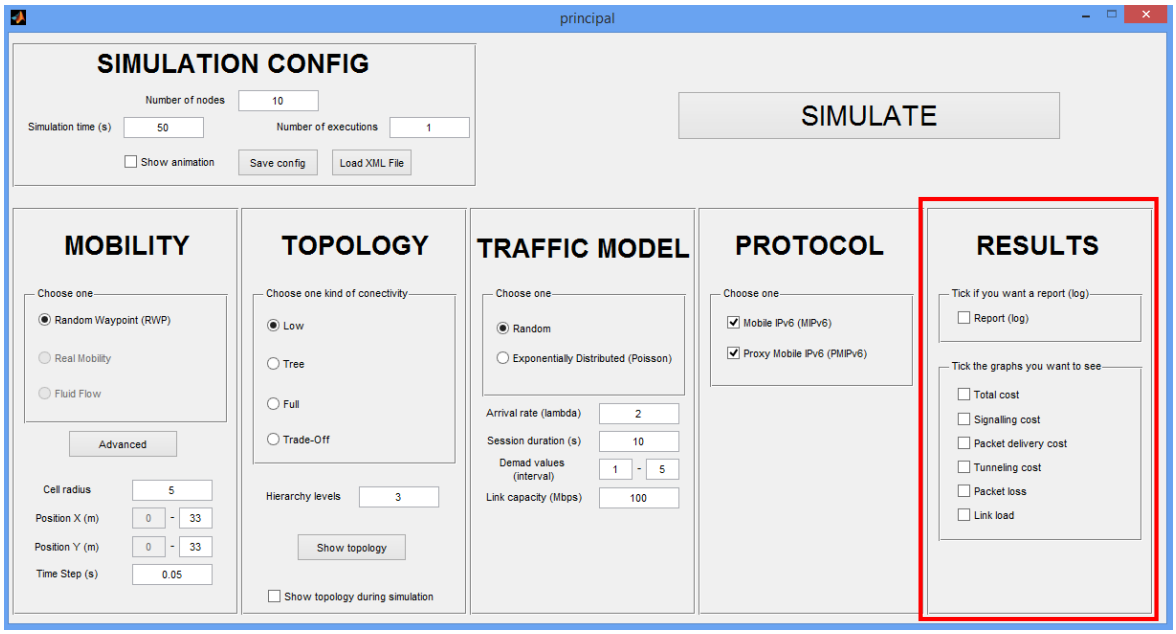


Figura 50: Ventana principal del simulador con el apartado Results añadido

Para la creación del archivo de log y de las gráficas se creó otro script de Matlab llamado "logAndGraphs.m" que analizaremos en los siguientes apartados.

3.3.9.1. Fichero de log

Es un fichero en formato ".txt" en el que se indica al usuario la configuración que ha elegido y donde se almacenan algunos resultados que no merecen la pena ser representados en gráficas al ser un solo número o constantes.

```
if (log == 1)
    reloj=clock;
    myClock=fix(reloj);
    minutes=myClock(5);
    seconds=myClock(6);
    if (minutes<10)
        if (seconds<10)
            myTime=sprintf('%d:0%d:0%d',myClock(4),myClock(5),myClock(6));
        else
            myTime=sprintf('%d:0%d:%d',myClock(4),myClock(5),myClock(6));
        end
    else
        if (seconds<10)
            myTime=sprintf('%d:%d:0%d',myClock(4),myClock(5),myClock(6));
        else
            myTime=sprintf('%d:%d:%d',myClock(4),myClock(5),myClock(6));
        end
    end
```

Figura 51: Fragmento de código de la creación del archivo log en "logAndGraphs"

Y, finalmente, tras la ejecución y creación del fichero log, éste tendría un aspecto como se muestra en las siguientes imágenes:

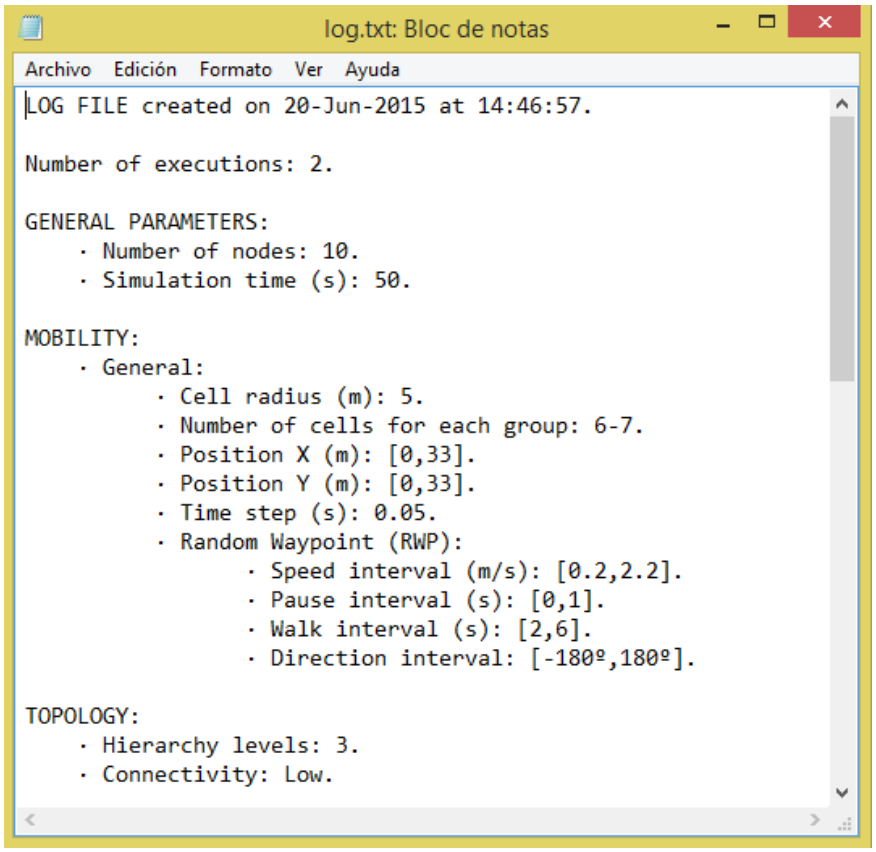


Figura 52: Fichero log. Parte 1

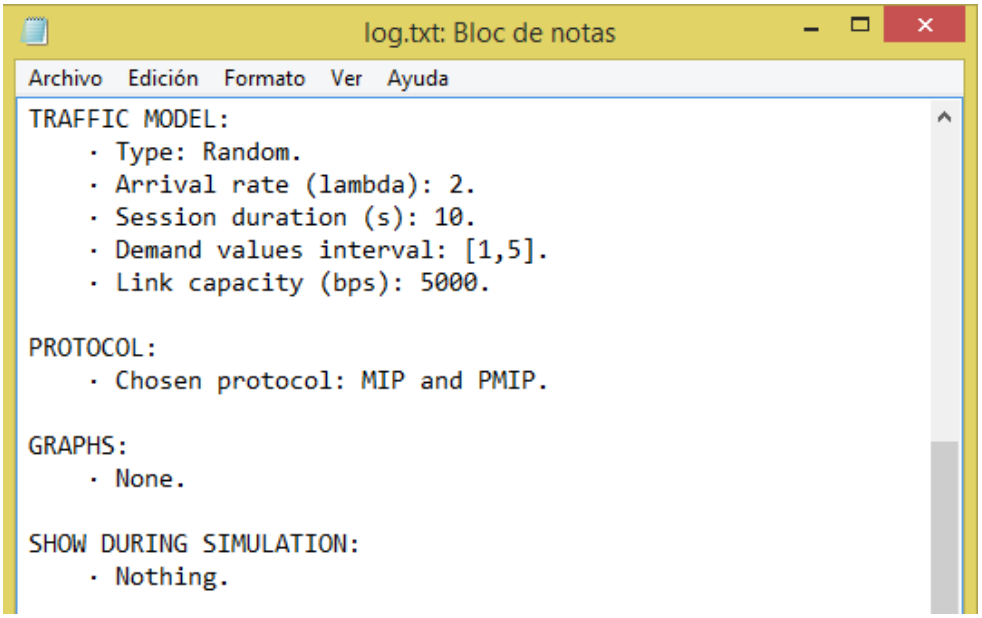


Figura 53: Fichero log. Parte 2

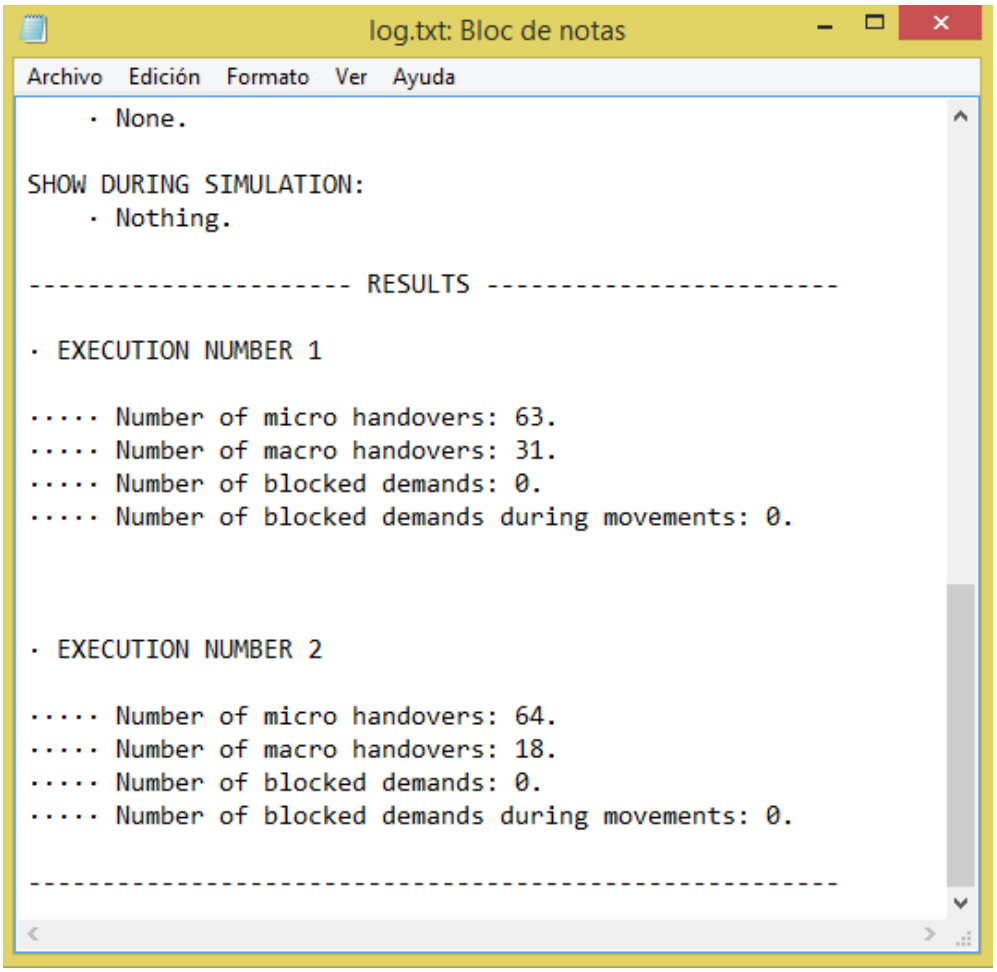


Figura 54: Fichero log. Parte 3

3.3.9.2. Gráficas

Cabe destacar que las gráficas se hacen respecto a la última simulación. Si, por ejemplo, se hicieran dos simulaciones, las gráficas mostrarían los resultados de la última.

Se deja como trabajo futuro hacer múltiples repeticiones en las que se pueda cambiar algún parámetro y hacer estudios estadísticos sobre las funciones.

En la siguiente imagen podemos ver como ejemplo cómo se han ido creando las gráficas distinguiendo si el usuario ha elegido mostrar MIPv6, PMIPv6 o ambos. En este caso, la gráfica del coste de señalización:

```
if (signallingCost == 1)
    switch xml.protocol.type
        case 0 %MIP
            x=results{xml.simulation.repetitions}.MIP.timeSignallingCostTotal;
            y=results{xml.simulation.repetitions}.MIP.signallingCostTotal;
            figure('Units', 'pixels', ...
            'Position', [400 200 500 375]);

            plot(x,y,'Color','b','linewidth',3);
            hTitle  = title ('SIGNALLING COST');
            hXLabel = xlabel('Simulation time (s)');
            hYLabel = ylabel('Cost (bytes)');

            set( gca                       , ...
                'FontName'   , 'Helvetica' );
                set([hTitle, hXLabel, hYLabel], ...
                'FontName'   , 'AvantGarde');
                set([hXLabel, hYLabel]  , ...
                'FontSize'   , 10          );
```

Figura 55: Fragmento de código de la creación de las gráficas del coste de señalización

Y todas las gráficas son similares, mostrando MIPv6 cuando el usuario elige MIPv6 o mostrando PMIPv6 si el usuario así lo requiere.

Después de crear todas las gráficas, sólo basta con llamar a "logAndGraphs.m" desde "execution.m", pasándole los parámetros necesarios:

```
fprintf ('Saving and showing results...');
logAndGraphs(inOrder,results,s_graphs.LOG,s_graphs.TOTAL_COST,s_graphs.SIG
```

Figura 56: Llamada a "logAndGraphs.m" desde "execution.m"

Si el usuario decide mostrar los resultados para los dos protocolos, en las gráficas se realiza una comparativa. No obstante, todos los tipos de gráficas serán vistos en el siguiente capítulo de este documento.

3.3.10. Últimas mejoras del simulador

Con ya toda la maquinaria en marcha, se decidió realizar unas últimas mejoras sobre el simulador, como cambiar algunas palabras y tipo de letra y colores para que la interfaz gráfica fuese más amigable de cara al usuario:

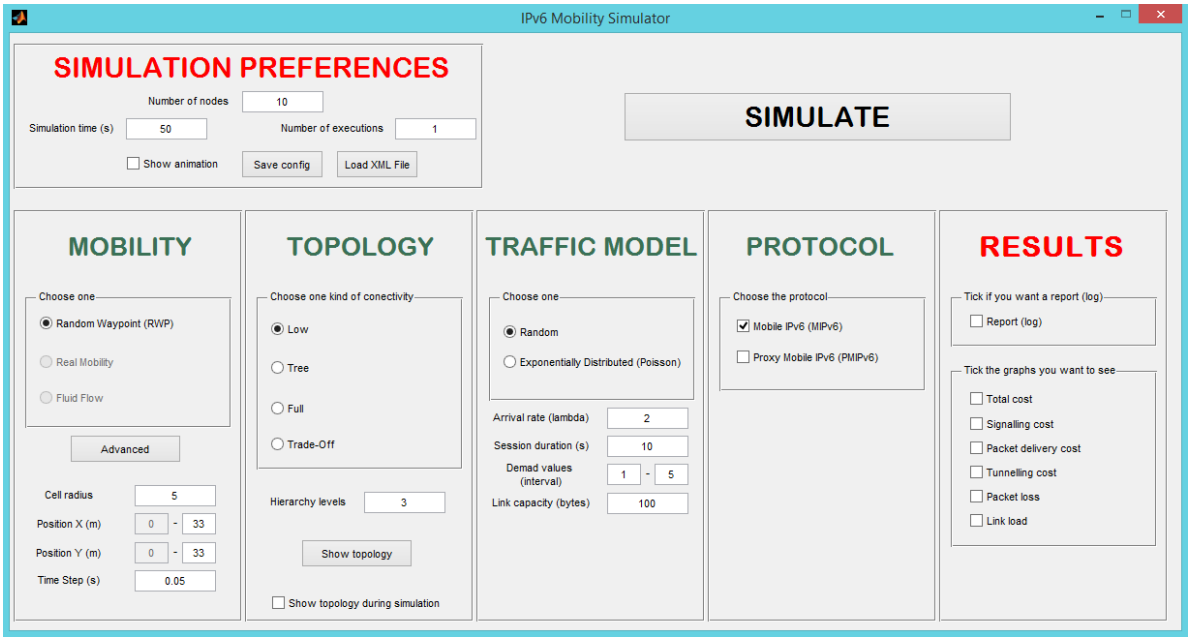


Figura 57: Diseño final de la ventana principal del simulador

4. RESULTADOS

Se realizará una simulación de prueba de 7200 segundos (2 horas) para mostrar todos los resultados y gráficas. Se utilizará la configuración mostrada en la siguiente imagen:

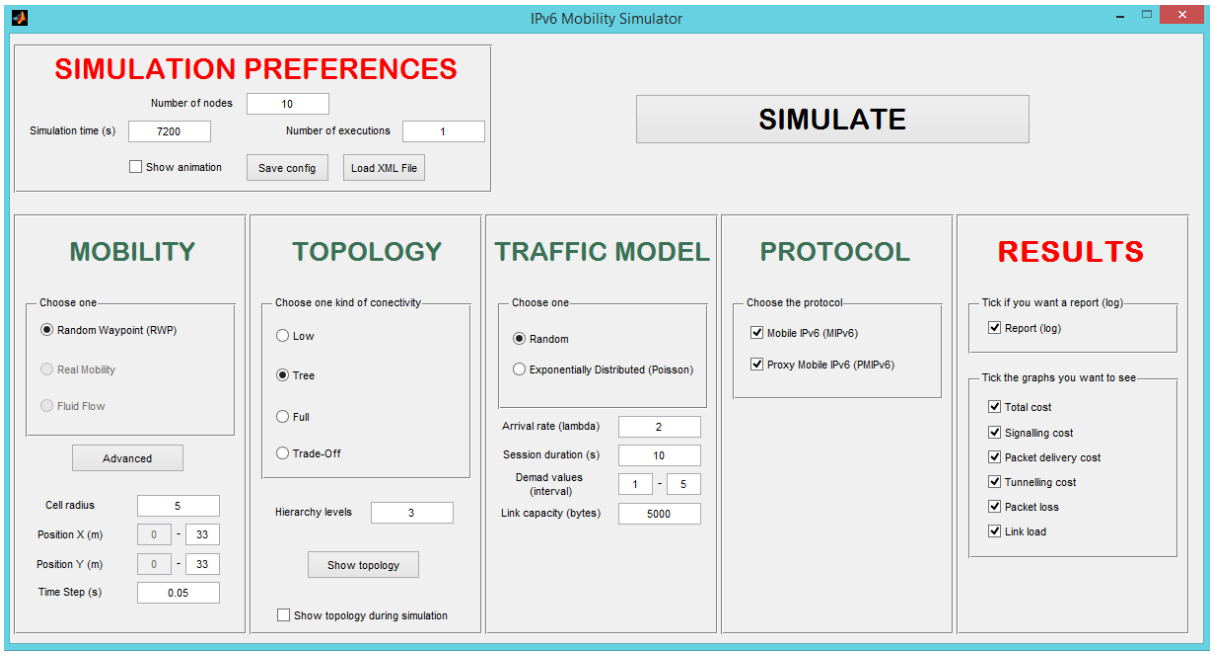


Figura 58: Configuración utilizada en el test de 7200 segundos

Tras pulsar el botón "**Simulate**", comenzará la simulación con los datos que hemos introducido. Irán apareciendo mensajes diciendo el estado de la simulación y cuando finalice, se mostrará esto:

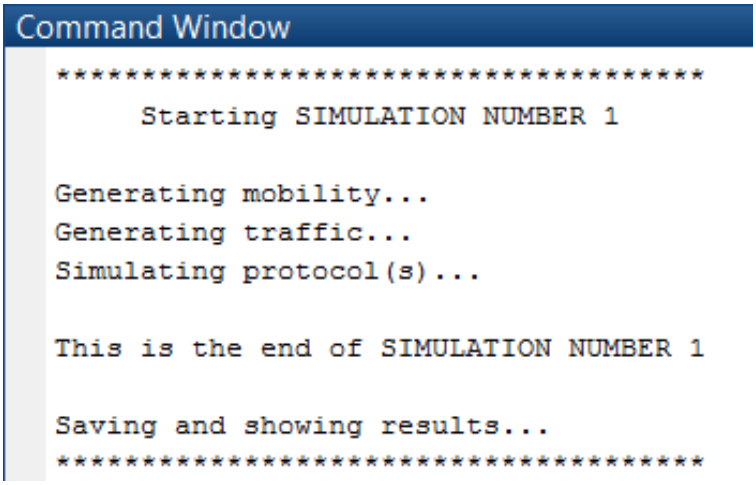


Figura 59: Ventana de comandos durante la simulación

Si hubiéramos indicado más repeticiones, haría lo mismo con todas. Igualmente, si se hubiera seleccionado mostrar la animación y/o la topología durante la simulación, el simulador las hubiera mostrado. En nuestro caso, no hemos indicado eso y, por tanto, muestra sólo la imagen anterior.

A continuación, analizaremos los resultados obtenidos de esta simulación de 7200 segundos.

4.1. Estructura de datos resultante

Después de la simulación, todos los resultados obtenidos han sido guardados en la variable "**results**", donde existe un campo para el protocolo MIPv6 y otro para PMIPv6, como se puede observar en las siguientes imágenes:

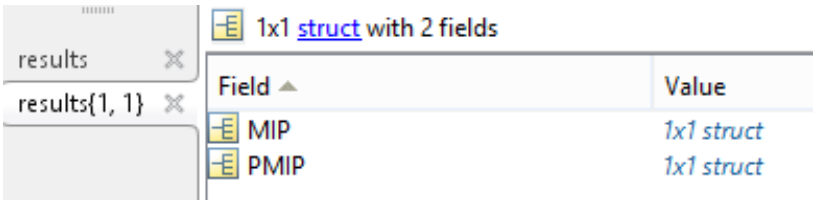


Figura 60: Variable "results" tras la simulación

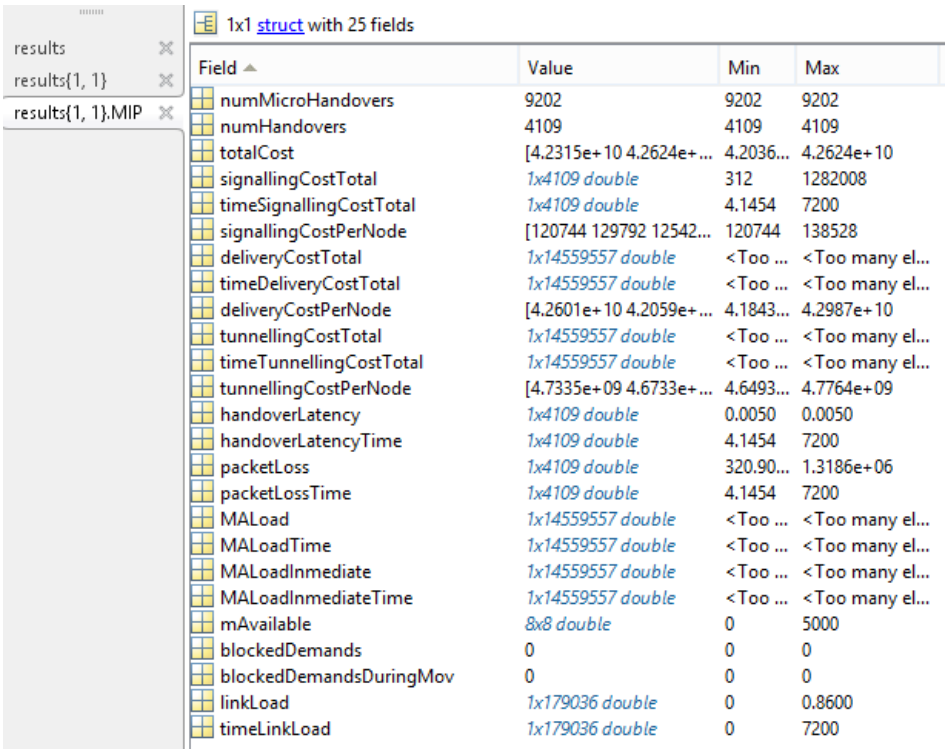
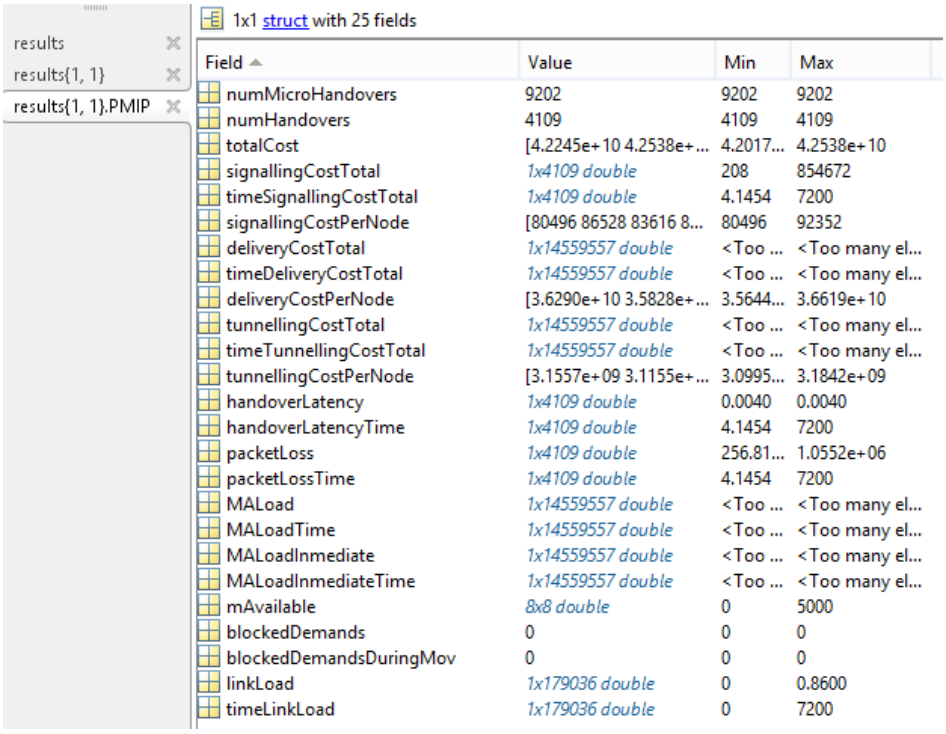


Figura 61: Valores obtenidos tras la simulación para MIPv6



Field	Value	Min	Max
numMicroHandovers	9202	9202	9202
numHandovers	4109	4109	4109
totalCost	[4.2245e+10 4.2538e+...	4.2017...	4.2538e+10
signallingCostTotal	*1x4109 double*	208	854672
timeSignallingCostTotal	*1x4109 double*	4.1454	7200
signallingCostPerNode	[80496 86528 83616 8...	80496	92352
deliveryCostTotal	*1x14559557 double*	<Too ...	<Too many el...
timeDeliveryCostTotal	*1x14559557 double*	<Too ...	<Too many el...
deliveryCostPerNode	[3.6290e+10 3.5828e+...	3.5644...	3.6619e+10
tunnellingCostTotal	*1x14559557 double*	<Too ...	<Too many el...
timeTunnellingCostTotal	*1x14559557 double*	<Too ...	<Too many el...
tunnellingCostPerNode	[3.1557e+09 3.1155e+...	3.0995...	3.1842e+09
handoverLatency	*1x4109 double*	0.0040	0.0040
handoverLatencyTime	*1x4109 double*	4.1454	7200
packetLoss	*1x4109 double*	256.81...	1.0552e+06
packetLossTime	*1x4109 double*	4.1454	7200
MALoad	*1x14559557 double*	<Too ...	<Too many el...
MALoadTime	*1x14559557 double*	<Too ...	<Too many el...
MALoadInmediate	*1x14559557 double*	<Too ...	<Too many el...
MALoadInmediateTime	*1x14559557 double*	<Too ...	<Too many el...
mAvailable	*8x8 double*	0	5000
blockedDemands	0	0	0
blockedDemandsDuringMov	0	0	0
linkLoad	*1x179036 double*	0	0.8600
timeLinkLoad	*1x179036 double*	0	7200

Figura 62: Valores obtenidos tras la simulación para PMIPv6

Pero estos datos, por sí solos, dicen mucho pero pueden ser un poco difíciles de entender. Por eso llegamos al siguiente paso: representarlos en gráficas.

4.2. Resultados experimentales

Podemos ver el archivo de *log* que se ha creado tras la simulación:

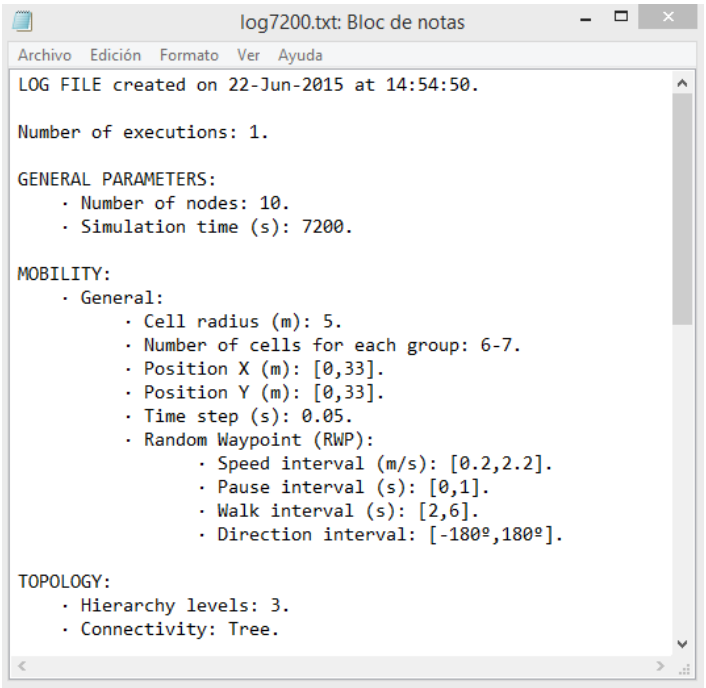


Figura 63: Fichero de log tras la simulación. Parte 1

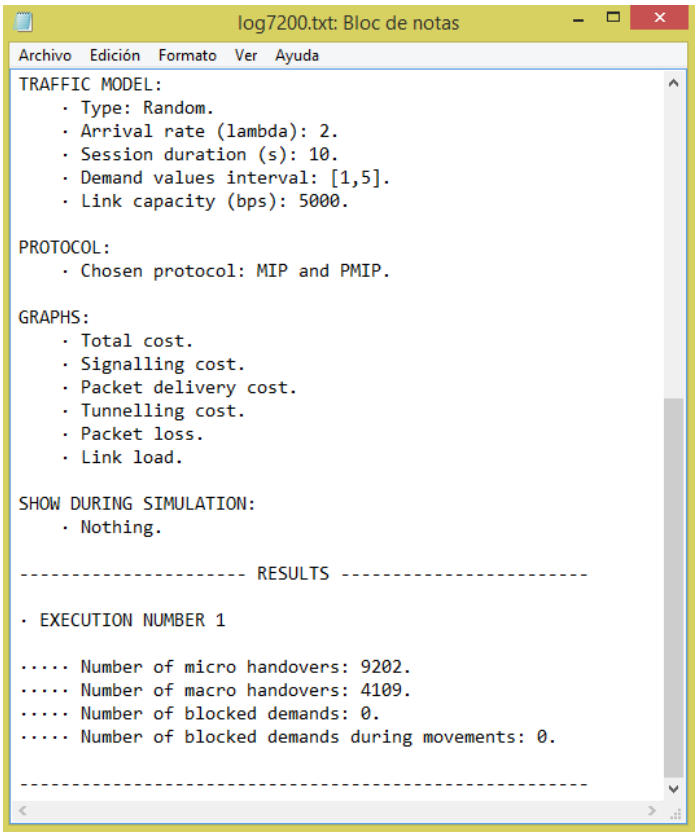


Figura 64: Fichero de log tras la simulación. Parte 2

Y a continuación, se mostrarían todas las gráficas que hemos seleccionado. En los siguientes apartados se presentará todas las gráficas posibles que se pueden obtener.

4.2.1. MIPv6

Empezamos representando el coste total por cada nodo, **Total Cost**:

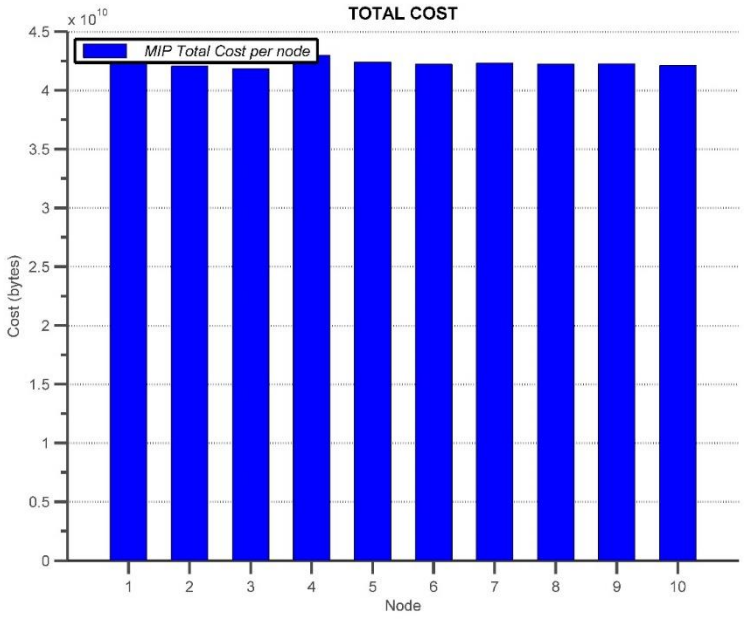


Figura 65: Gráfica de Total Cost para MIPv6

También, obtenemos la gráfica del coste de señalización en función del tiempo. Como se trata de una variable acumulativa, el coste será cada vez mayor:

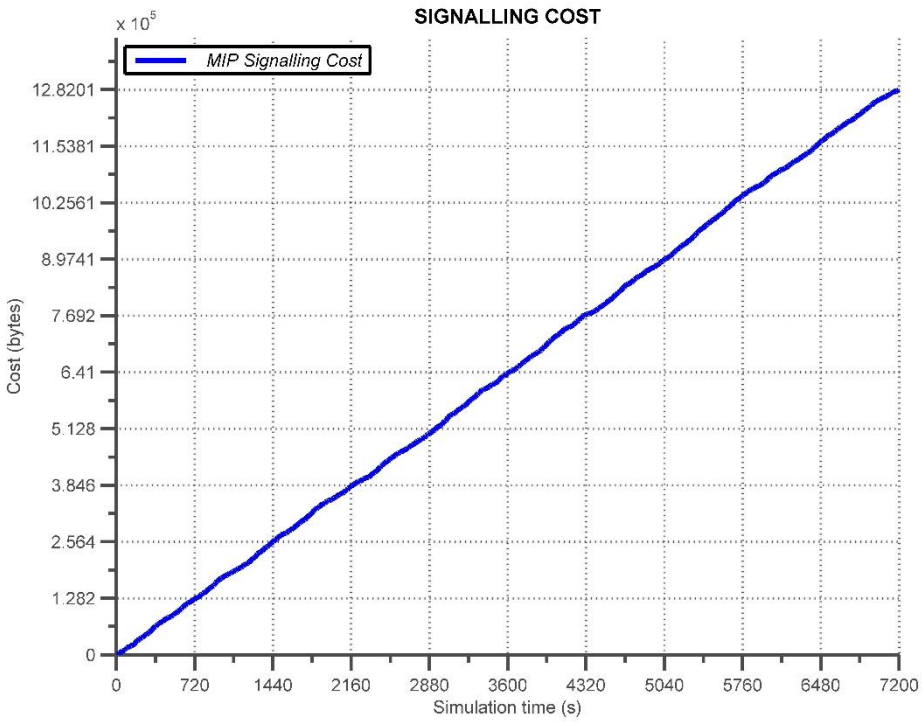


Figura 66: Gráfica del coste de señalización para MIPv6 en función del tiempo

Además, podemos obtener la gráfica del coste de señalización por nodo:

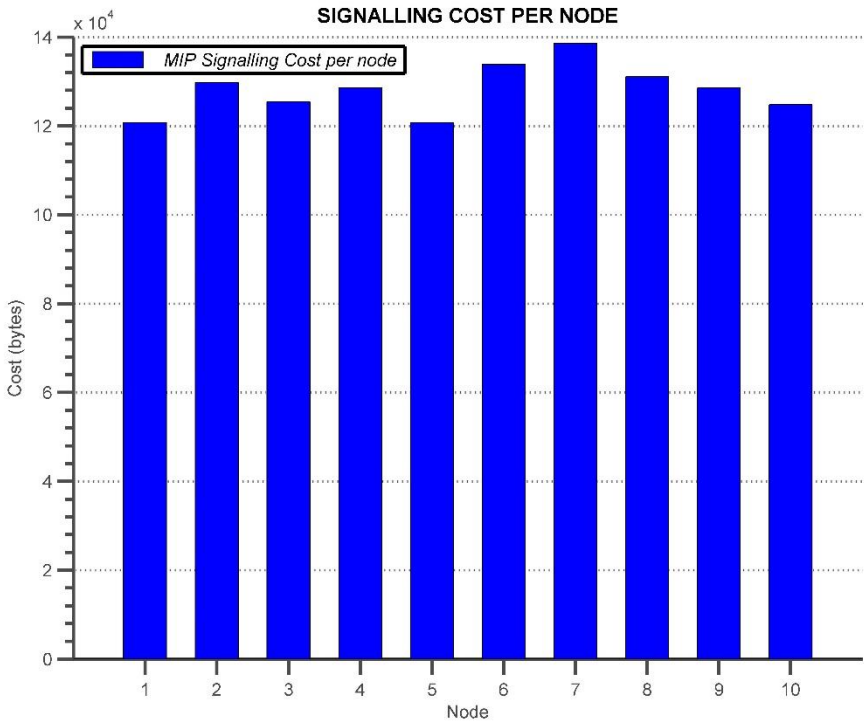


Figura 67: Gráfica del coste de señalización por nodo para MIPv6

Igual que en el caso anterior, obtenemos la gráfica del coste de entrega de paquetes, **Packet Delivery Cost**, en función del tiempo:

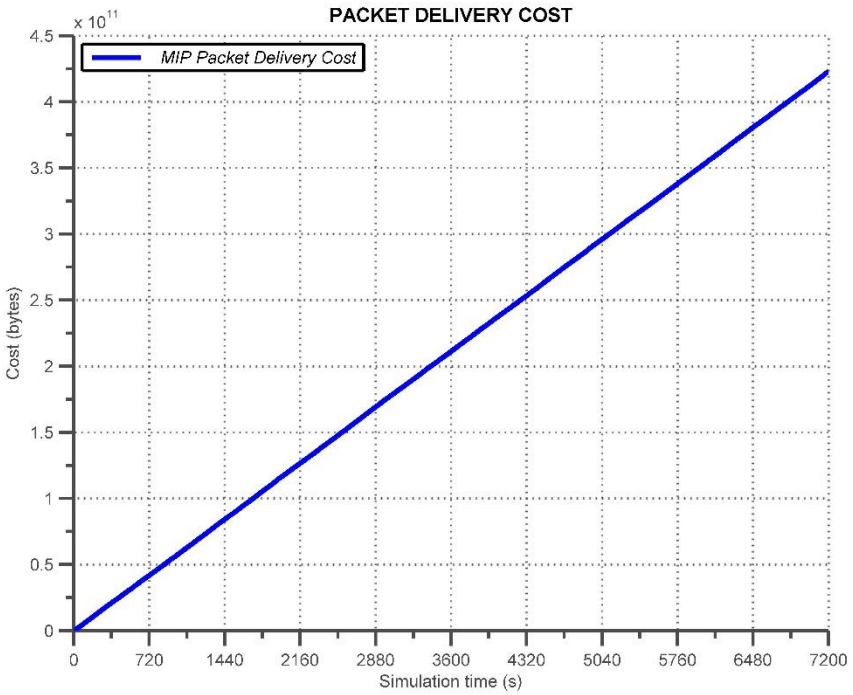


Figura 68: Gráfica del coste de entrega de paquetes para MIPv6

Y por cada nodo:

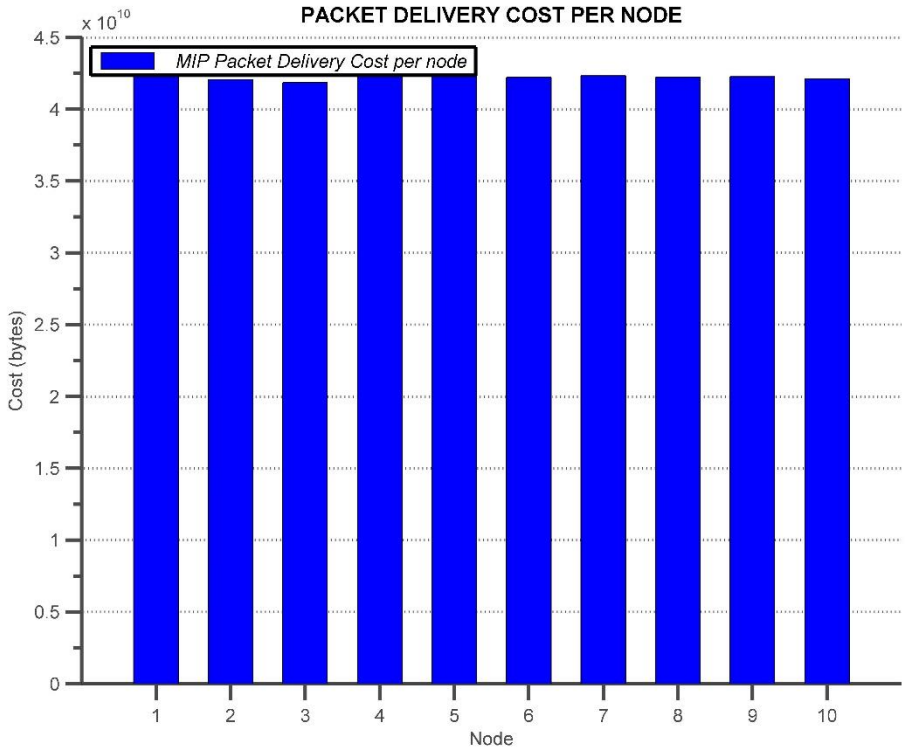


Figura 69: Gráfica del coste de entrega de paquetes por nodo para MIPv6

Del mismo modo, mostramos obtenemos la gráfica del coste *tunnelling*, **Tunnelling Cost**, en función del tiempo:

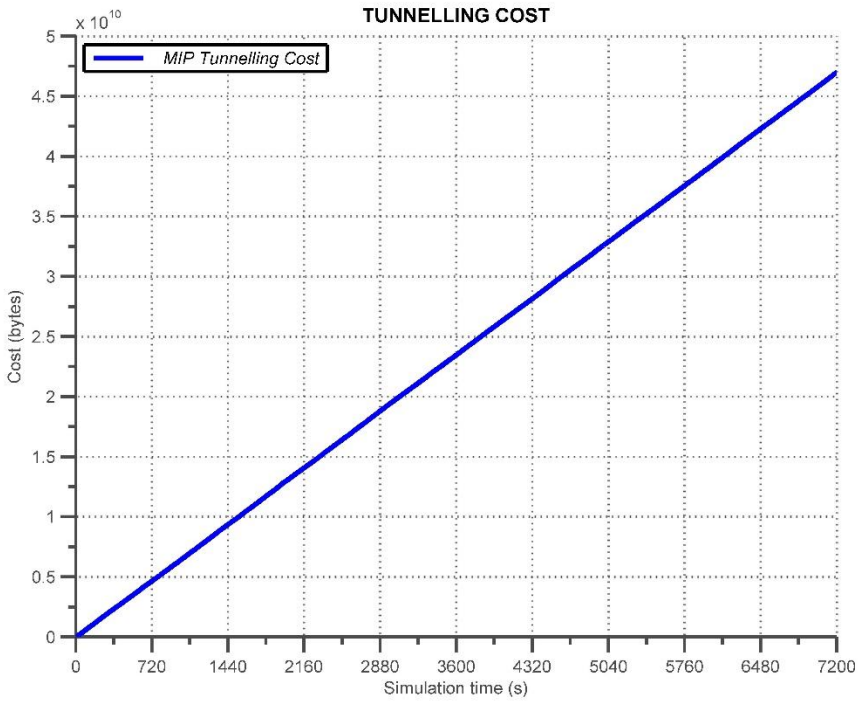


Figura 70: Gráfica del coste de tunnelling para MIPv6

Y por cada nodo:

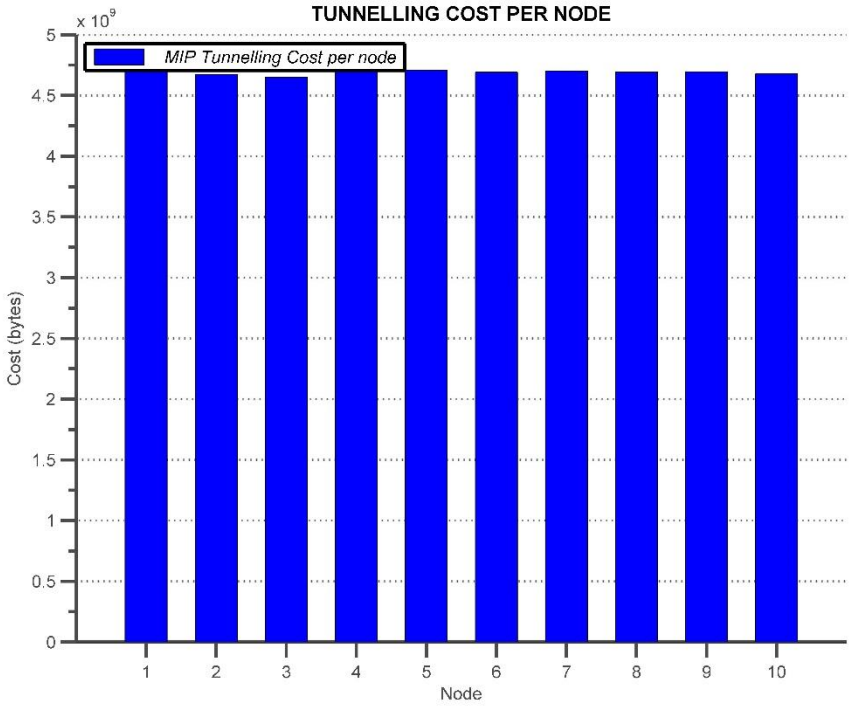


Figura 71: Gráfica del coste de tunnelling por nodo para MIPv6

Por definición, el coste de entrega de paquetes igual a la suma de los datos que se envían y el coste de *tunnelling*. Y es lo que se representa en la siguiente gráfica, la cual se puede comprobar que es idéntica a la *Figura 69:*

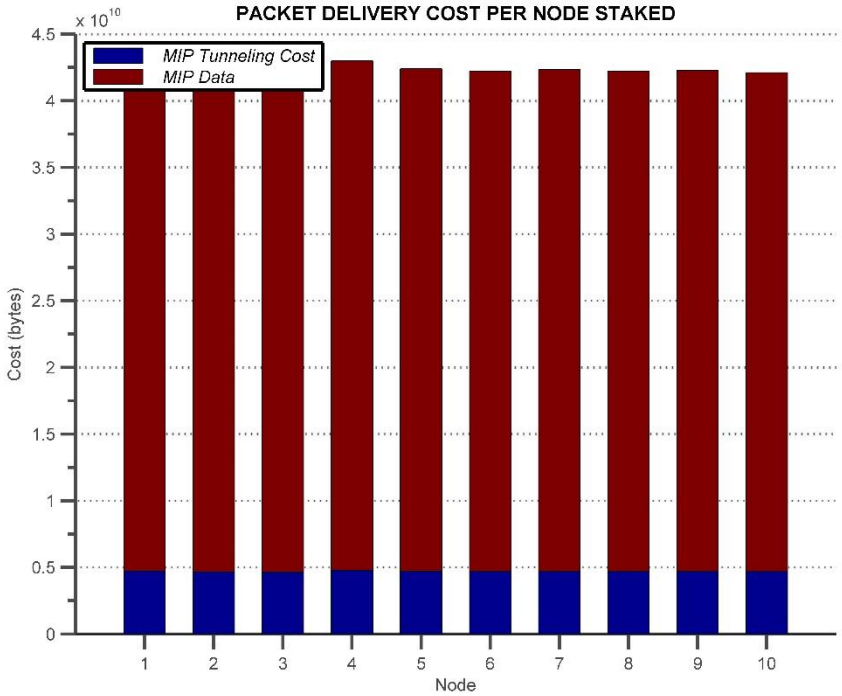


Figura 72: Gráfica apilada del coste de entrega de paquetes por nodo para MIPv6

También se muestra la gráfica de pérdida de paquetes, **Packet Loss**, en función del tiempo:

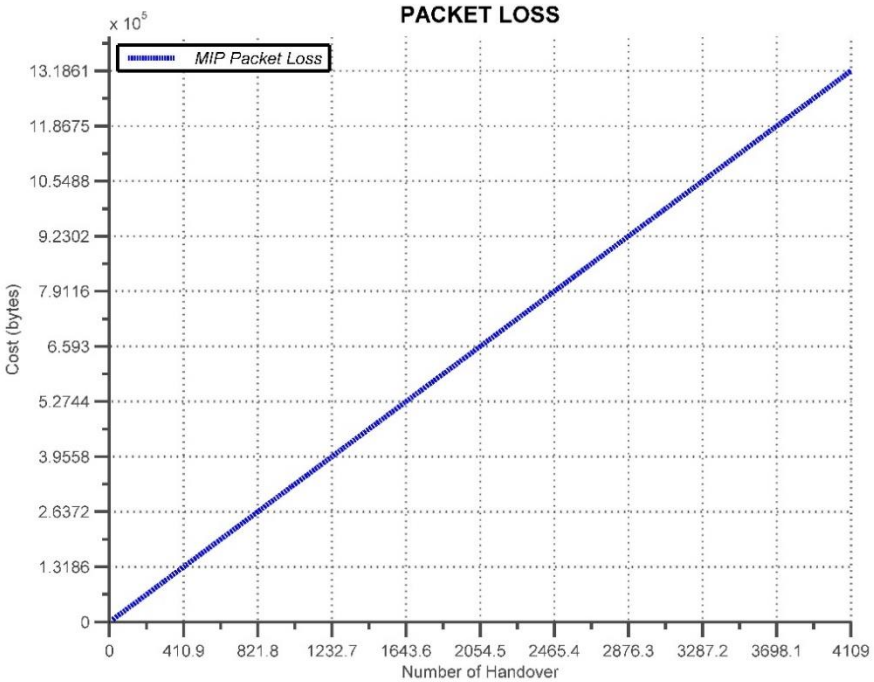


Figura 73: Gráfica de la pérdida de paquetes para MIPv6

Y la última gráfica que se nos muestra es la de la carga de la red, **Link Load**, en función del tiempo:

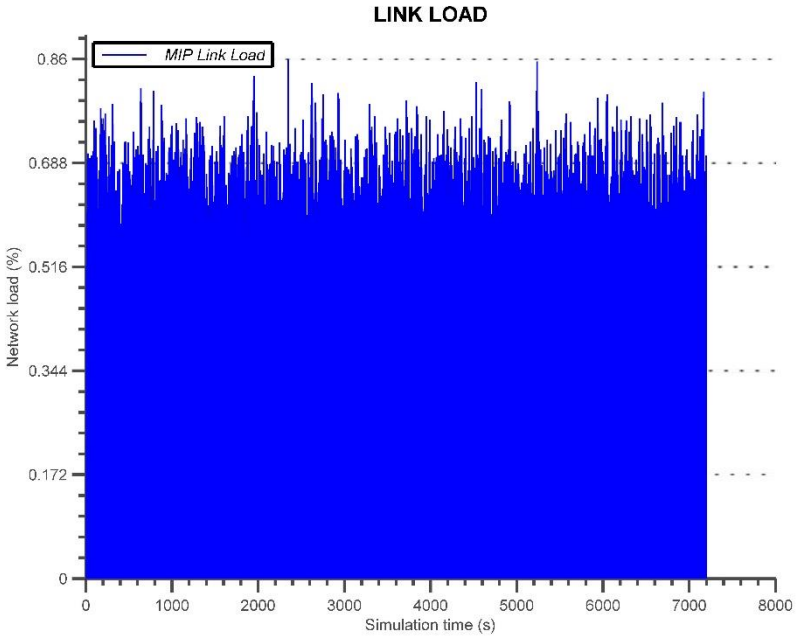


Figura 74: Gráfica de carga de la red para MIPv6

4.2.2. PMIPv6

Al igual que con MIPv6, empezamos representando el coste total por cada nodo, **Total Cost**:

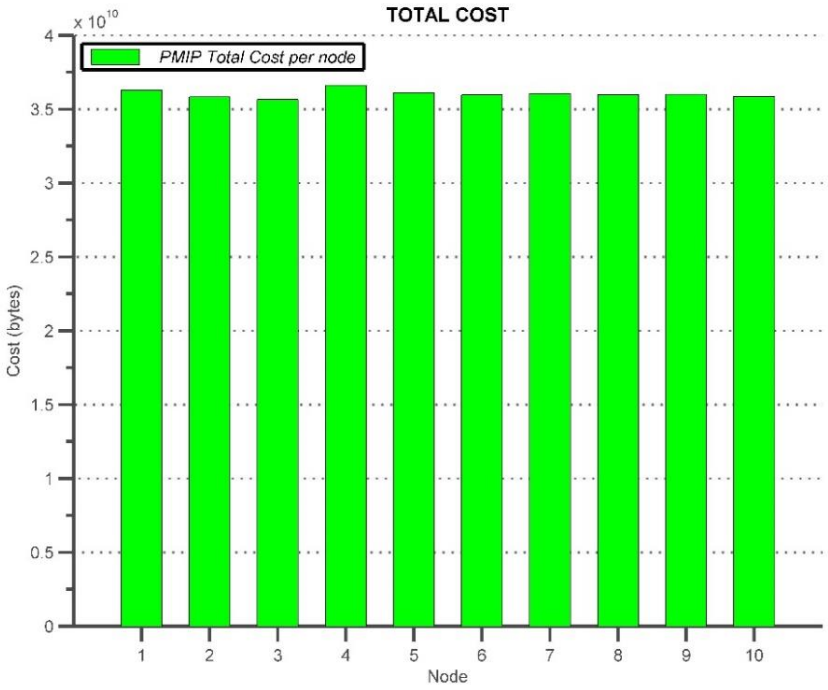


Figura 75: Gráfica de Total Cost para PMIPv6

Gráfica del coste de señalización en función del tiempo:

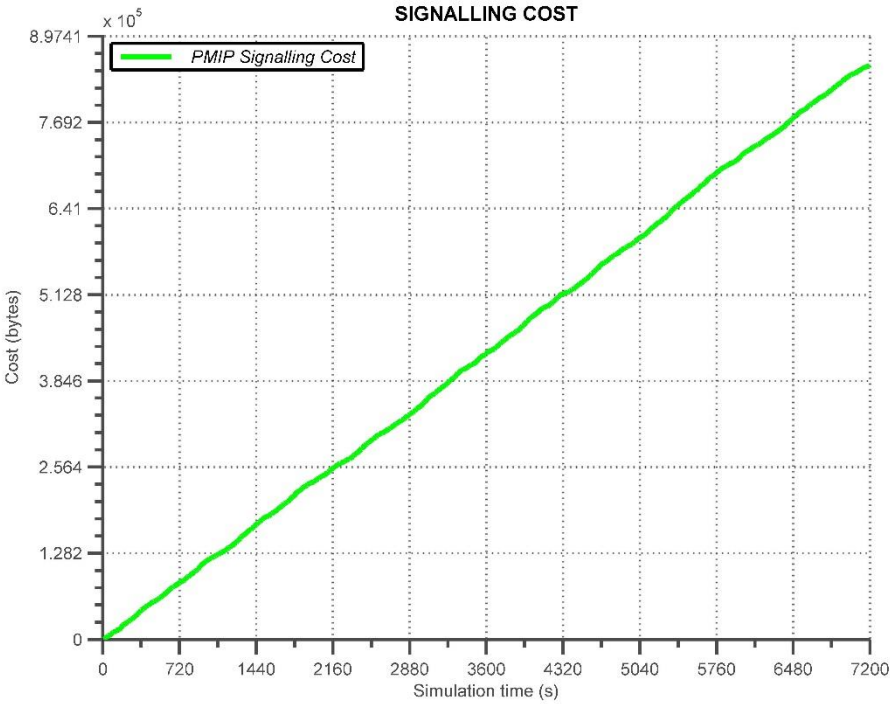


Figura 76: Gráfica del coste de señalización para PMIPv6 en función del tiempo

Y gráfica del coste de señalización por nodo:

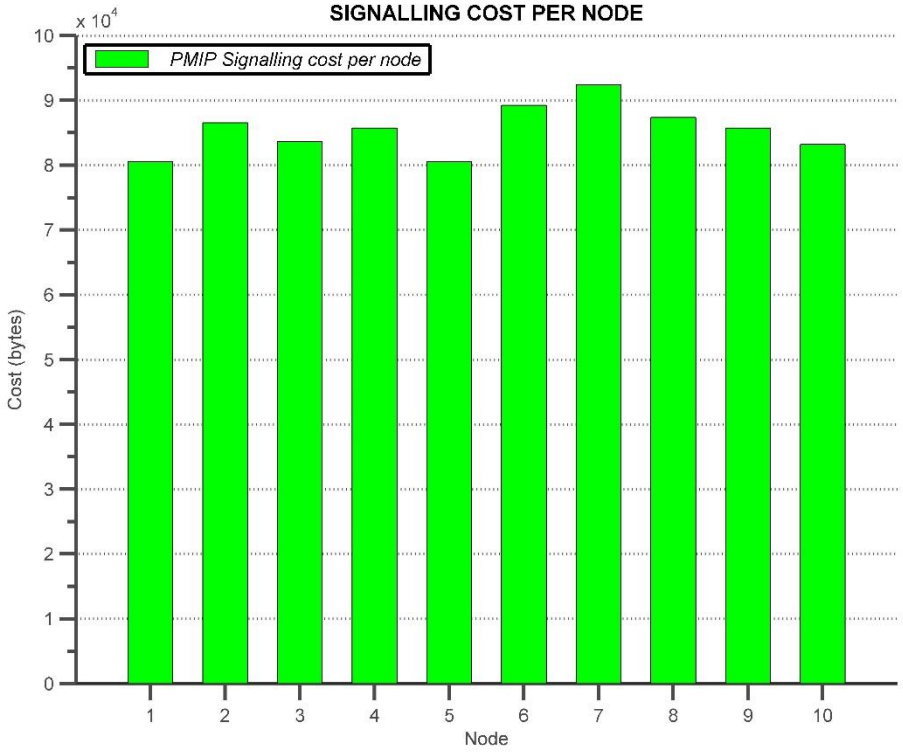


Figura 77: Gráfica del coste de señalización por nodo para PMIPv6

Gráfica del coste de entrega de paquetes, **Packet Delivery Cost**, en función del tiempo:

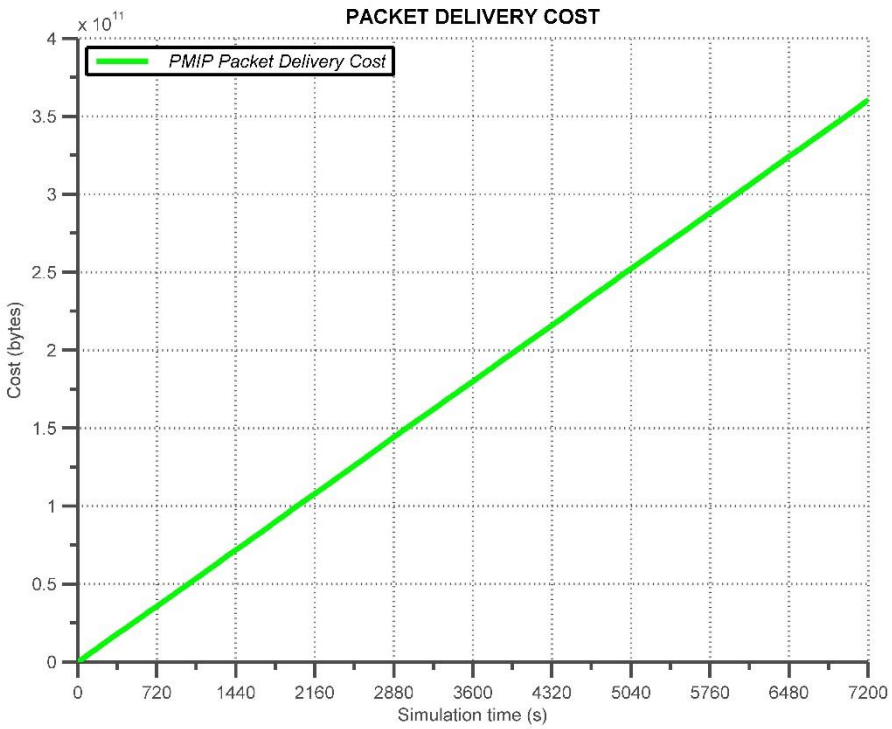


Figura 78: Gráfica del coste de entrega de paquetes para PMIPv6

Y por cada nodo:

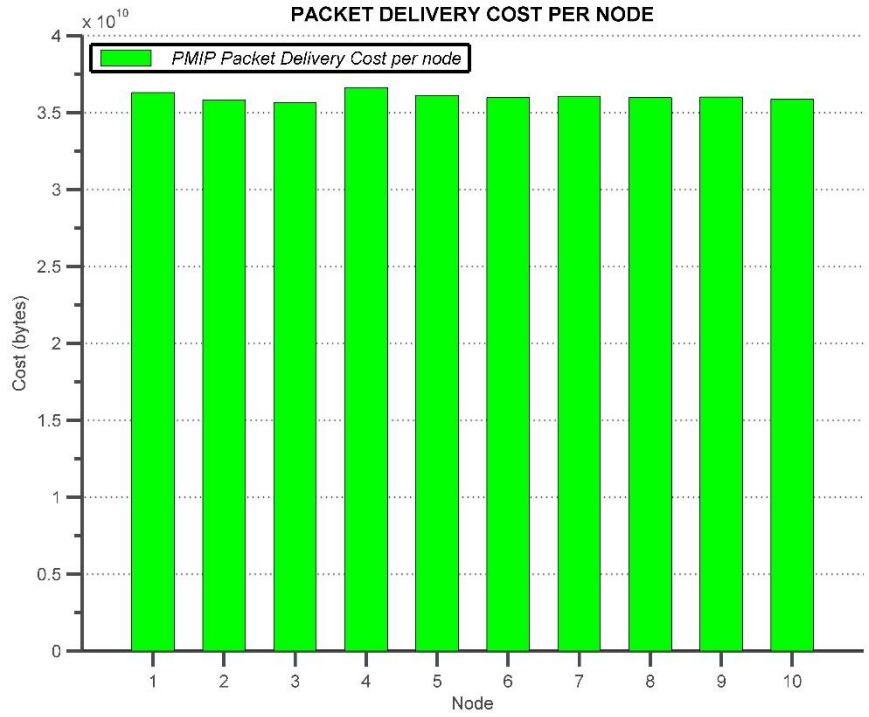


Figura 79: Gráfica del coste de entrega de paquetes por nodo para PMIPv6

Gráfica del coste *tunnelling*, **Tunnelling Cost**, en función del tiempo:

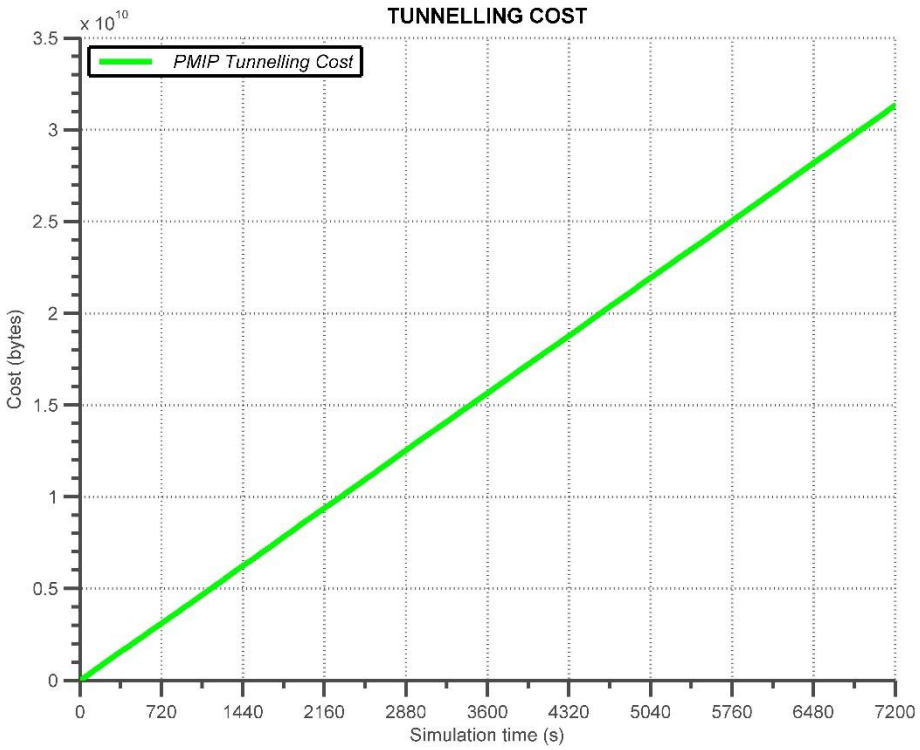


Figura 80: Gráfica del coste de tunnelling para PMIPv6

Y por cada nodo:

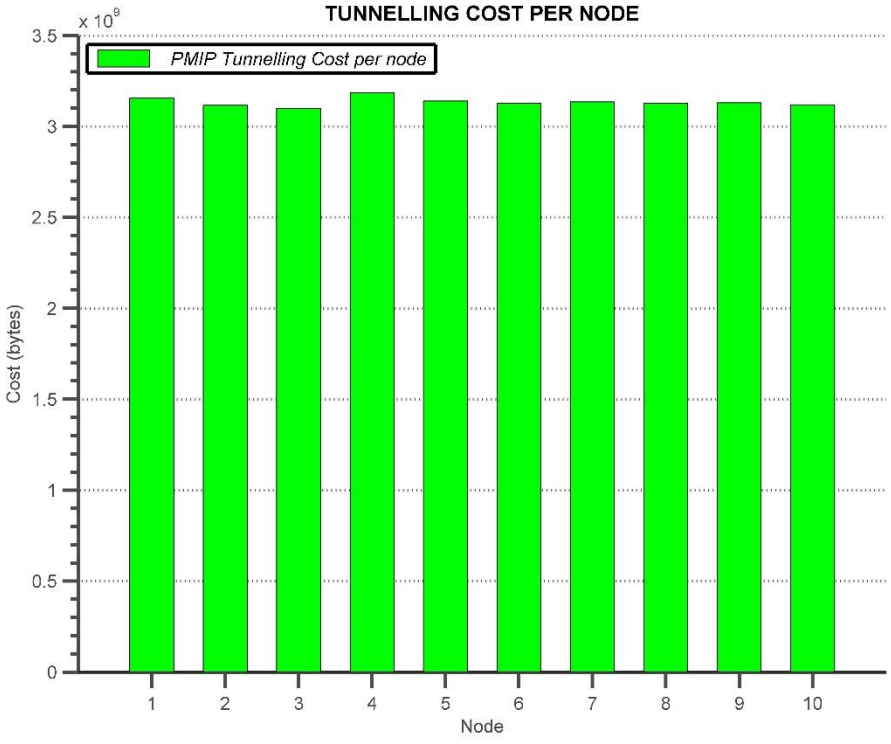


Figura 81: Gráfica del coste de tunnelling por nodo para PMIPv6

Al igual que hicimos con el protocolo MIPv6, se puede comprobar que la siguiente gráfica sumando el coste de *Tunnelling* y datos enviados es idéntica a la *Figura 79:*

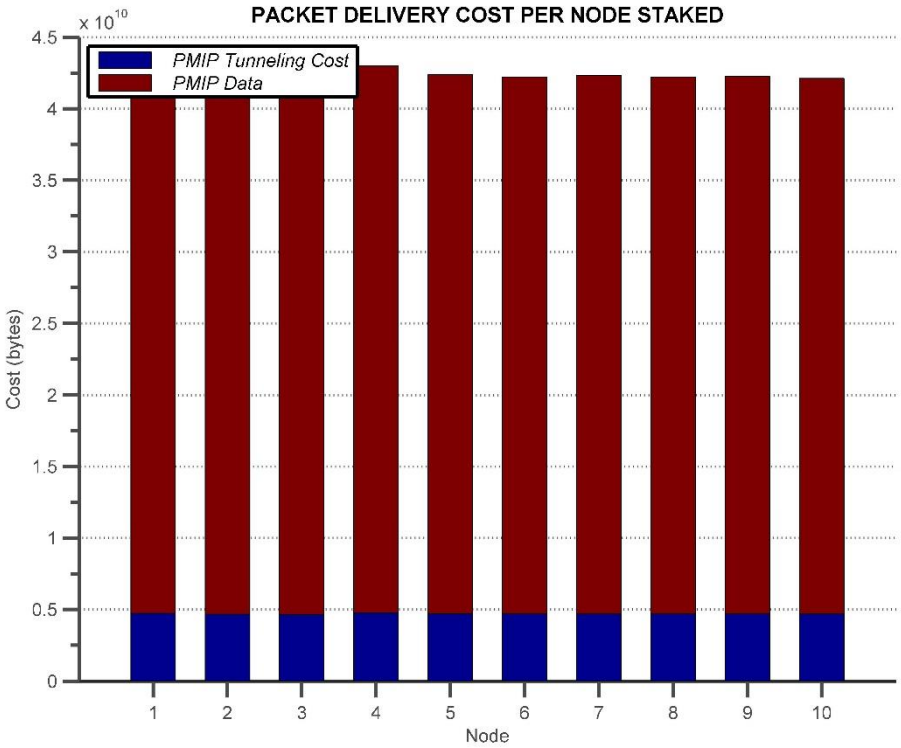


Figura 82: Gráfica apilada del coste de entrega de paquetes por nodo para PMIPv6

También se muestra la gráfica de pérdida de paquetes, **Packet Loss**, en función del tiempo:

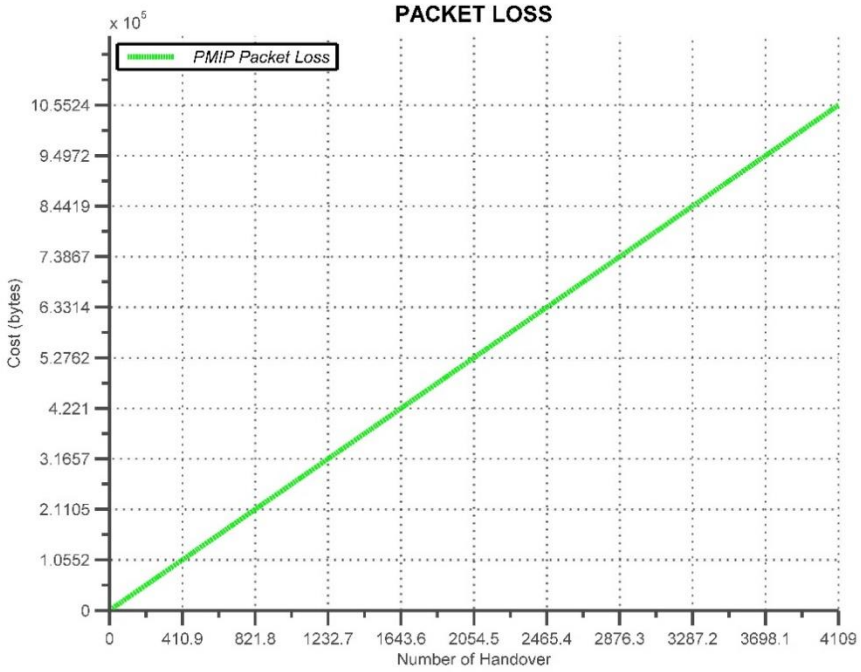


Figura 83: Gráfica de la pérdida de paquetes para PMIPv6

Y la última gráfica que se nos muestra es la de la carga de la red, **Link Load**, en función del tiempo:

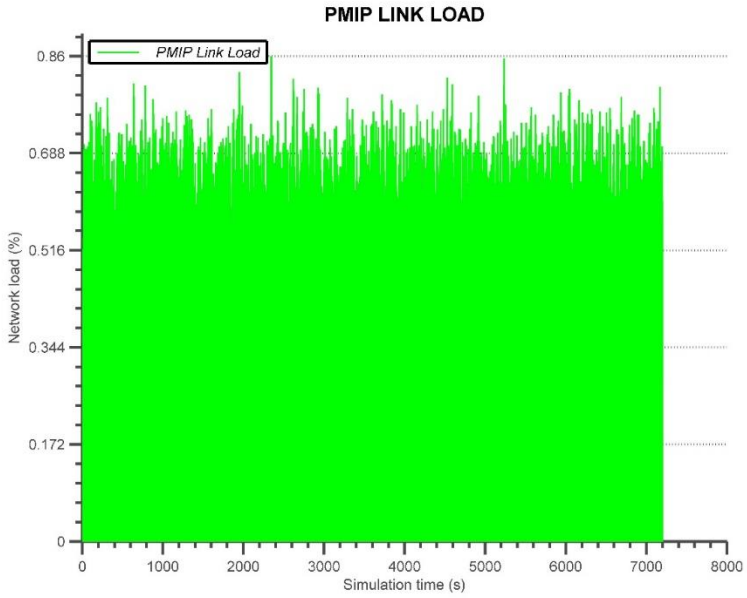


Figura 84: Gráfica de carga de la red para PMIPv6

4.2.3. Comparativa MIPv6-PMIPv6

En este apartado, vamos a comparar los gráficas de MIPv6 y PMIPv6. Empezamos con el coste total, **Total Cost**, donde el coste total por nodo en PMIPv6 es menor que utilizando MIPv6:

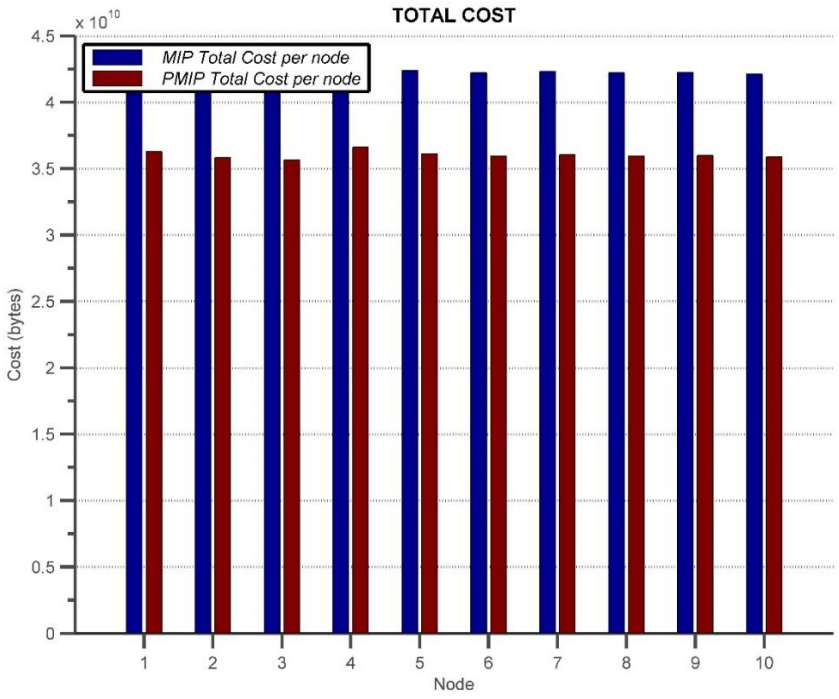


Figura 85: Comparativa MIPv6-PMIPv6 del coste total

Seguimos con el coste de señalización, **Signalling Cost**:

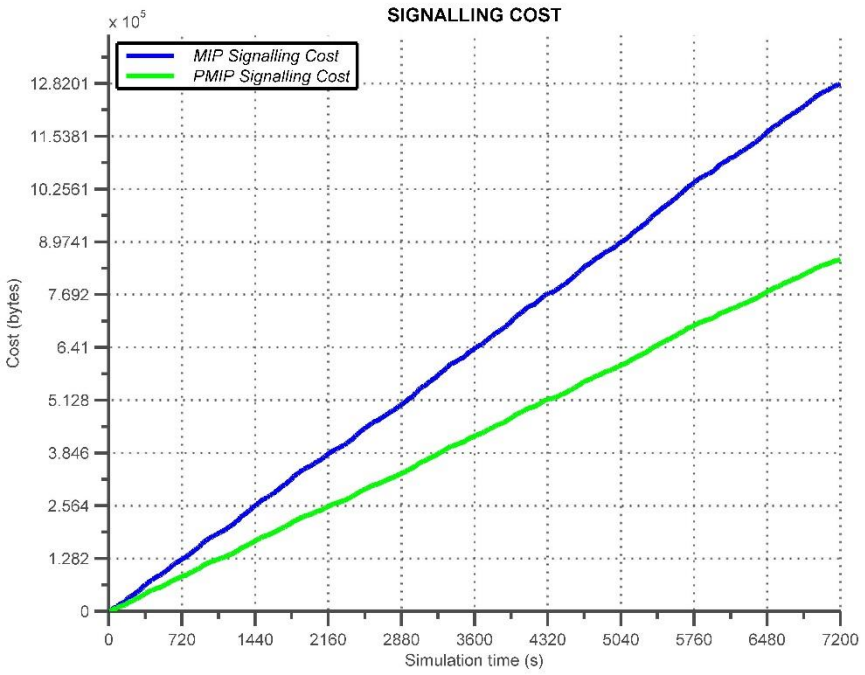


Figura 86: Comparativa MIPv6-PMIPv6 del coste de señalización en función del tiempo

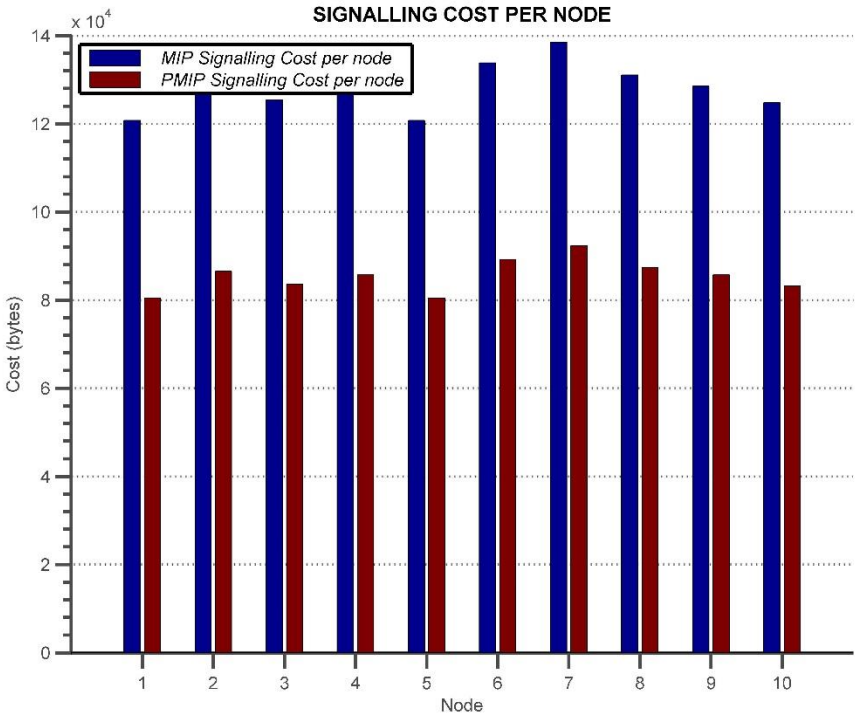


Figura 87: Comparativa MIPv6-PMIPv6 del coste de señalización por nodo

Observando las gráficas, se puede ver que PMIPv6 genera menos coste de señalización que MIPv6.

Seguimos con el coste de entrega de paquetes, **Delivery Cost**:

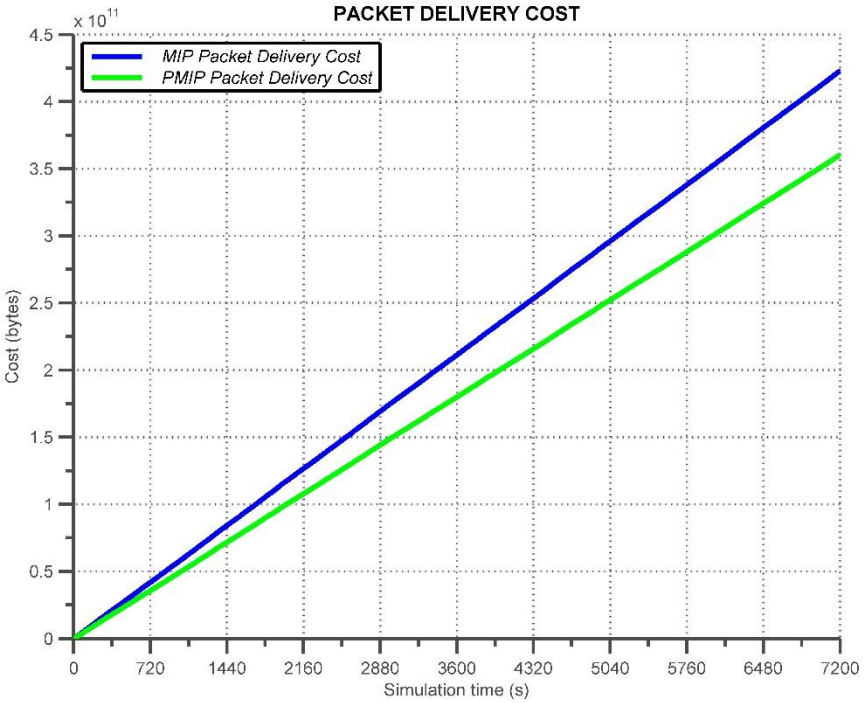


Figura 88: Comparativa MIPv6-PMIPv6 del coste de entrega de paquetes en función del tiempo

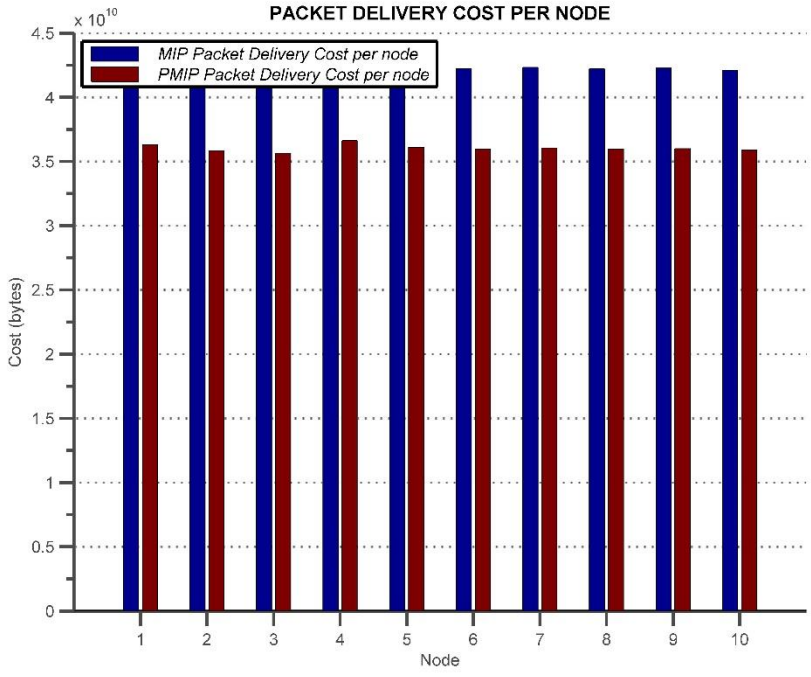


Figura 89: Comparativa MIPv6-PMIPv6 del coste de entrega de paquetes por nodo

Nuevamente, se comprueba que con PMIPv6 hay menos coste de entrega de paquetes.

Continuamos con el coste de *tunnelling*, **Tunnelling Cost**:

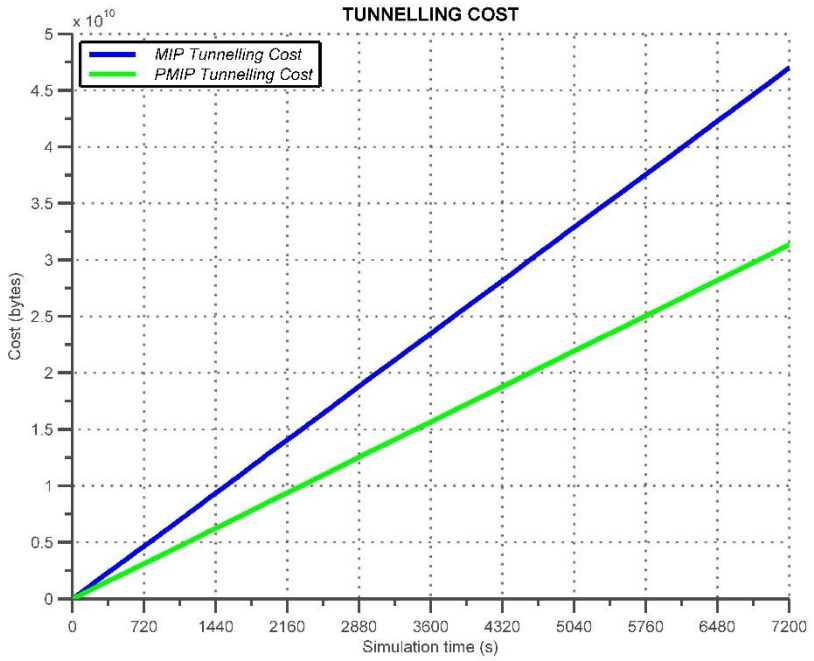


Figura 90: Comparativa MIPv6-PMIPv6 del coste de tunnelling en función del tiempo

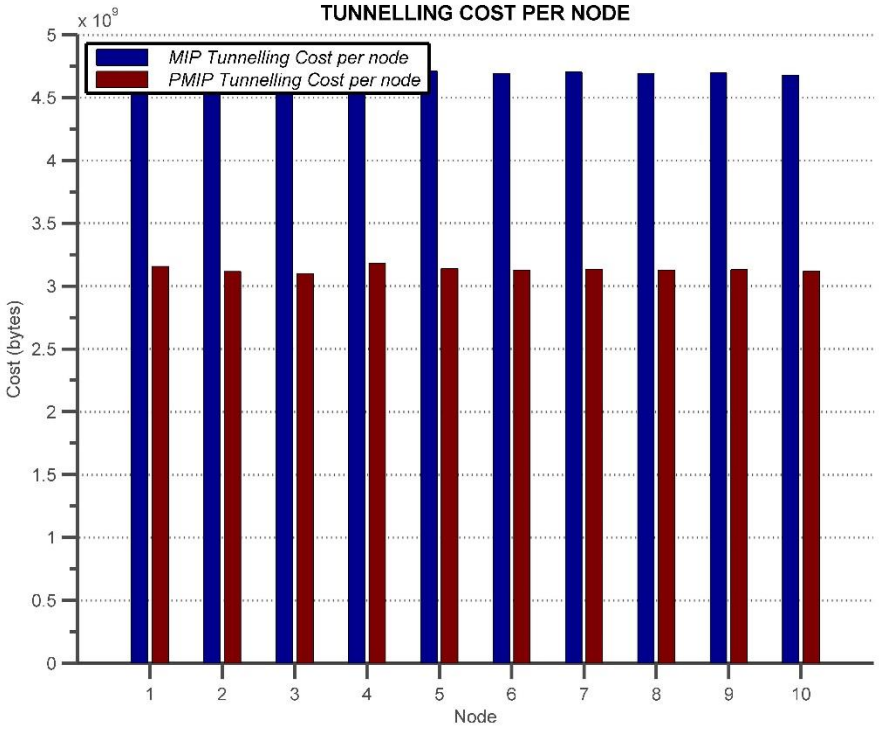


Figura 91: Comparativa MIPv6-PMIPv6 del coste de tunneling por nodo

Y en coste de tunnelling vuelve a ganar PMIPv6, con bastante menos coste que el que se produciría con MIPv6.

Respecto a la pérdida de paquetes, **Packet Loss**, se puede comprobar que se pierden menos paquetes utilizando PMIPv6 como protocolo:

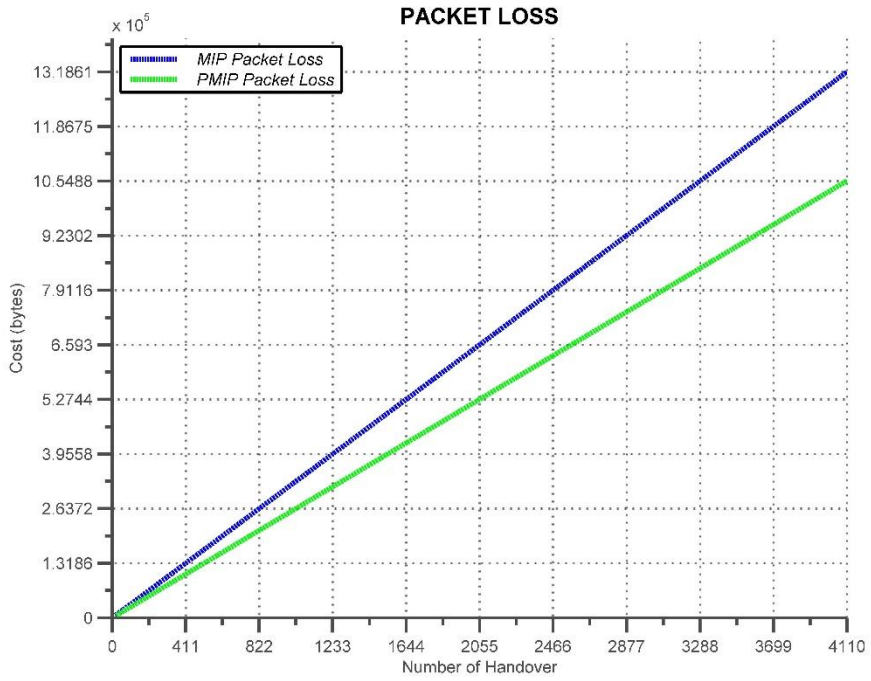


Figura 92: Comparativa MIPv6-PMIPv6 de la pérdida de paquetes

Por último, respecto a la carga de la red, **Link Load**, ésta es la misma y, si varía, la diferencia es mínima:

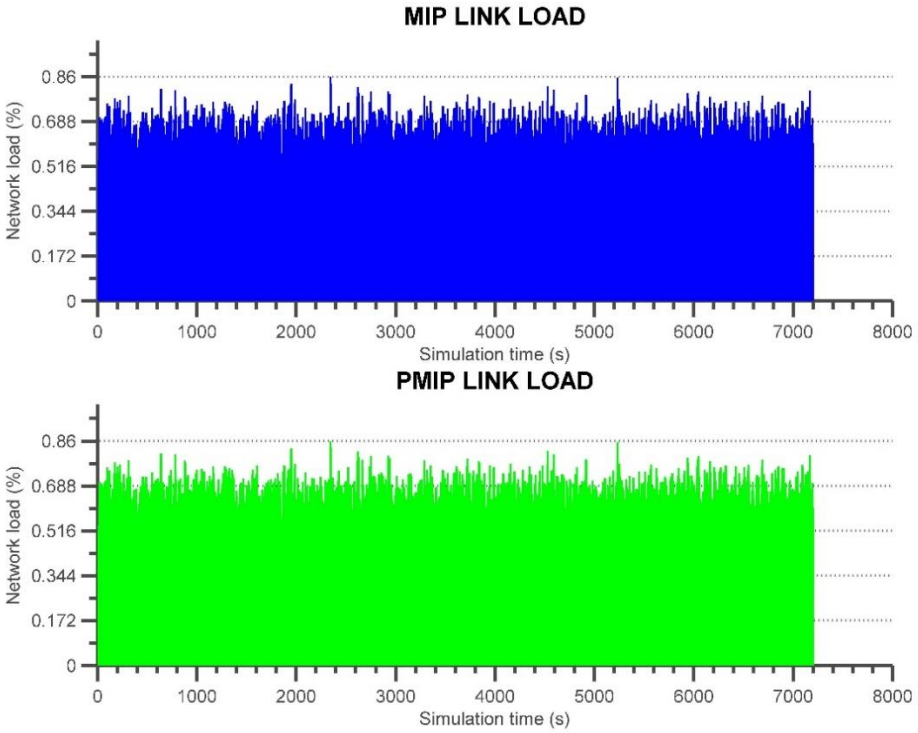


Figura 93: Comparativa MIPv6-PMIPv6 de la carga de la red

5. CONCLUSIONES Y TRABAJO FUTURO

5.1. Conclusiones

Las conclusiones que se han sacado de la realización de este Trabajo Fin de Grado se han ido estableciendo durante el desarrollo del mismo. A continuación, se muestra un resumen de ellas:

- Se han conocido en profundidad los protocolos MIPv6 y PMIPv6 para redes de próxima generación, así como los diferentes modelos de movilidad y tráfico.
- Se ha desarrollado un simulador de eventos discretos que permite analizar el comportamiento de MIPv6 y PMIPv6. Este simulador se ha diseñado para ser modular y fácilmente ampliable con nuevos protocolos, topologías, modelos de movimiento o modelos de tráfico que se quieran desarrollar.
- Según los resultados obtenidos, PMIPv6 muestra un mejor comportamiento que MIPv6. Esto es debido al propio funcionamiento de PMIPv6, en el que se gestiona la movilidad desde el punto de vista de la red, permitiendo que el nodo móvil no participe en ninguna tarea relacionada con la gestión de la movilidad.
- Tras la finalización del Trabajo, queda patente que la simulación es una herramienta muy útil para ciertos temas, como el de este TFG.

5.2. Trabajo futuro

Por último, se establecen unas líneas de trabajo futuro. Este simulador está en principio pensado para el ámbito educativo y de investigación, por tanto surgen multitud de nuevas vías de investigación y desarrollo a través de la línea de trabajo seguida en este Trabajo Fin de Grado. Además, como cada parte del simulador está independizada, se abren muchas posibles opciones. A continuación, algunas posibles líneas de trabajo futuro:

- Implementar otros tipos de modelos de movimiento (como “Real Mobility” o “Fluid Flow”) y de tráfico.
- Hacer que sea posible ejecutar la simulación un número *x* de veces pero no siempre igual, sino pudiendo cambiar parámetros para cada ejecución.

- Realizar nuevas gráficas y estudios estadísticos sobre las gráficas ya existentes.
- Desarrollar distintos algoritmos de *clustering* de las celdas de cobertura siguiendo otros modelos existentes en las redes de próxima generación.
- Desarrollar nuevos protocolos de gestión de la movilidad, como los nuevos mecanismos distribuidos (DMM, "*Distributed Mobility Management*") que están siendo estandarizados en el IETF en la actualidad.

6. BIBLIOGRAFÍA

- G. Perkins. D. Johnson and J. Arkko. "Mobility Support in IPv6". RFC 6275. July 2011.

- S. Gundavelli, K. Leung. V, Devaraplli, K. Chowdhury and B. Patil. "Proxy Mobile IPv6". RFC 5213. August 2008.

- T. Narten, E. Nordmark, W. Simpson, H. Soliman. "Neighbour Discovery for IPv6". RFC 4861. September 2007.

- Jie Li, Hsiao-Hwa Chen. "Mobility Support for IP-Based Networks". IEEE Communications Magazine. October 2005.

- Deguang Lee, Xiaoming Fu, Dieter Hogrefe. "A Review of Mobility Support Paradigms for the Internet". IEEE Communications Surveys & Tutorials. 1st Quarter 2006.

- J. Carmona Murillo, J. L: González Sánchez, A. Gazo Cervero, F. J. Rodríguez Pérez, L. Martínez Bravo, R. Martín Espada, J. A. Zarandieta Morán. "Análisis y evaluación de la potencialidad del protocolo Mobile IP". Documento disponible en: http://gitaca.es/agila2/pdfs/MobileIP.pdf.

- D. Cortés Polo, J. L. González Sánchez. "Análisis y optimización del *handover* en redes Mobile IP". Octubre 2008. Documento dispoible en: http://gitaca.unex.es/uploads/ES/CISCO-Mobile-IP.pdf.

- Javier Díaz, Mauricio Deamsi, Matías Robles, Germán Vodopivec. "Movilidad en IPv6". Facultad de Informática. Universidad Nacional de la Plata. Documento disponible en: http://sedici.unlp.edu.ar/bitstream/handle/10915/20873/Documento_completo.pdf?sequence=1.

- Josep Mangues Bafalluy, Albert Cabellos Aparicio, René Serral Gracia, Jordi Domingo Pascual, Antonio Gómez Skarmeta, Tomás P. de Miguel, Marcelo Bagnullo, Alberto García Martínez. "Movilidad IP. Macromovilidad, micromovilidad, calidad de servicio y seguridad". CTTC, UPC, UMU, UMP, UC3M.

- Ki-Sik Kong, Wonjun Lee, Youn-Hee Han, Myung-Ki Shin, HeungRyeol You. "Mobility Management For All-IP Mobile Networks: Mobile IPv6 VS. Proxy Mobile IPv6". IEEE Wireless Communications. April 2008.

- Sara Berzosa Calpe, Rafael Vidal Ferrer. "Redes Celulares 4G basadas en Mobile IPv6". Universidad Politécnica de Cataluña. Junio 2004. Documento disponible en: http://www.raco.cat/index.php/Buran/article/view/178913.

- Marco Di Renzo. "On System-Level Analysis and Design of 5G Mobile/Wireless Networks: The Magic of Stochastic Geometry". King's College London. Centre for Telecommunications Research. London, UK. February 2015.

- Klaus Wehrle, Mesut Günes, James Gross. "Modeling and Tools for Network Simulation". ISBN: 978-3-642-12330-6. Springer-Verlag Berlin Heidelberg. 2010.

- Radhika Ranjan Roy. "Handbook of mobile ad hoc networks". ISBN: 978-1-4419-6048-1. Springer Science+Business Media. 2011.

- Jesús Manuel Calle Cancho. "Proyecto Fin de Carrera. Estudio y análisis para el despliegue de la movilidad en IPv6". Universidad de Extremadura, Escuela Politécnica – Ingeniería Informática. Julio 2013.

- Voronoi en Matlab

  Disponible en: http://es.mathworks.com/help/matlab/ref/voronoi.html

- Localizar puntos dentro de un polígono

  Disponible en: http://es.mathworks.com/help/matlab/ref/inpolygon.html

- Sitefinder

  Disponible en: http://www.sitefinder.ofcom.org.uk/search

- Advanced topics in Communication Systems

  Disponible en:
  http://www.iitg.ernet.in/engfac/krs/public_html/lectures/ee635/A3.pdf

- Matlab Central

  Disponible en: http://es.mathworks.com/matlabcentral/

- Trello

  Disponible en: https://trello.com/

- Gantt Project

  Disponible en: http://www.ganttproject.biz/

Todos los enlaces web de la bibliografía han sido visitados por última vez el 26 de Julio de 2015.