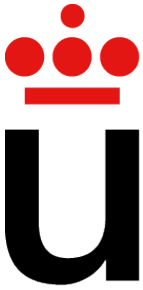


Universidad
Rey Juan Carlos

GRADO EN INGENIERÍA TELEMÁTICA

Curso Académico 2022/2023

Trabajo Fin de Grado

ESTUDIO DE PROTOCOLOS DE ENCAMINAMIENTO DINÁMICO EN MANETS CON ESCENARIOS DE MOVILIDAD EN MININET WIFI

Autor : Justo Martín Collado

Tutor : José Centeno González

Dedicado a

mis padres, que me lo han dado todo,

y a mi familia y amigos por apoyarme siempre.

Agradecimientos

Gracias a mi tutor José Centeno por su guía indispensable en la realización de este trabajo. Gracias también a mi amiga Paula Gallego por ayudarme en la revisión de éste.

Resumen

Este Trabajo de Fin de Grado muestra el uso de escenarios de movilidad en Mininet WiFi para el estudio de distintos protocolos de encaminamiento dinámico en Redes móviles Ad Hoc (MANETs).

Se han utilizado distintas tecnologías para llevar el trabajo a cabo: el emulador de redes inalámbricas Mininet WiFi, cuatro protocolos de enrutamiento dinámico distintos (B.A.T.M.A.N., B.A.T.M.A.N. Advanced, OLSR y OLSRv2), y para el análisis estadístico de los datos se han creado scripts en Python usando la librería Pandas.

Este estudio ha tenido como principal objetivo el comprobar como afecta la movilidad de los nodos en MANET al tiempo que tardan estos protocolos en recalcular sus tablas de encaminamiento y en estudiar cómo se comporta la emulación de un escenario con movilidad controlada en Mininet WiFi.

Para este estudio se han diseñado y programado escenarios de movilidad determinista y aleatoria. También se han diseñado y creado scripts para la obtención de estadísticas y para la generación de gráficas que faciliten y automaticen el análisis de los datos obtenidos en la realización de las pruebas.

Se han realizado cinco pruebas para cada uno de los cuatro escenarios distintos de movilidad, y se ha analizado qué protocolo tiene un mayor tiempo de desconexión total, media, mínima y máxima, y la cantidad de paquetes que generan cada uno a fin de compararlos.

Con las gráficas generadas se puede analizar para cada protocolo la velocidad a la hora de recalcular las tablas de encaminamiento, y se puede estudiar la estabilidad de cada uno de ellos en los distintos escenarios observando cómo estos parámetros cambian con la distinta movilidad de los nodos.

Se proponen como posibles trabajos futuros a desarrollar a partir de este TFG la generación o eliminación en tiempo real de nodos a la red cuando ya se ha iniciado el escenario en Mininet

WiFi o añadir soporte en Mininet WiFi para otros protocolos de encaminamiento dinámico en MANETs como AODV o ZRP.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este Trabajo de Fin de Grado se ha realizado un estudio de los protocolos de encaminamiento dinámico en redes MANETs. Las redes MANETs (Mobile Ad Hoc Networks) son redes en las que los nodos son a la vez clientes y encaminadores del tráfico. En ellas nodos son los encargados de enrutar el tráfico para que sea posible el envío de información entre dos o más nodos. Además, estas redes pueden cambiar fácilmente su topología mediante el movimiento de los nodos que la componen, haciendo que los protocolos de enrutamiento dinámico habituales como OSPF o IS-IS no sean capaces de mantener unas tablas de encaminamiento coherentes debido a la velocidad en la que los cambios han de hacerse o por la cantidad de tráfico que generan y que la red no es capaz de procesar.

Para la realización de este trabajo se ha empleado la herramienta Mininet WiFi. Mininet WiFi es un emulador de Redes Definidas por Software. Este emulador permite virtualizar nodos con sus interfaces WiFi de manera que mediante scripts en Python se pueda crear una topología de red en las que los nodos sean estáticos o que puedan tener movilidad.

En este trabajo se ha analizado cómo el movimiento de los nodos en Mininet WiFi afecta a la velocidad de actualización de las tablas de encaminamiento y a la cantidad de paquetes generados por los protocolos B.A.T.M.A.N., B.A.T.M.A.N. Advanced, OLSR y OLSRv2 mediante distintos escenarios de movilidad.

En los sucesivos capítulos explicaremos como se ha desarrollado este trabajo:

- En el capítulo 2, Tecnologías Relacionadas, se podrá entender cada una de las tecnologías y protocolos usados para la realización del trabajo.

- En el capítulo 3, Objetivos, se indican los objetivos establecidos al comenzar el trabajo.
- En el capítulo 4, Implementación, se describe el código de los escenarios y de los scripts para el análisis de datos.
- En el capítulo 5, Pruebas, se explica la metodología de las pruebas así como el análisis de los experimentos.
- En el capítulo 6, Conclusiones, se pueden encontrar las conclusiones de este trabajo así como posibles líneas de trabajo futuro.
- En los apéndices A y B se aporta el código desarrollado. Así mismo todo el código está también disponible online en `https://github.com/Klego/TFG-MANETS`.

Capítulo 2

Tecnologías Relacionadas

En este capítulo se explicarán las tecnologías usadas en el proyecto así como una descripción de las mismas.

2.1. Redes móviles Ad Hoc: MANETs

Las redes ad hoc son redes descentralizadas formadas entre dispositivos vecinos con capacidad de comunicarse entre sí, sin depender de ninguna infraestructura externa. Los nodos actúan como encaminadores para permitir la comunicación con otros nodos que están fuera de su rango de radio. Dentro de este tipo de redes hay un conjunto de redes llamadas Redes Móviles Ad Hoc (*Mobile Ad Hoc Networks: MANETs*), que son redes ad hoc donde los nodos tienen capacidad de movilidad haciendo más difícil el envío y recepción de datos entre los dispositivos que las componen.

Este tipo de redes son fácilmente desplegables en poco tiempo ya que no es necesario el uso de infraestructura adicional externa para la comunicación de los nodos, además de que son flexibles y robustas debido a que con la movilidad de éstos, los paquetes se pueden enrutar por distintos caminos si un nodo de la red no está disponible.

En cambio este tipo de redes también tienen ciertos problemas asociados, como la posible limitación de acceso a la energía por parte de los nodos, la poca capacidad de cómputo y la heterogeneidad de los mismos. Además debido a la movilidad las rutas de encaminamiento pueden cambiar produciendo más pérdidas de paquetes y errores en la transmisión, los nodos pueden dejar la red porque el rango de la antena no alcance al resto de la red debido a la

movilidad haciendo que la red se pueda fraccionar. Todos los nodos en este tipo de redes tienen que colaborar entre sí y son autónomos, es decir, no hay ningún sistema que controle en todo momento como se comportan éstos.

Hay distintos tipos de MANETs dependiendo el uso para el que han sido diseñadas:

- Redes Inalámbricas de Sensores (WSN), son redes donde los nodos son sensores que monitorizan condiciones físicas o ambientales.
- Redes Híbridas formadas por infraestructura de red convencional y redes ad hoc.
- Redes Ad Hoc Vehiculares (VANETs), usadas en la comunicación de vehículos con los dispositivos de la vía y entre sí.

2.2. Emulador de redes inalámbricas: Mininet WiFi

Mininet WiFi [2] es una extensión al emulador de redes SDN llamado Mininet [3]. Mininet WiFi añade la funcionalidad de virtualizar redes inalámbricas mediante la creación de estaciones y puntos de acceso WiFi virtuales. Estas funcionalidades son proporcionadas por los *drivers* estándar y *80211_hwsim* para Linux, los cuales permiten generar escenarios que emulan atributos y características como la posición y el movimiento relativo de un nodo WiFi a su punto de acceso.

2.2.1. Arquitectura de Mininet WiFi

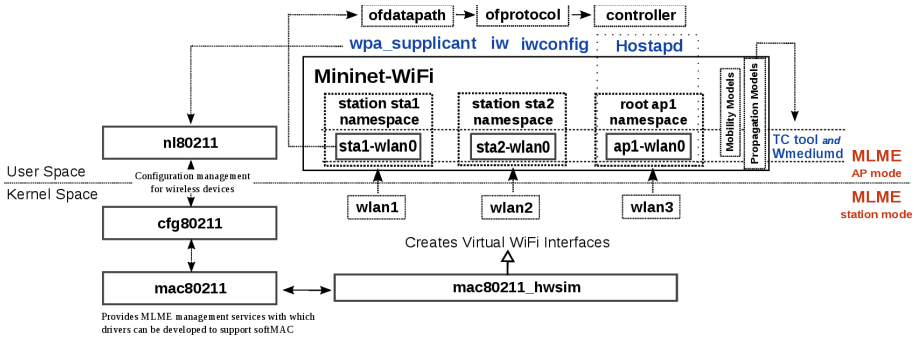


Figura 2.1: Arquitectura de Mininet WiFi

La arquitectura principal con sus componentes se puede apreciar en la figura 2.1 [4]. En esta arquitectura se puede ver el módulo *mac80211_hwsim* que es el responsable de la virtualización

de las interfaces WiFi. Mininet WiFi usa utilidades como *iwconfig* para configurar y obtener información de las interfaces.

Los dos tipos de componentes principales en Mininet WiFi son las estaciones y los puntos de acceso, los cuales están interconectados a otras estaciones o puntos de acceso mediante los espacios de nombres (*namespaces*) de Linux.

- **Estaciones**: Dispositivos que se pueden conectar a un punto de acceso o a otras estaciones. Cada estación posee una interfaz de red inalámbrica virtual (sta-wlan0).

- **Puntos de Acceso**: Equipos que gestionan estaciones asociadas y son virtualizadas a través de *hostapd* utilizando interfaces inalámbricas virtuales para poder asociarse a otros puntos de acceso.

2.2.2. Módulo de emulación del medio inalámbrico: Wmediumd

Un elemento de suma importancia en Mininet WiFi es cómo este realiza la simulación del medio inalámbrico mediante el uso de ***Wmediumd***. *Wmediumd* usa el módulo de kernel *mac80211_hwsim* para virtualizar el mismo medio para todos los nodos inalámbricos. Esto quiere decir que todos los nodos internamente están en rango para poder comunicarse y descubrir otras interfaces de red.

Mininet WiFi es el encargado de emular la posición y el rango la señal inalámbrica conectando o desconectando estaciones entre sí o con puntos de acceso, haciendo que se permita o deniegue la transferencia de paquetes de una interfaz a otra. Así mismo, si se quiere emular redes móviles ad hoc se requiere el uso de este módulo de control del medio inalámbrico.

2.3. Protocolos de encaminamiento dinámico para MANETs

Los protocolos de encaminamiento dinámico utilizados en redes móviles inalámbricas tienen como característica particular permitir que las tablas de encaminamiento cambien rápidamente ajustándose a los cambios frecuentes en la topología del escenario. En esta sección se explica cada uno de los protocolos estudiados en este trabajo a fin de mostrar las diferencias entre ellos.

Por tanto, para este tipo de redes existen protocolos de encaminamiento dinámico específicos que además se pueden dividir en varios tipos [5]:

- **Proactivos**: Los nodos que mantienen en todo momento en su tabla de encaminamiento las rutas a todos los nodos. OLSR, OLSRv2, B.A.T.M.A.N. y B.A.T.M.A.N. Advanced pertenecen a esta categoría.
- **Reactivos**: Los nodos sólo calculan la ruta al nodo destino cuando necesitan enviarle datos, por tanto no mantienen tablas de encaminamiento completas a todos los nodos.
- **Híbridos**: Se comportan como protocolos Proactivos cuando se envían datos a nodos cercanos a ellos, es decir, mantienen tablas de encaminamiento para los nodos cercanos y se comportan como un protocolo Reactivo cuando tienen que enviar mensajes a nodos más alejados de ellos en la red.
- **Geográficos**: Nodos que basan el encaminamiento en las coordenadas geográficas que los nodos retransmiten periódicamente.

2.3.1. Better Aproach To Mobile Ad-hoc Networking: B.A.T.M.A.N.

B.A.T.M.A.N. [6] es un protocolo de encaminamiento dinámico proactivo para redes móviles Ad-Hoc. La diferencia de B.A.T.M.A.N. con otros protocolos proactivos es que cada nodo no tiene toda la ruta hasta un determinado nodo destino, sino que unicamente almacena cuál es el mejor nodo adyacente para transmitir la información hacia él.

B.A.T.M.A.N. analiza datos como la pérdida y velocidad de propagación de los paquetes y no usa la información de la topología (como si hace OLSR). Así, B.A.T.M.A.N. elige como siguiente salto al nodo basándose en datos estadísticos calculados de manera local.

El protocolo detecta la presencia en la red de nodos con interfaces B.A.T.M.A.N. llamados **Originadores**. Estos envían mensajes llamados OGMs (*Originator Messages*) por el puerto 4305 mediante UDP. Estos OGM son los que usa B.A.T.M.A.N. para conocer la presencia de un Originador en la red.

En una red con B.A.T.M.A.N. cada nodo transmite periódicamente un OGM a sus nodos vecinos informando de su presencia en la red. Los nodos vecinos que reciben los OGM retransmiten esa información a otros nodos de los que antes hayan recibido un OGM haciendo que la

red esté inundada de estos mensajes.

Como en la práctica puede haber pérdidas de paquetes por interferencias, colisiones o congestión, el número de OGMs recibidos de un Originador vecino es el usado para estimar qué nodo debe ser la puerta de enlace hacia ese Originador. Con el fin de encontrar la mejor puerta de enlace a un nodo, B.A.T.M.A.N. cuenta el número de OGMs que ha recibido de un Originador por cada uno de sus vecinos y, por tanto, el vecino por el que se haya recibido más OGMs de ese nodo, es la mejor ruta y resulta escogido como puerta de enlace para encaminar los paquetes hacia ese Originador.

2.3.2. B.A.T.M.A.N. Advanced

B.A.T.M.A.N. Advanced [7] es una implementación de B.A.T.M.A.N. que opera en el **nivel dos del modelo OSI** en forma de módulo del kernel. Ha sido creado como un módulo del kernel para que las continuas lecturas y escrituras de las tablas de encaminamiento no tengan el impacto excesivo en la CPU que sí tienen los demonios en espacio de usuario.

B.A.T.M.A.N. Advanced opera de manera íntegra en el nivel dos del modelo OSI, así no solo la información de encaminamiento es transportada directamente dentro de tramas de Ethernet, sino que todo el tráfico es transportado usando tramas Ethernet.

Este protocolo encapsula toda la trama original en una nueva trama con tipo de protocolo ***Ethertype 0x0842*** y reenvía todo el tráfico hasta que llega al destino. Esto hace que los nodos parezcan que están unidos mediante un enlace local (todos los nodos creen que son vecinos entre sí, es decir, están a un salto) y no sean afectados por la topología o los cambios en la red. La manera más fácil de entender B.A.T.M.A.N. Advanced es pensar en un switch distribuido, donde cada nodo que corre B.A.T.M.A.N. Advanced es un puerto de ese switch.

En cada uno de los nodos que usan este protocolo se genera una nueva interfaz virtual llamada ***bat0***. Esta interfaz de red es para el sistema un puerto que permite acceder al switch distribuido y es la que permite la comunicación con el resto de nodos. Para que esta comunicación entre interfaces *bat0* sea posible, B.A.T.M.A.N. Advanced confía en las interfaces de Ethernet que se conectan a *bat0* mediante un *bridge* de Linux como describe la figura 2.2.

El encaminamiento de datos mediante B.A.T.M.A.N. Advanced es el siguiente:

1. La interfaz *bat0* envía la trama Ethernet (*Type: 0x0842*) con los datos encapsulados del

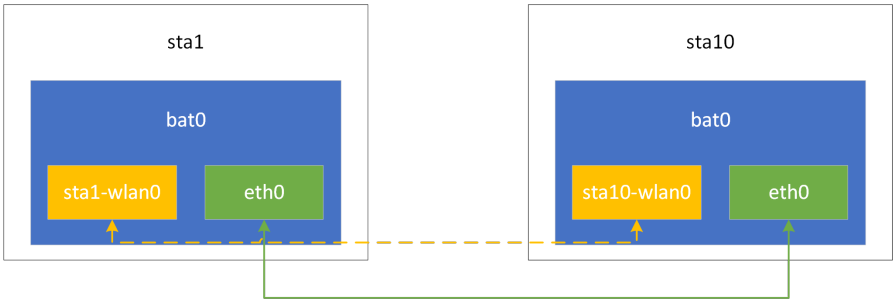


Figura 2.2: Interconexión de interfaces en B.A.T.M.A.N. Advanced

usuario.

2. En cada nodo que ejecute B.A.T.M.A.N. Advanced, existe una tabla de MAC (como en todo switch), la cual dice a qué nodo tiene que enviar la trama. Para elegir esta interfaz, el protocolo usa el mismo mecanismo que usa B.A.T.M.A.N. con los OGMs.

3. Seguidamente, cuando llega a un esta trama a un nodo de la red:

   - Si no es para él, mira su tabla de MAC y lo vuelve a reenviar mediante la interfaz *bat0*.
   - Si es para él, desencapsula la trama y opera con ella como normalmente haría.

Este diseño permite ciertas características que es importante tener en cuenta:

- Permite que encima de B.A.T.M.A.N. Advanced se use cualquier protocolo como IPv4, IPv6, DHCP, etc.
- No es necesario que los nodos tengan una IP inicialmente para poder comunicarse.

2.3.3. Optimized Link State Routing Protocol: OLSR

OLSR es un protocolo proactivo de MANETs para el encaminamiento de paquetes mediante la ruta más corta. [5] [8]

Cada nodo mantiene una tabla de nodos vecinos. Estos vecinos están clasificados en dos tipos: vecinos a un salto y vecinos a dos saltos. Los vecinos a un salto son los nodos vecinos que tienen conexión inalámbrica con el nodo y los vecinos a dos saltos son los que tienen conexión inalámbrica con los vecinos a un salto. Cada nodo selecciona un subconjunto de nodos vecinos a

un salto que le da acceso a todos los nodos vecinos a dos saltos. Estos vecinos son denominados **MPRs** (*multipoint relays*). Para garantizar que la topología de la red no ha cambiado y que si cambia se puedan efectuar los cambios necesarios en las tablas de encaminamiento, los nodos seleccionados como MPR envían de manera periódica mensajes **TC** (*topology control*) a los nodos que les han seleccionado a ellos como MPR.

Además, para el cálculo de la mejor ruta de un nodo a otro, se elige como camino más corto el mismo que siguen los paquetes de topología de red, es decir, el camino más corto de un nodo a otro es a través de los MPR de cada nodo.

Para seleccionar los MPRs de un nodo, los nodos envían mensajes **HELLO** de manera periódica a los vecinos que están a un solo salto. Estos mensajes contienen una lista con los vecinos a uno y dos saltos del nodo que envía el mensaje. Según la información de estos mensajes recibida por un nodo, este elige como MPRs al menor número de nodos que le dan acceso a todos los vecinos a dos saltos, de manera que cuando todos los nodos de la red han seleccionado a su MPR, la información puede ser encaminada a cualquier nodo de la red.

2.3.4. OLSRv2

OLSRv2 [9] es el sucesor de OLSR [8]. También es un protocolo proactivo para MANETs cuya diferencia principal es que se ha modularizado de manera que la elección de los vecinos ahora es llevada a cabo por el protocolo ***Neighborhood Discovery Protocol (NHDP)*** [10]. El formato de los mensajes están definidos en el documento ***Generalized Mobile Ad Hoc Network Packet/Message Format*** [11]. Los tipos, longitudes y valores de los mensajes (**TLVs**) están definidos en el RFC ***Representing Multi-Value Time in Mobile Ad Hoc Networks*** [12]. Las fluctuaciones en el envío de mensajes (*jitter* en inglés), están definidos en el documento ***Jitter Considerations in Mobile Ad Hoc Networks*** [13].

Para la elección de la ruta más corta cada nodo selecciona dos conjuntos de MPRs que le dan acceso a todos los nodos vecinos a dos saltos. Estos conjuntos son los **MPRs de inundación** y los **MPRs de encaminamiento**. Los MPRs de inundación son los que reenvían los mensajes de la topología de la red y los MPR de encaminamiento son aquellos enrutan el tráfico siendo nodos intermedios. Para la elección de estos conjuntos, a los mensajes HELLO, se le añade otro campo dentro del mensaje: **MPR_WILLING Message TLV** que indican cómo de dispuesto está un nodo para ser MPR de inundación y/o MPR de encaminamiento. Este parámetro es

configurado por el administrador de la red y podría usarse por ejemplo la disponibilidad de energía o la capacidad de computación de cada nodo para saber si están o no dispuestos a ser MPRs [9][Sección 5.4.8]. Con estos nuevos conjuntos de MPRs OLSRv2 asegura un mejor control del tráfico que OLSR.

Además, OLSRv2 no se basa unicamente en el descubrimiento de vecinos mediante saltos para calcular los MPRs y así la ruta más corta, sino que además usa métricas de red adicionales elegidas por los MPRs para encaminar el tráfico. A cada mensaje HELLO se le añade una metrica adicional elegida por los nodos para elegir a los MPRs. Está métrica puede ser el número de nodos distintos a los que el nodo candidato a ser MPR tenga enlace, por ejemplo. [9][Apéndice B.1]

2.4. Librería de Análisis de Datos: Pandas

Pandas [14] es un paquete de Python especializado en el manejo y análisis de datos, diseñado para trabajar con datos relacionales (tablas Excel o SQL), abstrayendo al programador de la dificultad que podría ser usar directamente una librería como Numpy [1] o Matplotlib [2].

Pandas usa dos clases de objetos principales para trabajar con los datos: ***series*** y ***dataframes***.

Los objetos *series* son arrays de una dimensión con datos del mismo tipo y de tamaño inalterable. Además, cada elemento de la serie está asociado a un índice para poder acceder a ese elemento.

```
0                10.10.0.1
1        02:00:00:00:65:05
2        02:00:00:00:65:07
3        02:00:00:00:65:10
4        02:00:00:00:65:04
                 ...
73070    02:00:00:00:65:06
73071    02:00:00:00:65:01
73072    02:00:00:00:65:09
73073    02:00:00:00:65:07
73074            10.10.0.3
```

Figura 2.3: Uso de una *Serie* con una captura en Wireshark

Los objetos *dataframes* son un conjunto de datos en forma de tabla donde cada columna es un objeto serie, es decir, todos los datos de una misma columna son del mismo tipo, pero

[1] https://numpy.org/

[2] https://matplotlib.org/

pueden ser distintos a los de otras columnas. Los *dataframes* contienen dos índices, uno para indicar la fila y otro para la columna. Además a éstos sí se les puede alterar el tamaño. Ambas clases de objetos se pueden ver en las figuras 2.3 y 2.4.

```
         No.        Time            Source  Destination Protocol  Length  \
0          1    0.000000         10.10.0.1  10.10.0.255  OLSR v1     122
1          2    0.040862  02:00:00:00:65:05    Broadcast   802.11     100
2          3    0.040862  02:00:00:00:65:07    Broadcast   802.11     100
3          4    0.040885  02:00:00:00:65:10    Broadcast   802.11     100
4          5    0.040886  02:00:00:00:65:04    Broadcast   802.11     100
...      ...         ...               ...          ...      ...     ...
73070  73071  598.876077  02:00:00:00:65:06    Broadcast   802.11     100
73071  73072  598.876083  02:00:00:00:65:01    Broadcast   802.11     100
73072  73073  598.876088  02:00:00:00:65:09    Broadcast   802.11     100
73073  73074  598.876077  02:00:00:00:65:07    Broadcast   802.11     100
73074  73075  598.939207         10.10.0.3  10.10.0.255  OLSR v1     242

                                                    Info
0                    OLSR (IPv4) Packet,  Length: 40 Bytes
1      Beacon frame, SN=0, FN=0, Flags=........, BI=1...
2      Beacon frame, SN=0, FN=0, Flags=........, BI=1...
3      Beacon frame, SN=0, FN=0, Flags=........, BI=1...
4      Beacon frame, SN=0, FN=0, Flags=........, BI=1...
...                                                  ...
73070  Beacon frame, SN=0, FN=0, Flags=........, BI=1...
73071  Beacon frame, SN=0, FN=0, Flags=........, BI=1...
73072  Beacon frame, SN=0, FN=0, Flags=........, BI=1...
73073  Beacon frame, SN=0, FN=0, Flags=........, BI=1...
73074                OLSR (IPv4) Packet,  Length: 160 Bytes

[73075 rows x 7 columns]
```

Figura 2.4: *Dataframe* con los datos de una captura en Wireshark

Con el uso de pandas, se pueden leer los datos desde un .csv o una hoja de cálculo e importarlos en un *dataframe* para poder hacer búsquedas, filtrar datos y generar gráficas desde Python como se ha hecho en el apéndice B.1 [línea 35]. Además, se puede crear manualmente un *dataframe* con datos ya obtenidos desde otras fuentes para poder almacenar y ordenar mejor la información y para poder realizar las operaciones antes descritas (apéndice B.2 [línea 68]).

Capítulo 3

Objetivos

El objetivo general de este trabajo es analizar el comportamiento de los protocolos OLSR, B.A.T.M.A.N., B.A.T.M.A.N. Advanced y OLSR versión 2, para estudiar su desempeño en escenarios de movilidad en redes MANET.

Los objetivos específicos a cumplir son los siguientes:

1. Instalación, configuración y generación de escenarios con Mininet WiFi.
2. Estudio de los protocolos de encaminamiento dinámico para MANETs:
   - Better Aproach To Mobile Ad-hoc Networking (B.A.T.M.A.N.).
   - Better Aproach To Mobile Ad-hoc Networking Advanced (B.A.T.M.A.N. Advanced).
   - Optimized Link State Route (OLSR).
   - Optimized Link State Route versión 2 (OLSR version 2).
3. Implementación de la movilidad mediante marcas de tiempo en escenarios de Mininet WiFi.
4. Implementación de un script para analizar la cantidad de paquetes que generan los protocolos en los distintos escenarios de movilidad.
5. Implementación de un script para analizar la velocidad en la convergencia de las rutas de encaminamiento con los distintos protocolos.
6. Análisis comparativo de los resultados obtenidos.

Capítulo 4

Implementación

4.1. Escenarios de movilidad en Mininet WiFi

Para la generación de los escenarios en Mininet WiFi [2] se utiliza el lenguaje Python para definir los nodos que componen el escenario, los parámetros de red y en general, la configuración del mismo. En esta sección se describirá con detalle como generar un escenario de movilidad como los usados en este trabajo.

Se han creado dos tipos de escenarios: escenarios con movilidad determinista y escenarios con movilidad aleatoria. Se explicará como generar uno desde cero y sus diferencias en las secciones 4.1.1 y 4.1.2 respectivamente.

4.1.1. Escenarios de movilidad determinista

La figura 4.1 muestra un ejemplo de configuración de un escenario de Mininet WiFi con movilidad determinista. Para realizar esta configuración, primero se ha de crear la red. Para ello se usa el módulo *wmediumd* necesario para virtualizar tarjetas de red. Seguidamente se añaden los nodos al escenario con sus parámetros y se inicia el modelo de propagación que usa el emulador para simular el comportamiento de un medio inalámbrico.

A continuación hay que indicarle al emulador qué nodos se añaden a la red generada en el paso anterior y los parámetros de esa red. Es en este paso donde se define qué tipo de enlace se va a usar, qué interfaz de red se va a conectar, el SSID de la red, el canal y el modo b/g/n en 2.4 GHz o ac en 5GHz. También es en este paso donde se le puede indicar a Mininet WiFi

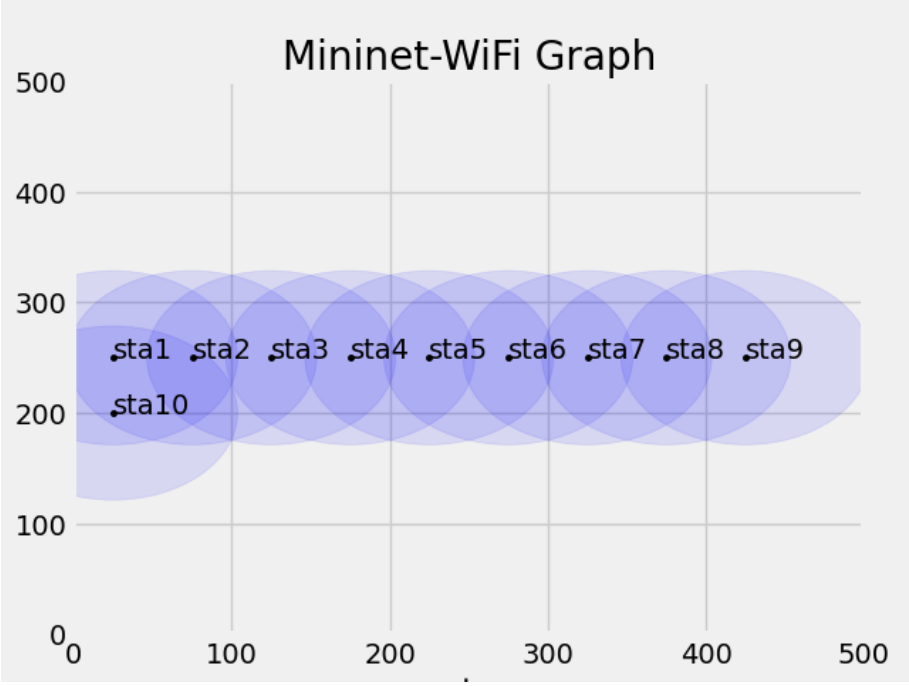


Figura 4.1: Escenario con movilidad determinista

si usar o no el modo *High Throughput* dado por el parámetro *ht_cap*. Este parámetro, que hace uso de la librería *hostapd*, permite usar un canal de transmisión cuyo ancho de banda sea de 40 MHz si se le indica, o en su defecto 20 MHz. Para la realización de los experimentos se ha dejado el valor de *ht_cap* por defecto después de una conversación con Ramón Fontes (creador de Mininet WiFi) [1]. En ella Ramón explica que al usar el rango estándar de los nodos la potencia es de -83 dBm y, con ésta, los nodos no pueden hacer un uso real de este modo. Por tanto, el uso de este parámetro puede producir errores en los experimentos o que OLSRv2 no funcione correctamente, como así habíamos observado en las primeras pruebas.

Además en este paso se busca si en los argumentos que se le pasan al script del escenario, están incluidos los demonios de los protocolos. Si están en la lista `protocols` (apéndice A.1[línea 34]), utiliza ese protocolo de encaminamiento y si no, muestra un mensaje de información. No se ha añadido el demonio *batmand* al diccionario porque al ejecutarlo de esta manera los nodos se bloqueaban y no tenían conexión unos con otros, teniendo que ser ejecutado mediante comandos en la consola de Mininet WiFi.

Seguidamente, se comprueba si el argumento *-p* ha sido incluido en la ejecución para mostrar la figura con los nodos.

[1] https://groups.google.com/g/mininet-wifi-discuss/c/MzHNR1bTRWM

Para activar el modelo de movilidad *ReplayingMobility* se debe indicar los nodos que se mueven y el archivo .dat del que se leen las coordenadas y los saltos de tiempo mediante la función `get_trace()`. La movilidad está prefijada de manera determinista para que los nodos se muevan a coordenadas específicas en instantes de tiempo determinados. Estos parámetros se determinadan por tres columnas en un archivo .dat con el siguiente formato:

```
75 200 90
125 200 150
175 200 210
225 200 270
275 200 330
325 200 390
375 200 450
425 200 510
```

Cada una de estas columnas corresponde respectivamente a las coordenadas X e Y en un plano de dos dimensiones y a la marca de tiempo en la que el nodo modifica sus coordenadas en el escenario. De esta manera podemos hacer que un nodo esté el tiempo que consideremos oportuno en cada posición.

Para ejecutar la movilidad con marcas temporales, ya sean las coordenadas fijadas de manera determinista o generadas aleatoriamente, es necesario usar una función auxiliar cuyo código fuente se puede encontrar en el apéndice A.1 (líneas 72-97) y en A.2 (líneas 99-117). Esta función fija la primera posición y la marca de tiempo en la gráfica de dos dimensiones, lee el fichero .dat donde están las coordenadas con su marca de tiempo e incluye ambos parámetros en sendas listas (`p.pos` y `p.time`) ya predefinidas en *ReplayingMobility* de manera que las use para generar el movimiento de los nodos.

Para finalizar, se llama a la interfaz de comandos de Mininet WiFi (CLI) la cual se ejecuta en bucle, por lo que cualquier función auxiliar que se necesite emplear en el escenario tiene que ser llamada antes.

El código fuente completo se puede ver en el apéndice A.1.

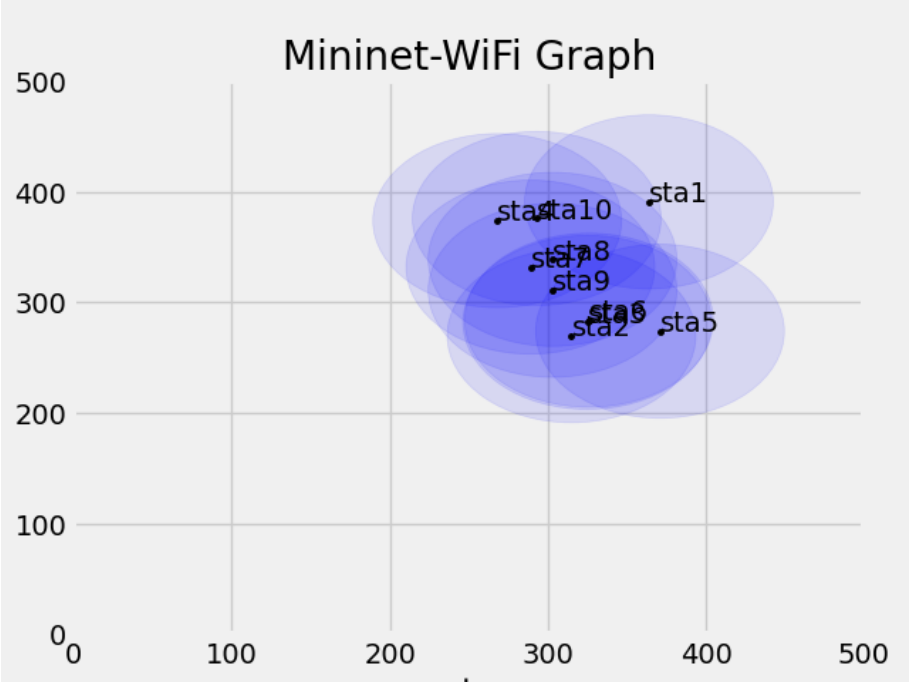


Figura 4.2: Escenario con movilidad aleatoria

4.1.2. Escenarios de movilidad aleatoria

Para generar un escenario con movilidad aleatoria como en la figura 4.2, se añade una función al modelo de visto en la sección 4.1.1 que genera de manera aleatoria las posiciones iniciales de los nodos en el escenario y es llamada en la función `get_trace()`. Esta aleatoriedad se controla mediante una semilla al inicio para que el escenario genere siempre las mismas posiciones en todos los nodos mientras se use la misma semilla, con el fin de comparar el comportamiento de todos los protocolos con el mismo escenario aleatorio.

Se define un escenario en el que todos los nodos se mueven. Para la generación de los archivos .dat de cada uno de los nodos, se ha creado un script simple y bastante parecido a la función que genera la primera posición (véase el apéndice A.2.1). Este script crea los ficheros .dat de cada uno de los nodos que hay en el escenario y escribe las coordenadas de manera aleatoria. No obstante, las marcas de tiempo son las mismas siempre para poder garantizar que todos los escenarios duran la misma cantidad de tiempo en ejecutarse a fin de que las pruebas en los distintos escenarios (capítulo 5) sean similares y esto no influya en los resultados obtenidos.

El código fuente de un escenario de este tipo se puede leer en el apéndice A.2.

4.2. Scripts de Análisis de Datos

4.2.1. Cantidad Media de Paquetes por Protocolo

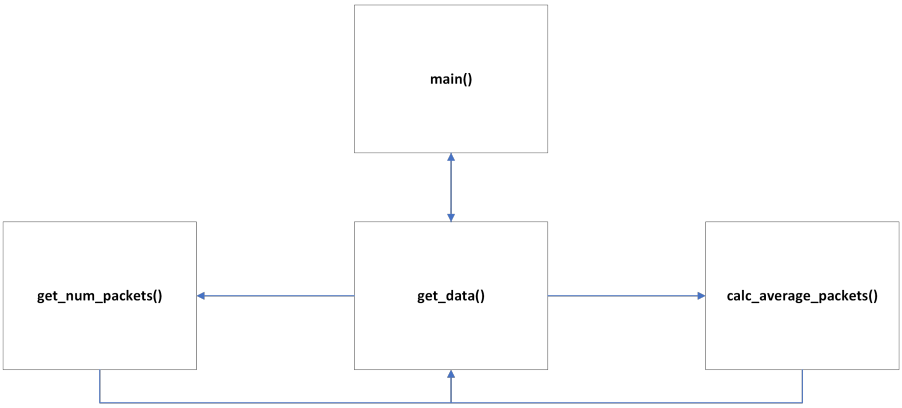


Figura 4.3: Diagrama del Analizador de Paquetes

Para el análisis de la cantidad media de paquetes generadas por cada uno de los protocolos en las pruebas realizadas en un escenario, se ha desarrollado un script en Python usando la librería Pandas [14] para el análisis de los datos. Este script consta de una función principal y tres funciones auxiliares (`get_data()`, `get_num_packets()` y `calc_average_packets()`) como describe la figura 4.3.

La función principal del script comprueba que el número de argumentos es correcto, almacena en variables las rutas donde están los archivos .csv que se han exportado desde la captura de paquetes hecha con la interfaz *hwsim0*, llama a la función auxiliar `get_data()` que procesa los datos en las rutas almacenadas anteriormente y genera la gráfica con los resultados obtenidos.

Para la generación de la gráfica, se crea un *dataframe* con los datos obtenidos de la función `get_data()`. En este *dataframe* que se ha creado, los protocolos son las filas y los datos son las columnas.

Seguidamente con el método `plot.bar()` se indica que se quiere dibujar una gráfica donde el eje de abscisas son los protocolos (si no se le indica ninguno, usa el parámetro *index* por defecto) y en el eje de ordenadas la cantidad media de paquetes generados por cada protocolo. Esta cantidad de paquetes se ha calculado haciendo la media de los paquetes generados en las cinco pruebas, para cada protocolo, en cada uno de los escenarios como se explica en la

sección 5.1. Además, para que se muestre el número exacto encima de la gráfica como en la figura 4.4, se accede al primer valor del contenedor dentro de *bar*.

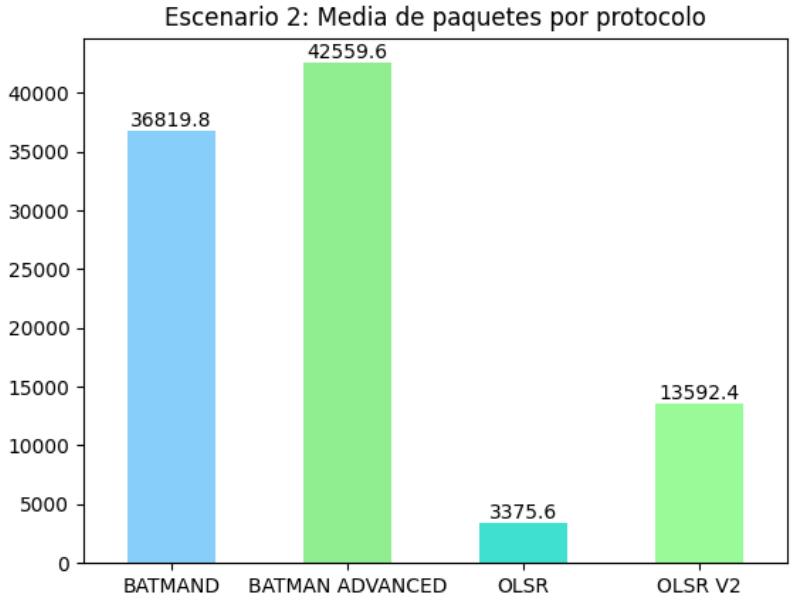


Figura 4.4: Cantidad media de paquetes por protocolo en un escenario

La función auxiliar `get_data()` toma la ruta donde se encuentran los datos de cada protocolo con su escenario ya seleccionado y genera un diccionario con la librería *Collections* para poder crear diccionarios anidados más fácilmente. A continuación se listan los ficheros .csv dentro de esa ruta, se llama a la función de Pandas `read_csv()` que lee los ficheros .csv y los agrega a un *dataframe*.

Usando la función auxilar `get_num_packets()` que filtra los datos por protocolo, éstos datos son agregados a un diccionario anidado donde una clave es el número de prueba a la que pertenecen los datos y la segunda clave es el dato que ha obtenido, en este caso el numero total de paquetes. Para finalizar, llama a `calc_average_packets()` a la que le pasa este diccionario anidado para que devuelva el valor medio de la cantidad de paquetes de todas las pruebas en un mismo escenario y devuelve este valor a la función principal.

A la función auxiliar `get_num_packets()` se le pasan dos argumentos: el *dataframe* generado después de leer el fichero .csv y una cadena que identifica al protocolo, necesaria para saber sobre qué protocolo se quiere actuar. Esta función filtra los datos en el *dataframe* según el nombre de la columna *Protocol* dentro del fichero .csv como se puede ver en la figura 4.5 y

se queda con el número de filas que contienen a ese protocolo en la columna.

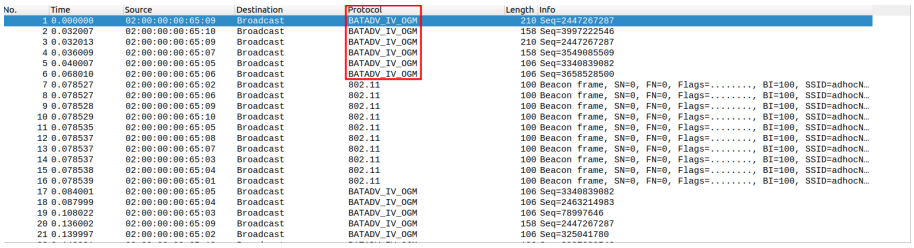


Figura 4.5: Columna *Protocol* usada para filtrar

Para finalizar con este script, la función `calc_average_packets()` a la que se le pasa el diccionario anidado que ha sido descrito en la función `get_data()`, obtiene los valores del diccionario, los añade a una lista y calcula la media de los datos, la cual devuelve a `get_data()` que esta a su vez devuelve a la función principal.

El script en su conjunto se puede leer en el apéndice B.1.

4.2.2. Tiempos de Convergencia de las Tablas de Encaminamiento por Protocolo

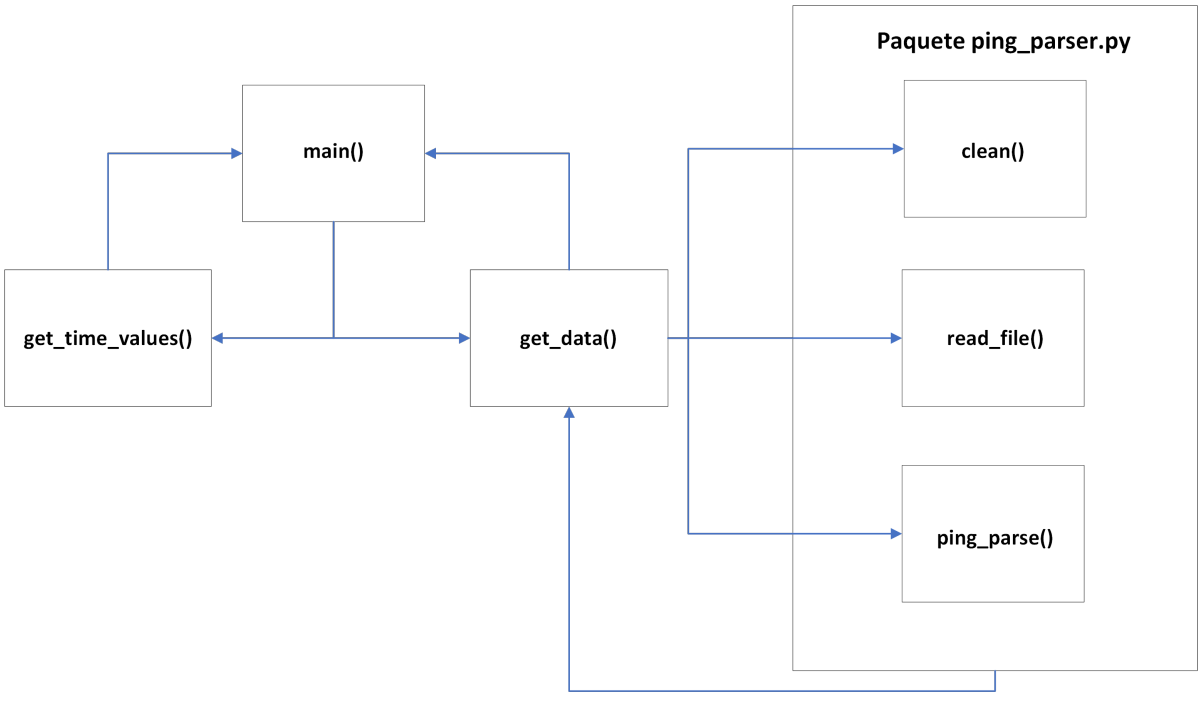


Figura 4.6: Diagrama del Analizador de *pings*

Este script realizado también con Python, analiza la diferencia entre las secuencias de *icmp_seq* en la salida del comando *ping* cuyo origen y destino son dos nodos del escenario.

El script está compuesto de una función principal y dos funciones auxiliares, pero, además, una de éstas, hace uso de otro paquete escrito también en Python llamado *ping_parser.py*. Dentro de este paquete hay 3 funciones auxiliares extra que son las encargadas de procesar los ficheros que contienen la salida del comando *ping* para su posterior análisis. El diagrama del script se puede observar en la figura 4.6.

El programa principal almacena las rutas de los directorios donde se encuentran los ficheros que guardan las salidas de cada una de las pruebas y llama a las funciones auxiliares `get_data()` y `get_time_values()`, las cuales procesan los datos. Estos datos, son añadidos a un *dataframe* que almacena todos los datos y genera las gráficas de cada uno de los tiempos medios de convergencia de las tablas de encaminamiento, donde el eje de abscisas corresponde a cada protocolo usado en las pruebas y el eje de ordenadas corresponde a los segundos que está desconectado, como se puede observar en la figura 4.7.

El script puede consultarse en el apéndice B.2.

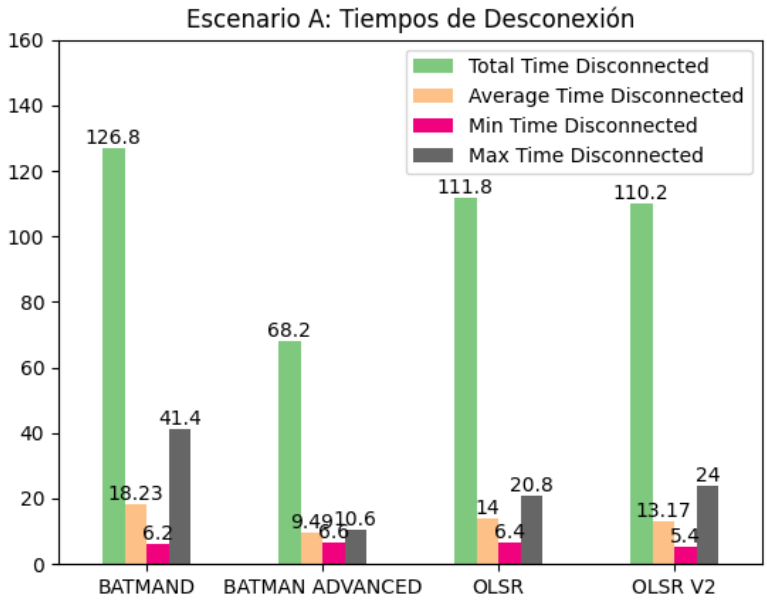


Figura 4.7: Tiempos medios de todos los protocolos en un escenario

La función que usa el paquete *ping_parser.py* es `get_data()`. Ésta llama a las funciones dentro de *ping_parser.py*, las cuales limpian los archivos de salida del comando *ping* elimi-

nando las lineas de error como *Destination Host Unreachable* o *Time to live exceeded* para poder seguidamente leer los nuevos archivos ya limpiados y mediante el uso de expresiones regulares [15] filtrar todos los *icmp_seq*. A continuación, se leen todas las secuencias de ICMP guardadas en la lista y cuando la diferencia entre el valor actual leído y el anterior es superior a cuatro, se almacena en otra lista de tiempos como se puede leer en el apéndice B.2.1.

Con esta lista se generan los valores estadísticos como el **tiempo total, medio, máximo y mínimo** devolviéndolos a la función `get_data()`. Estos valores se integran en un diccionario anidado al igual que en la subsección 4.2.1.

Igualmente a lo realizado en el *packet_analyzer.py*, en la función `get_time_values()` estos datos son ordenados, y se vuelve a realizar la media de todos los datos para sacar un valor estadístico único de cada medida con cada protocolo en un mismo escenario después de realizar todas las pruebas.

Capítulo 5

Pruebas y Análisis de los Resultados

5.1. Metodología utilizada

En la elaboración de las pruebas realizadas en cada escenario se ha seguido la misma metodología tanto a la hora de realizar las pruebas, como en el número y tiempo de duración de éstas o al capturar la información y almacenarla para su posterior análisis, de manera que no haya diferencias entre unas y otras.

Cada experimento dura diez minutos. Durante la realización de la prueba, los nodos cambian de posición un total de diez veces, por tanto, se mantienen en la misma posición un minuto. Además, se realizan cinco experimentos por protocolo en cada escenario.

La captura de paquetes se realiza mediante la interfaz de red virtual *hwsim0* que es capaz de capturar todos los paquetes que se envían y reciben en el escenario. Para asegurar que las pruebas realizadas duran siempre diez minutos, se usa el comando de Linux *timeout* al cual le pasamos los segundos que debe estar activa la captura.

Para las capturas se envían mensajes ICMP mediante el comando *ping* desde un nodo de la red como origen hasta otro nodo de la misma red como destino. También para asegurar que el tiempo de la realización de las pruebas es el mismo, se utiliza el mismo comando *timeout*. Tanto para la captura de paquetes como para el envío de los mensajes ICMP, el tiempo total es de seiscientos segundos.

5.2. Escenario A

5.2.1. Descripción del escenario

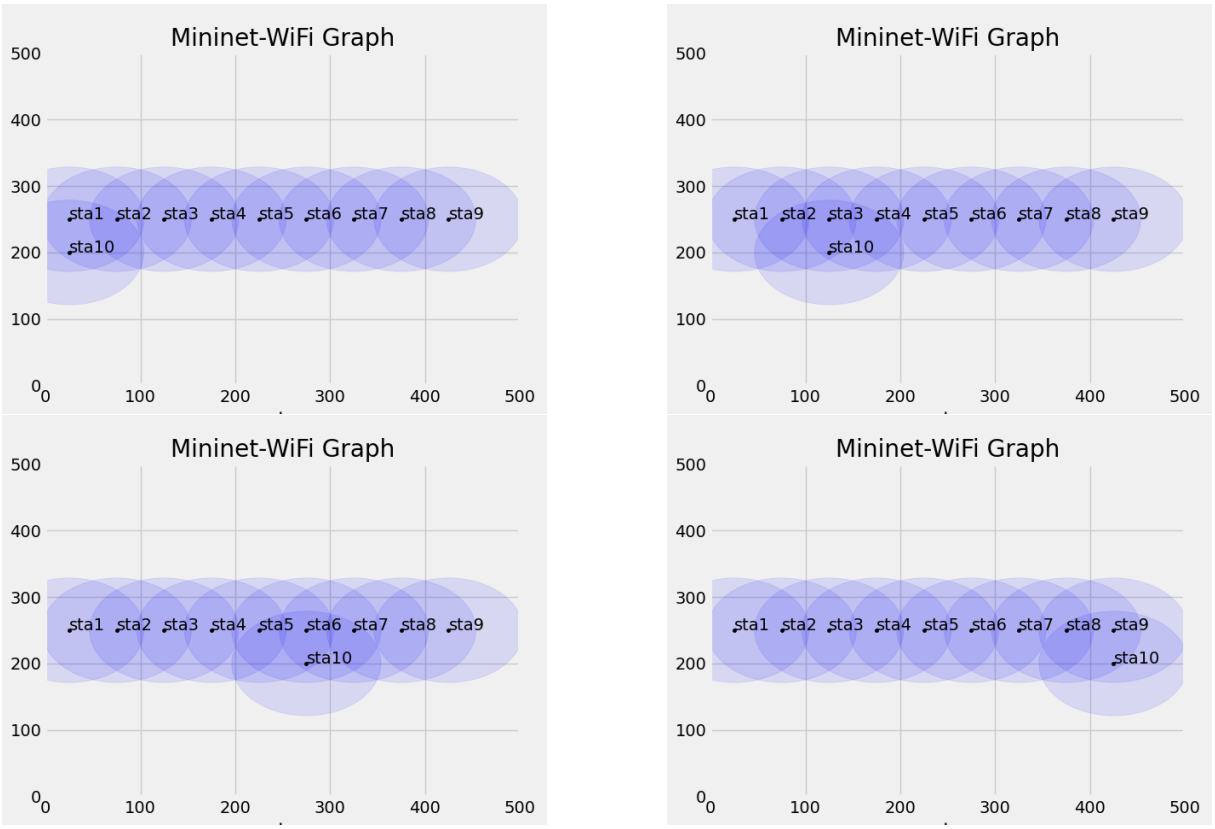


Figura 5.1: Escenario A. Movimiento de los nodos mostrado en varios instantes de tiempo

En este escenario, el nodo *sta10* es el único que se mueve mientras que los otros están a una distancia mínima de veinticinco metros a él y cada nodo estático a cincuenta metros de sus vecinos. El nodo *sta10* empieza frente al nodo *sta1* y se va desplazando hasta que llega a estar frente al nodo *sta9*.

Debido a la separación de los nodos con el nodo en movimiento, hay varios caminos por los que un nodo que se encuentre cerca de sta10 puede enviar y recibir datos de éste, por tanto aunque es un escenario de movilidad, no es muy exigente para los protocolos a la hora de volver a calcular las tablas de encaminamiento. El movimiento de los nodos en este escenario se muestra en las figuras 5.1 y 5.3.

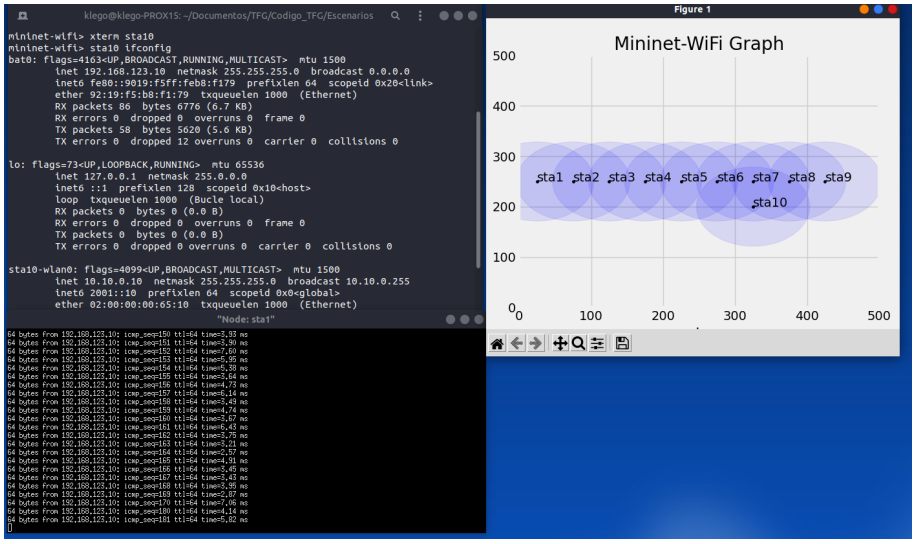


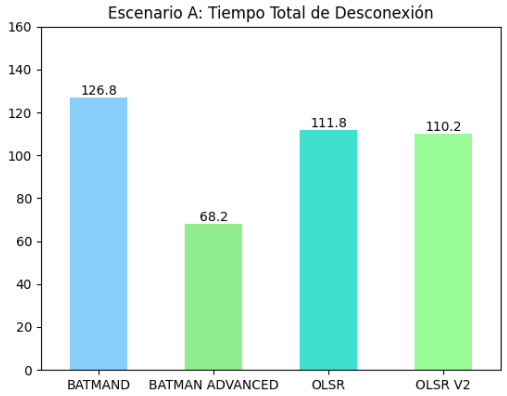
Figura 5.2: Escenario A. Ejecución del escenario en un instante de tiempo

Figura 5.3: Diagrama de movilidad del Escenario A

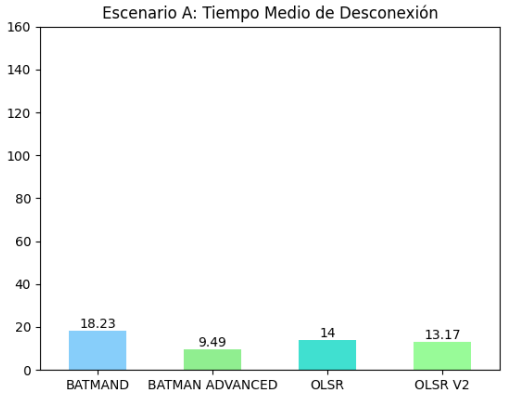
5.2.2. Resultados del escenario

Las gráficas 5.4a y 5.4b indican, el tiempo total y el tiempo medio en el que dos nodos del escenario no tienen conectividad entre si. Como se puede apreciar, B.A.T.M.A.N. Advanced es el protocolo con menor tiempo de desconexión seguido de OLSRv2. Además, las gráficas 5.4c y 5.4d nos indican el tiempo mínimo y máximo que tardan los protocolos en establecer las rutas entre los dos nodos del escenario. En estas gráficas podemos ver que OLSRv2 es quien tiene el menor tiempo mínimo seguido de B.A.T.M.A.N. Los protocolos que tienen el tiempo más alto a la hora de recalcular las rutas en este escenario son B.A.T.M.A.N. y OLSRv2.

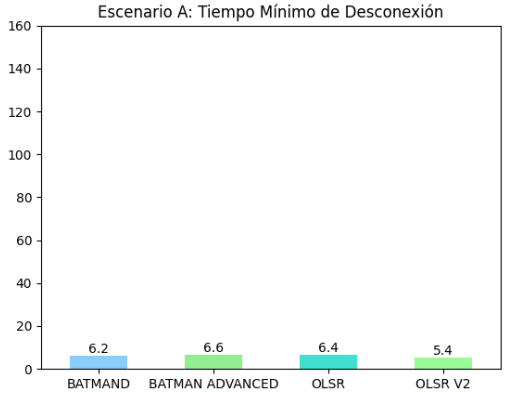
En la figura 5.5 podemos ver que el protocolo que genera mayor número de paquetes en este escenario es B.A.T.M.A.N. Advanced seguido de B.A.T.M.A.N. Los que menor número de paquetes generan son OLSR seguido de OLSRv2



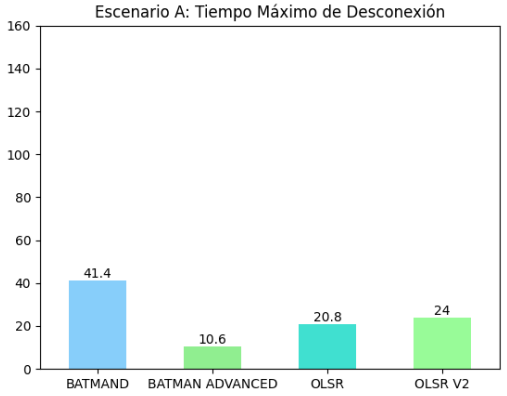
(a) Tiempo Total de Desconexión



(b) Tiempo Medio de Desconexión



(c) Tiempo Mínimo de Desconexión



(d) Tiempo Máximo de Desconexión

Figura 5.4: Escenario A. Tiempos de Desconexión

5.2.3. Análisis de los resultados

Se puede ver una mejora notable en B.A.T.M.A.N. Advanced con respecto a su predecesor, pues esta nueva implementación de B.A.T.M.A.N. supone una mejora notable en la estabilidad de las rutas calculadas como se puede apreciar en las gráficas 5.4b y 5.4a.

Sin embargo, aunque OLSRv2 y B.A.T.M.A.N. tengan el menor tiempo mínimo en recalcular las rutas como se puede ver en la figura 5.4c, en la gráfica 5.4d se observa que el mejor protocolo es B.A.T.M.A.N. Advanced porque en el peor de los casos es bastante mejor. Esto indica que aunque OLSRv2 es el más rápido en volver a calcular las rutas, no es tan estable como lo es B.A.T.M.A.N. Advanced.

También se puede apreciar que B.A.T.M.A.N es el protocolo que peor parado sale en este

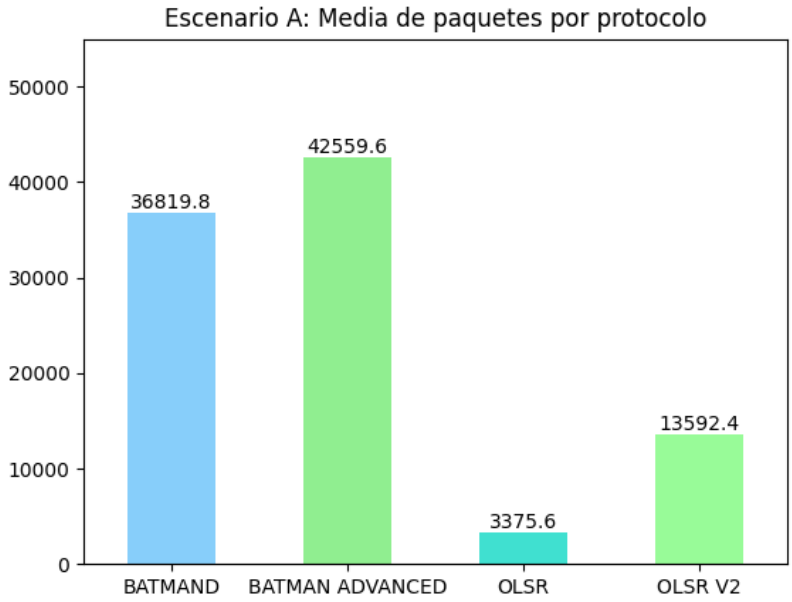


Figura 5.5: Escenario A. Cantidad de paquetes generados por protocolo

escenario porque es el menos estable y además el que mayor tiempo de desconexión presenta. OLSR es algo mejor que B.A.T.M.A.N., pero tampoco supone una mejora abismal en comparación con los nuevos protocolos B.A.T.M.A.N. Advanced y OLSRv2.

Es importante señalar que la diferencia entre los tiempos de OLSR y OLSRv2 no es tan grande como la de B.A.T.M.A.N. y B.A.T.M.A.N. Advanced. Y aunque OLSRv2 mejore en general a OLSR, esta mejora no es tan significativa.

En cuanto al número de paquetes, parece que el protocolo con mejor estabilidad es el que más paquetes envía como se puede apreciar en la figura 5.5. Sin embargo, se puede ver que aunque OLSRv2 envía bastante menos paquetes que B.A.T.M.A.N. Advanced lo cual se traduce en una menor congestión de tráfico en el escenario, su estabilidad general y su desempeño es menor.

Para terminar con el análisis de este escenario, B.A.T.M.A.N. es el segundo protocolo que más paquetes envía y en contraste con los tiempos de las gráficas 5.4, genera demasiado tráfico que no se ve reflejado en una mejor eficiencia y desempeño al contrario que B.A.T.M.A.N. Advanced.

5.3. Escenario B

5.3.1. Descripción del escenario

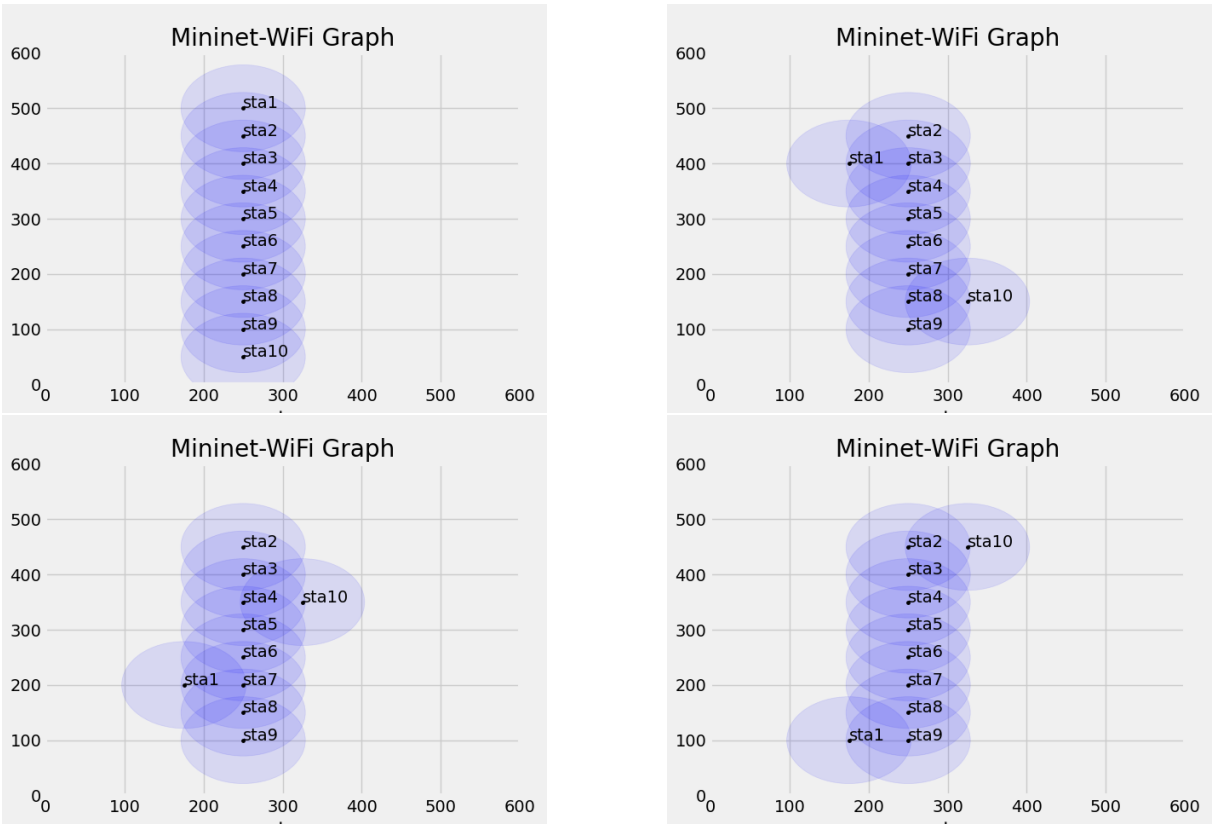


Figura 5.6: Escenario B. Movimiento de los nodos mostrado en varios instantes de tiempo

En este escenario, se intenta complicar a los nodos el la creación de rutas moviendo los nodos origen y destino del comando *ping* (*sta1* y *sta10*) como se puede apreciar en las figuras 5.6 y 5.8. Este escenario es bastante más exigente para los nodos porque cada nodo solo tiene en todo momento en la cobertura de su antena unicamente otro nodo y además estos cambian constantemente, haciendo más complicado el recalcular las entradas en la tabla de encaminamiento.

5.3.2. Resultados del escenario

En la gráfica 5.9a se puede apreciar que el protocolo con menos cantidad de desconexiones es B.A.T.M.A.N. Advanced seguido de su predecesor B.A.T.M.A.N. y OLSRv2. OLSR es el que más tiempo total está desconectado. Sin embargo, en tiempo medio de desconexión (figura

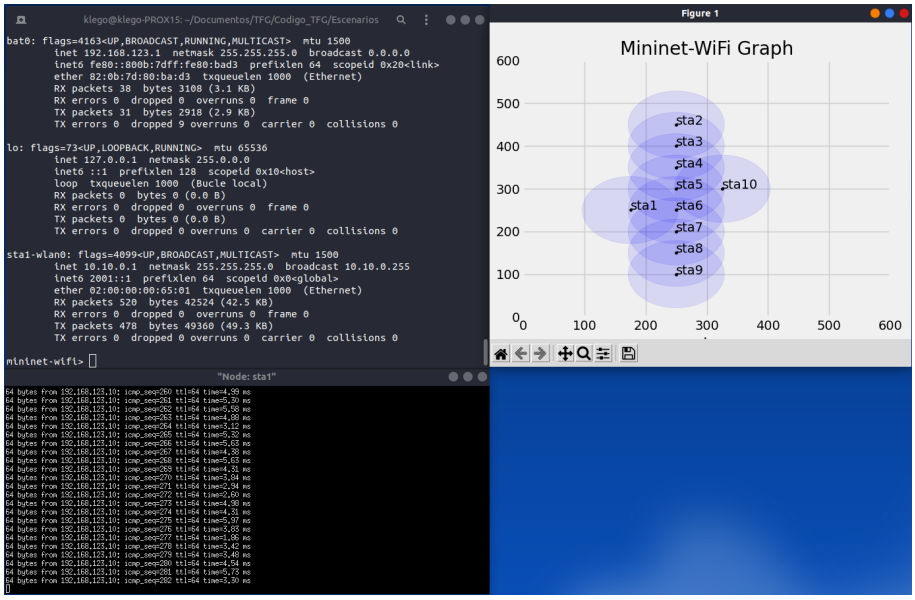


Figura 5.7: Escenario B. Ejecución del escenario en un instante de tiempo

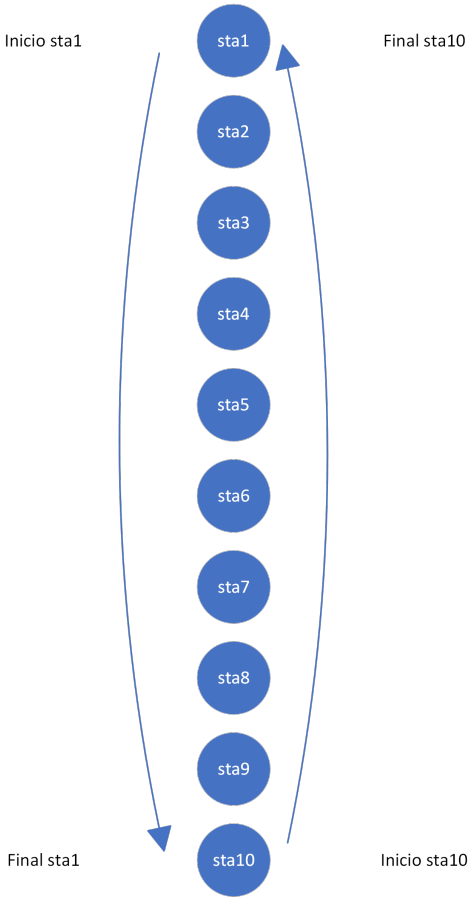
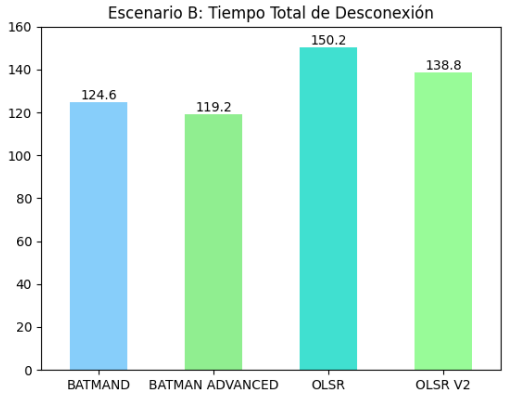
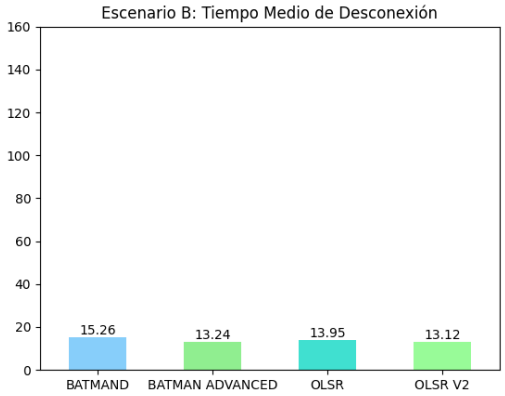


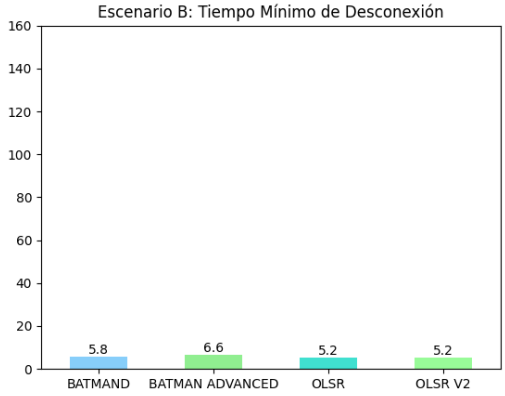
Figura 5.8: Diagrama de movilidad del Escenario B



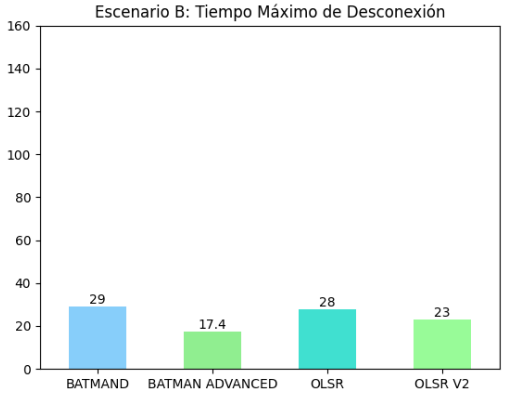
(a) Tiempo Total de Desconexión



(b) Tiempo Medio de Desconexión



(c) Tiempo Mínimo de Desconexión



(d) Tiempo Máximo de Desconexión

Figura 5.9: Escenario B. Tiempos de Desconexión

5.9b), se puede observar que mientras B.A.T.M.A.N. Advanced y OLSRv2 están prácticamente a la par, OLSR tiene menor tiempo y el que peores datos muestra es B.A.T.M.A.N.

En la gráfica 5.9c se puede ver que OLSR y OLSRv2 tienen el menor tiempo, seguido de B.A.T.M.A.N. y B.A.T.M.A.N. Advanced con el mayor tiempo mínimo. En cambio, en la gráfica que muestra el tiempo máximo (figura 5.9d), se ve que es B.A.T.M.A.N Advanced quien tiene el menor tiempo seguidos de OLSRv2, OLSR y B.A.T.M.A.N. en último lugar.

En cuanto a la generación de paquetes, B.A.T.M.A.N. Advanced sigue siendo el que más paquetes genera seguido de B.A.T.M.A.N., OLSRv2 y OLSR.

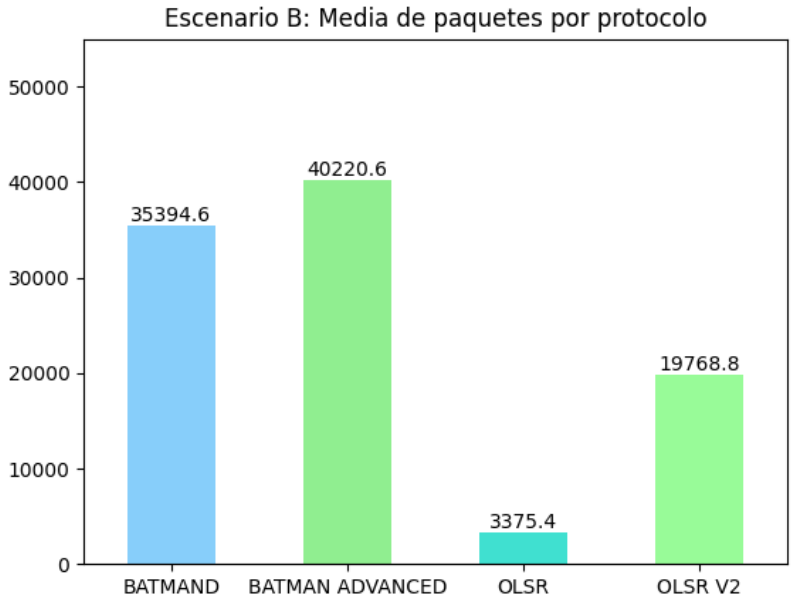


Figura 5.10: Escenario B. Cantidad de paquetes generados por protocolo

5.3.3. Análisis de los resultados

Viendo las gráficas 5.9a y 5.9b se puede observar que OLSR y OLSRv2 son los protocolos que más desconexiones presentan, es decir, se efectúan más cambios en las rutas, y por tanto, son menos estables. Sin embargo, tanto OLSRv2 como B.A.T.M.A.N Advanced son los protocolos más rápidos a la hora de recalcular las rutas, y aunque B.A.T.M.A.N. sea más estable que OLSR y OLSRv2, el tiempo que tarda en calcular las rutas es mucho más alto, con lo cual, en media, está más tiempo desconectado.

También se puede apreciar en las gráficas 5.9c y 5.9d que en el mejor de los casos, OLSRv2 y OLSR tienen la ventaja de ser más rápidos en la generación de entradas en la tabla de encaminamiento, aunque en el peor de los casos, B.A.T.M.A.N. Advanced es el protocolo que mejor se comporta seguido de OLSRv2 y OLSR. B.A.T.M.A.N., en cambio, es el peor protocolo también en este escenario porque es el que más tiempo medio de desconexión presenta y en el peor de los casos también es el que más tarda en recalcular las rutas.

En este escenario se puede ver que si el origen y el destino del comando *ping* se mueven y además unicamente hay un camino único, los protocolos B.A.T.M.A.N. Advanced y OLSRv2 están muy igualados en el tiempo medio que tardan sus tablas de encaminamiento en ser cohe-

rentes, pero se observa que B.A.T.M.A.N. Advanced es mejor protocolo en generar rutas que OLSRv2 (tiene menor tiempo total de desconexión) y en el peor de los casos también se comporta mejor, ya que tiene un menor tiempo máximo.

Si también tenemos en cuenta la cantidad de paquetes generados por cada protocolo, esta vez B.A.T.M.A.N. Advanced vuelve a generar un mayor número de paquetes, pero esta cantidad ya no se corresponde a un menor tiempo medio de desconexión. En cambio, OLSRv2, con menos de la mitad de paquetes que B.A.T.M.A.N. Advanced, es capaz de mantener la conexión activa el mismo tiempo aunque tenga más desconexiones.

B.A.T.M.A.N. sigue generando demasiado tráfico para el tiempo medio que está desconectado y OLSR es el que menor número de paquetes genera y aunque es mejor que B.A.T.M.A.N, sigue estando a medio camino entre B.A.T.M.A.N. y su sucesor.

5.4. Escenario C

5.4.1. Descripción del escenario

Este escenario es un punto intermedio entre los escenarios A y B (secciones 5.2 y 5.3). En este escenario, el nodo origen del *ping* sigue siendo *sta1* y el nodo destino es *sta10*. La diferencia con el escenario A es que ahora los nodos pares están separados a 50 metros de *sta10* haciendo que éste tenga un único camino cuando está frente a un nodo impar y que tenga varios caminos cuando está frente a los nodos pares. Además, entre los nodos estáticos, solo hay una ruta posible para encaminar el tráfico. Esta característica de tener un único camino es la agregada en el escenario B. El diagrama de movilidad se puede observar en las figuras 5.11 y 5.13.

5.4.2. Resultados del escenario

En la gráfica 5.14a se puede apreciar el protocolo con mayor tiempo de desconexión es B.A.T.M.A.N. seguido por OLSRv2 y OLSR. B.A.T.M.A.N. Advanced tiene el tiempo más bajo.

En cuanto al tiempo medio de desconexión (figura 5.14b), OLSRv2 es el que tiene el mayor tiempo sequido de B.A.T.M.A.N. Los protocolos con menor tiempo son B.A.T.M.A.N. Advan-

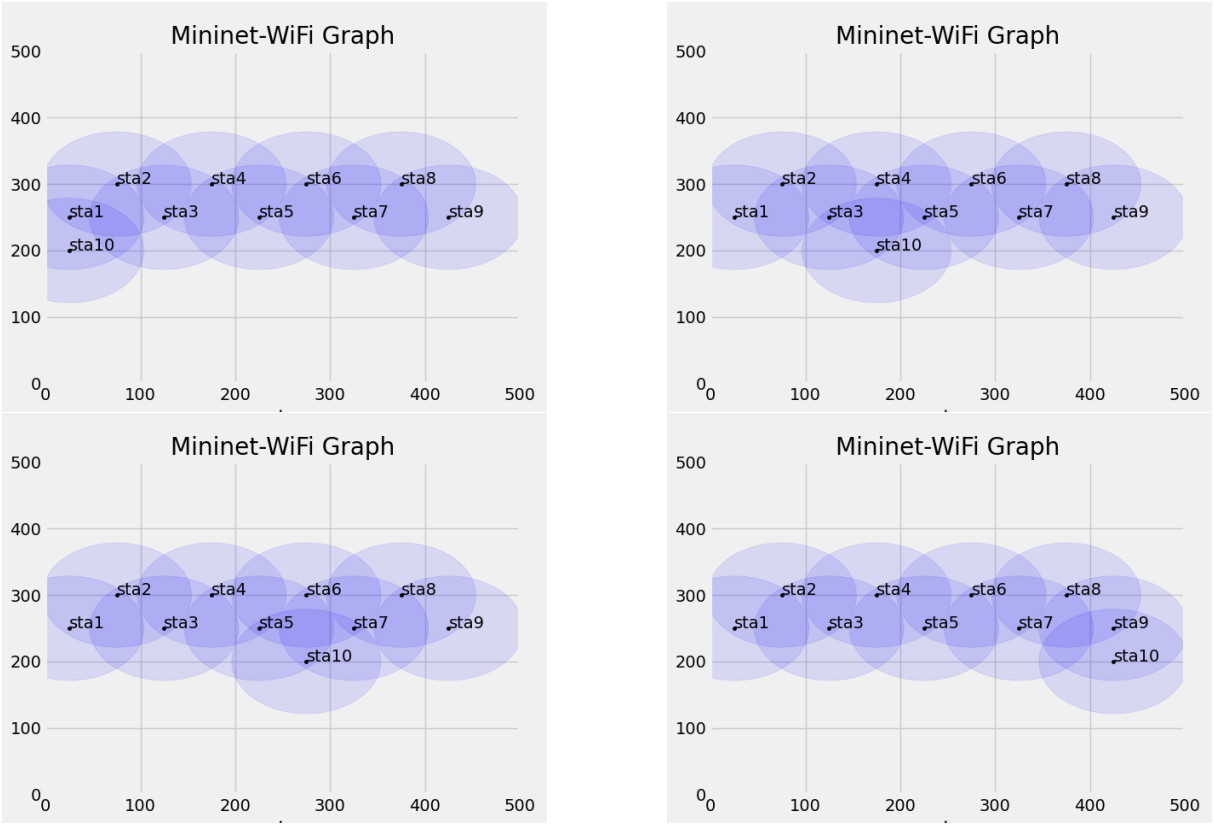


Figura 5.11: Escenario C. Movimiento de los nodos mostrado en varios instantes de tiempo

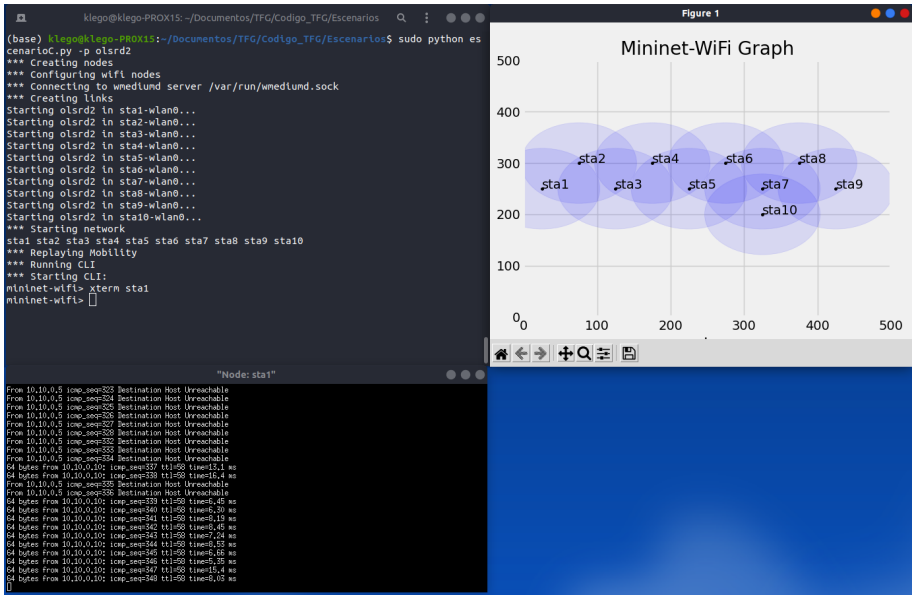


Figura 5.12: Escenario C. Ejecución del escenario en un instante de tiempo

ced y OLSR.

El protocolo con el menor tiempo mínimo es B.A.T.M.A.N. Advanced seguido muy de

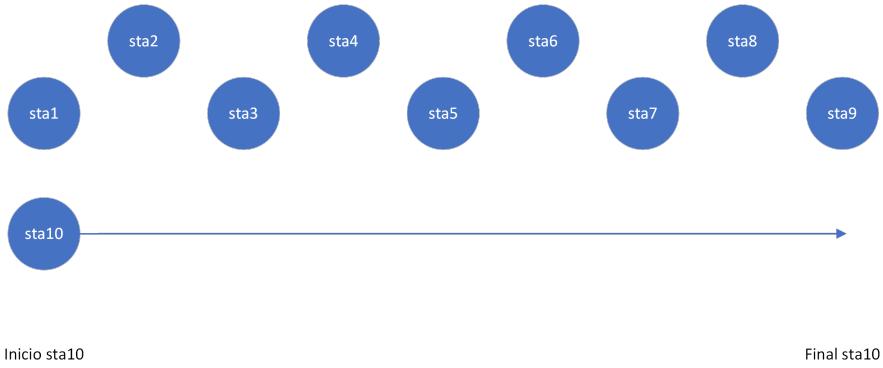
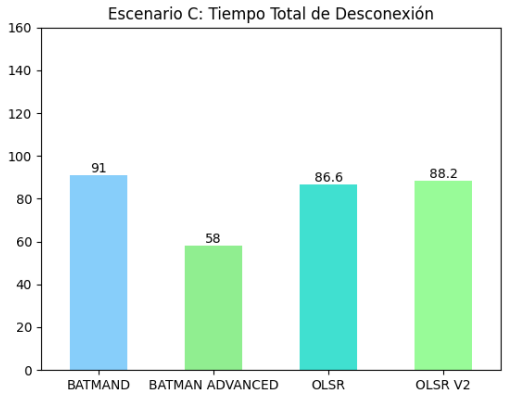
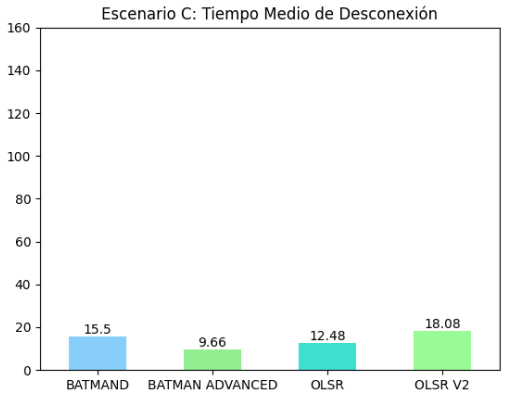


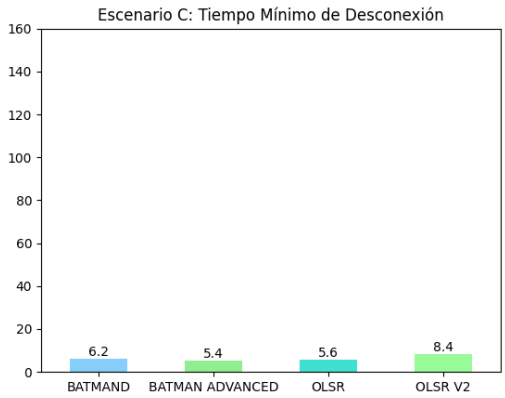
Figura 5.13: Diagrama de movilidad del Escenario C



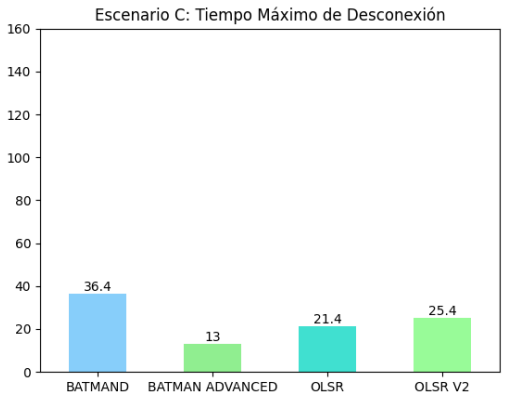
(a) Tiempo Total de Desconexión



(b) Tiempo Medio de Desconexión



(c) Tiempo Mínimo de Desconexión



(d) Tiempo Máximo de Desconexión

Figura 5.14: Escenario C. Tiempos de Desconexión

cerca por OLSR, como se puede observar en la figura 5.14c. Los protocolos que mayor tiempo mínimo presentan son B.A.T.M.A.N. y OLSRv2.

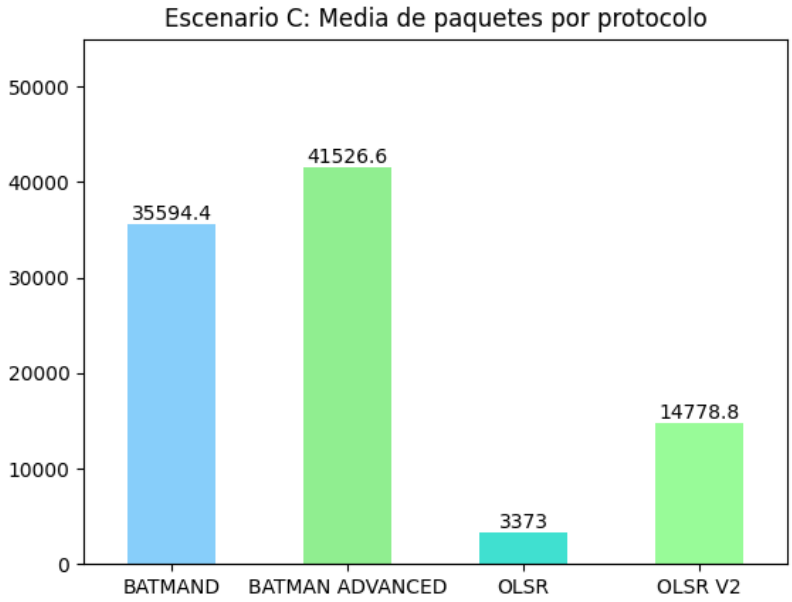


Figura 5.15: Escenario C. Cantidad de paquetes generados por protocolo

En cambio, en cuanto al tiempo máximo de desconexión se refiere, el protocolo con el menor tiempo es B.A.T.M.A.N. Advanced seguido de OLSR, OLSRv2 y por último B.A.T.M.A.N. (véase la figura 5.14d).

En cuanto a cantidad de paquetes (gráfica 5.15), se observa lo mismo que en las gráficas 5.5 y 5.10. B.A.T.M.A.N. Advanced y B.A.T.M.A.N. son los protocolos que más tráfico generan en la red seguidos de OLSRv2 y OLSR siendo el que menor número de paquetes genera.

5.4.3. Análisis de los resultados

En este escenario se puede ver que el mejor protocolo es sin duda B.A.T.M.A.N. Advanced. Es el protocolo más estable si se observa la figura 5.14a y el que menos tiempo tarda en recalcular las rutas de encaminamiento(figura 5.14b), ya sea en el mejor o peor de los casos (figuras 5.14c y 5.14d).

En cambio, se puede observar que OLSRv2 es el segundo protocolo con la peor estabilidad siendo B.A.T.M.A.N. el que peor estabilidad parece tener, ya que su tiempo total desconectado es superior al resto. También es el protocolo que más tiempo está desconectado de media, lo que indica que es el que más tarda en recalcular las rutas, aunque en el peor de los casos no es

el que más tiempo tarda, en el mejor de los casos sí.

B.A.T.M.A.N. es el protocolo menos estable de todos y el segundo que más tiempo tarda en recalcular las rutas en la tabla de encaminamiento ya que en el peor de los casos es el que más tarda con bastante diferencia frente al resto además de ser el peor en el mejor de los casos que B.A.T.M.A.N. Advanced u OLSR.

OLSR es un protocolo que no es el mejor en estabilidad, ni en velocidad en volver a generar las entradas en la tabla de encaminamiento tanto en el mejor escenario posible como en el peor, pero tampoco es el peor de los protocolos en este escenario cuatro. Se comporta incluso mejor que OLSRv2.

Si atendemos a la cantidad de paquetes que genera cada protocolo, ahora la cantidad de paquetes de B.A.T.M.A.N. Advanced si tiene impacto en la mejora en los tiempos del protocolo como en el el escenario dos (véase sección 5.2), sin embargo la cantidad de paquetes que genera B.A.T.M.A.N. para el poco impacto que tiene en los tiempos de desconexión sigue siendo igual de elevado que en el resto de escenarios ya descritos.

OLSR sigue generando muy poco tráfico en comparación con el resto pero parece que su desempeño es mejor que B.A.T.M.A.N. e incluso que OLSRv2 en este escenario, teniendo este último un mayor impacto en la congestión del tráfico en la red.

5.5. Escenario D

5.5.1. Descripción del escenario

En este escenario todos los nodos cambian de posición a coordenadas aleatorias cada minuto, por tanto, es el escenario donde más se pone a prueba a los protocolos de encaminamiento dinámico ya que al cambiar todos los nodos de posición constantemente, todas las tablas de todos los nodos cambian. En este escenario, el nodo origen del comando *ping* usado para realizar los experimentos es *sta5* y el nodo destino sigue siendo *sta10*.

5.5.2. Resultados del escenario

En la gráfica 5.18a se puede apreciar que B.A.T.M.A.N. Advanced es el protocolo con menos tiempo total seguido de B.A.T.M.A.N. En cambio, OLSR y OLSRv2 son los dos protocolos

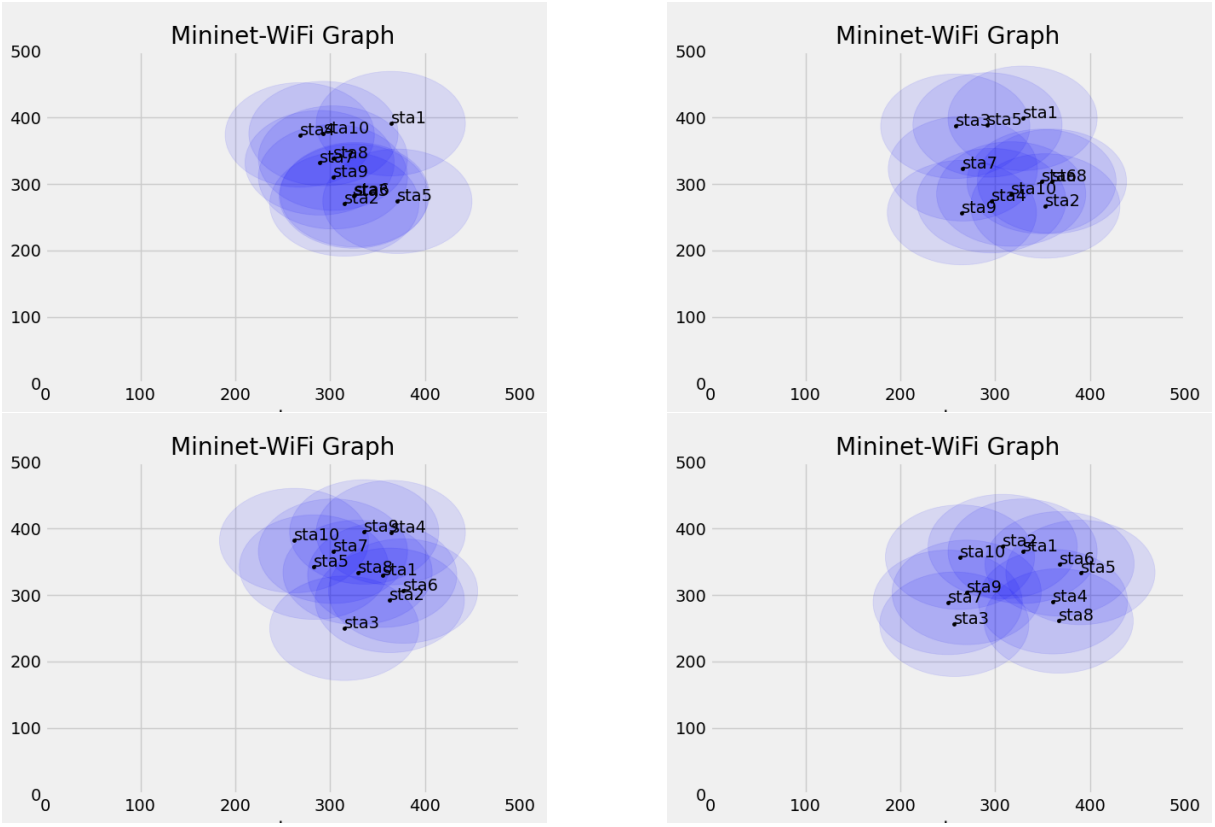


Figura 5.16: Escenario D. Movimiento de los nodos mostrado en varios instantes de tiempo

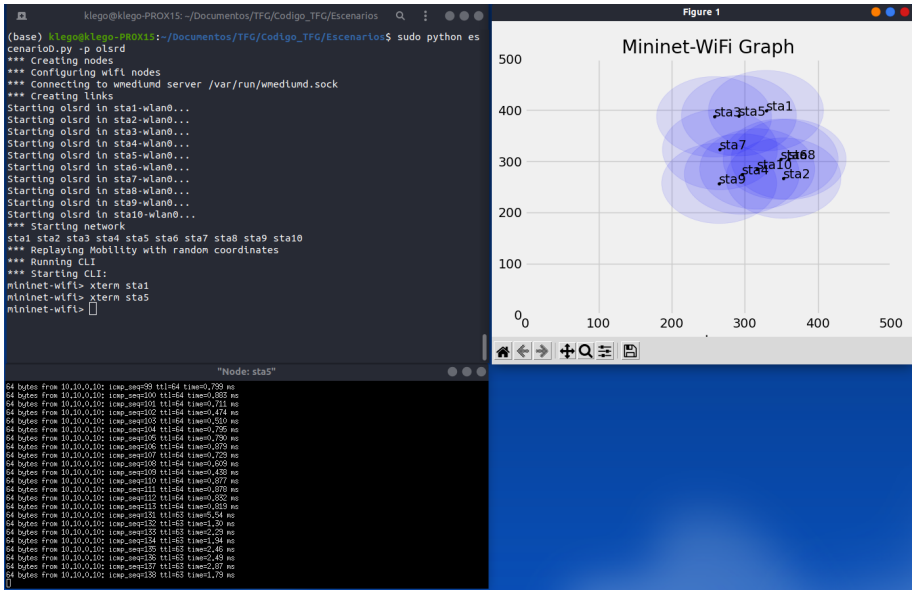
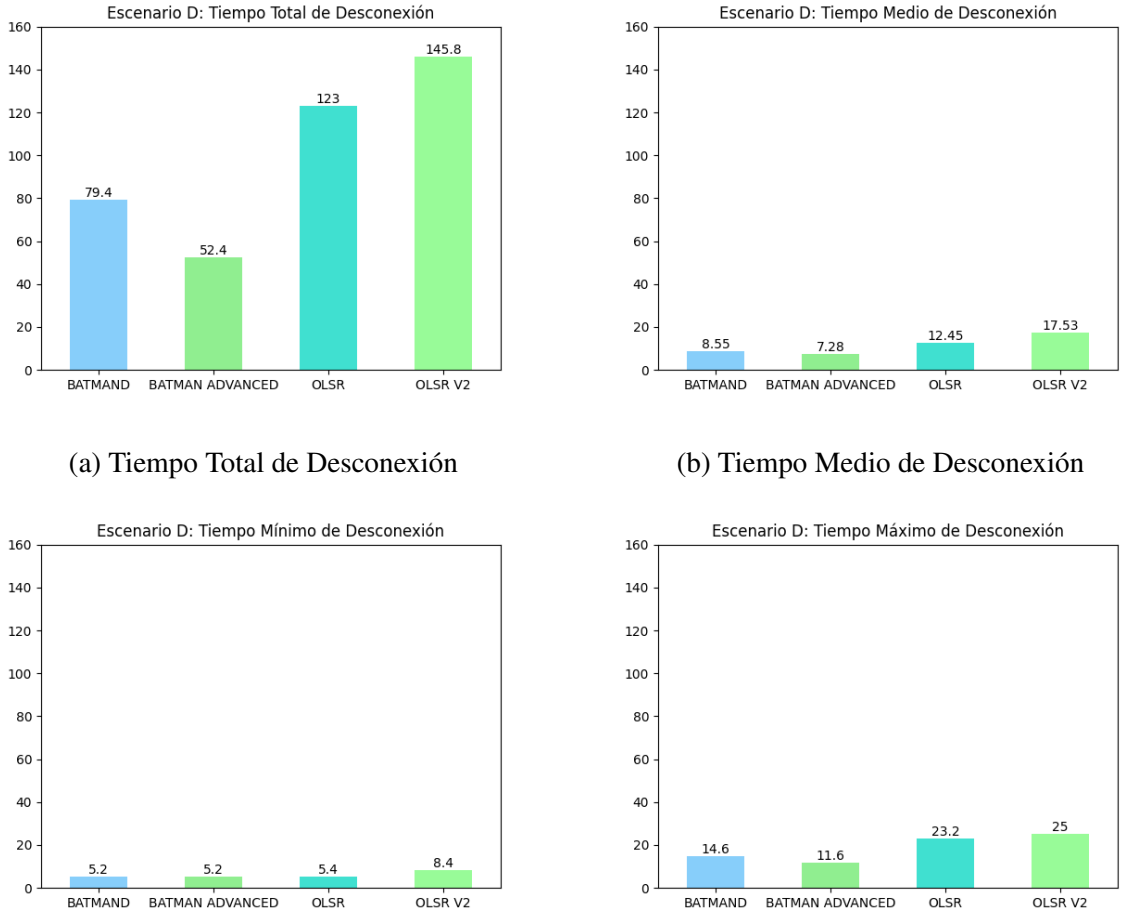


Figura 5.17: Escenario D. Ejecución del escenario en un instante de tiempo

con mayor tiempo, siendo OLSRv2 el que mayor tiempo presenta.

Si se mira la gráfica 5.18b, se observa que nuevamente los procolos B.A.T.M.A.N. y su



(a) Tiempo Total de Desconexión

(b) Tiempo Medio de Desconexión

(c) Tiempo Mínimo de Desconexión

(d) Tiempo Máximo de Desconexión

Figura 5.18: Escenario D. Tiempos de Desconexión

versión Advanced son los que menor tiempo medio de desconexión tienen seguidos nuevamente por OLSR y OLSRv2.

También se puede ver en la gráfica 5.18c que B.A.T.M.A.N. y B.A.T.M.A.N. Advanced tienen el mismo tiempo mínimo y el menor, seguido muy de cerca por OLSR. En cambio OLSRv2 es el que mayor tiempo mínimo presenta.

Además, en la gráfica 5.18d B.A.T.M.A.N. Advanced sigue teniendo el menor tiempo seguido de B.A.T.M.A.N. Los protocolos OLSR y OLSRv2 son los que mayor tiempo máximo de desconexión tienen, siendo OLSRv2 el que más tiene de los dos.

En cuanto a cantidad de paquetes, esta vez el que más paquetes ha generado en el escenario es B.A.T.M.A.N. y el siguiente es B.A.T.M.A.N. Advanced. OLSR vuelve a ser el que menor número de paquetes crea y OLSRv2 está entre B.A.T.M.A.N. Advanced y OLSR en generación

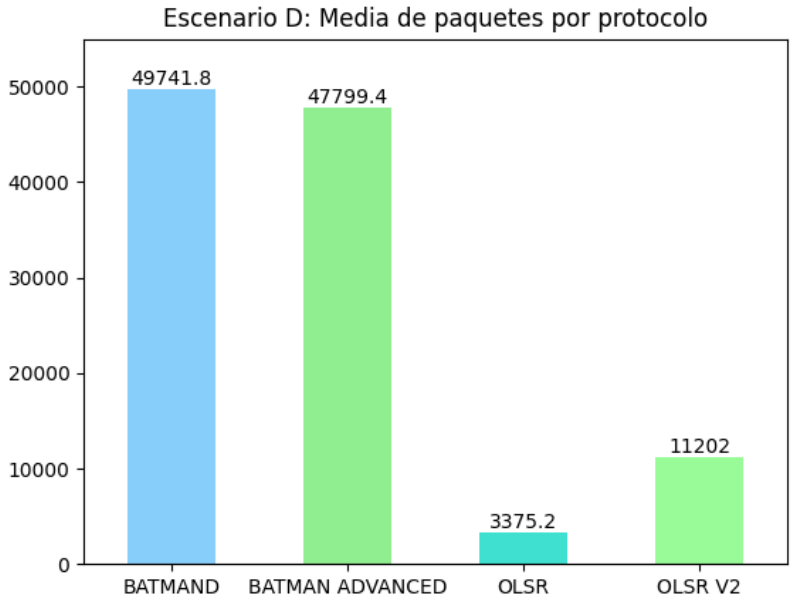


Figura 5.19: Escenario D. Cantidad de paquetes generados por protocolo

de paquetes.

5.5.3. Análisis de los resultados

Se observa que en este escenario donde todos los nodos se mueven, el desempeño de los protocolos B.A.T.M.A.N. y B.A.T.M.A.N. Advanced es muy superior a los protocolos OLSR y OLSRv2. Para empezar, según se puede ver en la gráfica 5.18a, B.A.T.M.A.N. Advanced y B.A.T.M.A.N. son mucho más estables que OLSR y OLSRv2 siendo este último el que menor estabilidad tiene.

Además, a la hora de recalcular las entradas en las tablas de *routing*, al protocolo al que menor tiempo le lleva es B.A.T.M.A.N. Advanced seguido de B.A.T.M.A.N. como se puede apreciar en la figura 5.18b. En cambio OLSR tarda algo más, no obstante, en el mejor de los casos está a la par de estos. Sin embargo, OLSRv2 tarda bastante más, ya sea en el mejor de los casos o en el peor, como demuestran las figuras 5.18c y 5.18d, dando como resultado un tiempo medio de desconexión muy superior a los protocolos B.A.T.M.A.N. y B.A.T.M.A.N. Advanced.

En cuanto a la cantidad de tráfico que generan (véase la figura 5.19), B.A.T.M.A.N. genera más tráfico que su sucesor. Además en este escenario esta cantidad de paquetes sí se ve reflejada

en un mejor rendimiento general. B.A.T.M.A.N. Advanced no es el protocolo que más tráfico genera, aunque por poco. Ademas de no ser el protocolo que más tráfico genera, sigue siendo el mejor protocolo en tiempo de desconexión, de lo que se deduce, que los cambios hechos en el protocolo han dado resultado.

En cambio, en cuanto a OLSR y OLSRv2, OLSR sigue estando en la misma posición generando el menor tráfico y su desempeño no es el peor aunque tampoco el mejor. Sin embargo, OLSRv2 genera más tráfico y se comporta peor en este escenario.

Capítulo 6

Conclusiones

Según el estudio realizado en este Trabajo de Fin de Grado, B.A.T.M.A.N. Advanced parece tener el mejor comportamiento como protocolo de encaminamiento dinámico en MANETs, por su estabilidad y velocidad a la hora de realizar los cambios en las tablas de encaminamiento. Sin embargo, hay que tener en cuenta que tanto B.A.T.M.A.N. como B.A.T.M.A.N. Advanced generan mucho tráfico, lo que es necesario tener presente a la hora de diseñar la red y el hardware de los nodos, y que si no se considera, puede generar excesivo tráfico en la misma. Hay que aclarar que en el estudio de este TFG no se ha tenido en cuenta las necesidades de cómputo requeridas en cada nodo por los protocolos, y por tanto, si se tuviese esto en cuenta estas conclusiones podrían cambiar.

También se ha podido observar que si existe poca movilidad en los nodos en la red, y la cantidad de tráfico de ésta supone un problema, es posible usar OLSR u OLSRv2, que aunque no son tan estables y rápidos a la hora de recalcular las rutas por cambios en la topología, no generan tanta cantidad de tráfico. No obstante, es necesario valorar que si hay normalmente un único camino por el que los paquetes pueden ser encaminados desde un nodo origen al destino, es mejor usar OLSR y no su sucesor.

Resulta significativo que si se produce movilidad constante de los nodos en la red, parece ser lo más recomendable usar B.A.T.M.A.N. Advanced debido a que los protocolos OLSR y OLSRv2 parecen tener bastantes problemas a la hora de tratar con cambios abundantes y muy continuados en la topología de la red.

Solo parece aconsejable el uso de B.A.T.M.A.N. en escenarios donde la cantidad de nodos fuese pequeña debido a su más fácil implementación comparado con B.A.T.M.A.N. Advanced,

y siempre que no fuese necesaria la velocidad de cálculo de rutas de este último, y cuando fuese una red donde todos los nodos se movieran considerablemente.

6.1. Consecución de objetivos

Se han cumplido los objetivos descritos en el capítulo 3: Se ha conseguido estudiar las diferencias entre los protocolos B.A.T.M.A.N, OLSR, OLSRv2 y B.A.T.M.A.N. Advanced.

También se ha estudiado la generación de movilidad determinista en los escenarios controlando el tiempo en que los nodos cambian de posición, y así mismo se ha estudiado la generación de movilidad aleatoria de los nodos con Mininet WiFi.

Además se ha conseguido desarrollar unos scripts que automatizan el análisis de los datos y generación de gráficas para su correspondiente análisis como se ha descrito en el capítulo 5.

6.2. Trabajos futuros

En la realización de este TFG se han encontrado posibles derivados del mismo para la realización de nuevos trabajos en el futuro:

1. Creación de un modelo de movilidad en Mininet WiFi en el que sea posible controlar o la velocidad con la que los nodos se mueven o el tiempo que tardan en cambiar de posición sin el uso de marcas temporales, mediante la modificación de *mobility.py* en la implementación de Mininet WiFi.

2. Estudio de la implicación en el uso de *High Throughput* (opción *ht_cap='HT40+'*) en Mininet WiFi.

3. Generación o eliminación en tiempo real de nodos a la red cuando ya se ha iniciado el escenario en Mininet WiFi.

4. Adición de soporte en Mininet WiFi para otros protocolos de encaminamiento dinámico en MANETs como AODV o ZRP.

Apéndice A

Escenarios de movilidad en Mininet WiFi

A.1. Escenarios de movilidad determinista

Código fuente de los escenarios que usan movilidad determinista mediante marcas temporales.

```
from mininet.log import setLogLevel, info
from mn_wifi.link import wmediumd, adhoc
from mn_wifi.cli import CLI
from mn_wifi.net import Mininet_wifi
from mn_wifi.replaying import ReplayingMobility
from mn_wifi.wmediumdConnector import interference
import sys
import os
import warnings


def topology(args):
        warnings.filterwarnings("ignore")
        net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)
        info("*** Creating nodes\n")
        kwargs = {}
        sta1 = net.addStation('sta1', ip='10.10.0.1/24',
        ↪  position='25,250,0', mac='02:00:00:00:65:01', **kwargs)
        sta2 = net.addStation('sta2', ip='10.10.0.2/24',
        ↪  position='75,250,0', mac='02:00:00:00:65:02', **kwargs)
```

```
19          sta3 = net.addStation('sta3', ip='10.10.0.3/24',
            ↪  position='125,250,0', mac='02:00:00:00:65:03', **kwargs)
20          sta4 = net.addStation('sta4', ip='10.10.0.4/24',
            ↪  position='175,250,0', mac='02:00:00:00:65:04', **kwargs)
21          sta5 = net.addStation('sta5', ip='10.10.0.5/24',
            ↪  position='225,250,0', mac='02:00:00:00:65:05', **kwargs)
22          sta6 = net.addStation('sta6', ip='10.10.0.6/24',
            ↪  position='275,250,0', mac='02:00:00:00:65:06', **kwargs)
23          sta7 = net.addStation('sta7', ip='10.10.0.7/24',
            ↪  position='325,250,0', mac='02:00:00:00:65:07', **kwargs)
24          sta8 = net.addStation('sta8', ip='10.10.0.8/24',
            ↪  position='375,250,0', mac='02:00:00:00:65:08', **kwargs)
25          sta9 = net.addStation('sta9', ip='10.10.0.9/24', position='425,
            ↪  250, 0', mac='02:00:00:00:65:09', **kwargs)
26          sta10 = net.addStation('sta10', ip='10.10.0.10/24',
            ↪  mac='02:00:00:00:65:10', **kwargs)
27
28          net.setPropagationModel(model="logDistance", exp=4)
29
30          info("*** Configuring wifi nodes\n")
31          net.configureWifiNodes()
32
33          info("*** Creating links\n")
34          protocols = ['batman_adv', 'olsrd', 'olsrd2']
35          for proto in args:
36                  if proto in protocols:
37                          kwargs['proto'] = proto
38                  if 'proto' not in kwargs:
39                          info("*INFO: No protocol selected*\n")
40
41          net.addLink(sta1, cls=adhoc, intf='sta1-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
42          net.addLink(sta2, cls=adhoc, intf='sta2-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
43          net.addLink(sta3, cls=adhoc, intf='sta3-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
```

```
44          net.addLink(sta4, cls=adhoc, intf='sta4-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
45          net.addLink(sta5, cls=adhoc, intf='sta5-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
46          net.addLink(sta6, cls=adhoc, intf='sta6-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
47          net.addLink(sta7, cls=adhoc, intf='sta7-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
48          net.addLink(sta8, cls=adhoc, intf='sta8-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
49          net.addLink(sta9, cls=adhoc, intf='sta9-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
50          net.addLink(sta10, cls=adhoc, intf='sta10-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
51
52          net.isReplaying = True
53          path = os.path.dirname(os.path.abspath(__file__)) +
            ↪  '/replayingMobility/escenario2/'
54          get_trace(sta10, '{}node.dat'.format(path), 10)
55
56          if '-p' in args:
57                  net.plotGraph(max_x=500, max_y=500)
58
59          info("*** Starting network\n")
60          net.build()
61
62          info("\n*** Replaying Mobility\n")
63          ReplayingMobility(net)
64
65          info("*** Running CLI\n")
66          CLI(net)
67
68          info("*** Stopping network\n")
69          net.stop()
70
71
72  def get_trace(sta, file_, numeral):
```

```
73          file_ = open(file_, 'r')
74          raw_data = file_.readlines()
75          file_.close()
76          sta.p = []
77          sta.time = []
78          if numeral == 10:
79                  pos = (25, 200, 0)
80                  tim = 30
81          elif numeral == 1:
82                  pos = (25, 250, 0)
83                  tim = 30
84          else:
85                  pos = (0, 0, 0)
86                  tim = 30
87          sta.position = pos
88          sta.time.append(tim)
89          for data in raw_data:
90                  line = data.split()
91                  x = line[0]  # First Column
92                  y = line[1]  # Second Column
93                  t = line[2]  # Third column
94                  pos = float(x), float(y), 0.0
95                  tim = float(t)
96                  sta.p.append(pos)
97                  sta.time.append(tim)
98
99
100 if __name__ == '__main__':
101         setLogLevel('info')
102         topology(sys.argv)
```

A.2. Escenarios de movilidad aleatoria

Código fuente del escenario D que usa la movilidad con marcas temporales y coordenadas aleatorias para emular un movimiento aleatorio de los nodos.

```
from mininet.log import setLogLevel, info
from mn_wifi.link import wmediumd, adhoc
from mn_wifi.cli import CLI
from mn_wifi.net import Mininet_wifi
from mn_wifi.replaying import ReplayingMobility
from mn_wifi.wmediumdConnector import interference
import random
import sys
import os
import warnings


def topology(args):
        seed = 2016
        warnings.filterwarnings("ignore")
        net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)

        info("*** Creating nodes\n")
        kwargs = {}

        sta1 = net.addStation('sta1', ip='10.10.0.1/24',
        ↪  mac='02:00:00:00:65:01', **kwargs)
        sta2 = net.addStation('sta2', ip='10.10.0.2/24',
        ↪  mac='02:00:00:00:65:02', **kwargs)
        sta3 = net.addStation('sta3', ip='10.10.0.3/24',
        ↪  mac='02:00:00:00:65:03', **kwargs)
        sta4 = net.addStation('sta4', ip='10.10.0.4/24',
        ↪  mac='02:00:00:00:65:04', **kwargs)
        sta5 = net.addStation('sta5', ip='10.10.0.5/24',
        ↪  mac='02:00:00:00:65:05', **kwargs)
        sta6 = net.addStation('sta6', ip='10.10.0.6/24',
        ↪  mac='02:00:00:00:65:06', **kwargs)
        sta7 = net.addStation('sta7', ip='10.10.0.7/24',
        ↪  mac='02:00:00:00:65:07', **kwargs)
        sta8 = net.addStation('sta8', ip='10.10.0.8/24',
        ↪  mac='02:00:00:00:65:08', **kwargs)
```

```
29          sta9 = net.addStation('sta9', ip='10.10.0.9/24',
            ↪  mac='02:00:00:00:65:09',  **kwargs)
30          sta10 = net.addStation('sta10', ip='10.10.0.10/24',
            ↪  mac='02:00:00:00:65:10', **kwargs)
31
32          net.setPropagationModel(model="logDistance", exp=4)
33
34          info("*** Configuring wifi nodes\n")
35          net.configureWifiNodes()
36
37          info("*** Creating links\n")
38
39          protocols = ['batman_adv', 'olsrd', 'olsrd2']
40          for proto in args:
41                  if proto in protocols:
42                          kwargs['proto'] = proto
43                  if 'proto' not in kwargs:
44                          info("*INFO: No protocol selected*\n")
45
46          net.addLink(sta1, cls=adhoc, intf='sta1-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
47          net.addLink(sta2, cls=adhoc, intf='sta2-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
48          net.addLink(sta3, cls=adhoc, intf='sta3-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
49          net.addLink(sta4, cls=adhoc, intf='sta4-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
50          net.addLink(sta5, cls=adhoc, intf='sta5-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
51          net.addLink(sta6, cls=adhoc, intf='sta6-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
52          net.addLink(sta7, cls=adhoc, intf='sta7-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
53          net.addLink(sta8, cls=adhoc, intf='sta8-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
54          net.addLink(sta9, cls=adhoc, intf='sta9-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
```

```
55          net.addLink(sta10, cls=adhoc, intf='sta10-wlan0', ssid='adhocNet',
            ↪  mode='g', channel=5, **kwargs)
56
57          net.isReplaying = True
58          path = os.path.dirname(os.path.abspath(__file__)) +
            ↪  '/replayingMobility/escenario5/'
59          random.seed(seed)
60          get_trace(sta1, '{}node_sta1.dat'.format(path))
61          get_trace(sta2, '{}node_sta2.dat'.format(path))
62          get_trace(sta3, '{}node_sta3.dat'.format(path))
63          get_trace(sta4, '{}node_sta4.dat'.format(path))
64          get_trace(sta5, '{}node_sta5.dat'.format(path))
65          get_trace(sta6, '{}node_sta6.dat'.format(path))
66          get_trace(sta7, '{}node_sta7.dat'.format(path))
67          get_trace(sta8, '{}node_sta8.dat'.format(path))
68          get_trace(sta9, '{}node_sta9.dat'.format(path))
69          get_trace(sta10, '{}node_sta10.dat'.format(path))
70
71          if '-p' in args:
72                  net.plotGraph(max_x=500, max_y=500)
73
74          info("*** Starting network\n")
75          net.build()
76
77          info("\n*** Replaying Mobility with random coordinates\n")
78          ReplayingMobility(net)
79
80          info("*** Running CLI\n")
81          CLI(net)
82
83          info("*** Stopping network\n")
84          net.stop()
85
86
87  def first_position():
88          min_x = 250
89          min_y = 250
```

```
90          max_x = 400
91          max_y = 400
92          coord_x = random.randint(min_x, max_x)
93          coord_y = random.randint(min_y, max_y)
94          initial_pos = (coord_x, coord_y, 0)
95
96          return initial_pos
97
98
99  def get_trace(sta, file_):
100         file_ = open(file_, 'r')
101         raw_data = file_.readlines()
102         file_.close()
103         sta.p = []
104         sta.time = []
105         pos = first_position()
106         tim = 30
107         sta.position = pos
108         sta.time.append(tim)
109         for data in raw_data:
110                 line = data.split()
111                 x = line[0]  # First Column
112                 y = line[1]  # Second Column
113                 t = line[2]  # Third column
114                 pos = float(x), float(y), 0.0
115                 tim = float(t)
116                 sta.p.append(pos)
117                 sta.time.append(tim)
118
119
120 if __name__ == '__main__':
121         setLogLevel('info')
122         topology(sys.argv)
```

A.2.1. Generador de coordenadas aleatorias

Script que genera de manera aleatoria las coordenadas en los nodos para el escenario D.

```
import os
import random


def main():
        min_x = 250
        min_y = 250
        max_x = 400
        max_y = 400
        timestamps = [90, 150, 210, 270, 330, 390, 450, 510, 560]
        seed = 2022
        path = os.path.dirname(os.path.abspath(__file__)) +
        ↪  '/replayingMobility/escenario5/'
        random.seed(seed)
        for i in range(1, 11):
                filename = "node_sta" + str(i) + ".dat"
                filepath = f"{path}/{filename}"
                with open(filepath, 'w') as f:
                        for j in range(0, 9):
                                line = str(random.randint(min_x, max_x)) +
                                ↪  " " + str(random.randint(min_y,
                                ↪  max_y)) + " " + str(timestamps[j]) +
                                ↪  "\n"
                                f.write(line)


if __name__ == "__main__":
        main()
```

Apéndice B

Análisis de Datos

B.1. Analizador de Cantidad de Paquetes

Código fuente del script que analiza la cantidad de paquetes generados por cada uno de los protocolos en un escenario.

```
import pandas as pd
import matplotlib.pyplot as plt
import sys
import os
import collections


def get_num_packets(df, protocol):
        protocol_packets = 0
        if protocol == "batman":
                protocol_packets = df.loc[df["Protocol"] == "BAT_BATMAN"]
        elif protocol == "olsr":
                protocol_packets = df.loc[df["Protocol"] == "OLSR v1"]
        elif protocol == "batman_adv":
                protocol_packets = df.loc[(df["Protocol"] ==
                ↪  "BATADV_IV_OGM") | (df["Protocol"] ==
                ↪  "BATADV_UNICAST_TVLV")]
        elif protocol == "olsrv2":
                protocol_packets = df.loc[df["Protocol"] == "packetbb"]

```

```
19          num_packets = len(protocol_packets.index)
20          return num_packets
21
22
23  def calc_average_packets(num_packets):
24          total_packets = [val["Total Protocol Packets"] for key, val in
            ↪  num_packets.items() if "Total Protocol Packets" in val]
25                  avg_packets = (sum(total_packets)) / len(total_packets)
26          return avg_packets
27
28
29  def get_data(path, protocol):
30          id_test = 1
31          data_packet = collections.defaultdict(dict)
32          for packet_file in os.listdir(path):
33                  if packet_file.endswith(".csv"):
34                          filepath = f"{path}/{packet_file}"
35                          df = pd.read_csv(filepath)
36                          data_packet[str(id_test)]["Total Protocol
                            ↪  Packets"] = get_num_packets(df, protocol)
37                          id_test += 1
38
39          avg_packets = calc_average_packets(dict(data_packet))
40          return avg_packets
41
42
43  def main(args):
44          if len(args) != 2:
45                  print("Uso: packet_analyzer.py <número de escenario>")
46                  sys.exit(1)
47          escenario = args[1]
48          path_batmand = os.path.dirname(os.path.abspath(__file__)) +
            ↪  "/BATMAND/" + escenario
49          path_batman_adv = os.path.dirname(os.path.abspath(__file__)) +
            ↪  "/BATMAN_ADV/" + escenario
50          path_olsr = os.path.dirname(os.path.abspath(__file__)) + "/OLSR/"
            ↪  + escenario
```

```
51          path_olsrv2 = os.path.dirname(os.path.abspath(__file__)) +
            ↪ "/OLSRV2/" + escenario
52
53          num_avg_packets_batmand = get_data(path_batmand, "batman")
54          num_avg_packets_batman_adv = get_data(path_batman_adv,
            ↪ "batman_adv")
55          num_avg_packets_olsr = get_data(path_olsr, "olsr")
56          num_avg_packets_olsrv2 = get_data(path_olsrv2, "olsrv2")
57
58          df = pd.DataFrame([num_avg_packets_batmand,
            ↪ num_avg_packets_batman_adv, num_avg_packets_olsr,
            ↪ num_avg_packets_olsrv2], index=['BATMAND', 'BATMAN ADVANCED',
            ↪ 'OLSR', 'OLSR V2'], columns=['Total Packets'])
59          np = df.plot.bar(y="Total Packets", color=["lightskyblue",
            ↪ "lightgreen", "turquoise", "palegreen"], rot=0, ylim=(0,
            ↪ 55000), title="Escenario " + escenario + ": Media de paquetes
            ↪ por protocolo", legend=False)
60          np.bar_label(np.containers[0])
61          plt.show()
62
63
64  if __name__ == '__main__':
65          main(sys.argv)
```

B.2. Analizador de Tiempos de Desconexión

Código fuente del scrip que analiza los tiempos de desconexión de los protocolos en un escenario y genera las gráficas correspondientes.

```
1   import sys
2   import matplotlib.pyplot as plt
3   import os
4   import ping_parser as pp
5   import collections
6   import pandas as pd
7
8
```

```
9  def get_data(path):
10         id_prueba = 1
11         data_protocol = collections.defaultdict(dict)
12         for file in os.listdir(path):
13                 if file.endswith(".txt"):
14                         filepath = f"{path}/{file}"
15                         pp.clean(filepath)
16                         data_read = pp.read_file(filepath)
17                         data = pp.ping_parse(data_read)
18                         data_protocol[str(id_prueba)]["total_time"] =
                           ↪  data[0]
19                         data_protocol[str(id_prueba)]["average_time"] =
                           ↪  data[1]
20                         data_protocol[str(id_prueba)]["min_time"] =
                           ↪  data[2]
21                         data_protocol[str(id_prueba)]["max_time"] =
                           ↪  data[3]
22                         id_prueba += 1
23
24         return dict(data_protocol)
25
26
27 def get_time_values(data_protocol):
28         final_values = []
29         total_time = [val["total_time"] for key, val in
           ↪  data_protocol.items() if "total_time" in val]
30         avg_total = (sum(total_time)) / len(total_time)
31         final_values.append(float("{:.2f}".format(avg_total)))
32
33         avg_time = [val["average_time"] for key, val in
           ↪  data_protocol.items() if "average_time" in val]
34         avg_avg = (sum(avg_time)) / len(avg_time)
35         final_values.append(float("{:.2f}".format(avg_avg)))
36
37         min_time = [val["min_time"] for key, val in data_protocol.items()
           ↪  if "min_time" in val]
38         avg_min = (sum(min_time)) / len(min_time)
```

```
39          final_values.append(float("{:.2f}".format(avg_min)))
40
41          max_time = [val["max_time"] for key, val in data_protocol.items()
            ↪  if "max_time" in val]
42          avg_max = (sum(max_time)) / len(min_time)
43          final_values.append(float("{:.2f}".format(avg_max)))
44
45          return final_values
46
47
48  def main(args):
49          if len(args) != 2:
50                  print("Uso: ping_analyzer.py <número de escenario>")
51                  sys.exit(1)
52          escenario = args[1]
53
54          path_batmand = os.path.dirname(os.path.abspath(__file__)) +
            ↪  "/BATMAND/" + escenario
55          path_batman_adv = os.path.dirname(os.path.abspath(__file__)) +
            ↪  "/BATMAN_ADV/" + escenario
56          path_olsr = os.path.dirname(os.path.abspath(__file__)) + "/OLSR/"
            ↪  + escenario
57          path_olsr2 = os.path.dirname(os.path.abspath(__file__)) +
            ↪  "/OLSRV2/" + escenario
58
59          data_batmand = get_data(path_batmand)
60          final_values_batmand = get_time_values(data_batmand)
61          data_batman_adv = get_data(path_batman_adv)
62          final_values_batman_adv = get_time_values(data_batman_adv)
63          data_olsr = get_data(path_olsr)
64          final_values_olsr = get_time_values(data_olsr)
65          data_olsr2 = get_data(path_olsr2)
66          final_values_olsr2 = get_time_values(data_olsr2)
67
```

```
68          df = pd.DataFrame([final_values_batmand, final_values_batman_adv,
            ↪  final_values_olsr, final_values_olsr2], index=['BATMAND',
            ↪  'BATMAN ADVANCED', 'OLSR', 'OLSR V2'], columns=['Total Time
            ↪  Disconnected', 'Average Time Disconnected', 'Min Time
            ↪  Disconnected', 'Max Time Disconnected'])
69
70          tt = df.plot.bar(y='Total Time Disconnected',
            ↪  color=["lightskyblue", "lightgreen", "turquoise",
            ↪  "palegreen"], rot=0, ylim=(0, 160), title="Escenario " +
            ↪  escenario + ": Tiempo Total de Desconexión", legend=False)
71          tt.bar_label(tt.containers[0])
72
73          at = df.plot.bar(y='Average Time Disconnected',
            ↪  color=["lightskyblue", "lightgreen", "turquoise",
            ↪  "palegreen"], rot=0, ylim=(0, 160), title="Escenario " +
            ↪  escenario + ": Tiempo Medio de Desconexión", legend=False)
74          at.bar_label(at.containers[0])
75
76          mt = df.plot.bar(y='Min Time Disconnected', color=["lightskyblue",
            ↪  "lightgreen", "turquoise", "palegreen"], rot=0, ylim=(0, 160),
            ↪  title="Escenario " + escenario + ": Tiempo Mínimo de
            ↪  Desconexión", legend=False)
77          mt.bar_label(mt.containers[0])
78
79          maxt = df.plot.bar(y='Max Time Disconnected',
            ↪  color=["lightskyblue", "lightgreen", "turquoise",
            ↪  "palegreen"], rot=0, ylim=(0, 160), title="Escenario " +
            ↪  escenario + ": Tiempo Máximo de Desconexión", legend=False)
80          maxt.bar_label(maxt.containers[0])
81
82          allt = df.plot.bar(rot=0, colormap='Accent', ylim=(0, 160),
            ↪  title="Escenario " + escenario + ": Tiempos de Desconexión")
83          plt.show()
84
85
86  if __name__ == '__main__':
87          main(sys.argv)
```

B.2.1. Paquete de funciones auxiliares para filtrar la salida del comando *ping*

Código fuente de las funciones auxiliares que filtran la salida del comando *ping* y que es usado en el script B.2.

```
import re


def clean(file):
    with open(file) as f:
                lines = f.readlines()
        with open(file+"_cleaned", 'w') as write_f:
                for line in lines:
                        if "Destination Host Unreachable" not in line or
                        ↪  "Time to live exceeded" not in line:
                                write_f.write(line)
        write_f.close()
        f.close()


def read_file(file):
        with open(file+"_cleaned") as f:
                data = f.read()

        return data


def ping_parse(data):
        aux = 0
        time = []
        total_t = 0
        elapsed_time = 4
        icmp_seq = re.findall(r'icmp_seq=(\d+).*?', data)

        for seq in icmp_seq:
                if aux + elapsed_time < int(seq):
                        connect_time = int(seq) - aux
```

```
32                          time.append(connect_time)
33                          aux = int(seq)
34                  else:
35                          aux += 1
36
37          for t in time:
38                  total_t += t
39          avg_t = float("{:.2f}".format(total_t/len(time)))
40          min_t = min(time)
41          max_t = max(time)
42          returned_values = [total_t, avg_t, min_t, max_t]
43
44          return returned_values
```

Bibliografía

[1] Bernabe Dorronsoro, Patricia Ruiz, Gregoire Danoy, Yoann Pigne, and Pascal Bouvry. *Introduction to Mobile Ad Hoc Networks*, pages 1–26. Wiley-IEEE Press, 2014.

[2] Ramón Fontes. Github mininet wifi. `https://github.com/intrig-unicamp/mininet-wifi`.

[3] Bob Lantz. Mininet documentation. `http://mininet.org/walkthrough/`.

[4] Ramón Fontes. Web mininet wifi. `https://mininet-wifi.github.io/`.

[5] Universidad Rey Juan Carlos GSyC. *Protocolos de Routing. Routing entre robots móviles*, chapter 3, pages 58–90. GSyC, Universidad Rey Juan Carlos, 2021. `https://gsyc.urjc.es/`.

[6] Axel Neumann y Corinna Aichele y Marek Lindner y Simon Wunderlich. Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.). Internet-Draft draft-wunderlich-openmesh-manet-routing-00, Internet Engineering Task Force, April 2008. `https://datatracker.ietf.org/doc/draft-wunderlich-openmesh-manet-routing/00/`.

[7] Axel Neumann and Corinna Aichele y Marek Lindner y Simon Wunderlich. Open-mesh: B.a.t.m.a.n. advanced documentation, 2013. `https://www.open-mesh.org/projects/batman-adv/wiki/Wiki`.

[8] T. Clausen y P. Jacquet. Optimized link state routing protocol (olsr). RFC 3626, RFC Editor, October 2003. `http://www.rfc-editor.org/rfc/rfc3626.txt`.

[9] T. Clausen y C. Dearlove y P. Jacquet y U. Herberg. The optimized link state routing protocol version 2. RFC 7181, RFC Editor, April 2014. http://www.rfc-editor.org/rfc/rfc7181.txt.

[10] Thomas H. Clausen, Christopher Dearlove, and Justin Dean. Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP). RFC 6130, April 2011.

[11] Cédric Adjih, Christopher Dearlove, Justin Dean, and Thomas H. Clausen. Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format. RFC 5444, February 2009.

[12] Christopher Dearlove and Thomas H. Clausen. Representing Multi-Value Time in Mobile Ad Hoc Networks (MANETs). RFC 5497, March 2009.

[13] Thomas H. Clausen, Christopher Dearlove, and Brian Adamson. Jitter Considerations in Mobile Ad Hoc Networks (MANETs). RFC 5148, February 2008.

[14] Wes McKinney y Chang She. Pandas api reference, 2022. https://pandas.pydata.org/docs/reference/index.html.

[15] Enrique Soriano Salvador y Gorka Guardiola Muzquiz. *Fundamentos de Sistemas Operativos: Una aproximacion practica a Linux*, chapter 2.20, pages 91–93. GSyC, Universidad Rey Juan Carlos, 1.02 edition, octubre 2022. https://honecomp.github.io/.

[16] Ramon Fontes and Christian Rothenberg. *Wireless Network Emulation with Mininet-WiFi*. Christian Esteve Rothenberg, 2019.

[17] Ramon Fontes, Samira Afzal, Samuel Brito, Mateus Santos, and Christian Esteve Rothenberg. Mininet-WiFi: emulating Software-Defined wireless networks. In *2nd International Workshop on Management of SDN and NFV Systems, 2015(ManSDN/NFV 2015)*, Barcelona, Spain, November 2015.

[18] Ramon Fontes. Mininet-wifi's mailing list, 2022. https://groups.google.com/g/mininet-wifi-discuss.

[19] Jorge Luzón López. Protocolos de encaminamiento de redes inalámbricas. Trabajo fin de grado, Universidad Rey Juan Carlos, 2021.