



ugr

Universidad
de Granada

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

**Diseño e implementación de un
laboratorio virtual de redes
inalámbricas**

Autor

Adrián Ripoll Casas

Director

Gabriel Maciá Fernández



Dpto. de Teoría de la Señal, Telemática y Comunicaciones



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, Septiembre de 2016

Diseño e implementación de un laboratorio virtual de redes inalámbricas

Adrián Ripoll Casas

Palabras Clave

Laboratorio, Virtual, Redes Inalámbricas, Máquinas Virtuales, Ad hoc

Resumen

Hoy en día vivimos en una sociedad en la que cada vez existen más dispositivos inalámbricos permanentemente conectados, ya sean ordenadores, teléfonos, dispositivos vestibles, vehículos, etc. Además, cada día que pasa se crean nuevos dispositivos, redes o protocolos de red sobre los que basar las redes. Es por ello que existe la necesidad de probar dichas redes, protocolos relacionados con las mismas o dispositivos.

Para eso, actualmente se cuenta con, o bien simuladores de red cuyo funcionamiento se basa en eventos discretos, o bien entornos reales de experimentación. Sin embargo, dichas herramientas son insuficientes, bien por su alto coste, como es el caso de los entornos reales, o bien por la imposibilidad de trabajar con los dispositivos durante las simulaciones. En este proyecto se pretende diseñar e implementar un laboratorio virtual de redes inalámbricas sobre una red cableada ya existente que aúne las bondades tanto de los simuladores (bajo coste, escalabilidad, velocidad), como de los entornos reales (poder trabajar en tiempo real con los dispositivos a probar).

En primer lugar, se realizará un análisis de los principales factores que deben tenerse en cuenta en la creación de un laboratorio de redes tales como los modelos de movilidad o el direccionamiento IP. Acto seguido se creará un diseño que permita contemplar los factores previamente analizados. Dicho diseño nos permitirá virtualizar una red inalámbrica dentro de una red cableada, ya sea con máquinas virtuales o con ordenadores reales. Una vez finalizado el diseño, se llevará a cabo la implementación de un prototipo de Laboratorio Virtual de Redes Inalámbricas basado en el diseño realizado que resuelva alguno de los factores indicados en el análisis. Dicha implementación se realizará con Máquinas Virtuales con un Sistema Operativo Ubuntu y harán uso de un programa desarrollado con Python que creará una interfaz de red virtual que será la utilizada para comunicarse entre Hosts, los cuales a su vez se conectarán a otra máquina, Controller, que será la encargada de crear y mantener la estructura de la red indicada también haciendo uso de un programa desarrollado en Python, así como de un archivo JSON que contenga la estructura de red deseada. Además, los Hosts harán uso del protocolo AODV, lo que permitirá crear una red Inalámbrica Ad hoc.

A continuación, se evaluará el funcionamiento de la implementación en dos escenarios distintos, uno con solo dos dispositivos que actúen de Hosts y uno de Controller que permita corroborar que se realice correctamente todo el proceso de creación de la red y la comunicación entre dos dispositivos que tengan conectividad

directa. El segundo escenario contendrá cuatro dispositivos Hosts y un Controller y servirá para, en primer lugar, comprobar que si no se hace uso de AODV no exista conectividad entre dispositivos que no deban tenerla y, finalmente, que haciendo uso de AODV existe conectividad entre todos los dispositivos que conforman la red.

Finalmente, gracias al proceso de evaluación, hemos aprendido que, haciendo uso de un solo ordenador, podemos crear nuestro propio Laboratorio Virtual de Redes Inalámbricas con una red de varios dispositivos conectados en una red inalámbrica Ad hoc que hace uso del protocolo AODV. Esto permitirá probar nuevos dispositivos, protocolos o las capacidades de una red con un bajo coste, rápida configuración y toda la libertad que proporciona un entorno de pruebas real.

Design and implementation of a virtual laboratory of wireless networks

Adrián Ripoll Casas

Keywords

Laboratory, Virtual, Wireless Networks, Virtual Machines, Ad hoc

Abstract

We live in a world surrounded by permanently connected wireless devices such as computers, phones, wearables devices, vehicles, etc., which number increases every day. Moreover, each passing day new devices, networks or network protocols on which to base the networks are created. That is why there is a need to test such devices, networks or protocols related to them.

For that purpose, nowadays we have network simulators whose operation is based on discrete events or actual experimental environments. However, these tools are insufficient, due to their high costs, as in the case of actual environments, or by the inability to work with the devices during simulations. This project aims to design and implement a virtual laboratory of wireless networks over an existing wired network that combines the benefits of both simulators (low cost, scalability, speed), as in real environments (being able to work in real time with the testing devices).

First of all, an analysis of the main factors to be taken into account in the creation of a laboratory network models such as mobility and IP routing will be performed. Then a design which would cover the factors previously analyzed will be created. Such design will allow us to virtualize a wireless network into a wired network, either with virtual machines or real computers. Once the design is done, an implementation of a prototype Virtual Laboratory of Wireless Networks will be carried out based on the previous design in order to resolve some of the factors mentioned in the analysis. This implementation will be done with Virtual Machines operating with Ubuntu and will use a program developed with Python that will create a virtual network interface that will be used to communicate between hosts. These hosts will be connected with another machine, the Controller, which will be responsible for creating and maintaining the network structure previously indicated, also using a program developed in Python, as well as a JSON file containing the desired network structure. In addition, Hosts will use the AODV protocol, which will allow to create a Wireless Ad hoc network.

Next, the implementation will be evaluated in two different scenarios to test if it performs correctly. The first scenario will be one with only two devices that act Hosts and one Controller in order to corroborate that the entire process of creating the network and the communication between two devices directly connected functions properly. The second stage will contain four Hosts and a Controller devices and serve to first check if without using AODV there is no connectivity between devices that should not have it and finally making use of AODV there is connectivity between all devices in

the network.

Finally, thanks to the evaluation process, we have learnt that, using a single computer, we can create our own Virtual Lab Wireless Network with a network of multiple devices connected in an Ad hoc wireless network that uses the AODV protocol. This will allow to test new devices, protocols or the capabilities of a network with low cost, quick setup and all the freedom of a real testing environment.

Yo, **Adrián Ripoll Casas**, alumno de la titulación GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76659994M, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Adrián Ripoll Casas

Granada, 5 de Septiembre de 2016.

D. **Gabriel Maciá Fernández** Profesor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como director del Trabajo Fin de Grado de D. Adrián Ripoll Casas

Informa:

Que el presente trabajo, titulado ***Diseño e implementación de un laboratorio virtual de redes inalámbricas***, ha sido realizado bajo su supervisión por **Adrián Ripoll Casas**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 5 de Septiembre de 2016.

El director:

Gabriel Maciá Fernández

Agradecimientos

En primer lugar, quisiera agradecer a mi familia todo lo que han hecho por mí, sin ellos no estaría aquí. A mis padres por apoyarme incluso con las decisiones más difíciles y a mi hermana por aceptarme como compañero de piso.

También me gustaría mencionar a mis compañeros y amigos que han estado conmigo tanto en los buenos momentos como en los malos, disfrutando y sufriendo por igual.

Finalmente, dar las gracias a los profesores y profesoras que me han formado estos años hasta llegar aquí, en especial a mi director de proyecto, Gabriel Maciá, por haberme aguantado pacientemente y ayudado cuando lo he necesitado.

Por ello, gracias a todos de corazón.

Índice General

Capítulo 1 - Introducción	1
1.1 Propósito y objetivos	4
1.2 Metodología	4
1.3 Estructura del documento	5
Capítulo 2 - Planificación	6
2.1 Recursos.....	6
2.1.1 Recursos humanos	6
2.1.2 Recursos hardware	6
2.1.3 Recursos software	7
2.2 Planificación.....	7
2.2.1 Estudio previo	7
2.2.2 Estado del arte	7
2.2.3 Análisis y diseño	7
2.2.4 Implementación	7
2.2.5 Evaluación.....	7
2.2.6 Redacción de la memoria	8
2.3 Costes.....	8
2.3.1 Recursos humanos	8
2.3.2 Recursos tecnológicos	8
2.3.3 Presupuesto	9
Capítulo 3 - Estado del arte	10
3.1 Redes inalámbricas	10
3.1.1 IEEE 802.11	13
3.1.1 IEEE 802.15.....	15
3.2 Redes ad hoc inalámbricas	18
3.2.1 AODV	24
3.3 Tunneling	30
3.4 Virtualización.....	31
Capítulo 4 - Análisis.....	35
4.1 Modelos de movilidad	35
4.2 Transmisión y propagación	39
4.3 Direccionamiento IP	40

4.4 Cobertura, interferencias y ruido	40
Capítulo 5 - Diseño	43
5.1 Hosts	44
5.2 Controller	44
5.3 Conexiones.....	44
5.4 Funcionamiento	45
Capítulo 6 - Implementación	47
6.1 Hosts	48
6.2 Controller	51
6.3 Instalación.....	55
Capítulo 7 - Evaluación	56
7.1 Conectividad entre 2 Hosts	56
7.2 Conectividad entre 4 Hosts	61
Capítulo 8 – Conclusiones.....	68
8.1 Conclusiones del trabajo.....	68
8.2 Futuras ampliaciones	69
Referencias.....	71

Índice de figuras

Figura 1.1. Crecimiento global de dispositivos inalámbricos. Fuente: Cisco VNI Mobile, 2016	1
Figura 1.2. Número de apps disponibles en cada tienda de aplicaciones. Fuente: Statista, 2015	1
Figura 1.3. Número medio de aplicaciones y tiempo invertidos por persona por mes. Fuente: The Nielsen Company, 2015	2
Figura 1.4. Red vehicular.....	3
Figura 2.2.1. Diagrama de Gantt con la distribución temporal del proyecto	7
Figura 4.1.1. Patrón de movimiento de un nodo móvil utilizando random walk model. Fuente: King Fahd University of Petroleum & Minerals [10]	35
Figura 4.1.2. Patrón de movimiento de un nodo móvil utilizando random waypoint model. Fuente: King Fahd University of Petroleum & Minerals [10]	36
Figura 4.1.3. Patrón de movimiento de un nodo móvil utilizando random direction model. Fuente: King Fahd University of Petroleum & Minerals [10]	36
Figura 4.1.4. Patrón de movimiento de un nodo móvil utilizando Gauss-Markov mobility model. Fuente: King Fahd University of Petroleum & Minerals [10]	37
Figura 4.1.5. Patrón de movimiento de un grupo de tres nodos móviles utilizando Reference Point Group mobility model. Fuente: King Fahd University of Petroleum & Minerals [11]	38
Figura 4.1.6. Patrón de movimiento de un grupo de seis nodos móviles utilizando Pursue mobility model. Fuente: King Fahd University of Petroleum & Minerals [11]	38
Figura 4.1.7. Patrón de movimiento de un grupo de siete nodos móviles utilizando nomadic community mobility model. Fuente: King Fahd University of Petroleum & Minerals [11]	39
Figura 5.1. Esquema del Laboratorio	43
Figura 5.4.1 Diagrama de flujo de los hosts.....	45
Figura 5.4.2 Diagrama de flujo del controller	46
Figura 6.1. Estructura de red	47
Figura 7.1.1 Escenario Evaluación 1.....	56
Figura 7.1.2 Consola host 1.....	57
Figura 7.1.3 Consola host 2.....	57
Figura 7.1.4 Consola controller	57
Figura 7.1.5 Intercambio magic word entre controller y host 1.....	58
Figura 7.1.6 Envío de dirección IP y asignación a tun0.....	58
Figura 7.1.7 Ping de host 2 a host 1 (consola)	59
Figura 7.1.8 Ping de host 2 a host 1 (wireshark).....	59
Figura 7.1.9 Traceroute de host 1 a host 2	59
Figura 7.1.10 Ping request host 2 a host 1 visto desde el controller	60
Figura 7.1.11 Ping reply host 1 a host 2 visto desde el controller.....	60
Figura 7.2.1 Escenario evaluación 2.....	61
Figura 7.2.2 Archivo sysctl.conf	62
Figura 7.2.3 Pings host 1 (sin AODV).....	63
Figura 7.2.4 Pings host 2 (sin AODV).....	63
Figura 7.2.5 Pings host 3 (sin AODV).....	64

Figura 7.2.6 Pings host 4 (sin AODV).....	64
Figura 7 2.7 Ping de host 2 a host 3 (Controller)	65
Figura 7.2.8 Ping de host 2 a host 3 (Host 1)	65
Figura 7.2.9 AODV host 1.....	65
Figura 7.2.10 AODV host 1	66
Figura 7.2.11 Ping de host 1 a host 4.....	66
Figura 7.2.12 Tabla de routing de AODV host 1	66
Figura 7.2.13 Tabla de routing de AODV host 2	67
Figura 8.2.1 MANETLab.....	69

Capítulo 1 - Introducción

En la actualidad el número de dispositivos inalámbricos crece constantemente. De acuerdo a un estudio realizado por Cisco, entre 2015 y 2020 se producirá un aumento a razón del 8% de nuevos dispositivos cada año [\[1\]](#).

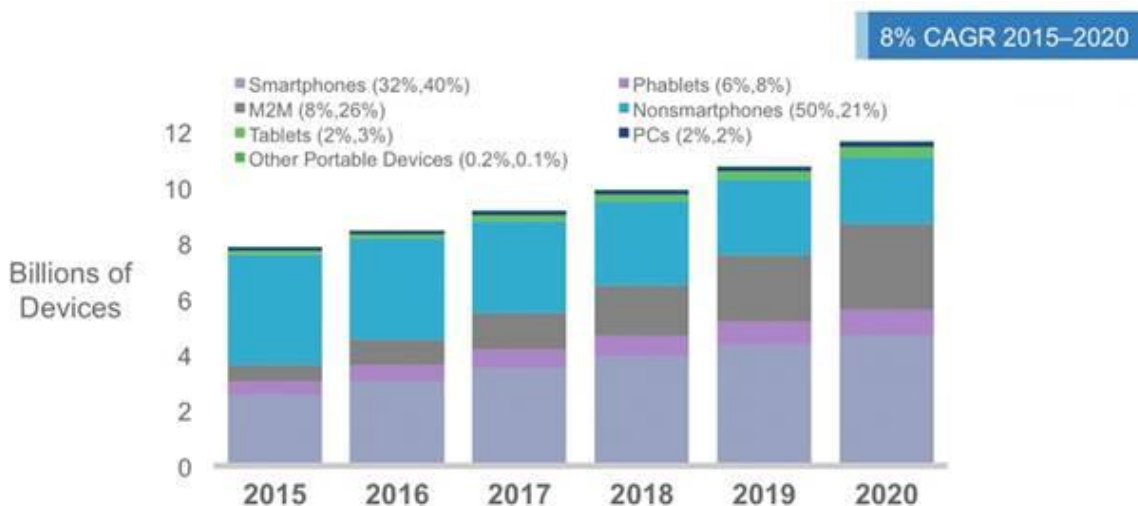
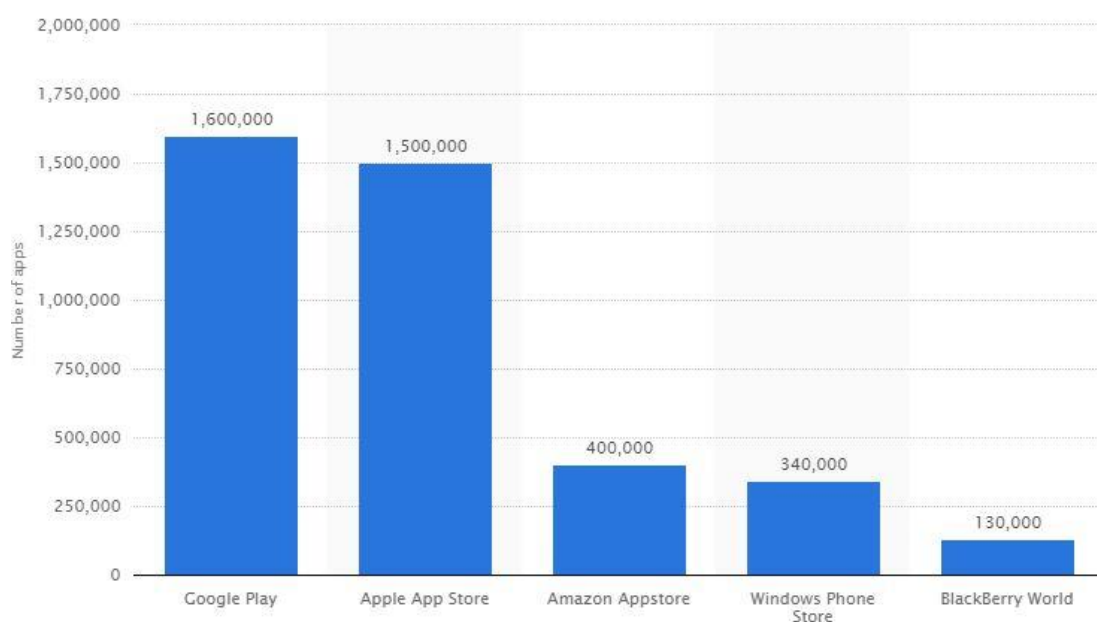


Figura 1.1. Crecimiento global de dispositivos inalámbricos. Fuente: Cisco VNI Mobile, 2016

Los números entre paréntesis indican los datos de 2015 y 2020.

Además, cada uno de estos dispositivos puede tener más de un servicio que realice conexiones, por ejemplo, tomando como referencia los smartphones, en la siguiente figura podemos apreciar la cantidad de aplicaciones disponibles para cada S.O:



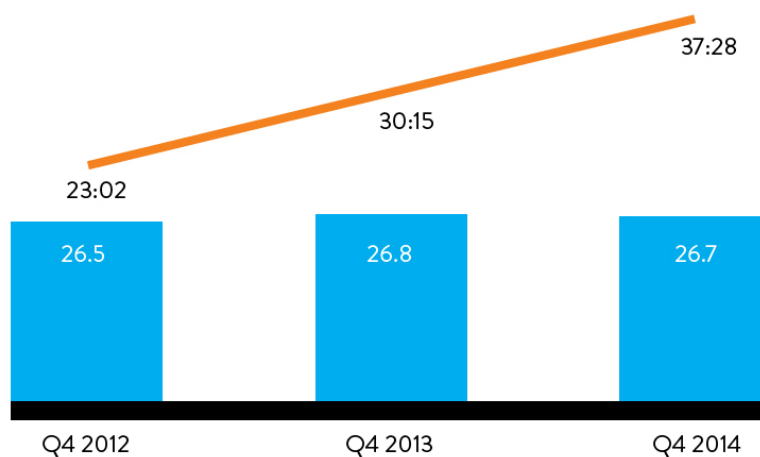
© Statista 2015

Figura 1.2. Número de apps disponibles en cada tienda de aplicaciones. Fuente: Statista, 2015

Si bien no todas las aplicaciones disponibles necesitarán de una conexión para funcionar, podemos asumir que un número significativo de ellas lo hará. Por otro lado, los smartphones son cada vez más relevantes en la vida cotidiana de las personas y ocupan cada vez más de su tiempo, por lo que es imprescindible que las redes asociadas a los mismos funcionen sin problemas:

AVERAGE NUMBER OF APPS USED AND TIME PER PERSON PER MONTH

n



Read As: In Q4 2014, smartphone users accessed 26.7 apps per month on average, and spent 37 hours and 28 minutes per month on apps.

Source: Nielsen

Copyright © 2015 The Nielsen Company

Figura 1.3. Número medio de aplicaciones y tiempo invertidos por persona por mes. Fuente: The Nielsen Company, 2015

Otro ejemplo de comunicación inalámbrica en auge son las redes vehiculares (VANET):

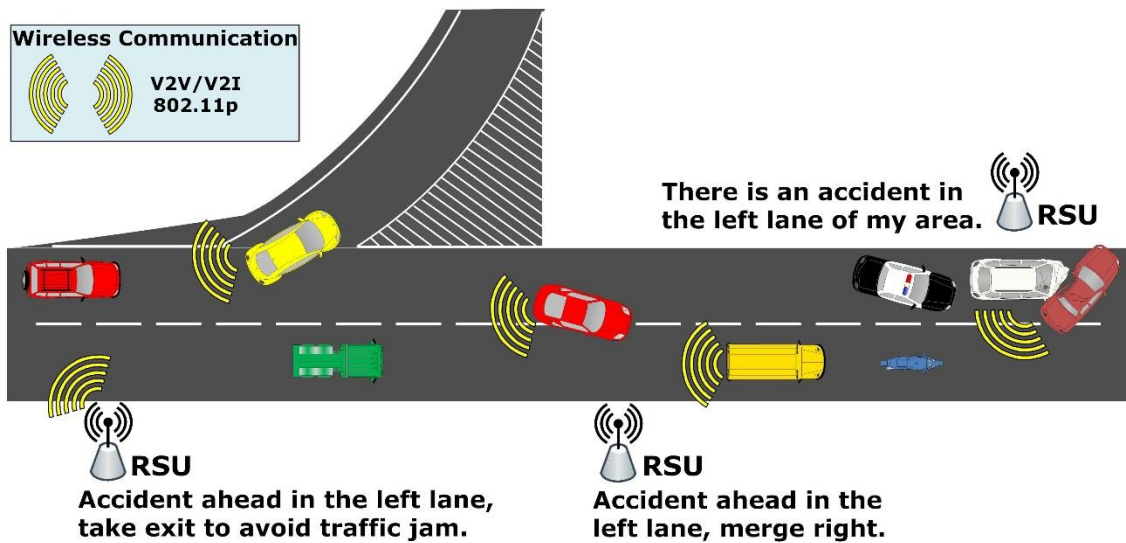


Figura 1.4. Red vehicular

En estas redes los vehículos envían información relevante para el conductor de un vehículo a otro. Dada la naturaleza de la información compartida, es necesario que la red y los protocolos que soportan la comunicación sean rápidos y a prueba de fallos.

Todos estos requisitos llevan a la necesidad de probar las redes, protocolos o dispositivos que las forman antes de su llegada al público. Para ello, existen actualmente dos métodos principales:

- Entornos reales en los que se cuenta con los dispositivos que se desean probar o la implementación de nuevos protocolos. Estos entornos permiten trabajar directamente con el producto final y los resultados que obtienen son los mismos que se obtendrían una vez el producto llegue al mercado. Sin embargo, este tipo de entornos acarrea una serie de problemas como pueden ser el alto coste que conlleva adquirir cada uno de los dispositivos necesarios para la prueba, factores externos que pueden afectar a las pruebas como la propagación multirayectoria, analizar el seguimiento de la trayectoria de los mensajes o problemas de área asociados a los distintos radios de cobertura de los dispositivos, que puede variar desde pocos metros (tecnología Bluetooth o Wi-Fi) hasta varios kilómetros (antenas de telefonía)
- Un simulador de redes es un software que predice el comportamiento de una red de dispositivos. Los principales usos de un simulador de redes son:
 - Validación de redes para empresas, data centers, redes de sensores, etc.
 - Estudio del impacto de modificaciones o adiciones a una red existente.
 - Testeo de nuevos protocolos de red

Como mínimo, un simulador debe permitir al usuario lo siguiente:

- Modelar la topología de red especificando los nodos y enlaces de dicha red.

- Modelar el tráfico entre nodos
- Proporcionar medidas de la actuación de la red
- Visualizar el flujo de paquetes
- Llevar un registro de paquetes/eventos para su posterior análisis

Ejemplos de distintos tipos de simuladores son:

- OMNeT++: Es un simulador de eventos discretos extensible, modular, basado en C++. Open source [\[2\]](#)
- ns-3: Es un simulador de eventos discretos, diseñado principalmente para investigación y educación. Software libre [\[3\]](#)
- NetSim: Simulador perteneciente a Cisco, es una aplicación que permite simular redes basadas en Cisco. Software propietario [\[4\]](#)
- Riverbed Modeler: Anteriormente denominado OPNET, es una solución de gestión del rendimiento de aplicaciones que tiene integradas la monitorización de experiencia del usuario final, la captura de transacciones de alta definición y analíticas avanzadas. Software propietario [\[5\]](#)

1.1 Propósito y objetivos

La gran importancia de las redes inalámbricas en nuestra sociedad, junto con los inconvenientes de los métodos de prueba citados anteriormente lleva al desarrollo de este Trabajo.

El propósito del mismo es la creación de una solución a medio camino entre los entornos reales y los simuladores que aúne las ventajas de ambos, tales como la escalabilidad, el bajo coste o la posibilidad de manipular los elementos involucrados en el experimento en tiempo real, eliminando a su vez los problemas de radio de cobertura, factores externos o la necesidad de adquirir cada uno de los dispositivos.

Para ello, se espera completar los siguientes objetivos. En primer lugar, la realización de un diseño genérico que permita crear laboratorios virtuales de redes inalámbricas adaptados a necesidades específicas. Una vez conseguido, el siguiente objetivo a completar será la implementación de un primer prototipo de laboratorio virtual de redes inalámbricas ad hoc de acuerdo al diseño realizado.

1.2 Metodología

- Para ello, se espera completar los siguientes objetivos. En primer lugar, la Estudio del estado del arte y aspectos a tener en cuenta para este trabajo.
- Análisis de la problemática asociada a las redes inalámbricas.
- Diseño e Implementación de una solución al problema planteado.
- Evaluación de la implementación y análisis de los resultados.

1.3 Estructura del documento

El documento está organizado en siete capítulos, siendo el primero esta introducción, y el resto:

- **Capítulo 2:** En este capítulo se muestran los recursos utilizados en el trabajo, así como la planificación del mismo y una estimación de costes.
- **Capítulo 3:** En este capítulo se relata el estado del arte relacionado con este trabajo, específicamente, se tratarán aspectos como las redes inalámbricas, las redes ad hoc inalámbricas, el tunneling y la virtualización.
- **Capítulo 4:** Este capítulo se centra en el análisis de los factores a tener en cuenta a la hora de realizar un laboratorio virtual de redes inalámbricas.
- **Capítulo 5:** En este capítulo se explica el diseño realizado para solucionar los problemas enumerados en el capítulo anterior.
- **Capítulo 6:** En este capítulo se detalla un prototipo del diseño anterior realizado como implementación para este trabajo.
- **Capítulo 7:** En este capítulo se llevan a cabo una serie de experimentos con el fin de comprobar el correcto funcionamiento de la implementación realizada.
- **Capítulo 8:** Este capítulo recoge las conclusiones del trabajo, así como un estudio de futuras ampliaciones para el mismo.

Capítulo 2 - Planificación

En este capítulo se presenta un cálculo de recursos y costes involucrados en el trabajo, así como la planificación del proyecto, desglosado en diferentes fases.

2.1 Recursos

En la realización del trabajo interviene una serie de recursos que podemos dividir en recursos humanos, recursos hardware y recursos software.

2.1.1 Recursos humanos

En este trabajo intervienen las siguientes personas:

- D. Gabriel Maciá Fernández, profesor titular del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada. Pertenece al grupo de investigación Network Security and Engineering Group (NESG). Es ingeniero de telecomunicación por la Universidad de Sevilla, en la que obtuvo su graduación en el año 1998. Posteriormente, su experiencia profesional se desarrolló en el ámbito empresarial durante 7 años, en los que ha trabajado en departamentos técnico-tecnológicos de empresas del ámbito de la energía (Enditel – Endesa) y de las telecomunicaciones (Vodafone S.A.), en la que trabajó en el departamento de O&M conmutación de red troncal y red de acceso. Se incorporó a la Universidad de Granada en el año 2005, y obtuvo su grado de doctor en 2007, con una tesis doctoral titulada “Ataques de denegación de servicio a baja tasa contra servidores”, la cual ha obtenido el premio LA CAIXA a la mejor tesis doctoral en comercio electrónico, en la XXVIII convocatoria de premios del colegio de ingenieros de telecomunicación (COIT). Actualmente, su interés investigador está centrado en el área de la seguridad en comunicaciones, con énfasis en la generación de ataques, la denegación de servicio, el diseño de protocolos robustos y los sistemas de detección de intrusiones. En este trabajo desempeña el rol de Director.
- Adrián Ripoll Casas, alumno de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada. Autor del presente trabajo.

2.1.2 Recursos hardware

En este trabajo se hace uso de los siguientes dispositivos:

- 1x PC portátil Asus G750JX-T4259H Intel Core i7-4700HQ (2.4 GHz, 6 MB), 16 GB RAM DDR3.

2.1.3 Recursos software

En este trabajo se combina el uso de recursos open source con software propietario:

- Sistemas Operativos: Windows 10, Ubuntu 10.04.
- Ofimática y gestión de proyectos: Office 365 Universitarios, Gantt Project, SmartGit.
- Diseño gráfico: Draw.io
- Entorno de desarrollo: Sublime Text 3.
- Otras herramientas: Oracle VM VirtualBox, Github, Wireshark.

2.2 Planificación

Este trabajo se puede dividir en 6 bloques que se pueden observar en el siguiente diagrama de Gantt:

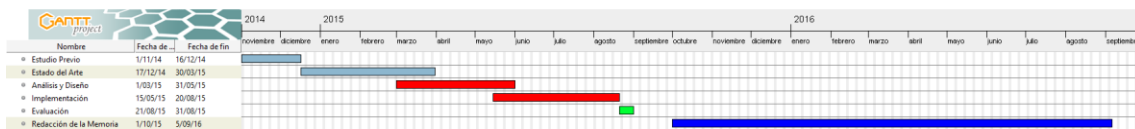


Figura 2.2.1. Diagrama de Gantt con la distribución temporal del proyecto

2.2.1 Estudio Previo

Durante este bloque se realiza un estudio del lenguaje de programación Python, así como de las diferentes configuraciones para el correcto funcionamiento de máquinas virtuales.

2.2.2 Estado del arte

En este bloque se lleva a cabo un estudio de las tecnologías y posibles soluciones existentes, total o parciales, relacionadas con el ámbito del trabajo.

2.2.3 Análisis y Diseño

En este bloque se estudia la problemática asociada al trabajo y se procede a realizar un diseño que satisfaga los objetivos del proyecto teniendo en cuenta dicha problemática.

2.2.4 Implementación

En este bloque se implementa una solución basada en el diseño anterior.

2.2.5 Evaluación

Una vez terminada la implementación, se procede a evaluar su funcionamiento.

2.2.6 Redacción de la memoria

La redacción de la memoria se planificó originalmente para realizarse en paralelo al resto de bloques, sin embargo, finalmente se llevó a cabo una vez se acabó el resto de bloques.

2.3 Costes

A continuación, se lleva a cabo una estimación de los costes del trabajo, según los recursos explotados durante su elaboración y un presupuesto final.

2.3.1 Recursos humanos

Haciendo uso del diagrama de Gantt se puede extraer el coste temporal de cada una de las fases del trabajo. Para su presupuesto, se aplica una media de 15€/hora, asumiendo el mismo coste para las diferentes tareas, y sin contar fines de semana o festivos. Además, se establece una media de 2 hora de trabajo por día. La elaboración de la memoria no se tiene en cuenta a la hora de hacer los cálculos. Por tanto, el presupuesto para los recursos humanos es:

Bloque	Extensión (horas)	Coste (euros)
Estudio previo	92	1380,00
Estado del arte	178	2385,00
Análisis y diseño	184	2760,00
Implementación	196	2940,00
Evaluación	22	330,00
TOTAL	672	10080,00

En total se han dedicado 672 horas, que equivalen a un trabajo a jornada completa (8 horas) 5 días a la semana durante 4 meses y 1 semana.

2.3.2 Recursos tecnológicos

Se incluyen los recursos hardware y los recursos software, si bien la gran mayoría de los recursos software son con licencia gratuita u open source:

Recurso	Coste (euros)
PC portátil Asus G750JX-T4259H	1689,00
Windows 10	0,00 (incluido con el ordenador)
Ubuntu 10.04	0,00
Office 365 Universitarios	79,00
Gantt Project	0,00
SmartGit	0,00
Draw.io	0,00
Sublime Text 3	0,00
Oracle VM VirtualBox	0,00
Github	0,00
Wireshark	0,00
TOTAL	1768,00

2.3.3 Presupuesto

Finalmente, se presenta el presupuesto final resultado de la suma de los recursos humanos y los recursos tecnológicos:

Concepto	Coste (euros)
Recursos humanos	10080,00
Recursos tecnológicos	1768,00
TOTAL	11848

Capítulo 3 - Estado del arte

En este capítulo se relatan diversos aspectos previos relacionados con el Proyecto que se deben conocer. En primer lugar, se describirán las redes inalámbricas y las redes ad hoc inalámbricas. Acto seguido se explicará en qué consiste el tunneling. Finalmente, se hablará sobre la virtualización, máquinas virtuales y herramientas de simulación.

3.1 Redes inalámbricas

Las redes inalámbricas son aquellas en las que la conexión entre nodos se realiza por medio de ondas electromagnéticas, sin necesidad de cables. Este tipo de redes permite reducir costes al no necesitar cableado Ethernet ni conexiones físicas entre nodos, sin embargo, necesita una seguridad más exigente y robusta contra intrusos. Existen diversos métodos de clasificación para las redes inalámbricas:

- Clasificación en función de la infraestructura: Permite clasificar las redes en dos bloques:
 - Basadas en infraestructura: Redes basadas en estaciones base y puntos de acceso. Generalmente, los paquetes realizan un solo salto entre la estación base y el punto de acceso. Ejemplos: redes celulares, redes WLAN.
 - Redes Ad hoc: Redes basadas en nodos remotos capaces de coordinarse para formar una red. Generalmente, los paquetes pueden realizar múltiples saltos para llegar a su destino. Ejemplos: redes de sensores, redes vehiculares, redes WLAN Ad hoc.
- Clasificación en función de la movilidad y la tecnología: Permite clasificar las redes en dos bloques:
 - Sistemas fijos: Pueden ser terrestres de banda ancha (Sistemas RLL) o estrecha (MMDS, LMDS) o vía satélite (difusión de TV).
 - Sistemas de comunicaciones móviles: Pueden ser terrestres (1G, 2G, 3G, 4G) o vía satélite.
- Clasificación en función del área de cobertura: Permiten clasificar las redes en dos bloques:
 - De corta distancia: Utilizadas en redes corporativas cuyas oficinas se encuentran en uno o varios edificios no muy alejados entre sí. Ejemplos: LMDS, MMDS, Wi-Fi.
 - De larga distancia: Utilizados para transmitir información en espacios que pueden variar desde una ciudad hasta varios países. Ejemplos: VSAT, telefonía móvil.
- Clasificación en función del número de usuarios: Permiten clasificar las redes en cuatro bloques:
 - Redes inalámbricas de área personal (WPAN): Son redes formadas por la interconexión de dispositivos centrados en el espacio de trabajo de un individuo (impresoras, ordenadores, teléfonos móviles, puntos de acceso). Se basan en el estándar IEEE 802.15
 - Redes inalámbricas de área local (WLAN): Redes formadas por dos o más dispositivos conectados de forma inalámbrica en un área limitada

como un hogar, escuela, laboratorio u oficinas. Funcionan como alternativa o extensión de las Redes de Área Local cableadas. Se basan en el estándar IEEE 801.11

- Redes inalámbricas de área metropolitana (WMAN): Redes de alta velocidad formadas por varios dispositivos conectados en un área de varios edificios o una ciudad completa. Se basan en el estándar IEEE 802.16
- Redes inalámbricas de área extensa (WWAN): Aquellas redes de dispositivos que abarcan varias ubicaciones físicas proveyendo servicio a una zona, país o incluso varios continentes. Se basan en GSM, GPRS, UMTS, HSPA, LTE, LTE-Advanced, Satélite.

Existen diversas funciones imprescindibles para el correcto funcionamiento de una red inalámbrica que podemos organizar dentro del marco del modelo OSI:

- Capa 2: Control de acceso al medio.
- Capa 3: Routing.
- Capa 4: Transporte.

Control de Acceso al Medio

La subcapa de control de acceso al medio proporciona direccionamiento y mecanismos de control de acceso al canal que hacen posible para varios dispositivos comunicarse mediante un único medio compartido. Para ello, se hace uso de distintos protocolos de acceso múltiple, los más destacados para redes inalámbricas son:

- Carrier sense multiple access with collision avoidance (CSMA/CA): Se utiliza principalmente en 802.11. Antes de empezar la transmisión, cada equipo anuncia su intención de transmitir para evitar colisiones entre los datos. De esta forma, el resto de equipos sabrán cuando hay colisiones y, en vez de transmitir cuando el medio quede libre, esperarán un tiempo aleatorio. Una ventaja de CSMA/CA, es que soluciona el problema del nodo oculto, es decir, una estación que quiere transmitir cree que el canal está libre, aunque en realidad está ocupado por otro nodo al que no escucha, para ello, se hará uso del intercambio de paquetes request to send (RTS) y clear to send (CTS).
- Slotted ALOHA: Una mejora del ALOHA original, divide el tiempo en intervalos correspondientes a una trama. Solo se puede iniciar una transmisión al principio del intervalo, por tanto, cada trama solo podrá colisionar durante el tiempo de trama.
- Mobile slotted ALOHA (MS-ALOHA): Utilizado principalmente en redes vehiculares, es un protocolo basado en conexión, aunque es reactivo a cambios en la topología y no incluye reservas que no sean para intercambio de datos.

Routing

El routing es una de las funciones más importantes en el diseño de una red, puesto que su deber es aceptar la información de una fuente y enviarla hasta su destino. Para ello, es necesario determinar la ruta a seguir, teniendo en cuenta diversos requisitos:

- Exactitud.
- Imparcialidad.
- Corrección.

- Simplicidad.
- Robustez.
- Estabilidad.
- Optimización.
- Eficiencia.

Podemos diferenciar los distintos algoritmos de routing clasificándolos en dos grandes bloques:

- Algoritmos estáticos y de mínimo coste: Eligen una ruta permanente para cada par de nodos origen-destino, dicha elección se basa en algoritmos de mínimo coste y no tiene en cuenta variables dinámicas. Se implementan mediante una matriz de encaminamiento central, que se almacena en un centro de control de la red y contiene el camino entre cada pareja de nodos origen-destino, sin embargo, con el fin de optimizar, no es necesario almacenar la ruta completa, solo el primer nodo de la ruta. Existen varios algoritmos de cómputo de coste, por ejemplo, Dijkstra o Bellman-Ford.
- Algoritmos adaptables: Las tablas de routing se actualizan de forma periódica y automática, intercambiando información de estado entre los nodos de la red. A su vez, se pueden dividir en tres tipos en función del lugar de actualización:
 - Centralizados: Se actualizan en un único nodo central que obtiene las rutas óptimas y posteriormente se distribuye a los nodos la información que necesiten. Su mayor desventaja es que genera alta carga y es lento. Un algoritmo adaptable centralizado es el encaminamiento basado en flujo.
 - Aislados: La decisión de rutas se realiza localmente. Cada nodo genera sus propias tablas considerando sólo información local. Son rápidos y no generan carga en la red, sin embargo, sus decisiones suelen ser poco adecuadas. Un ejemplo puede ser las inundaciones, en las que todos los paquetes se envían a todas las líneas excepto al origen, sin embargo, esto puede dar lugar a bucles, por lo que se utiliza un contador de saltos que es decrementado en cada nodo y al llegar a 0 deja de retransmitir el paquete.
 - Distribuidos: Cada nodo construye sus propias tablas usando información procedente de nodos cercanos. Consiguen rutas mejores que los algoritmos aislados y más rápido que los centralizados, sin embargo, los cambios se propagan lentamente. Un ejemplo es el protocolo routing information protocol (RIP), utilizado en redes IP, este protocolo hace uso del vector distancia. Se supone un conocimiento del coste de transmisión de cada nodo a sus vecinos. Periódicamente hay un intercambio de tablas entre los nodos. Haciendo uso de esta información, se estiman los costes y se actualizan las tablas de acuerdo al mínimo coste.
 - Jerárquicos: Divide la red en regiones. Cada nodo conoce todos los detalles de su región, pero nada de las restantes. Su mayor inconveniente es que puede desembocar en un aumento de la longitud de algunos caminos.

Transporte

Los protocolos de transporte se encargan de la comunicación extremo a extremo y de la multiplexación y demultiplexación de aplicaciones y puertos. Existen dos grandes protocolos de transporte que son los más usados:

- User datagram protocol (UDP): Tiene una funcionalidad best-effort, es decir, proporciona un servicio no orientado a conexión, no fiable, sin garantías de entrega ordenada y sin control de congestión. Se utiliza frecuentemente para aplicaciones multimedia, por ser tolerantes a fallos y sensibles a retardos.
- Transmission control protocol (TCP): Ofrece un servicio punto a punto, orientado a conexión y fiable mediante el uso de control de congestión, control de flujo, mecanismos de detección y recuperación de errores. Garantiza una entrega ordenada de la información, lleva a cabo transmisiones full-duplex e incorpora confirmaciones. TCP es adaptable a las condiciones dinámicas de la red.

3.1.1 IEEE 802.11

IEEE 802.11 es un conjunto de especificaciones de capa física (PHY) y capa de control de acceso al medio (MAC) para la implementación de redes de área local inalámbricas (WLAN). Creado y mantenido por el Institute of Electrical and Electronics Engineering (IEEE) LAN/MAN Standards Committee (IEEE 802). La versión inicial del estándar se publicó en 1997 y ha tenido diversos ajustes. Este estándar y sus ajustes son la base para productos inalámbricos que hacen uso de la marca Wi-Fi.

La familia 802.11 consiste en una serie de técnicas de modulación half-duplex por el aire que hacen uso del mismo protocolo básico. 802.11-1997 fue el primer estándar de la familia, sin embargo, en primero ampliamente aceptado fue 802.11b, seguido de 802.11a, 802.11g, 802.11n y 802.11ac. Otros estándares en la familia (c-f, h, j), son ajustes que se utilizan para ampliar los estándares existentes, y pueden incluir correcciones de especificaciones previas:

- 802.11-1997 (802.11 legacy): La versión original del estándar IEEE 802.11, publicada en 1997 y clarificada en 1999, aunque obsoleta actualmente. Especifica dos nuevos bits rates netos de 1 o 2 Mbit/s, además de corrección de errores hacia delante, así como tres tecnologías para la capa física: señales infrarrojas operando a 1 Mbit/s, salto de frecuencias en el espectro expandido operando a 1 o 2 Mbit/s y secuencia directa en el espectro expandido operando a 1 o 2 Mbit/s. Las dos últimas hacen uso de microondas en la banda de 2.4 GHz. También se define el protocolo carrier sense multiple access with collision avoidance (CSMA/CA) para la capa MAC.
- 802.11a (OFDM waveform): Es una corrección de IEEE 802.11 que define los requerimientos para el acceso múltiple por división de frecuencias ortogonales (OFDM). Originalmente descrita como la cláusula 17 de la especificación de 1999, actualmente está definida en la cláusula 18 de la especificación de 2012. Proporciona protocolos que permiten la transmisión y recepción de datos a ratios desde 1.5 a 54 Mbit/s teóricos. Utiliza el mismo protocolo de capa de enlace y el formato de trama que el estándar original, pero hace uso de OFDM en la capa física. Opera en la banda de 5 GHz con una velocidad máxima de 54Mbit/s, además de códigos de detección de errores. En la práctica, alcanza

unas velocidades de 20 Mbit/s. Se ha implementado a lo largo del mundo, particularmente en espacios corporativos. Aunque la corrección original ya no es válida, el término 802.11a aún se usa para describir interoperabilidad de sus sistemas a 5.8 GHz, 54 Mbit/s.

- 802.11b: Es una corrección de IEEE 802.11 de 1999 que extiende el throughput hasta los 11 Mbit/s teóricos y usa el protocolo CSMA/CA definido en el estándar original. Debido a la sobrecarga causada por CSMA/CA, en la práctica se alcanzan velocidades de 5.9 Mbit/s utilizando TCP y 7.1 Mbit/s con UDP. Los dispositivos que utilizan 802.11b experimentan interferencias de otros productos que operan en la banda de 2.4 GHz, tales como hornos microondas, dispositivos Bluetooth, teléfonos inalámbricos, etc. Se utiliza en configuraciones punto a multipunto, donde un punto de acceso se comunica mediante una antena omnidireccional con clientes móviles en su rango de cobertura.
- 802.11g: Es el tercer estándar de modulación para WLANs. Ratificado en 2003, es la cláusula 19 tanto de la especificación de 2007 como de la de 2012. Opera en la banda de 2.4 GHz al igual que 802.11b, pero hace uso del mismo esquema de transmisión OFDM que 802.11a. Alcanza velocidades de 54Mbit/s teóricas, 22Mbit/s en la práctica. Es totalmente compatible hacia atrás con hardware 802.11b, sin embargo, en una red 802.11g, la presencia de equipos 802.11b reducen significativamente la velocidad general de la red. El esquema de modulación de 802.11g copia el OFDM de 802.11a para velocidades de 6, 9, 12, 18, 24, 36, 48 y 54 Mbit/s, hace uso de complementary code keying (CCK), al igual que 802.11b para velocidades de 5.5 y 11 Mbit/s y de DBPSK/DQPSK+DSSS para 1 o 2 Mbit/s. Al igual que 802.11b, sufre interferencias de otros productos que operan en la banda de 2.4 GHz.
- 802.11n: Es una corrección de los estándares 802.11 previos publicada en 2009 que incluye el uso de múltiples antenas para aumentar velocidades. Opera tanto en la banda de 2.4 GHz como en la de 5 GHz, siendo opcional ésta última, alcanzando velocidades máximas desde 54 Mbit/s hasta los 600 Mbit/s. Da soporte a multiple-input multiple-output (MIMO), canales de 40 MHz en la capa física, agregación de tramas en la capa MAC y mejoras de seguridad entre otras características. Es compatible hacia atrás con dispositivos 802.11g, 802.11b y 802.11a.
- 802.11ac: Es una corrección de 802.11 publicada en 2013 construida sobre 802.11n. Proporciona un alto throughput en la banda de 5 GHz. La especificación tiene un throughput multiestación de hasta 1 Gbit/s y un throughput en enlace único de al menos 500 Mbit/s. Para ello, hace uso de canales más anchos (80 o 160 MHz), más flujos MIMO (hasta 8), enlace descendente multiusuario MIMO (hasta 4 clientes) y una modulación de alta densidad (hasta 256-QAM).

802.11b/g/n utilizan el espectro de 2.400-2.500 GHz, lo que se conoce como la banda 2.4 GHz, mientras que 802.11a/n utilizan 4.915-5.825 GHz, conocida como la banda 5 GHz. Cada espectro se subdivide en canales con una frecuencia central y un ancho de banda. La banda de 2.4 GHz se divide en 14 canales separados 5 MHz, empezando por el canal 1 centrado en 2.412 GHz. Por otro lado, la numeración del espectro de 5.725-5.875 GHz se organiza de forma diferente en cada país según sus propias regulaciones del espectro.

3.1.2 IEEE 802.15

IEEE 802.15 es un conjunto de especificaciones para la implementación de redes de área personal inalámbricas (WPAN). Creado y mantenido por el Institute of Electrical and Electronics Engineering (IEEE) LAN/MAN Standards Committee (IEEE 802). Hay 10 áreas principales de desarrollo, aunque no todas están activas. El número de grupos de trabajo en IEEE 802.15 varía en función del número de proyectos activos. Las 10 grandes áreas de desarrollo son:

- IEEE 802.15.1: WPAN/Bluetooth: Este grupo se basa en la tecnología Bluetooth. Define las especificaciones de las capas PHY y MAC para conectividad inalámbrica con dispositivos fijos, portátiles y móviles dentro de o entrando en un área personal de operación. Los estándares se publicaron en 2002 y 2005.
- IEEE 802.15.2: Coexistence: Este grupo se encarga de la coexistencia de WPANs con otros dispositivos operando en bandas de frecuencia sin licencia. El estándar se publicó en 2003 y el grupo de trabajo “entró en hibernación”.
- IEEE 802.15.3: High rate WPAN: Se divide a su vez en:
 - IEEE 802.15.3-2003: Es un estándar MAC y PHY para WPANs de alta velocidad (de 11 a 55 Mbit/s).
 - IEEE 802.15.3a: Fue un intento de proporcionar una banda ultra ancha de mayor velocidad como mejora para aplicaciones que involucran multimedia. Los miembros del grupo de trabajo no pudieron ponerse de acuerdo en la elección entre dos propuestas tecnológicas: multi-band orthogonal frequency division multiplexing (MB-OFDM) y direct sequence UWB (DS-UWB).
 - IEEE 802.15.3b-2006: Es una corrección publicada en 2006. Modifica 802.15.3 para mejorar la implementación e interoperabilidad de la capa MAC. Incluye varias optimizaciones, corrección de errores, aclaración de ambigüedades y añade clarificaciones editoriales preservando compatibilidad hacia atrás.
 - IEEE 802.15.3c-2009: Se publicó en 2009. El grupo desarrolló una capa PHY alternativa basada en ondas milimétricas para el estándar IEEE 802.15.3-2003. Esta mmWave WPAN está definida para operar en el rango 57-66 GHz. Permite velocidades muy altas, a corto alcance (10m) para aplicaciones incluyendo acceso a Internet de alta velocidad, descarga de contenido streaming, streaming en tiempo real y un bus de datos inalámbrico para sustituir el cable. Hay 3 modos de capa física: single carrier (SC) mode, hasta 5.3 Gbit/s, high speed interface (HSI), hasta 5 Gbit/s con portadora simple, y audio/visual (AV) mode, hasta 3.8 Gbit/s con OFDM).
- IEEE 802.15.4: Low rate WPAN: Trata con velocidades bajas (20-250 kbit/s), pero baterías de larga vida (años) y complejidad baja. El estándar define tanto la capa física como la capa de enlace. Opera en tres bandas de frecuencia posibles: 868/915/2450 MHz. La primera edición se publicó en 2003. Cuenta con 6 correcciones:
 - IEEE 802.15.4a: WPAN low rate alternative PHY: es una corrección que especifica capas físicas adicionales al estándar original. Su principal meta es proporcionar mayor rango de alta precisión y capacidades de

localización (1 metro o más de exactitud), mayor throughput agregado, añadiendo escalabilidad a las velocidades de transferencia, mayor rango y menor coste y consumo de potencia. Las dos líneas de partida elegidas fueron PHYs opcionales consistentes en una UWB pulse radio operando en el espectro sin licencia UWB y un chirp spread spectrum operando en 2.4 GHz.

- IEEE 802.15.4b: Revision and enhancement: Publicado en 2006, a este grupo se le encargó crear un proyecto para mejoras y aclaraciones del estándar IEEE 802.15.4-2003, tales como resolución de ambigüedades, reducir complejidades innecesarias, aumentar la flexibilidad en el uso de claves de seguridad, etc.
- IEEE 802.15.4c: Amendment for China: Publicado en 2009. Define una corrección de la capa física que añade nuevas especificaciones para ajustarse a los cambios de regulación de China que abrieron las bandas 314-316 MHz, 430-434 MHz y 779-783 MHz para uso de WPAN dentro del país.
- IEEE 802.15.4d: PHY and MAC amendment for Japan: A este grupo se le encargó definir una corrección del estándar IEEE 802.15.4-2006. La corrección define una nueva PHY y cambios a la MAC para dar soporte a la nueva banda disponible en Japón (950-956 MHz), a la vez que la coexistencia con sistemas de etiquetado en la banda.
- IEEE 802.15.4e: Amendment for industrial applications: A este grupo se le encargó una corrección de la capa MAC. La intención de esta corrección era la mejora y adición de funcionalidades a la capa MAC tales como mayor soporte a los mercados industriales y permitir compatibilidad con modificaciones propuestas para las WPAN chinas. Se hicieron mejoras específicas para añadir salto de canal y ranuras temporales variables compatibles con ISA100.11a. Se aprobó en 2011.
- IEEE 802.15.4f PHY and MAC amendment for active RFID: A este grupo se le encargó definir nuevas capas físicas y mejoras para la capa MAC requeridas para dar soporte a las nuevas PHYs para sistemas RFID activos bi-direccionales y aplicaciones de localización.
- IEEE 802.15.4g: Amendment for smart utility network: A este grupo se le encargó crear una corrección de la capa física para proporcionar un estándar para facilitar las aplicaciones de control de procesos a gran escala. En 2012 se lanzó el estándar radio 802.15.4g.
- IEEE 802.15.5: mesh networking: Proporciona un framework arquitectural que permite a los dispositivos WPAN promocionar redes malladas interoperables, estables y escalables. El estándar se divide en dos partes: low-rate WPAN mesh less, basado en IEEE 802.15.4-2006 MAC, y high-rate WPAN mesh networks, que utilizan IEEE 802.15.3b MAC. Las características comunes de las dos mallas incluyen inicialización de red, direccionamiento y unicasting multisalto. Además, la malla de baja velocidad soporta multicasting, broadcasting, portabilidad, trazado de ruta y funciones de ahorro de energía, mientras que la malla de alta velocidad soporta servicios multisalto de tiempo garantizado.
- IEEE 802.15.6: Body area networks: A este grupo se le encargó centrarse en un estándar inalámbrico de baja potencia y rango corto que se optimizara para

dispositivos y operaciones dentro de, sobre o alrededor del cuerpo humano con el fin de aplicarlo a aplicaciones médicas, electrónicas y entretenimiento. El borrador de un estándar para tecnologías body area network (BAN) se aprobó en 2011.

- IEEE 802.15.7: visible light communication: En 2011, este grupo de trabajo completó un borrador de estándares PHY y MAC para comunicación con luz visible (VLC).
- IEEE 802.15.8: Peer aware communications: Este grupo se encarga de desarrollar un estándar para peer aware communications (PAC) optimizado para comunicaciones e infraestructuras peer to peer con coordinación distribuida completa operando en bandas inferiores a 11 GHz. El estándar propuesto cuenta con velocidades escalables de hasta 10 Mbit/s. Algunas características propuestas son:
 - Descubrimiento de información de vecinos sin asociación.
 - Descubrimiento del número de dispositivos en la red.
 - Comunicaciones grupales con membresía simultánea en múltiples grupos (hasta 10).
 - Posicionamiento relativo.
 - Retransmisión multisalto.
 - Seguridad.
- IEEE 802.15.9: Key management protocol: Este grupo se encarga de desarrollar una práctica recomendada para el transporte de datagramas del protocolo de administración de claves (KMP). La recomendación definirá un framework de mensajes basado en elementos de información como método de transporte para los datagramas KMP y directrices para el uso de KMPs existentes con IEEE 802.15.4. La práctica recomendada no creará un nuevo KMP. Aunque IEEE 802.15.4 soporta seguridad de datagramas, no proporciona un mecanismo para el establecimiento de claves, lo que resulta en claves débiles.
- IEEE 802.15.10: Layer 2 routing: Este grupo se encarga de desarrollar una práctica recomendada para el routing de paquetes en redes 802.15.4 con cambios dinámicos, con mínimo impacto en el manejo de las rutas. El objetivo es extender el área de cobertura con el incremento del número de nodos. Las capacidades relacionadas con las rutas que la práctica recomendada proporcionará incluyen:
 - Establecimiento de rutas.
 - Reconfiguración dinámica de rutas.
 - Descubrimiento y adición de nuevos nodos.
 - Rotura de rutas establecidas.
 - Pérdida y recurrencia de rutas.
 - Recopilación en tiempo real del estado de los enlaces.
 - Soporte para broadcast y multicast.
 - Reenvío de tramas efectivo.

3.2 Redes ad hoc inalámbricas

Las redes ad hoc inalámbricas o WANET son un tipo de red inalámbrica descentralizada. La red no depende de infraestructuras pre-existentes, en su lugar, cada nodo participa en el encaminamiento mediante el reenvío de datos hacia otros nodos.

Una red ad hoc se refiere a cualquier conjunto de redes donde todos los nodos tengan el mismo estado dentro de la red y son libres de asociarse con cualquier otro dispositivo ad hoc en el rango de enlace. También se refiere a la habilidad de un dispositivo de red de mantener la información del estado de conexión para cualquier cantidad de dispositivos en un rango de un enlace.

Este tipo de red permite la adhesión de nuevos dispositivos simplemente estando en el rango de un nodo ya perteneciente a la red. El principal inconveniente de este tipo de redes radica en el número de saltos que debe recorrer la información antes de llegar a su destino.

El hecho de ser redes descentralizadas las hace aptas para aplicaciones no dependientes de nodos centrales. Son también aptas para situaciones de emergencia gracias a su mínima configuración y rápido despliegue. Según su aplicación, pueden ser clasificadas en:

- Mobile ad hoc network (MANET): Se trata de una red de dispositivos conectados inalámbricamente y que poseen propiedades de auto-configuración además de poseer cierta movilidad. Un tipo específico de este tipo de redes son las redes vehiculares (VANET), aunque este tipo de redes tienen retos específicos extra.
- Redes inalámbricas malladas (mesh): Es una red en malla implementada sobre una red inalámbrica LAN. Son aquellas redes en las que se mezclan la topología Ad hoc y una topología basada en infraestructura. Permiten que las tarjetas de red se comuniquen entre sí, independientemente del punto de acceso.
- Redes de sensores: Es una red de dispositivos equipados con sensores que colaboran en una tarea común. Están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación inalámbrica los cuales permiten formar redes descentralizadas. Se caracterizan por su facilidad de despliegue y por ser autoconfigurables. Otra de sus características es su gestión eficiente de la energía.

Entre las aplicaciones principales para las redes ad hoc podemos destacar:

- Aplicaciones militares, gracias a la seguridad que proporcionan.
- Computación distribuida, puesto que son fáciles de desplegar.
- Operaciones de emergencia, en el caso de que las redes convencionales no lleguen o estén caídas.
- Sanidad

Existen diversos retos relacionados con las redes Ad hoc inalámbricas:

- Esquema de acceso al medio (MAC).
- Routing.
- Multicasting.
- Transporte.

- Calidad de servicio (QoS).
- Seguridad.

Control de Acceso al Medio

El objetivo del control de acceso al medio (MAC) es gestionar el acceso al medio compartido (aire), para ello, debe tener en cuenta diversas cuestiones:

- Debe ser distribuido.
- Puesto que no hay nodos centrales, hace falta sincronización.
- Existe el problema del terminal oculto.
- Hay que intentar maximizar el flujo de la red, a la vez que se minimiza el retraso de acceso.
- Se debe repartir el ancho de banda con justicia.
- Hay que alcanzar un compromiso entre ahorro de batería y tasa de envío.

Las redes ad hoc inalámbricas cuentan con protocolos MAC específicos, puesto que los protocolos típicos de redes cableadas (CSMA) no son válidos al basarse en la medida del medio a transmitir. Se pueden clasificar estos protocolos en:

- Contention free: TDMA, CDMA, FDMA, son válidos para redes estructuradas y redes estáticas.
- Contention based:
 - Random access: ALOHA y ALOHA ranurado (ALOHA+TDMA), válidos para redes con poca carga, ofrece una tasa de envío muy reducida.
 - Reservation/collision resolution: Pueden ser iniciados por el emisor (sender-initiated) o iniciados por el receptor (receiver-initiated):
 - Multiple access collision avoidance (MACA): hace uso de paquetes adicionales RTS (request-to-send) y CTS (clear-to-send) así como de un mecanismo de back-off ante colisiones. Entre sus debilidades están las estaciones ocultas y la alta carga, además, si falla una transmisión se espera hasta que lo detecta el mecanismo de la capa de transporte.
 - Multiple access collision avoidance wireless (MACAW): incluye el envío de un ACK después del envío de datos, para evitar terminal expuesto, se hace uso de un DS (tiempo de envío). Para controlar el medio, el CTS incluye la IP del que envió el RTS.
 - MACA-BI: el receptor envía un ready-to-receive (RTR) cuando está preparado, el tiempo de los RTR se estima con los paquetes de datos, contempla la utilización de RTS.
- Protocolos con múltiples canales. Sus principales ventajas son que minimiza las colisiones, es fácil ofrecer determinadas calidades de servicio y múltiples canales aumentan el ancho de banda disponible. Sin embargo, aún no se ha definido a que nodo se le asigna qué canal.
 - Busy tone multiple access (BTMA): Define dos canales de transmisión, uno para datos y otro para control, si nadie ocupa el canal de control se emite una señal busy y se envían los datos.
 - Dual BTMA (DBTMA): Define un canal de transmisión para datos y otro para control. Hace uso de dos tonos de ocupado: B_r y B_t , utiliza RTS y

- CTS. Si no hay B_r , se envía un RTS, al recibirlo, si no hay B_t , se envía el CTS.
- HRMA: Se divide el tiempo en ranuras y adicionalmente en M frecuencias. f_0 se utiliza para sincronización, f y f' son parejas de frecuencias, f' se utiliza para hop reservation (HR), RTS, CTS y datos, f para ACK
- Protocolos con antenas direccionales: cada nodo cuenta con un array de antenas.
 - Directional MACA: Se envían RTS omnidireccionales, CTS y Datos son direccionales.

Routing

El objetivo del routing es encontrar el camino de un nodo origen a un nodo destino, para ello, debe tener en cuenta:

- Cambios de topología derivados de la movilidad.
- Tasa de error elevada en el canal
- Pérdida de paquetes por el entorno inalámbrico.
- Problemas de terminal oculto y expuesto.
- Requieren mecanismos efectivos: batería, cómputo y ancho de banda reducidos.

Las redes Ad hoc inalámbricas cuentan con protocolos de routing específicos, que pueden clasificarse en:

- Proactivos: Son extensiones de los protocolos de routing usados en redes cableadas. Buscan las rutas a seguir de forma periódica, suponiendo que serán útiles:
 - Destination sequenced distance Vector (DSDV): Se basa en el algoritmo Bellman-Ford: tabla con la distancia (número de saltos) mínima y el primer salto al destino. Incluye la actualización de tablas, números de secuencia para evitar bucles y count-to-infinity. Sus principales ventajas son que se conoce el siguiente salto a todos los destinos y permite adaptar los protocolos de redes cableadas a ad hoc, como desventajas destacan que no es apto para entornos de movilidad y si hay un cambio en la red hay que esperar que dicho nodo lo notifique.
 - Wireless routing protocol (WRP): También extiende de Bellman-Ford, presenta un mecanismo particular para evitar bucles. Utiliza 4 tablas: Distance table (DT) con información de los vecinos, routing table (RT) con el siguiente salto y predecesor de un destino, link cost table (LCT) con el coste para alcanzar un vecino y message retransmission list (MRL) con mensajes a enviar a los vecinos en forma de contador. Sus principales ventajas son que converge más rápidamente y no tiene que esperar que el destino actualice su estado para detectar una nueva ruta. Sus desventajas son que necesita mucho espacio en disco en cada nodo y no es apto para alta movilidad.
 - Optimized link state routing (OLSR): Protocolo de inundación eficiente, analiza el estado del enlace mediante un mecanismo multipoint relaying: Reduce el tamaño de los paquetes, hace uso de

- menos enlaces para enviar la información (MPRset) y envía Hello periódicos con los MPRset. Cuenta como ventajas que reduce la sobrecarga con respecto a otros protocolos proactivos y reduce el número de envíos. Sus principales desventajas son que hace uso de muchos paquetes de control y es inadecuado para alta movilidad.
- Fisheye state routing protocol (FSR) Es un protocolo jerárquico. Su idea principal es que la información en una zona focal es más precisa que en el exterior de dicha zona. El estado de los enlaces sólo se envía a los vecinos. La máxima frecuencia de actualización se encuentra a un salto. Sus ventajas son que cuenta con menos paquetes de control y está especialmente diseñado para redes grandes y con alta movilidad, sin embargo, hay que tener en cuenta que el número de saltos que define cada nivel es un parámetro clave.
- Reactivos: Solo se realiza la búsqueda de un camino hacia un destino cuando es necesario y no antes:
- Dynamic source routing protocol (DSR): Pretende reducir el ancho de banda que suponen los paquetes de control, su principal característica es que es beacon-less (no manda mensajes hello), busca construir el camino óptimo a un destino mediante la inundación de RouteRequest, hace uso de número de secuencia y TTL. Cada nodo almacena el camino completo hasta el destino solicitado. Su principal ventaja es que no necesita envío periódico de paquetes de control. Sus desventajas son que no maneja localmente la rotura de un enlace y que iniciar una conexión es más lento que con protocolos proactivos.
 - Ad hoc on-demand distance vector routing (AODV): La principal diferencia con DSR es que sólo almacena el siguiente salto al destino, adicionalmente, utiliza un número de secuencia de destino para determinar un camino actualizado hacia el destino. Sus principales ventajas son que las rutas se establecen por necesidad, solo almacena el siguiente salto y hace uso de un número de secuencia del destino. Sus desventajas son la existencia de varios RouteReply en respuesta, beacons periódicos y que no maneja localmente la rotura de un enlace.
 - Temporally ordered routing algorithm (TORA): Iniciado por la fuente, establece las rutas rápidamente, minimiza paquetes de control y es libre de bucles. Cada nodo posee información local de primer salto y puede detectar particiones. Sus funciones básicas son la creación de rutas, el mantenimiento de rutas y el borrado de rutas. Cuenta con 3 paquetes de control distintos: Query (QRY), update (UPD) y clear (CLR). Su principal ventaja es que restringe los paquetes de control a una zona, mientras que su mayor desventaja es que un cambio local en la ruta puede no ser óptimo.
- Híbridos: Combinan el uso de protocolos proactivos y reactivos:
- Zone routing protocol (ZRP): la idea principal es utilizar un protocolo proactivo en una zona pequeña y uno reactivo para la red global. Proactivo: intra-zone routing protocol (IARP), reactivo: inter-zone routing protocol (IERP). Su mayor ventaja es que hace uso de menos

paquetes de control que los protocolos reactivos y proactivos, sin embargo, si existen zonas solapadas existe un exceso de los mismos.

Multicast

Se define el multicast routing como el establecimiento de un árbol de rutas en el que se distribuyen los grupos multicast. Un mensaje pasa por una rama de un árbol una única vez. Sin embargo, en redes Ad hoc, el árbol no se mantiene debido a la movilidad, por tanto, hará falta la utilización de protocolos exclusivos. Estos protocolos tienen que tener en cuenta los siguientes factores:

- Robustez
- Eficiencia
- Reducido número de paquetes de control
- Calidad de servicio
- Gestión de recursos

Dependiendo de su funcionamiento, los protocolos se pueden dividir en iniciados por la fuente o por el receptor.

- Multicast ad hoc on-demand distance vector routing protocol (MAODV): Se divide en dos fases, inicialización del árbol y mantenimiento del mismo. Destacan dos conceptos clave:
 - Group leader: primero en unirse
 - Group hellos: se envían periódicamente

Se hace uso de mensajes multicast activation (MACT) y RREP para el establecimiento de ruta y árbol.

Transporte

Para el transporte se requiere un protocolo que cuente con control de congestión y confiable, por lo que se descartan UDP y TCP tradicional. Dicho protocolo tiene como objetivos:

- Mínima sobrecarga debido a cabeceras
- Mecanismos de control de congestión
- Adaptarse dinámicamente a cambios de topología
- Interacción cross-layer

Para lograr estos objetivos, se desarrollan distintos protocolos a partir de TCP:

- Feedback-based TCP (TCP-F): Requiere una capa de enlace confiable y un protocolo de routing que pueda facilitarle información. Su objetivo es reducir la degradación de throughput debida a las roturas de los caminos de información. Si un enlace cae existe un punto de fallo (FP) que genera una notificación de fallo de ruta (RFN), cuando el emisor recibe un RFN entra en estado snooze, tras la recepción de un restablecimiento de ruta (RRN) vuelve a estado conectado. Sus principales ventajas son una solución sencilla al problema de congestión a la vez que proporciona control de la misma. Por contra, requiere una modificación de la librería TCP (debido al estado snooze) y la ventana de congestión tras un RFN puede no corresponder con la necesaria.
- TCP with explicit link failure notifications (TCP-ELFN): Similar a TCP-F, el FP

genera ELFN para el emisor, que contiene un ICPM error y un route error con la información de la rotura el enlace en él. Para comprobar si el enlace está active genera pruebas periódicas en el estado standby. Sus ventajas son que desacopla control de congestión y detección de rotura de enlaces y una menor dependencia del protocolo de routing para reestablecer la ruta. Sus desventajas son que las particiones de la red implican muchos paquetes de control (lo que conlleva pruebas periódicas) y que, al igual que TCP-F, la ventana de congestión tras un RFN puede no corresponder con la necesaria.

- Ad hoc TCP (ATCP): No modifica TCP, sino que añade una capa entre red y transporte. Dicha capa distingue entre distintos estados:
 - Normal.
 - LOSS. 3 o más ACKs duplicados: No reenvía ACKs, pone TCP en persist (evita retransmisiones) y reenvía los paquetes.
 - CONGESTED. Si recibe una notificación de congestión (ECN). Quita TCP de persist, envía el paquete a TCP y pasa a CONGESTED (no realiza ninguna operación).
 - DISCONN. ICMP destination unreachable (DUR). Pone TCP en persist y envía pruebas periódicas.

Sus principales ventajas son que es compatible con TCP y que separa el control de congestión con la rotura de rutas. Sus desventajas que depende del algoritmo de routing para detectar roturas y la implementación de la capa ATCP.

- Split-TCP: Separa los dos requerimientos de TCP en ad hoc: Control de congestión (localmente) y confiabilidad (extremo a extremo). Se divide una sesión TCP en segmentos o zonas con nodos intermedios proxy. Sus principales ventajas son que mejora el throughput, la injusticia del mismo y la movilidad es menos crítica. Por otro lado, requiere una modificación de TCP y viola la garantía extremo a extremo.

Calidad de Servicio

Para garantizar calidad de servicio (QoS), hacen falta una serie aspectos a tener en cuenta:

- Parámetros QoS: dependen de la aplicación:
 - Multimedia: velocidad de transmisión y retardo.
 - Militar: seguridad y confianza
 - Sensores: mínimo consumo de batería
- Routing preparado para QoS: camino óptimo a un destino depende de los parámetros de QoS.
- Marco QoS: sistema completo.

Seguridad

A la hora de proporcionar seguridad a las comunicaciones, las soluciones tradicionales pasan por TTPs (trusted third parties): Certification authority (CA), que almacena la correspondencia entre usuario y clave pública, o key distribution center (KDC), que gestiona claves de sesión para los usuarios de su base de datos.

Si nos centramos en las redes ad hoc, las distintas soluciones pueden encuadrarse en dos apartados: Aproximación asimétrica, en la que la autoridad está parcial o totalmente distribuida, o aproximación simétrica (hardcoded).

Las amenazas de las redes ad hoc atacan a todas las capas: física, enlace, red, transporte, multicapa. Las más comunes atacan la capa de red (algoritmo de routing), mediante sinkholes, dropping, jamming o inundaciones. Es por ello que existen protocolos de routing seguros:

- Security-aware ad hoc routing protocol (SAR): Utiliza niveles de confianza, cada nivel hace uso de una clave simétrica, las cuales se distribuyen a niveles iguales o superiores.
- Authenticated routing for ad hoc networks (ARAN): Incluye encriptación asimétrica en el proceso. Utiliza certificados, no trata la distribución de claves y hace uso de autenticación extremo a extremo.

3.2.1 AODV

La posterior utilización del protocolo AODV en la implementación realizada motiva un estudio más profundo de este protocolo, para tener en cuenta en la fase de diseño posibles restricciones.

Ad hoc on-demand distance vector routing (AODV), es un protocolo publicado en Julio de 2003 por C. Perkins del Nokia Research Center, E. Belding-Royer de la Universidad de Santa Bárbara, California y S. Das de la Universidad de Cincinnati y recogido en el RFC 3561 [\[6\]](#).

El protocolo AODV está diseñado para redes ad hoc móviles con poblaciones de hasta decenas o millares de nodos móviles. AODV puede encargarse de ratios de baja, moderada o relativamente elevada movilidad, así como una variedad de niveles de tráfico de datos. AODV está diseñado para usarlo en redes donde los nodos pueden confiar entre sí, ya sea mediante el uso de claves preconfiguradas, o porque se conoce que no hay nodos maliciosos. Se ha diseñado para reducir la diseminación de tráfico de control y eliminar exceso de cabeceras en tráfico de datos con el fin de mejorar la escalabilidad y el rendimiento.

El algoritmo de AODV hace posible un routing dinámico, de arranque automático y multisalto entre nodos móviles participantes en el establecimiento y mantenimiento de una red ad hoc. AODV permite a los nodos móviles obtener rutas hacia nuevos destinos rápidamente, y no requiere que mantengan las rutas hacia destinos no activos en la comunicación. Además, autoriza a los nodos móviles a responder en el caso de roturas de enlaces y cambios en la topología de la red de forma rápida. La operación de AODV es libre de bucles, y, evitando el problema de la “cuenta hasta el infinito” de Bellman-Ford ofrece una rápida convergencia cuando se produce un cambio de topología. Cuando se rompe un enlace, AODV notifica al grupo de nodos afectados para que invaliden las rutas que hacen uso del enlace roto. Una característica distintiva de AODV es su uso de un número de secuencia de destino para cada entrada de ruta. Dicho número es creado por el destino para ser incluido junto a cualquier información de ruta que envía a los nodos que la soliciten. La utilización de estos números asegura libertad de bucles y es sencillo de programar. A la hora de elegir entre dos rutas hacia el mismo destino, se seleccionará la ruta que tenga el mayor número de secuencia.

Hay tres tipos de mensaje definidos por AODV: Route requests (RREQs), route replies (RREPs) y route errors (RERRs). Estos mensajes se reciben vía UDP, y el procesamiento habitual de cabeceras IP se les aplica. Por tanto, se espera que el generador de los mensajes utilice su dirección IP como dirección IP origen para los mensajes. Para mensajes broadcast, se utiliza la IP 255.255.255.255.

Mientras los extremos de una comunicación tengan rutas válidas entre ellos, AODV no interviene en la misma. Cuando se necesite una nueva ruta a un nuevo destino, el emisor enviará por broadcast un RREQ para encontrar dicha ruta. La ruta se determinará o bien cuando el RREQ alcance el destino, o bien al pasar por un nodo intermedio con una ruta reciente, la cual será válida si su número de secuencia es al menos tan grande como el del RREQ. La ruta se hará efectiva enviando un RREP unicast de vuelta al dueño del RREQ.

Los nodos monitorizan el estado del enlace de los saltos siguientes en rutas activas. Cuando se produce una rotura de enlace en una ruta activa, se utiliza un mensaje RERR para notificar al resto de nodos la pérdida de ese enlace. El RERR indica aquellos destinos que han pasado a ser no alcanzables por el enlace roto.

AODV es un protocolo de routing y se encarga de la gestión de la tabla de routing. La información se debe mantener incluso para rutas de corta vida. En cada entrada de la tabla se utilizan los siguientes campos:

- Dirección IP destino.
- Número de secuencia destino.
- Flag de número de secuencia destino válido.
- Flags de estado y routing (válido, inválido, reparable, siendo reparado).
- Interfaz de Red
- Cuenta de saltos (número de saltos necesarios para alcanzar el destino).
- Siguiendo salto
- Lista de precursores
- Tiempo de vida

La gestión del número de secuencia es crucial para evitar bucles de routing, incluso cuando se rompe un enlace y un nodo pasa a no ser alcanzable para suministrar su propia información sobre su número de secuencia. Un destino pasa a ser inalcanzable cuando se rompe o desactiva un enlace. Cuando ocurre esto, la ruta se invalida para operaciones que involucren su número de secuencia y se marca la entrada en la tabla de routing como inválida.

Estructura de los mensajes

El formato de los mensajes route request (RREQ), route reply (RREP), route error (RERR) y route reply acknowledgment (RREP-ACK) definidos para AODV es el siguiente:

- Route request (RREQ): Se estructura de la siguiente forma:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type										J R G D U Reserved										Hop Count																			
										RREQ ID																													
										Destination IP Address																													
										Destination Sequence Number																													
										Originator IP Address																													
										Originator Sequence Number																													

- Type: 1
- J: Flag join; reservado para multicast.
- R: Flag repair; reservado para multicast.
- G: Flag gratuitous RREP; indica cuando un RREP gratuito debería ser enviado al nodo especificado en la dirección IP destino.
- D: Flag destination only; indica que solo el destino puede responder a este RREQ.
- U: Unknown sequence number; indica que el número de secuencia destino es desconocido.
- Reserved: Enviado como 0; ignorado en recepción.
- Hop count: El número de saltos desde la dirección IP origen hasta el nodo que maneja la petición.
- RREQ ID: Número de secuencia único que identifica el RREQ en conjunción con la dirección IP del origen.
- Destination IP address: La dirección IP del nodo destino del que se desea la ruta.
- Destination sequence number: El último número de secuencia recibido por el origen para cualquier ruta hacia el destino.
- Originator IP address: La dirección IP del nodo que origina el route request.
- Originator sequence number: El número de secuencia actual para ser utilizado en la entrada de la ruta hacia el origen del route request.

- Route reply (RREP): Su estructura es la siguiente:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
+-----+																																							

- Type: 2
- R: Flag repair; utilizado para multicast.

- A: Acknowledgment requerido.
- Reserved: Enviado como 0; ignorado en recepción.
- Prefix size: Si es distinto de cero, los 5 bits especifican que el siguiente salto indicado puede ser utilizado por cualquier nodo con el mismo prefijo como el destino solicitado.
- Hop count: El número de saltos desde la dirección IP origen y la dirección IP destino. Para rutas multicast indica el número de saltos hasta el miembro del árbol multicast enviando el RREP.
- Destination IP address: La dirección IP del destino para el que se suministra la ruta.
- Destination sequence number: El número de secuencia de destino asociado a la ruta.
- Originator IP address: Dirección IP del nodo que creó el RREQ para el que se suministra la ruta.
- Lifetime: Tiempo en milisegundos durante el cual los nodos que reciban el RREP consideran la ruta válida.

El prefix size permite a un router de subred proporcionar una ruta para cada host en la subred definido por el prefijo de ruta, el cual es determinado por la dirección IP del router de subred y el prefix size. Para garantizar esta característica, el router tiene que garantizar alcanzabilidad a todos los hosts que comparten el prefijo.

El bit "A" se utiliza cuando el enlace por el que se envía el RREP es no fiable o unidireccional.

- Route error (RERR): Su formato es el siguiente:

```

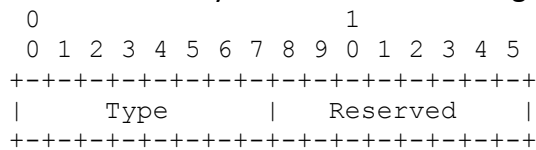
0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      |N|      Reserved      |      DestCount      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Unreachable Destination IP Address (1)      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Unreachable Destination Sequence Number (1)      |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Additional Unreachable Destination IP Addresses (if needed) |
+-----+-----+-----+-----+-----+-----+-----+-----+
|Additional Unreachable Destination Sequence Numbers (if needed)|
+-----+-----+-----+-----+-----+-----+-----+-----+

```

- Type: 3
- N: Flag no delete; fijado cuando un nodo ha realizado una reparación local de un enlace y los nodos superiores no deberían borrar la ruta.
- Reserved; Enviado como 0; ignorado en recepción.
- DestCount: El número de destinos no alcanzables incluidos en el mensaje, al menos debe ser 1.
- Unreachable destination IP address: La dirección IP del destino que ha pasado a ser inalcanzable debido a una rotura de enlace.
- Unreachable destination sequence number: El número de secuencia en la entrada de la tabla de routing para el destino listado en el campo anterior.

El mensaje RERR se envía cuando la rotura de un enlace causa que uno o más destinos pasen a ser inalcanzables desde algún nodo vecino.

- Route reply acknowledgment (RREP-ACK): Este mensaje debe ser enviado como respuesta a un RREP con el bit "A" fijado. Generalmente se hace cuando hay riesgo de enlaces unidireccionales que prevengan la terminación del ciclo de route discovery. Se estructura de la siguiente forma:



- Type: 4
- Reserved: Enviado como 0; ignorado en recepción.

Operación

A continuación, se describen las siguientes operaciones llevadas a cabo por AODV: descubrimiento de rutas, establecimiento de rutas, mantenimiento de conectividad, rotura de enlaces.

- Descubrimiento de rutas: AODV es un protocolo reactivo, por tanto, el descubrimiento de rutas funciona bajo demanda, es decir, cuando un nodo quiere comunicarse con otro del que no conoce la ruta, se inicia el descubrimiento de dicha ruta. Para ello, se difunde por broadcast un paquete RREQ. El campo hop count se inicializa a cero y se incrementará cada vez que se retransmita el RREQ. Cada uno de los nodos involucrados se encarga de mantener dos contadores: el número de secuencia, que asegura que las rutas no sean obsoletas, y un identificador del RREQ, que junto con la dirección IP origen, identifican cada RREQ. El número de secuencia aumenta cuando el nodo adquiere información sobre un nuevo vecino, mientras que el identificador de RREQ cuando se inicia un RREQ. El nodo fuente del RREQ lo rellena con su dirección IP, su número de secuencia actual y su identificador de RREQ, así como la dirección IP de destino y su último número de secuencia conocido. Cuando un nodo recibe un RREQ, en primer lugar, actualiza su tabla de routing con el número de secuencia y el siguiente salto hacia el origen del RREQ. A continuación, comprueba su tabla para ver si conoce la ruta hacia el destino, es decir, o bien es el destino, o bien tiene grabada una ruta cuyo temporizador no ha expirado y tiene un número de secuencia como mínimo igual al del RREQ. Si se cumplen estos requisitos, se considera que el nodo tiene una ruta reciente hacia el destino y genera un mensaje RREP. En caso contrario, vuelve a difundir el mensaje a sus vecinos, actualizando el número de secuencia de destino al máximo si el almacenado en su tabla es mayor. Cuando un nodo recibe un RREQ, almacena la dirección IP origen y el RREQ ID, de esta forma, si vuelve a recibir ese RREQ, se descarta.
- Establecimiento de rutas: Cuando un nodo recibe un RREQ y tiene una ruta reciente con la que puede responder, genera un mensaje RREP de vuelta. En caso de que el enlace por el que se envía el mensaje no sea fiable, se activará el flag de acknowledgment, con el fin de que al recibir el RREP, el nodo debe enviar de vuelta un RREP-ACK. Si el nodo que responde es el destino, el campo hop count estará a cero, en caso contrario, indicará el número de saltos entre el nodo que ha generado el RREP y el destino. Dicho RREP se envía de forma unicast al siguiente salto hacia el nodo fuente. El nodo que reciba el RREP incrementará el

hop count en 1 y actualizará su tabla de routing con la información del nodo destino. A continuación, volverá a enviar de forma unicast el RREP al siguiente salto. Esto se repite hasta que se alcance el nodo fuente, lo que establece la ruta a seguir. Si posteriormente el nodo fuente recibiera un RREP con un número de secuencia igual o mayor al almacenado en la tabla de routing pero con menor hop count, se cambiaría la ruta almacenada por la nueva recibida. Si un nodo intermedio recibe más de un RREP para un mismo par fuente/destino, el nodo compara el número de secuencia y el de saltos con la información de su tabla de routing. Si el número de secuencia es mayor que el almacenado, o si es el mismo pero el número de saltos es menor, el nodo actualizará su tabla y retransmitirá el RREP a la fuente. En caso contrario, se descarta el RREP y no se reenvía.

- Gratuitous RREP: Cuando se recibe un RREQ y se responde con RREP, el RREQ deja de transmitirse, lo que puede provocar que el nodo destino no reciba ningún RREQ si todos son respondidos por nodos intermedios, con lo cual, no aprenderá la ruta hasta el nodo fuente, lo cual puede causar problemas si se intenta establecer una sesión TCP, por ejemplo. Para evitar que esto ocurra, debe activarse el flag de RREP gratuito desde la fuente. Si este flag está activado, los nodos intermedios que reciban el RREQ y respondan con un RREP, también enviarán de forma unicast un RREP al nodo destino. La dirección IP del destino se colocará en el campo originator IP address, mientras que la IP del nodo fuente se colocará en el destination IP address. El nodo que crea el RREP gratuito usa su distancia a la fuente como hop count y envía el mensaje al siguiente salto hacia el nodo destino.
- Mantenimiento de conectividad: A través de la transmisión de los paquetes, los nodos de la red mantienen actualizada la información relativa a la conectividad local. Cuando se rompe un enlace, si se utiliza un protocolo de capa 2 fiable, los nodos pueden desentenderse de la detección de los enlaces rotos. Sin embargo, si no existe una capa MAC que proporcione estas prestaciones, los nodos harán un uso proactivo de los mensajes HELLO. Para utilizar estos mensajes, si un nodo no ha transmitido nada dentro de un intervalo predefinido, difundirá a sus vecinos un mensaje que informará de que sigue activo. Este mensaje, denominado HELLO, es un RREP no solicitado que contiene la dirección del nodo que lo emite junto a su número de secuencia actual. Este mensaje solo se transmite a los vecinos, puesto que contiene un TTL igual a uno. Los vecinos que reciban este paquete actualizarán su información de conectividad local para incluir al emisor del HELLO. En caso de no recibir ninguno en un tiempo definido con anterioridad, significa que la conectividad ha cambiado y la información de la ruta para ese vecino deberá ser actualizada.
- Rotura de enlaces: Cuando se rompe un enlace con un vecino, el nodo copia la distancia al destino en el campo cuenta de saltos de la tabla de routing para ese vecino. Esto se utilizará para volver a descubrir una ruta hacia dicho destino. A continuación, marca el enlace como inválido e incrementa su número de secuencia. Acto seguido, comprueba en la tabla de routing si ese vecino era el siguiente salto de alguna de sus rutas activas, en caso afirmativo, se genera un mensaje RERR. Este mensaje contiene una entrada por cada destino cuya ruta se ha visto afectada por la rotura del enlace, así como el número de secuencia de

cada destino incrementado en uno. Este mensaje se difundirá a todos los vecinos e invalidará todas las rutas afectadas. Cuando se recibe un RERR, se invalidarán las rutas mencionadas en el mensaje si esas rutas usan la fuente del RERR como siguiente salto. Si esto ocurre, el nodo difunde su propio RERR con los destinos perdidos. Si un mensaje llega hasta el nodo fuente de una comunicación, éste invalida las rutas afectadas y, si necesita una ruta hacia el destino perdido, reiniciará un proceso de descubrimiento de rutas. También se generará un mensaje RERR si se recibe un paquete de datos para el que no se tiene ninguna ruta. En este caso, se difundirá un mensaje RERR con la dirección IP del nodo destino por cada paquete de datos con origen desconocido.

3.3 Tunneling

Puesto que el objetivo del proyecto es crear una red virtual inalámbrica sobre una red cableada ya existente, será necesario crear túneles entre las interfaces de ambas redes para poder mantener una correcta comunicación. De ahí que sea necesario realizar un estudio previo del tunneling.

Se denomina tunneling a la técnica consistente en encapsular un protocolo de red sobre otro, lo que crea un túnel por el que fluye la información. De esta forma, permite acceso a una red a servicios disponibles en otra distinta, ya sea por ser inseguros o difíciles de implementar, por ejemplo, proporcionar acceso a una red corporativa a trabajadores remotos. Debido al encapsulamiento que se lleva a cabo en este proceso, también se consigue ocultar el tráfico que va por el túnel, lo que añade seguridad.

El establecimiento del túnel se implementa incorporando una PDU determinada dentro de otra para transmitirla de un extremo a otro sin que sea necesaria una interpretación intermedia de la PDU encapsulada. Generalmente, el protocolo sobre el que se encapsula opera en una capa igual o superior al protocolo encapsulado.

Existen diversos tipos de túnel, siendo los más populares los túneles SSH, HTTP y L2TP:

- Túnel SSH: Consiste en un túnel encriptado creado mediante una conexión con el protocolo SSH. Se utiliza para enviar información sin encriptar sobre un canal encriptado.
- Túnel HTTP: En estos túneles, el protocolo HTTP actúa como envoltorio en comunicaciones entre varios protocolos de red. Se utiliza generalmente como medio de comunicación desde una localización de red con conectividad restringida, por ejemplo, detrás de NATs o firewalls, o servidores proxy.
- Layer 2 tunneling protocol (L2TP): Se utiliza para dar soporte a redes privadas virtuales (VPNs). No proporciona encriptación o confidencialidad por sí mismo, sino que confía en que por el túnel vaya un protocolo de encriptación que proporcione privacidad, generalmente se utiliza IPsec en conjunto con L2TP para solucionar este problema. Diferencia entre 4 modelos distintos de túnel:
 - Voluntary tunnel.
 - Compulsory tunnel – incoming call
 - Compulsory tunnel – remote dial

- L2TP multihop connection

Otra forma de crear túneles de capa 2 o capa 3 es mediante la creación de interfaces virtuales tun (capa 3) o tap (capa 2) en ambos extremos del túnel y permitir que funcionen con normalidad:

- Interfaces tun: Funcionan en el nivel IP o capa 3 y permiten conexiones punto a punto. Un uso típico es el establecimiento de conexiones VPN. Puesto que son capa 3, solo aceptan paquetes IP, no pueden ser utilizados en bridges y generalmente no soportan broadcasting.
- Interfaces tap: Funcionan en el nivel Ethernet o capa 2, por lo que se comportan con un adaptador de red normal. Puesto que operan en capa 2, aceptan cualquier protocolo de transporte y no se limitan a conexiones punto a punto. Pueden formar parte de un bridge y se utilizan en sistemas de virtualización para proporcionar adaptadores de red virtual a múltiples máquinas. Además, soportan broadcasting.

3.4 Virtualización

Se denomina virtualización a la creación de una versión virtual de un recurso tecnológico, tales como plataformas hardware, sistemas operativos, sistemas de almacenamiento, recursos de red, etc. Existen diversos tipos de virtualización:

- Virtualización hardware o virtualización de plataforma: Consiste en la creación de una máquina virtual que actúa como un ordenador real con un sistema operativo. El software ejecutado en estas máquinas está separado de los recursos hardware subyacentes.
- Virtualización de escritorio: Es el concepto de separar el entorno de escritorio de la máquina física. Un ejemplo de virtualización de escritorio es la infraestructura de escritorio virtual (VDI). En vez de interactuar con el ordenador utilizando un ratón, teclado y monitor directamente, el usuario utiliza otro ordenador o dispositivo móvil conectado al ordenador inicial.
- Virtualización de memoria: Consiste en el desacoplamiento de recursos de memoria RAM de sistemas individuales en un data center y la agregación de esos recursos en una memoria virtual disponible para todos los ordenadores.
- Virtualización de almacenamiento: Se divide a su vez en:
 - Virtualización de bloque: Se refiere a la abstracción de almacenamiento lógico o particiones del almacenamiento físico para que pueda ser accedido sin tener en cuenta el almacenamiento físico.
 - Virtualización de fichero: Responde a los desafíos de NAS (network-attached storage) eliminando las dependencias entre los datos accedidos y la localización física donde están almacenados.
- Virtualización de datos: Hace referencia a cualquier enfoque de gestión de datos que permita a una aplicación recuperar y manipular datos sin tener en cuenta datos técnicos tales como su formato en el origen o donde está localizado físicamente.
- Virtualización de red: Es el proceso de combinar recursos de red hardware y software, así como funcionalidades de red dentro de una sola red virtual. Ejemplos de virtualización de red son las Virtual LANs o las VPNs.

Una máquina virtual es un software que emula a un ordenador y puede ejecutar programas como hace un ordenador real. Una característica esencial de las máquinas virtuales es que los procesos que ejecutan están limitados por los recursos y abstracciones proporcionados por ellas. Se pueden clasificar en dos grandes categorías:

- Máquinas virtuales de sistema: Permiten a la máquina física subyacente multiplicarse entre varias máquinas virtuales, cada una ejecutando su propio sistema operativo. A la capa de software que permite la virtualización se la llama monitor de máquina virtual o hipervisor. Un monitor de máquina virtual puede ejecutarse o bien directamente sobre el hardware o bien sobre un sistema operativo.
- Máquinas virtuales de proceso: Se ejecuta como un proceso normal dentro de un sistema operativo y soporta un solo proceso. La máquina se inicia automáticamente cuando se lanza el proceso que se desea ejecutar y se detiene cuando éste finaliza. Su objetivo es el de proporcionar un entorno de ejecución independiente de la plataforma hardware y el sistema operativo. El ejemplo más conocido de este tipo de máquina virtual es la máquina virtual de Java.

En este trabajo se utilizará una máquina virtual de sistema. Ejemplos de monitores de máquina virtual son:

- VirtualBox: Es un hipervisor diseñado tanto para uso profesional como personal. Actualmente, VirtualBox se puede utilizar en Windows, Linux, Mac y Solaris y soporta un gran número de sistemas operativos para virtualizar. Open source. [\[7\]](#)
- Parallels: Es un hipervisor diseñado para correr un sistema operativo Windows en Mac que permite ejecutar en paralelo y de forma fluida aplicaciones de Windows y Mac con rapidez, control y de forma segura. Cuenta con distintas licencias que van desde 79,99€ hasta 99€/año. [\[8\]](#)
- VMware: Dispone de diversas herramientas de virtualización dependiendo de la solución requerida, por ejemplo, VMware Fusion se recomienda para usuarios domésticos que buscan ejecutar aplicaciones de Windows en un MAC, VMware Workstation Player es una aplicación de virtualización de escritorios que se ejecuta en uno o más sistemas operativos en el mismo ordenador, finalmente, VMware Workstation Pro es la evolución del anterior y está diseñado para profesionales. Dispone de licencias gratuitas de prueba. [\[9\]](#)

En concreto, para la implementación realizada en este trabajo se hará uso de VirtualBox por ser open Source, multiplataforma y satisfacer todas las necesidades del trabajo a pesar de proporcionar resultados peores que VMware por ejemplo.

En la actualidad, no existen soluciones de virtualización de redes inalámbricas, sino se hace uso de soluciones de simulación de red a la hora de hacer pruebas.

En líneas generales, los simuladores de red intentan modelar las redes reales. Puesto que el proceso de modelización es más barato que una implementación real, una gran cantidad de escenarios pueden analizarse con un menor coste. Sin embargo, no es posible modelar perfectamente todos los detalles de una red, y solo muestran los resultados al terminar la simulación, es decir, no permiten interactuar con el dispositivo

simulado en tiempo real. Esa es la mayor diferencia con respecto a la emulación o virtualización de una red. Un emulador de red pretende emular la red a la que se conectan los dispositivos, pero no los dispositivos en sí.

Existen multitud de simuladores de red con características diferentes, sin embargo, puesto que este trabajo no trata sobre simuladores, destacamos OPNET, NS2, NS3, OMNeT ++. OPNET es el único de los 4 que es software comercial. NS2 es el simulador más popular en entornos académicos gracias a ser open source y su gran cantidad de librerías de componentes. Sin embargo, NS2 presenta limitaciones de diseño que se intentan solucionar en NS3, el cual no es solo una actualización de NS2, sino un rediseño completo. Finalmente, OMNeT++ es otro simulador de red popular en entornos académicos que cuenta con una potente interfaz gráfica y un diseño modular, también open source.

- OPNET (actualmente Riverbed Modeler): es software comercial especializado en el desarrollo e investigación de redes. Debido a ser comercial, ofrece herramientas gráficas al usuario relativamente más potentes que las del resto. Se basa en un sistema de eventos discretos, lo que significa que el comportamiento del sistema puede simularse modelando los eventos en el sistema en el orden de los escenarios que el usuario haya establecido. Hace uso de una estructura jerárquica para organizar las redes. También ofrece herramientas de programación para que el usuario defina sus propios formatos. De acuerdo a sus creadores, sus características principales son:
 - Motor rápido de simulación de eventos discretos.
 - Múltiples librerías de componentes con código fuente.
 - Modelado orientado a objetos.
 - Entorno de modelado jerárquico.
 - Soporte para simulaciones inalámbricas escalable.
 - Interfaz gráfica de usuario.
 - Modelado inalámbrico personalizable.
 - Simulaciones de eventos discretos, híbridas y analíticas.
 - Núcleos de simulación paralela para 32 y 64 bits.
 - Soporte para computación en malla.
 - Interfaz abierta para integración de librerías externas.
- Network simulator 2 (NS2): Es un simulador de red open source orientado a objetos basado en eventos discretos. Utiliza tanto C++ como OTcl. C++ se utiliza para implementar el protocolo detallado, mientras que OTcl se utiliza para controlar el escenario de simulación y la programación de eventos.
- Network simulator 3 (NS3): Al igual que NS2, es un simulador de red open source basado en eventos discretos enfocado principalmente a investigación y educación. Sus principales diferencias entre NS2 y NS3 son:
 - Núcleo software: El núcleo de NS3 está escrito en C++ y Python (sustituyendo a OTcl). También se utilizan patrones de diseño avanzados de C++.
 - Atención al realismo: Las entidades de protocolo están diseñadas para ser más cercanas a ordenadores reales.
 - Integración software: Soporta la incorporación de más software de red open source y reduce la necesidad de reescribir modelos para

simulación.

- Soporte para virtualización: Se utilizan máquinas virtuales ligeras.
 - Rastreo de arquitectura: NS3 está desarrollando un framework de localización y recopilación de estadísticas con el fin de permitir personalización de las salidas sin tener que reconstruir el núcleo de simulación.
- OMNeT++: Es una plataforma de simulación de red open source basada en eventos discretos con soporte de interfaz gráfica de usuario (GUI). Se centra en las redes de comunicaciones. Sigue una arquitectura basada en componentes, también llamados módulos, programados en C++. Los componentes se ensamblan en otros componentes y modelos más grandes utilizando un lenguaje de alto nivel. Los módulos son reutilizables y se pueden combinar de diversos modos, lo cual es una de las principales características de OMNeT. Sus componentes principales son:
- Librería de núcleos de simulación.
 - Compilador para el lenguaje de descripción de topología NED.
 - Editor gráfico de red para ficheros NED.
 - GUI para ejecución de simulaciones.
 - Interfaz de usuario basada en comandos para ejecución de simulaciones.
 - Herramienta gráfica de pintado vectorial.
 - Herramienta de documentación de modelos.

Capítulo 4 - Análisis

En este capítulo se realiza un análisis de los aspectos y problemas a tratar a la hora de crear un laboratorio virtual de redes inalámbricas. Concretamente se hablará de modelos de movilidad, tiempos de transmisión y propagación, direccionamiento IP y cobertura, interferencias y ruido.

4.1 Modelos de movilidad

Los dispositivos conectados en una red inalámbrica se mueven de acuerdo a diversos patrones. Por tanto, para realizar simulaciones es necesario definir modelos realistas.

Los modelos de movilidad se diseñan para describir los patrones de movimiento de usuarios móviles y como su posición, velocidad y aceleración cambian a lo largo del tiempo. Podemos clasificarlos en cuatro grandes bloques:

- Modelos aleatorios: En estos modelos, los dispositivos se mueven libremente sin restricciones con independencia de otros nodos. Destacan el “random waypoint model” y sus variantes “random walk model” y “random direction model”. En las siguientes figuras se puede ver el patrón de movimiento de un nodo móvil dentro de un rectángulo de 300 x 600 m utilizando estos modelos:

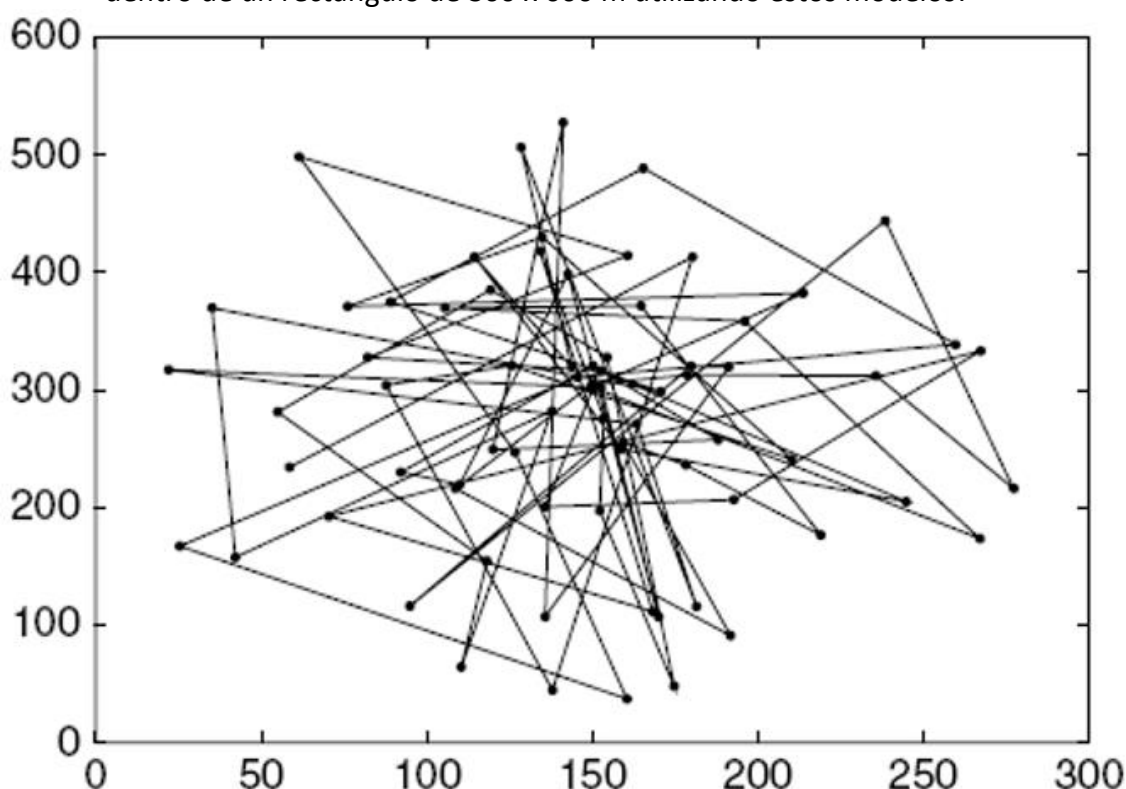


Figura 4.1.1. Patrón de movimiento de un nodo móvil utilizando random walk model. Fuente: King Fahd University of Petroleum & Minerals [\[10\]](#).

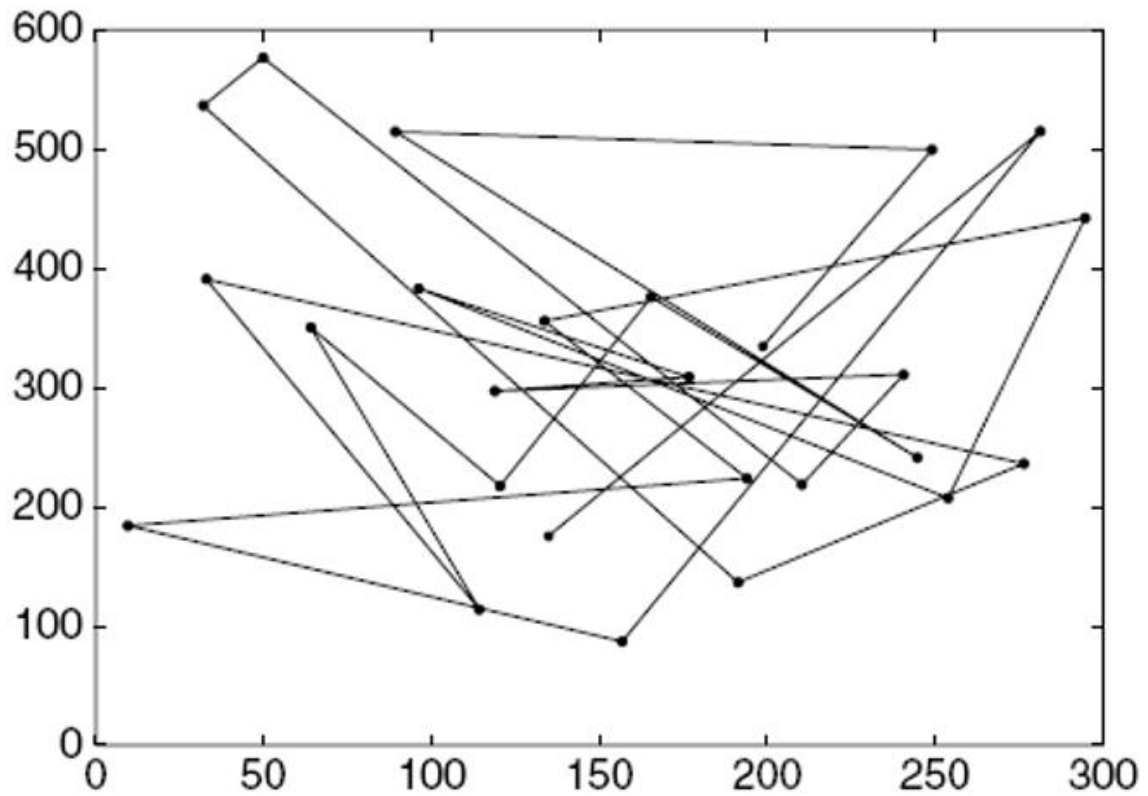


Figura 4.1.2. Patrón de movimiento de un nodo móvil utilizando random waypoint model. Fuente: King Fahd University of Petroleum & Minerals [10].

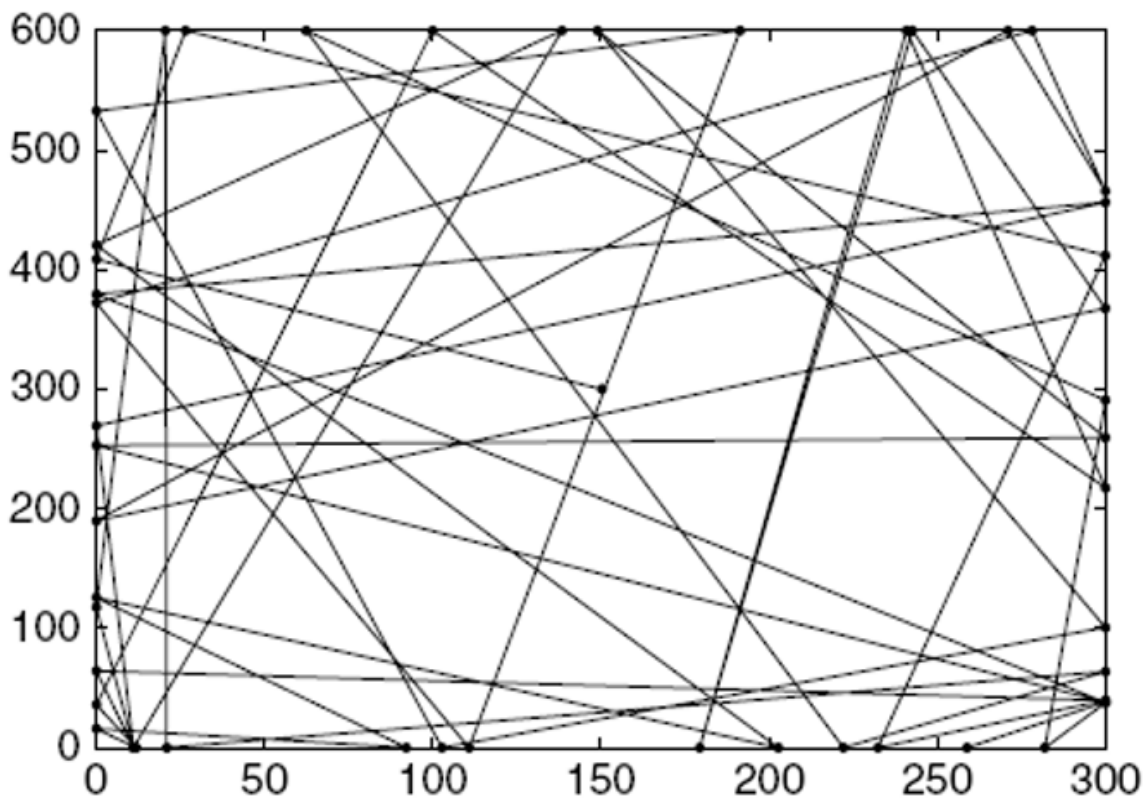


Figura 4.1.3. Patrón de movimiento de un nodo móvil utilizando random direction model. Fuente: King Fahd University of Petroleum & Minerals [10].

- Modelos con dependencia temporal: La movilidad de un dispositivo puede estar limitada por la aceleración, velocidad y cambio de dirección. El cambio de velocidad de un móvil puede depender de su velocidad previa. Además, las velocidades de un dispositivo en diferentes espacios de tiempo están correladas. Estas características se denominan dependencia temporal de la velocidad. Modelos que tienen en cuenta esta dependencia son el “Gauss-Markov mobility model” y el “smooth random mobility model”. En las siguientes figuras se puede ver el patrón de movimiento de un nodo móvil dentro de un rectángulo de 300 x 600 m utilizando Gauss-Markov mobility model:

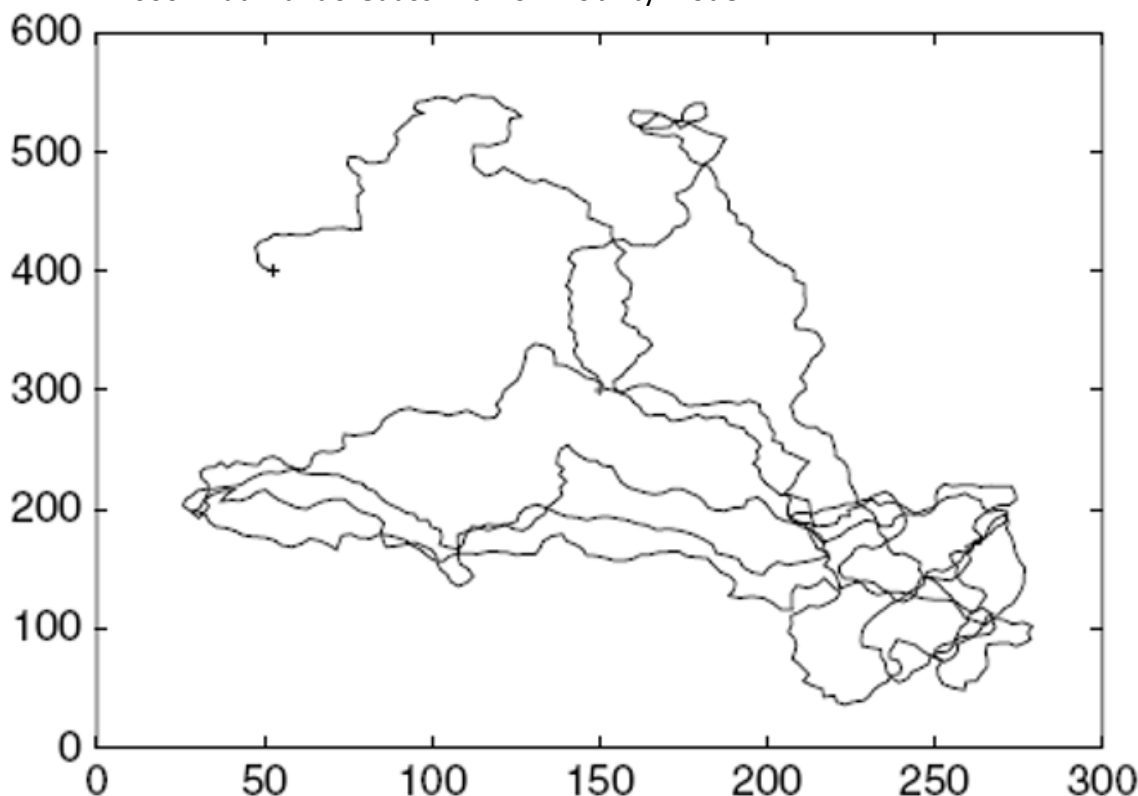


Figura 4.1.4. Patrón de movimiento de un nodo móvil utilizando Gauss-Markov mobility model. Fuente: King Fahd University of Petroleum & Minerals [\[10\]](#).

- Modelos con dependencia espacial: En los modelos aleatorios, los dispositivos no tienen en cuenta otros dispositivos, es decir, la posición, velocidad y dirección de un móvil no se ve afectada por otros dispositivos. Sin embargo, la movilidad de un dispositivo puede ser influenciada por otros dispositivos vecinos. Puesto que las velocidades de distintos dispositivos están correladas en espacio, se denomina a esta característica Dependencia Espacial de la velocidad. Modelos que tienen en cuenta esta dependencia son el “reference point group model” y un grupo de modelos de movilidad correlados denominados que incluyen el “column mobility model”, el “pursue mobility model” y el nomadic community mobility model”. En las siguientes figuras se puede ver el patrón de movimiento de un nodo móvil dentro de un rectángulo de 300 x 600 m utilizando reference point group model, pursue mobility model y nomadic community mobility model:

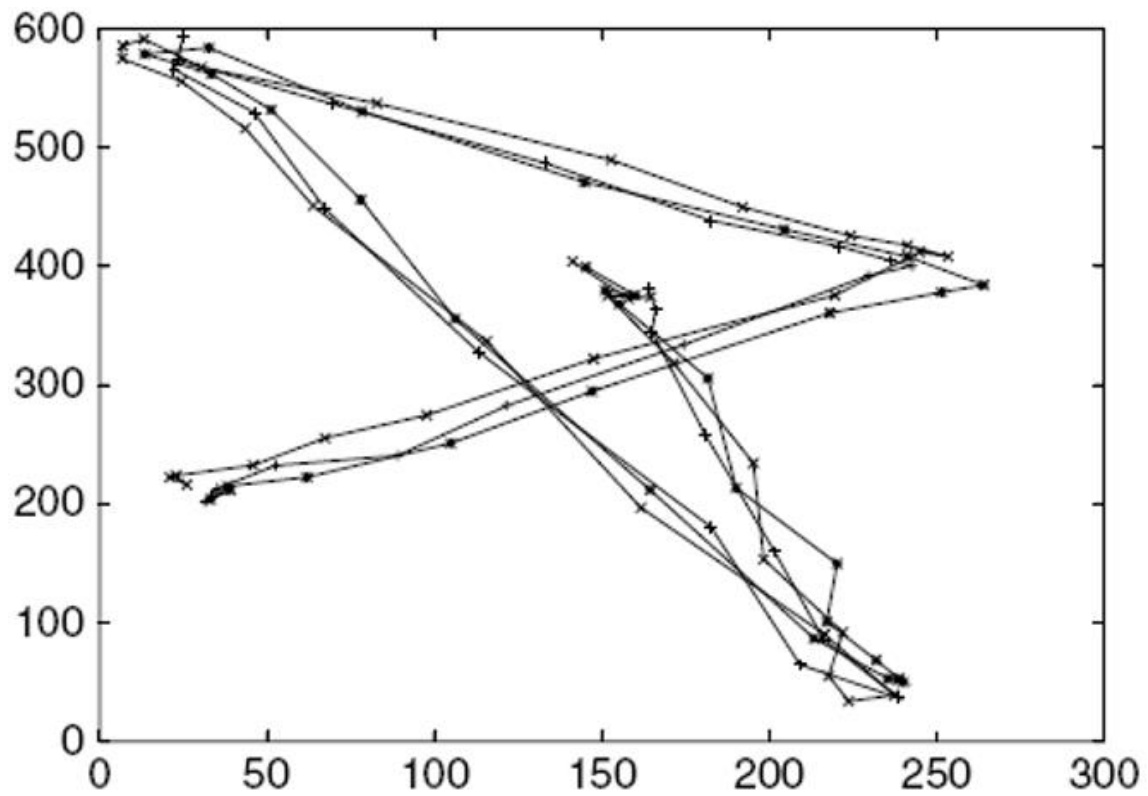


Figura 4.1.5. Patrón de movimiento de un grupo de tres nodos móviles utilizando Reference Point Group mobility model. Fuente: King Fahd University of Petroleum & Minerals [11].

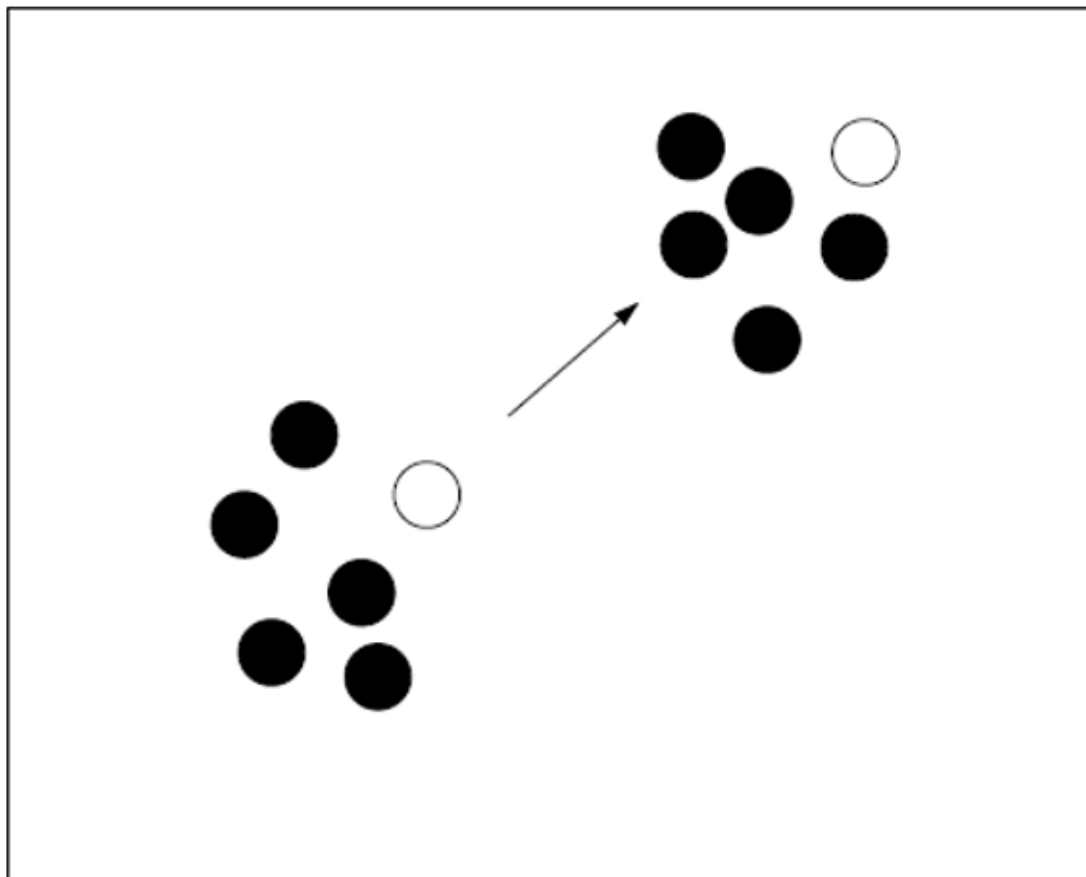


Figura 4.1.6. Patrón de movimiento de un grupo de seis nodos móviles utilizando Pursue mobility model. Fuente: King Fahd University of Petroleum & Minerals [11].

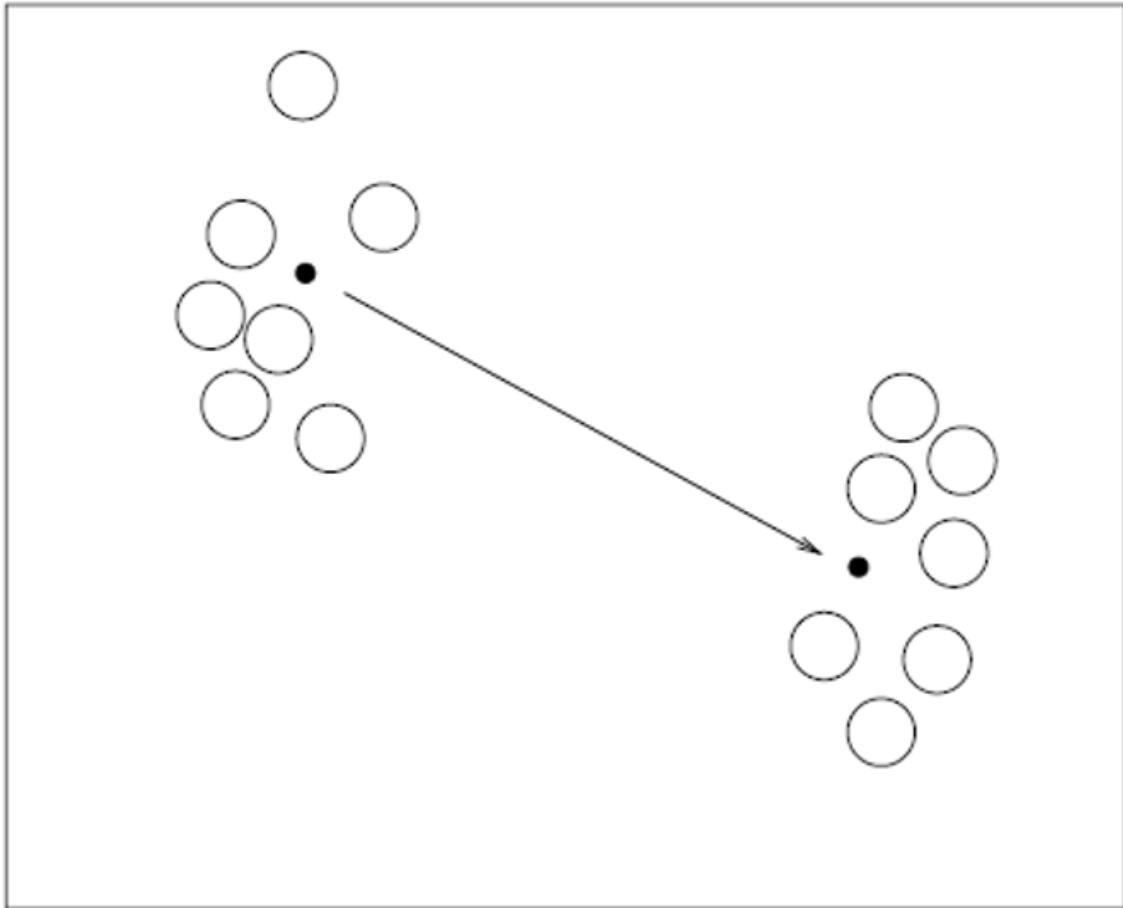


Figura 4.1.7. Patrón de movimiento de un grupo de siete nodos móviles utilizando nomadic community mobility model. Fuente: King Fahd University of Petroleum & Minerals [\[11\]](#).

- Modelos con restricciones geográficas: En los modelos aleatorios se permite a los dispositivos moverse libremente por todo el campo de la simulación. Sin embargo, en la realidad, el movimiento de un dispositivo suele estar sujeto al ambiente que lo rodea. Por tanto, los dispositivos pueden moverse de forma pseudo-aleatoria en caminos predeterminados. Modelos que tienen en cuenta esta característica e integran caminos y obstáculos son el “pathway mobility model” y el “obstacle mobility model”.

Puesto que este trabajo consiste en crear un laboratorio de redes inalámbricas sobre una red cableada, los nodos que forman nuestra red no dispondrán de movilidad por sí mismos. Por tanto, habrá que implementar distintos algoritmos de movilidad y aplicarlos a cada uno de los nodos en caso necesario.

4.2 Transmisión y propagación

El tiempo de transmisión es la cantidad de tiempo que transcurre desde el inicio hasta el final de la transmisión de un paquete, viene definido por el tamaño del paquete enviado y el Bit rate utilizado.

En las redes inalámbricas, la información se envía por aire a la velocidad de la luz, por tanto, los tiempos de propagación de los paquetes dependerán de la distancia a la que se encuentran emisor y receptor.

Se denomina tiempo de entrega de un paquete o latencia al tiempo transcurrido desde que se envía el primer bit y se recibe el último. Se calcula sumando el tiempo de transmisión y el tiempo de propagación.

Puesto que los nodos que compondrán nuestro laboratorio virtual son estáticos, siempre se encontrarán a la misma distancia, por lo tanto, los tiempos de propagación no variarán, para solucionar esto, habrá que asignar una posición a cada uno de los nodos y, en conjunto con los modelos de movilidad, ir calculando la distancia entre los nodos e incluir retrasos en la transmisión para emular los tiempos de propagación. Otro factor a tener en cuenta son los mensajes de difusión. Estos mensajes se envían a todos los nodos de la red, no a uno solo. Para poder enviar este tipo de mensajes, necesitamos que las interfaces de red utilizadas soporten broadcast, es decir, a la hora de utilizar interfaces virtuales, tendremos que utilizar interfaces tap o de capa 2.

4.3 Direccionamiento IP

Una dirección IP es un identificador de una interfaz dentro de una red IP. Consta de dos partes, la primera parte identifica la red y la segunda identifica los equipos dentro de la red. Para separar ambas partes se utiliza la máscara de red. Dicha máscara es una combinación de 32 bits expresados en 4 octetos separados por puntos.

Las direcciones IP se dividen en clases que definen redes grandes (tipo A), medianas (tipo B), pequeñas (tipo C), multicast (tipo D), y experimentales (tipo E).

A su vez, podemos dividir las redes de estas clases en subredes reservando bits del campo host.

Existen protocolos encargados de asignar direcciones IP a los nodos de una red, por ejemplo, DHCP, sin embargo, al crear una red virtual sobre otra red ya existente, tendremos que asignar nosotros las direcciones manualmente a cada uno de los nodos. Para ello, uno de los nodos se hará cargo de ir asignando direcciones de una lista dada a cada uno de los nodos.

4.4 Cobertura, interferencias y ruido

Se define el término cobertura como el área geográfica en la que un dispositivo puede comunicarse. La cobertura depende de diversos factores, tales como la orografía del terreno, estructuras, la tecnología utilizada, la frecuencia de emisión y la eficiencia y sensibilidad del equipo de comunicaciones utilizado.

Se considera que se produce una interferencia cuando dos ondas análogas ocupan el mismo espacio al mismo tiempo. La perturbación resultante es la conjunción de las perturbaciones de cada onda. Diversos ejemplos de interferencias son:

- Interferencia electromagnética: es la perturbación que existe en cualquier

sistema electrónico causada por una fuente de radiación electromagnética.

- Diafonía: Existe diafonía entre dos circuitos cuando parte de las señales presentes en uno de ellos se manifiestan en el otro.
- Interferencia de canal adyacente: es causada por una radiación electromagnética espuria de un canal adyacente como resultado de un filtrado insuficiente, una sincronización incorrecta o un control de frecuencia incorrecto.

Dependiendo de la diferencia de fase de las ondas, se pueden producir los siguientes fenómenos:

- Interferencia constructiva: se produce cuando la diferencia de fases es un múltiplo entero de 2π . En este caso, la amplitud de la onda resultante es la suma de las amplitudes originales.
- Interferencia destructiva: se produce cuando la diferencia de fases es un múltiplo entero impar de π . En este caso, la amplitud de la onda resultante es la resta de las amplitudes originales.

En principio, puede definirse como ruido a cualquier señal indeseable en un sistema de telecomunicaciones. Sin embargo, tal definición es ambigua, puesto que permite categorizar como ruido fenómenos como las interferencias, que, hasta cierto punto, son controlables.

El ruido es un fenómeno natural, inevitable e incontrolable. Dicho de otra forma, el ruido siempre estará presente en un sistema de comunicaciones y contribuirá al deterioro de las señales. Según su origen, el ruido puede clasificarse en:

- Ruido artificial: debido a la actividad humana y originado principalmente en máquinas eléctricas. Este tipo de ruido puede reducirse tanto en la fuente como en el receptor. Puede clasificarse en tres clases principales:
 - Interferencia
 - Zumbido
 - Ruido impulsivo
- Ruido natural: A su vez, se divide en dos grandes grupos:
 - Ruido inherente a los componentes de un circuito o sistema:
 - Ruido térmico
 - Ruido de granalla
 - Ruido de partición
 - Ruido por defecto.
 - Ruido debido a fuentes naturales externas al sistema:
 - Ruido atmosférico
 - Ruido cósmico

Las interferencias y el ruido son factores que disminuyen la cobertura de los dispositivos.

Con el fin de otorgar mayor realismo al Laboratorio, tendremos que emular los efectos del ruido y las interferencias en el rango de cobertura de los dispositivos, además, la movilidad de un nodo también influye en su cobertura, por tanto, habrá que ir calculando para cada nodo con quien tiene conectividad en cada momento, de este

proceso deberá encargarse el nodo responsable de mantener la estructura de la red.

Capítulo 5 - Diseño

En este capítulo se relata la creación de un diseño de laboratorio virtual de redes inalámbricas basado en una red cableada. En primer lugar, se mostrará la estructura de la solución, y, acto seguido, se detallará cada uno de los apartados, siendo estos hosts, conexiones y controller. Finalmente, se explicará el funcionamiento interno del Laboratorio.

La estructura de nuestro laboratorio virtual seguirá el siguiente esquema:

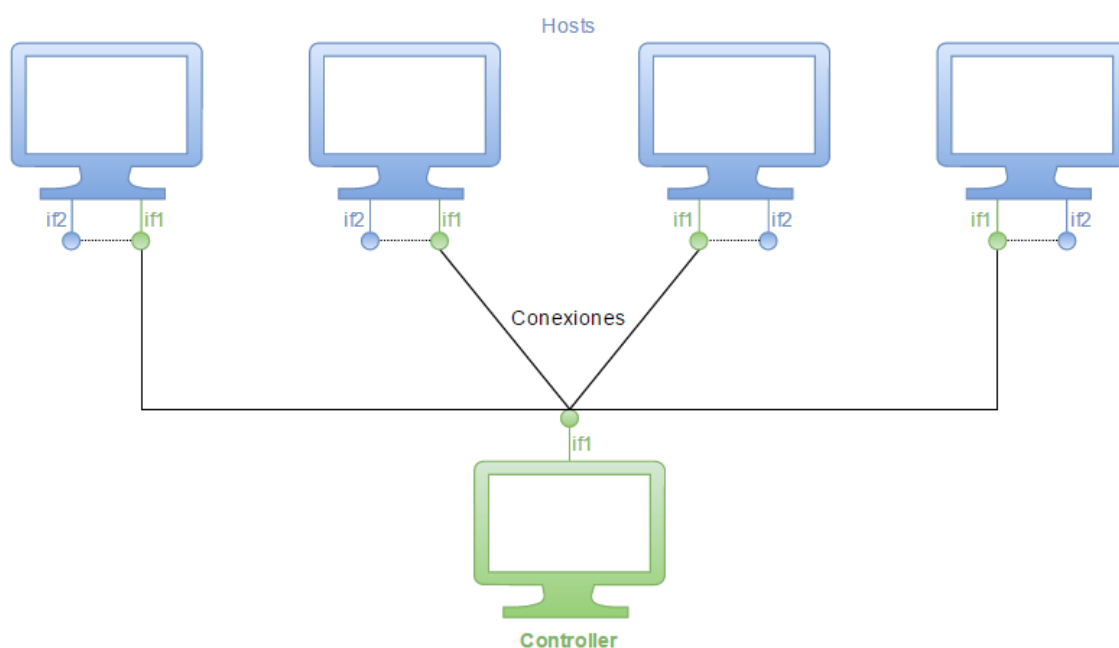


Figura 5.1. Esquema del Laboratorio

De acuerdo a este esquema, los dispositivos en azul son los dispositivos utilizados por el usuario (hosts e if2), mientras que los dispositivos en verde trabajan de forma invisible al usuario final (controller e if1). Podemos dividir el laboratorio en 3 bloques, un bloque formado por N dispositivos denominados hosts, otro bloque formado por el dispositivo denominado controller, y un último bloque, conexiones, que engloba las conexiones entre interfaces dentro de los hosts y entre hosts y controller.

A su vez, este esquema se interconectará a través del controller con una interfaz web mediante un archivo JSON. Esta interfaz web y la comunicación ya se ha implementado en un PFC anteriormente defendido “Desarrollo de una herramienta de simulación para un laboratorio MANET virtual con AODV”, realizado por Ángel Hita Albarracín y dirigido por Gabriel Maciá Fernández y Rafael Alejandro Rodríguez Gómez para la Universidad de Granada. La interconexión entre ambos trabajos se comenta en el apartado futuras ampliaciones del capítulo 8.

5.1 Hosts

Llamaremos hosts a los dispositivos con los que trabaje el usuario, ya sean ordenadores, móviles, máquinas virtuales, etc. Para cumplir con su cometido, necesitarán al menos 2 interfaces de red, una conectada en la misma red que el resto de dispositivos (if1), y otra que será la que se utilice para la red inalámbrica virtual (if2). Los hosts dispondrán de un programa que se encargará, en primer lugar, de establecer y mantener la conexión con el controller y, a continuación, configurar la interfaz que vamos a utilizar con la información que el controller le proporcione.

5.2 Controller

Denominaremos controller al dispositivo encargado de configurar la red deseada y emular las características de una red inalámbrica. Para ello contará con un programa al que habrá que hacerle saber la estructura de red, es decir, cuántos hosts forman el laboratorio y con quien podrá comunicarse directamente cada uno de ellos. Dicho programa será también el encargado de virtualizar las funciones de una red inalámbrica, tales como la movilidad, los distintos tiempos de propagación, etc. Además, se conectará a cada uno de los hosts para proporcionarles la información de configuración necesaria para sus interfaces y, finalmente, se encargará de reenviar los paquetes que le envíen cada uno de los hosts hacia el resto, respetando la estructura de red previamente administrada.

5.3 Conexiones

En este bloque describiremos las distintas conexiones entre las interfaces de los dispositivos. Las interfaces denominadas if2 serán las que funcionen como interfaces inalámbricas, por ello no estarán físicamente conectadas a ninguna red. Estas interfaces podrán ser físicas o virtuales y serán con las que trabaje el usuario. Por otro lado, las interfaces if1 serán interfaces ethernet conectadas todas a una misma red cableada.

En primer lugar, habrá que establecer una comunicación entre las interfaces if1 de los hosts y el controller, comunicación que se mantendrá abierta con el fin de retransmitir los paquetes de los Hosts hacia sus destinos.

Una vez establecida dicha comunicación, el programa de los hosts se encargará de crear un túnel que una la red cableada sobre la que funciona el Laboratorio y la red inalámbrica que estamos virtualizando, para ello, se creará una conexión entre las interfaces “inalámbricas” if2, y las interfaces ethernet if1, con el fin de que los hosts puedan enviar y recibir paquetes hacia y desde otros Hosts.

5.4 Funcionamiento

Una vez definidas las funciones de los 3 bloques, es hora de explicar el funcionamiento interno del laboratorio.

En primer lugar, destacar que como se ha mencionado previamente, las interfaces if2 serán las interfaces inalámbricas, es decir, las interfaces con las que trabajará el usuario.

Cada vez que el usuario envíe información a través de estas interfaces, el programa de los hosts cogerá esta información, la encapsulará y se la enviará al controller a través de su interfaz if1.

Una vez en el controller, éste consultará las vecindades de ese host, es decir, con quien tiene conexión el host emisor, y retransmitirá el paquete hacia el/los host/s receptores.

Finalmente, cuando los hosts reciban un paquete por su interfaz if1, lo desencapsulará y se lo pasará a su interfaz if2, que procesará el paquete y actuará en consecuencia.

Todo este proceso es invisible para el usuario final, que trabajará con su dispositivo como si estuviera en una red inalámbrica estándar. A continuación, se incluyen los diagramas de flujo del proceso seguido por los hosts y por el controller cada vez que les llega un paquete:

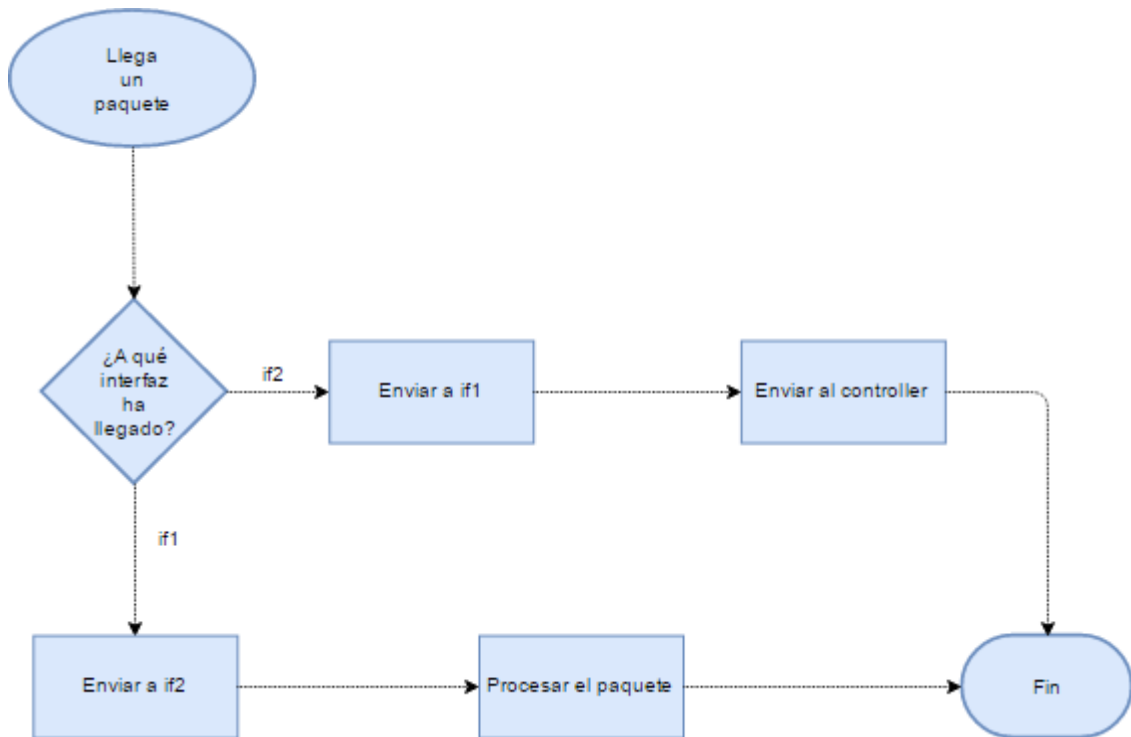


Figura 5.4.1 Diagrama de flujo de los hosts

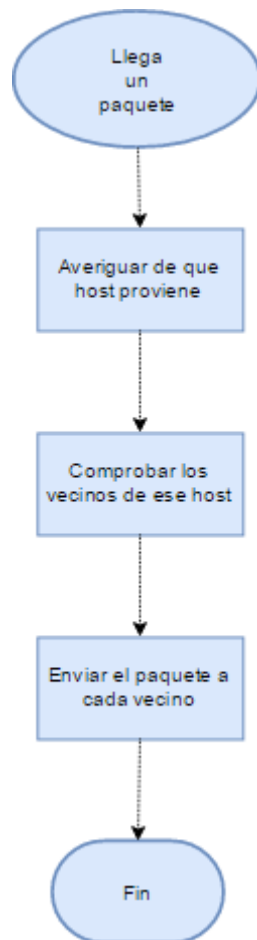


Figura 5.4.2 Diagrama de flujo del controller

Capítulo 6 - Implementación

En este capítulo se relata la implementación de un prototipo basado en el diseño del capítulo 5. Dicho prototipo no tendrá en cuenta todos los aspectos relatados en el capítulo 4, pero que permitirá construir un laboratorio virtual de redes ad hoc inalámbricas. Primero se detalla la estructura seguida, y, a continuación, se profundiza en el contenido de los hosts y el controller. Finalmente, se describirá el proceso a seguir para realizar una instalación de esta implementación.

Para llevar a cabo la creación del prototipo se van a utilizar una serie de máquinas virtuales conectadas siguiendo la siguiente estructura:

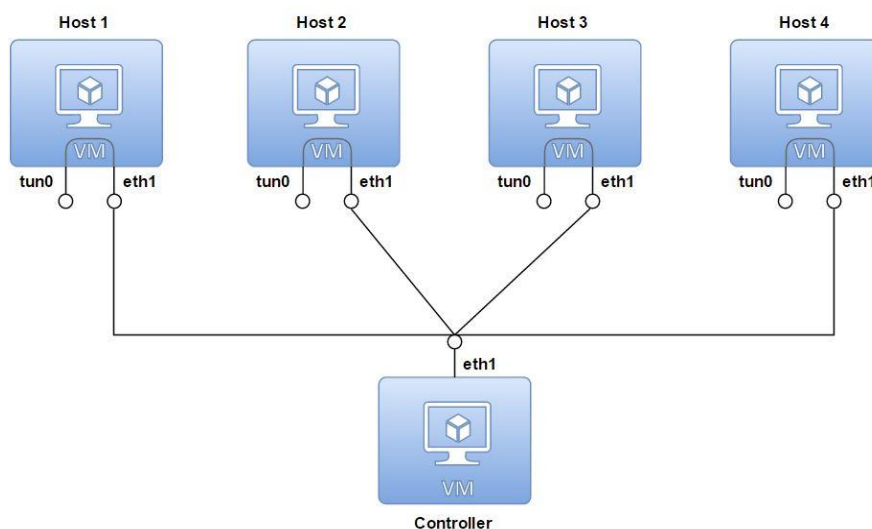


Figura 6.1. Estructura de red

En este ejemplo podemos ver un total de 5 máquinas virtuales, 4 que actuarán como hosts y una que actuará como controller. Para esta implementación se ha decidido utilizar la herramienta VirtualBox por ser multiplataforma y gratuita, lo que permitirá que sea más accesible de cara al usuario. Por otro lado, se ha elegido utilizar Ubuntu 10.04 como sistema operativo, ya que cumple con los requisitos necesarios para el correcto funcionamiento del laboratorio.

En primer lugar, hay que mencionar que todas las máquinas deben estar conectadas en la misma red. En el caso de este trabajo, se conectaron mediante un sistema de Host-only networking, que permite crear una red que contiene a la máquina física y a las máquinas virtuales, conectándolas todas entre sí [\[12\]](#). A continuación, podemos apreciar en el esquema que los hosts hacen uso de otra interfaz denominada tun0 conectada a la

interfaz eth1. Estas interfaces virtuales las creará el programa Python desarrollado y son las que utilizará el usuario para comunicarse con el resto de hosts. Estas interfaces serán interfaces TAP, es decir, interfaces que simularán capa 2, lo que permitirá la transmisión de mensajes broadcast como los RREQ de AODV. La otra opción es utilizar interfaces TUN, sin embargo, estas interfaces son de capa 3 solamente. El programa Python de los hosts será el encargado de crear el túnel entre las interfaces tun0 y las eth1 dentro de los hosts. Las conexiones entre las interfaces eth1 de los hosts con la interfaz eth1 del controller se realizarán mediante sockets. El programa de los hosts será el que cree el socket en primer lugar y lo ponga a la espera hasta que el programa del controller cree su socket y establezca conexión con cada uno de los hosts. Estos sockets serán sockets UDP, puesto que se desea que su efecto sea mínimo en las aplicaciones que puedan ejecutar los usuarios. En el caso de utilizar TCP, podría darse la situación de utilizar tunneling TCP sobre una aplicación que ya utiliza TCP, con resultados perjudiciales para la aplicación [13]. Finalmente, queda crear el túnel entre las interfaces tun0 y las eth1, para ello, para realizar las tareas de lectura y escritura en las interfaces se ha optado por utilizar la herramienta `select[]` de Python en vez de `thread[]`, puesto que esta última añade una complejidad innecesaria, además de en el caso de leer de sockets e interfaces, la herramienta `select[]` es más rápida.

Para emular la conectividad de una red ad hoc inalámbrica se hará uso de una clase específicamente creada para este proyecto y de un archivo JSON de entrada con la estructura de la red deseada, así como de dos programas principales, uno para las máquinas que actúan de hosts y otro para la máquina que ejerza de controller. Además, en los hosts se ejecutará la herramienta AODV-UU, el cual se trata de una implementación del protocolo AODV realizada por la Universidad de Upsala [14].

6.1 Hosts

En primer lugar, los requisitos que deben cumplir las máquinas que actuarán como hosts son los siguientes:

- Sistema operativo Linux con kernel v2.6.x
- Módulo netfilter instalado
- Instalar AODV-UU
- Python 2.x

Para este proyecto se han utilizado máquinas virtuales con Ubuntu 10.04 por cumplir los requisitos anteriormente citados.

A continuación, se procede a detallar el contenido del archivo `host.py` creado para este proyecto:

En primer lugar, nos encontramos con las importaciones de librerías necesarias para el funcionamiento del programa, seguidas de la función de ayuda y la declaración de las variables globales del programa:

```
import os, sys
from socket import *
from fcntl import ioctl
from select import select
```



```

import getopt, struct
import subprocess

#Help function
def usage(status=0):
    print "Usage: host.py"
    sys.exit(status)

#Global variables
MAGIC_WORD = "VirtualWLANLab"
TUNSETIFF = 0x400454ca
IFF_TAP = 0x0002
TUNMODE = IFF_TAP
PORT = 50000

```

La función de ayuda imprimirá por pantalla la línea de comandos que hay que introducir para iniciar el programa, mientras que las variables globales sirven para:

- MAGIC_WORD: Esta variable contiene una contraseña que habrá que compartir y comparar con la del controller.
- TUNSETIFF, IFF_TAP, TUNMODE: Variables con códigos hexadecimales utilizadas para la creación de la interfaz tun0.
- PORT: Puerto en el que estará escuchando el programa para que se conecte el controller.

Acto seguido nos encontramos con el parseo de argumentos, que, en el caso de este programa, el único admitido será “-h” para invocar la función de ayuda:

```

#Parsing the arguments
opts = getopt.getopt(sys.argv[1:], "h")

for opt, optarg in opts[0]:
    if opt == "-h":
        usage()

```

Una vez terminado el parseo, se procede a crear la nueva interfaz tipo TAP y a darle un nombre, en este caso, tun0, sin embargo, para que esté totalmente operativa necesita una dirección IP que se le asignará más adelante:

```

#Creating the new interface
f = os.open("/dev/net/tun", os.O_RDWR)
ifs = ioctl(f, TUNSETIFF, struct.pack("16sH", "tun%d",
TUNMODE))
ifname = ifs[:16].strip("\x00")

```

Después de crear la interfaz, se realiza la conexión con el controller, para ello, en primer lugar, se crea el socket UDP y a continuación se pone a escuchar en el puerto especificado hasta que el controller intente conectar:

```

#Preparing socket
s = socket(AF_INET, SOCK_DGRAM)

```

El primer intercambio entre el controller y el host es la `MAGIC_WORD`, que deberá coincidir entre ambos dispositivos para poder conectar correctamente. Una vez hecho este intercambio, se solicita al controller una dirección IP de su lista de direcciones y, al recibirla, se asignará a la interfaz `tun0` creada anteriormente:

```
#Establishing connection with the controller
try:
    s.bind(("", PORT))
    while 1:
        word,peer = s.recvfrom(1500)
        if word == MAGIC_WORD:
            break
        print "Bad magic word for %s:%i" % peer(i)
    s.sendto(MAGIC_WORD, peer)
    #Retreiving IP from the controller and assign it to the
    new interface
    ip,peer = s.recvfrom(1500)
    print "Assigned to interface %s" %ifname + " the IP
    address: %s" %ip
    os.system("ifconfig %s" %ifname + " %s" %ip)
    print "Connection with %s:%i established" % peer
```

Al llegar a este punto ya tendremos una interfaz totalmente operativa, sin embargo, hay que “conectarla” a la interfaz `eth1` que es la interfaz utilizada para conectarse al controller:

```
#Starting the wireless emulator functions (host side)
while 1:
    r = select([f,s],[],[])[0][0]
    if r == f:
        s.sendto(os.read(f,1500),peer)
    else:
        buf,p = s.recvfrom(1500)
        os.write(f, buf)
except KeyboardInterrupt:
    print "Stopped by user."
```

Para realizar dicha “conexión” se utilizará la herramienta `select[]`, que utilizaremos para leer tanto la interfaz como el socket.

En primer lugar, si recibimos algún paquete en la interfaz, es decir, información que se desea enviar a otros hosts, esa información se recogerá y se enviará por el socket al controller, que será el encargado de reenviarla a los hosts vecinos con los que tengamos conectividad.

Por otro lado, si se recibe algo por el socket, es decir, información de otros hosts enviada por el controller, esa información se recogerá y se escribirá en la interfaz.

Con esto termina el programa diseñado, sin embargo, este programa por sí mismo solamente permite comunicarnos con nuestros vecinos directos, no con todos los hosts de la red. Para ello se hará uso del protocolo AODV mediante AODV-UU [\[14\]](#).

6.2 Controller

En primer lugar, los requisitos que debe cumplir la máquina que actuará como controller son los siguientes:

- Sistema operativo Linux
- Python 2.x

Para este proyecto se han utilizado máquinas virtuales con Ubuntu 10.04 por cumplir los requisitos anteriormente citados.

El controller está formado por tres archivos desarrollados para este proyecto:

- Un archivo JSON data.json que contiene la estructura de la red a emular.
- Un archivo Python NetScheme.py que servirá para almacenar la información de la estructura proporcionada por el JSON
- Un archivo Python controller.py que se encargará de conectarse con los hosts y ejercer de intermediario entre ellos asegurando la conectividad especificada en el archivo data.json.

En primer lugar se procede a detallar el contenido de los archivos citados anteriormente, empezando por el archivo data.json. Este archivo está formado por dos líneas:

```
{
    "hosts": [1, 2, 3, 4],
    "neighbors": [[2, 3], [1, 3, 4], [1, 2], [2]]
}
```

- La primera de ella, denominada “hosts”, contiene una lista de los hosts que forman nuestra red, numerados con un número natural siguiendo el orden que el usuario desee.
- La segunda línea, denominada “neighbors”, contiene una lista de las listas de vecinos con los que tiene conectividad cada uno de los hosts de la primera lista identificados por su número y siguiendo el mismo orden, es decir, en el caso del JSON anterior:
 - El host 1 tendrá conectividad directa con los hosts 2 y 3
 - El host 2 tendrá conectividad directa con los hosts 1, 3 y 4
 - El host 3 tendrá conectividad directa con los hosts 1 y 2
 - El host 4 tendrá conectividad directa con el host 2

A continuación, se indica el contenido del archivo NetScheme.py:

```
class NetScheme:
    netList={}

    def __init__(self, hosts, neighbors):
        j=0
        for i in hosts:
```

```

        self.setNeighbors(i,neighbors[j])
        j=j+1

def addHost(self,host):
    self.netList[host]=[]

def getHost(self,i):
    return self.netList[i]

def delHost (self,host):
    self.netList.pop(host)

def getHosts (self):
    return self.netList.keys()

def printHosts(self):
    print(self.netList.keys())

def setNeighbors(self,host,neighbors):
    self.netList[host]=[neighbors]

def getNeighbors(self,host):
    return self.netList[host]

def delNeighbors(self,host):
    self.netList[host]=[]

def printScheme(self):
    for j in self.netList:
        print"Host: %i" %j + " Neighbors: ",
self.netList[j]

```

Este archivo está constituido por una clase denominada `NetScheme` compuesta únicamente por una `netList` de Python y funciones para trabajar con ella. El objetivo de esta clase es extraer inicialmente la información del JSON y adaptarla al lenguaje Python para poder trabajar con el esquema de red cómodamente:

En primer lugar, nos encontramos con el constructor de la función, al cual habrá que pasarle las listas de hosts y vecinos asociados y las añadirá a la `netList`.

Acto seguido nos encontramos con la lista de funciones encargadas de trabajar con la clase:

- `addHost`: se encarga de añadir un host a nuestra clase
- `getHost`: devuelve el identificador de un host concreto
- `delHost`: elimina un host de la lista mediante su identificador
- `getHosts`: devuelve la lista completa de hosts y sus identificadores
- `printHosts`: imprime la lista completa de hosts y sus identificadores
- `setNeighbors`: añade la lista de vecinos de un host concreto
- `getNeighbors`: devuelve la lista de vecinos de un host concreto

- delNeighbors: elimina la lista de vecinos de un host concreto
- printScheme: imprime el esquema completo de hosts y sus vecinos asociados.

Finalmente se detalla el contenido del archivo controller.py que hará uso de los dos archivos anteriores para crear la red:

En primer lugar, nos encontramos con las funciones necesarias para el parseo del JSON, la función de ayuda y las variables globales del programa:

```
import os, sys
from socket import *
from fcntl import ioctl
from select import select
import getopt, struct
import json
from NetScheme import NetScheme

#Json's parsing functions
def _decode_list(data):
    rv = []
    for item in data:
        if isinstance(item, unicode):
            item = item.encode('utf-8')
        elif isinstance(item, list):
            item = _decode_list(item)
        elif isinstance(item, dict):
            item = _decode_dict(item)
        rv.append(item)
    return rv
def _decode_dict(data):
    rv = {}
    for key, value in data.iteritems():
        if isinstance(key, unicode):
            key = key.encode('utf-8')
        if isinstance(value, unicode):
            value = value.encode('utf-8')
        elif isinstance(value, list):
            value = _decode_list(value)
        elif isinstance(value, dict):
            value = _decode_dict(value)
        rv[key] = value
    return rv

#Help function
def usage(status=0):
    print "Usage: controller.py -t  
[ipTarget1,ipTarget2,...,ipTargetn]"
    print "You need to have a valid json in the same folder  
than controller.py"
    sys.exit(status)

#Global variables
MAGIC WORD = "VirtualWLANLab"
PORT = 50000
socketList = [0] * 256
peer = [0] * 256
#List of IPs that the controller will provide to the hosts.  
User configurable.
ip=["10.0.0.1/24","10.0.0.2/24","10.0.0.3/24","10.0.0.4/24"  
,"10.0.0.5/24","10.0.0.6/24","10.0.0.7/24"]
```

La función de ayuda imprimirá por pantalla la línea de comandos que hay que introducir para iniciar el programa, mientras que las variables globales sirven para:

- **MAGIC WORD:** Esta variable contiene una contraseña que habrá que compartir y comparar con la del host.
- **PORT:** El puerto en el que estarán escuchando los hosts.
- **socketList:** Una lista en la que se guardarán los sockets utilizados, uno con cada host.
- **peer:** Una lista en la que se guardará la información de cada dupla IP, Port al que nos conectamos.
- **ip:** Una lista personalizable que contiene las distintas direcciones IP que se irán asignando a los hosts. Es personalizable por el usuario.

Acto seguido se encuentran las funciones de parseo de argumentos y del JSON. En el caso del controller, habrá que pasarle como argumento la lista de direcciones IP de los hosts a los que debe conectarse y deberán situarse los archivos data.json y NetScheme.py en la misma carpeta que controller.py:

```
#Parsing the arguments
opts = getopt.getopt(sys.argv[1:], "t:h")

for opt, optarg in opts[0]:
    if opt == "-h":
        usage()
    elif opt == "-t":
        target = optarg.split(",")

#Parsing the json
with open('data.json') as data_file:
    data = json.load(data_file, object_hook=_decode_dict)

scheme=NetScheme(data["hosts"],data["neighbors"])
```

Una vez parseadas las entradas, se rellenan las listas socketList y peer con los sockets UDP y las duplas IP,Port de cada uno de los hosts. Hecho esto, se procede a conectarse con los hosts:

```
#Preparing the peers list and the sockets using the hosts
in the scheme provided
j=0
for i in scheme.netList.keys():
    socketList[i] = socket(AF_INET, SOCK_DGRAM)
    peer[i]=(target[j],PORT)
    print (peer[i])
    j+=1

#Establishing connections with the hosts
try:
    j=0
    for i in scheme.netList.keys():
        socketList[i].sendto(MAGIC_WORD, peer[i])
        word,peer[i] = socketList[i].recvfrom(1500)
        if word != MAGIC_WORD:
            print "Bad magic word for %s:%i" % peer[i]
            sys.exit(2)
            ##Sending IP to the host's new interface
            socketList[i].sendto(ip[j], peer[i])
            print "Connection with %s:%i established" %peer[i]
            j+=1
    print("Controller ready")
```

En primer lugar, se envía la MAGIC_WORD y se recibe otra desde el host, si ambas coinciden, se establece la conexión y se envía al host la dirección IP de su interfaz tun0.

Este proceso se realiza para cada uno de los hosts que forman la red.

Al llegar a este punto, ya están realizadas las conexiones entre los hosts y el controller y se tiene el esquema de red almacenado. Finalmente queda realizar las funciones de controller, para lo que se utilizará la herramienta `select[]`, a la que se le pasa la lista de sockets previamente creados:

```
#Starting the emulator functions (controller side)
while 1:
    inp, outp, exc = select(socketList, [], [])
    inp=inp[0]
    buf, p = inp.recvfrom(1500)
    i=0
    while inp!=socketList[i]:
        i+=1
    aux=scheme.getNeighbors(i)[0]
    for j in aux:
        socketList[j].sendto(buf, peer[j])

except KeyboardInterrupt:
    print "Stopped by user."
```

Una vez llegados a este punto, cada vez que se recibe algo por uno de los sockets, se almacena, se comprueba de que host proviene y se obtiene su lista de vecinos y se procede a reenviar el paquete a cada uno de ellos.

6.3 Instalación

Para realizar una instalación de esta implementación hay que llevar a cabo los siguientes pasos:

- Instalar VirtualBox y crear una máquina virtual para el controller y tantas máquinas para los hosts como se deseen, e instalar Ubuntu 10.04.
- Instalar AODV-UU [\[14\]](#) en las máquinas que actuarán como hosts.
- Descargar los programas explicados anteriormente del repositorio del trabajo. [\[15\]](#)
- Introducir el contenido de la carpeta Host en las máquinas que actuarán como hosts, y el contenido de la carpeta Controller en la máquina que actuará de controller.
- Modificar el archivo data.json con la estructura de red deseada.
- Activar el reenvío de paquetes en los hosts, para ello ejecutamos el comando “`sudo gedit /etc/sysctl.conf`” y descomentamos la línea `net.ipv4.ip_forward=1` si está comentada.
- Iniciar el programa host.py en los hosts mediante el comando `sudo “python host.py”`
- Iniciar el programa controller.py en el controller mediante el comando “`sudo python controller.py -t ipHost1,ipHost2,...,ipHostN`”
- Activar AODV en los hosts mediante los comandos “`sudo modprobe kaodv`” y “`sudo aodvd i tun0 -l -r 2 -w`”

Capítulo 7 - Evaluación

En este capítulo se realizarán diferentes pruebas en aras de comprobar el correcto funcionamiento de la implementación. Concretamente, en primer lugar, se realizará una prueba en la que participarán 2 hosts y el controller, con el fin de comprobar que la conectividad funciona correctamente. A continuación, una prueba con 4 hosts, pero sin utilizar AODV, para comprobar que no exista comunicación entre hosts que no deban tenerla. Finalmente, se repetirá la prueba anterior utilizando AODV para así verificar que todo funciona correctamente.

Tanto el controller como los hosts serán máquinas virtuales con el sistema operativo Ubuntu 10.04 y la herramienta utilizada es VirtualBox.

7.1 Conectividad entre 2 Hosts

En esta prueba se conectarán las interfaces tun0 de 2 hosts mediante el controller con el fin de comprobar que las conexiones, establecimientos de direcciones IP y la retransmisión de mensajes se llevan a cabo correctamente.

El escenario es el siguiente:

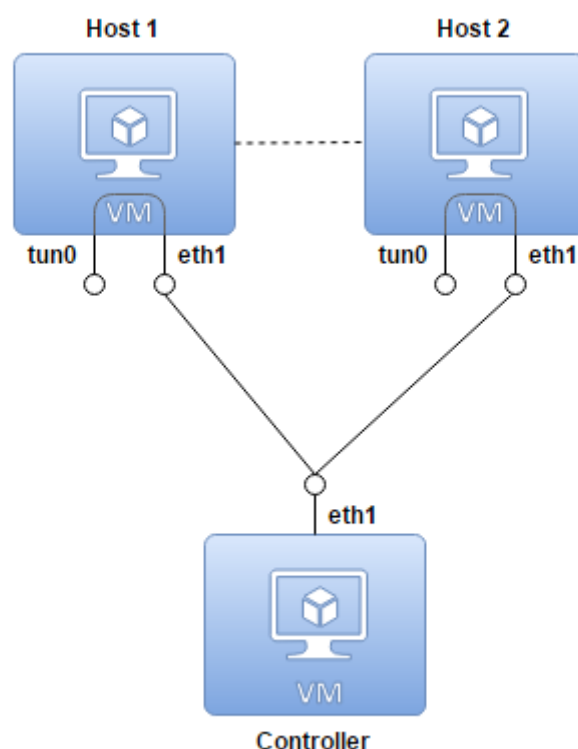


Figura 7.1.1 Escenario Evaluación 1

La interfaz eth1 del controller tendrá asociada la dirección IP 192.168.56.101/24, la interfaz eth1 del Host 1 será 192.168.56.102/24 mientras que la del Host 2 será 192.168.56.103/24. Por otro lado, a la interfaz tun0 del host 1 se le asignará la dirección 10.0.0.1/24 y a la interfaz tun0 del host 2 la dirección 10.0.0.2/24.

En primer lugar, configuramos el esquema de la red en el archivo JSON de tal forma que se corresponda con el escenario representado:

```
{
    "hosts": [1, 2],
    "neighbors": [[2], [1]]
}
```

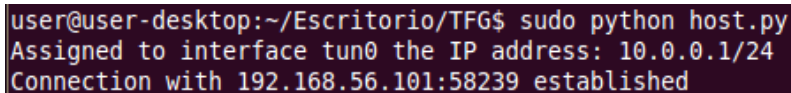
A continuación, lanzamos el archivo `host.py` en los dos hosts que forman parte de la prueba mediante el siguiente comando:

- `sudo python host.py`

Acto seguido, ejecutaremos en el controller el archivo `controller.py`:

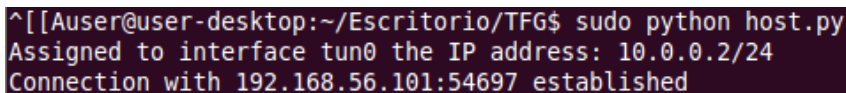
- `sudo python controller.py -t 192.168.51.102,192.168.56.103`

Una vez hecho esto, si todo ha funcionado correctamente, podremos ver los siguientes mensajes en las respectivas consolas de Linux:

A terminal window showing the execution of 'sudo python host.py'. The output indicates that the IP address 10.0.0.1/24 has been assigned to the tun0 interface and that a connection with 192.168.56.101:58239 has been established.

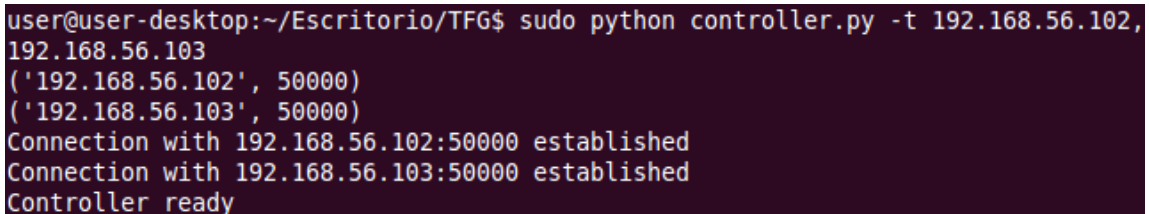
```
user@user-desktop:~/Escritorio/TFG$ sudo python host.py
Assigned to interface tun0 the IP address: 10.0.0.1/24
Connection with 192.168.56.101:58239 established
```

Figura 7.1.2 Consola host 1

A terminal window showing the execution of 'sudo python host.py'. The output indicates that the IP address 10.0.0.2/24 has been assigned to the tun0 interface and that a connection with 192.168.56.101:54697 has been established.

```
^[[Auser@user-desktop:~/Escritorio/TFG$ sudo python host.py
Assigned to interface tun0 the IP address: 10.0.0.2/24
Connection with 192.168.56.101:54697 established
```

Figura 7.1.3 Consola host 2

A terminal window showing the execution of 'sudo python controller.py -t 192.168.56.102, 192.168.56.103'. The output shows the controller receiving connections from both hosts and becoming ready.

```
user@user-desktop:~/Escritorio/TFG$ sudo python controller.py -t 192.168.56.102,
192.168.56.103
('192.168.56.102', 50000)
('192.168.56.103', 50000)
Connection with 192.168.56.102:50000 established
Connection with 192.168.56.103:50000 established
Controller ready
```

Figura 7.1.4 Consola controller

Para entrar más en detalle acerca del proceso de conexión, estudiaremos el intercambio de mensajes entre Host 1 y el Controller. En primer lugar, ambas Máquinas Virtuales se han enviado una “MAGICWORD” para identificarse, concretamente, “VirtualWLANLab”:

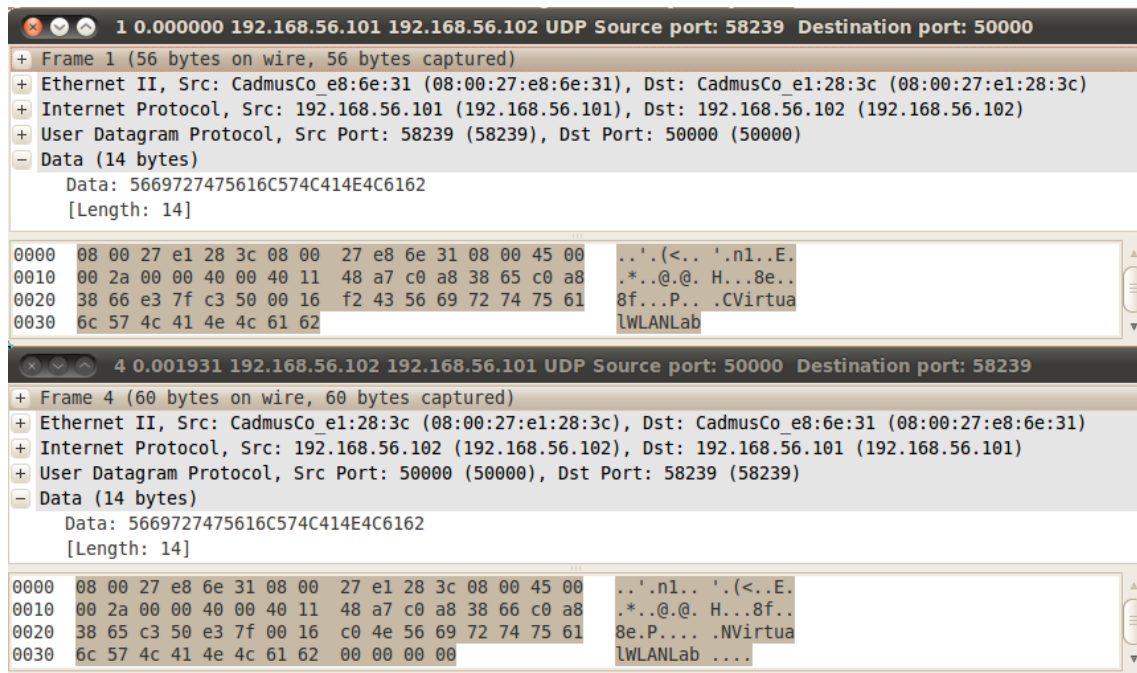


Figura 7.1.5 Intercambio magic word entre controller y host 1

Una vez realizado el intercambio, el controller enviará al host 1 un mensaje que contendrá la dirección IP para su nueva interfaz tun0:

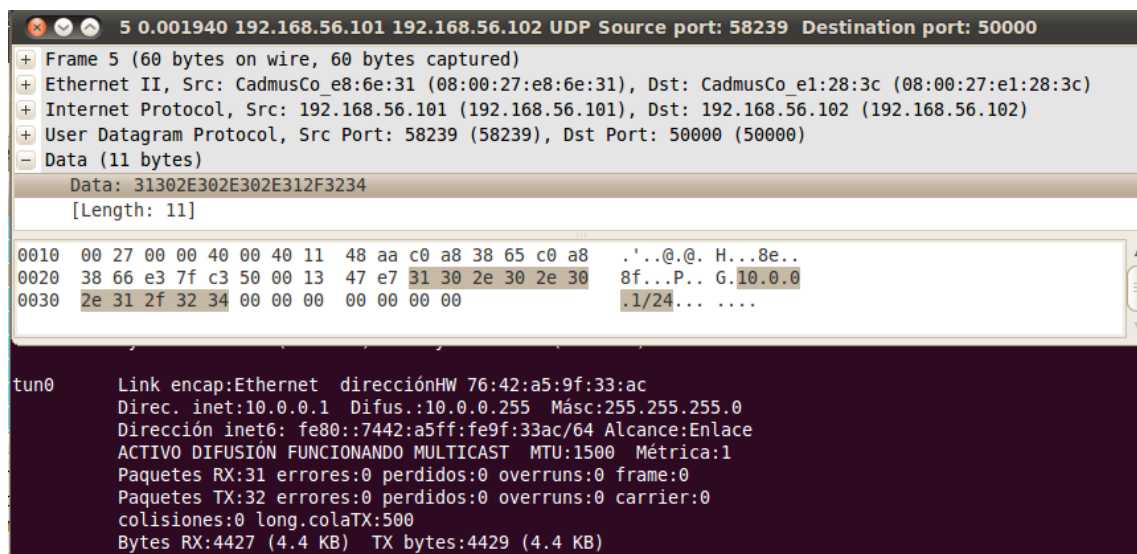


Figura 7.1.6 Envío de dirección IP y asignación a tun0

Una vez finalizada la conexión entre el controller y el host 1, se realizará la conexión con el Host 2 siguiendo el mismo procedimiento, cambiando la dirección IP por 10.0.0.2/24.

A continuación, ejecutaremos un ping desde el host 2 hacia el host 1 con el fin de comprobar si existe conectividad:

```
user@user-desktop:~$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=4.71 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.851 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.786 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.767 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.826 ms
^C
--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.767/1.589/4.718/1.565 ms
```

Figura 7.1.7 Ping de host 2 a host 1 (consola)

1	0.000000	62:7b:20:51:5b:a2	Broadcast	ARP	Who has 10.0.0.1? Tell 10.0.0.2
2	0.000012	76:42:a5:9f:33:ac	62:7b:20:51:5b:a2	ARP	10.0.0.1 is at 76:42:a5:9f:33:ac
3	0.000641	10.0.0.2	10.0.0.1	ICMP	Echo (ping) request
4	0.000652	10.0.0.1	10.0.0.2	ICMP	Echo (ping) reply
5	0.998933	10.0.0.2	10.0.0.1	ICMP	Echo (ping) request
6	0.998942	10.0.0.1	10.0.0.2	ICMP	Echo (ping) reply
7	1.997890	10.0.0.2	10.0.0.1	ICMP	Echo (ping) request
8	1.997898	10.0.0.1	10.0.0.2	ICMP	Echo (ping) reply
9	2.996867	10.0.0.2	10.0.0.1	ICMP	Echo (ping) request
10	2.996875	10.0.0.1	10.0.0.2	ICMP	Echo (ping) reply
11	3.996292	10.0.0.2	10.0.0.1	ICMP	Echo (ping) request
12	3.996299	10.0.0.1	10.0.0.2	ICMP	Echo (ping) reply
13	4.999111	76:42:a5:9f:33:ac	62:7b:20:51:5b:a2	ARP	Who has 10.0.0.2? Tell 10.0.0.1
14	4.999790	62:7b:20:51:5b:a2	76:42:a5:9f:33:ac	ARP	10.0.0.2 is at 62:7b:20:51:5b:a2

Figura 7.1.8 Ping de host 2 a host 1 (wireshark)

Además, hacemos uso del comando traceroute para comprobar el camino exacto desde la interfaz tun0 del host 1 a la interfaz tun0 del host 2:

```
user@user-desktop:~$ traceroute 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 30 hops max, 60 byte packets
1 user-desktop-3.local (10.0.0.2) 11.939 ms 12.056 ms 12.059 ms
```

Figura 7.1.9 Traceroute de host 1 a host 2

En estas imágenes vemos que, desde el punto de vista de las interfaces tun0, los mensajes que se intercambian no pasan por ningún punto intermedio, sin embargo, sabemos que pasan por el controller, al cual le llegan por un socket y los reenvía por otro, lo cual es un proceso invisible de cara a las interfaces tun0:

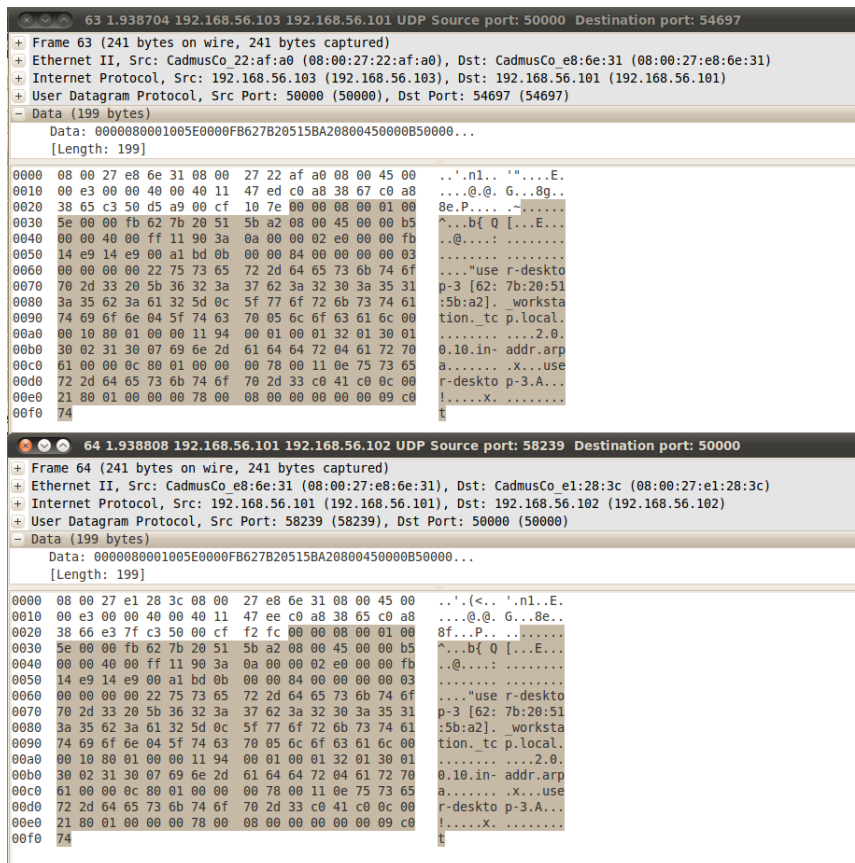


Figura 7.1.10 Ping request host 2 a host 1 visto desde el controller

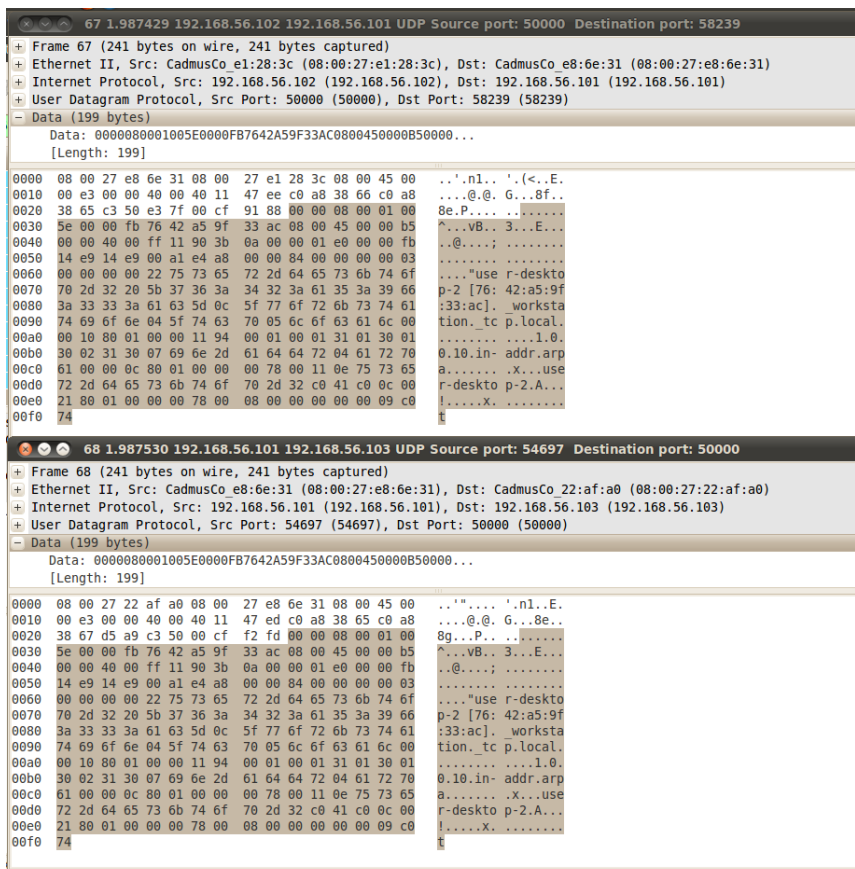


Figura 7.1.11 Ping reply host 1 a host 2 visto desde el controller

En estas imágenes, que se corresponden al primer Ping request/ping reply de la figura 6.1.8, vemos que el controller no conoce el contenido de los mensajes que se intercambian entre las interfaces tun0, simplemente reenvía lo que llega del host 2 al host 1 y viceversa, al ser estos dos los únicos hosts conectados entre sí.

7.2 Conectividad entre 4 Hosts

En esta prueba se conectarán las interfaces tun0 de 4 hosts mediante el controller con el fin de comprobar que sólo exista conectividad entre Hosts que se designen como conectados directamente. Una vez comprobado esto, se ejecutará una implementación del protocolo AODV en cada uno de los hosts para que todos puedan comunicarse entre sí y se comprobará su correcto funcionamiento. El escenario es el siguiente:

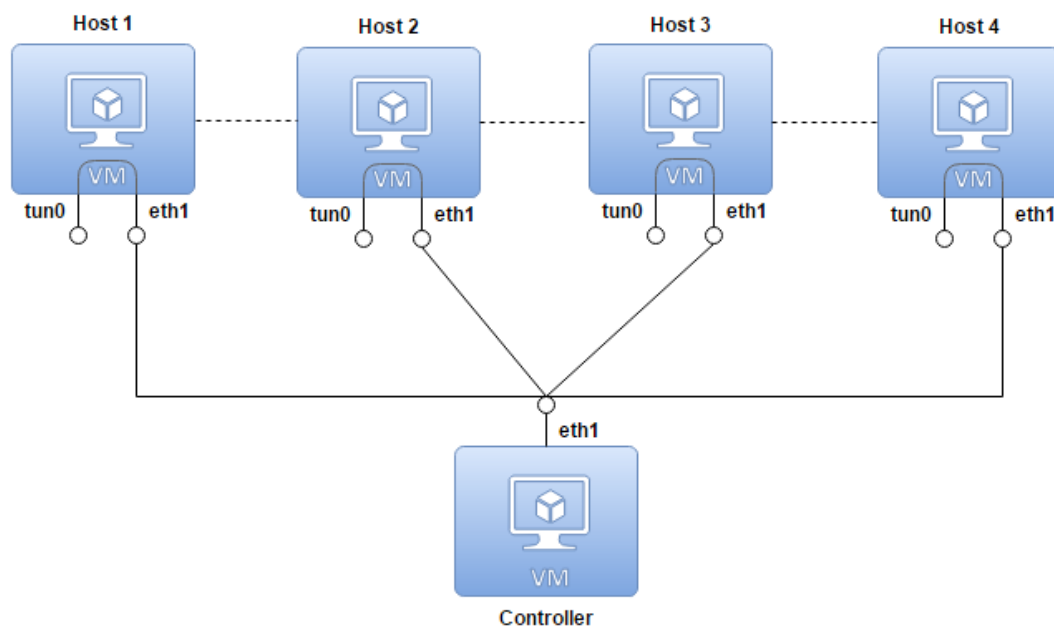


Figura 7.2.1 Escenario evaluación 2

Podemos ver que el host 1 solo podrá comunicarse directamente con el host 2, el host 2 se comunicará directamente con el host 1 y el host 3, el host 3 tiene permitido comunicarse directamente con el host 2 y el host 4 y, finalmente, el host 4 solamente tendrá comunicación directa con el host 3.

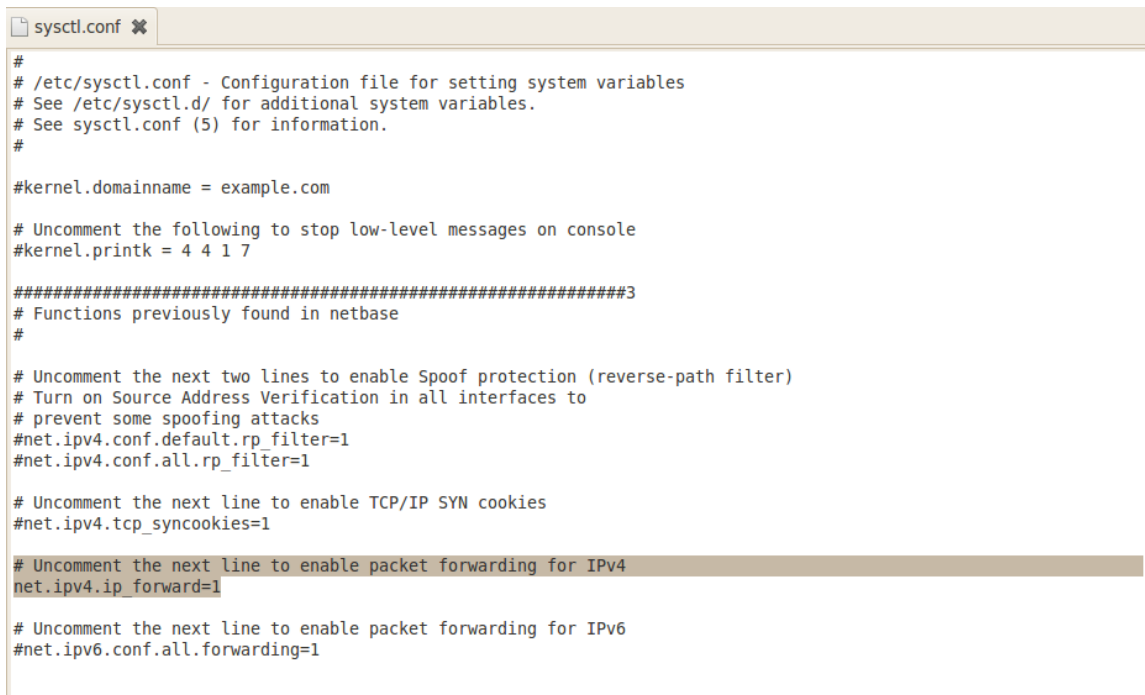
La interfaz eth1 del controller tendrá asociada la dirección IP 192.168.56.101/24, la interfaz eth1 del host 1 será 192.168.56.102/24 mientras que la del host 2 será 192.168.56.103/24, la interfaz eth1 del host 3 192.168.56.104/24 y la interfaz eth1 del host 4 192.168.56.105/24. Por otro lado, a la interfaz tun0 del host1 se le asignará la dirección 10.0.0.1/24, a la interfaz tun0 del host 2 la dirección 10.0.0.2/24, a la interfaz tun0 del host 3 10.0.0.3/24 y, por último, a la interfaz tun0 del host 4 10.0.0.4/24.

En primer lugar, configuramos el esquema de la red en el archivo JSON de tal forma que se corresponda con el escenario representado:

```
{
    "hosts": [1, 2, 3, 4],
    "neighbors": [[2], [1, 3], [2, 4], [3]]
}
```

A continuación, activamos el reenvío de paquetes en los hosts, para ello, accedemos al archivo `sysctl.conf` situado en la carpeta `/etc/` y descomentamos la línea perteniente al reenvío de paquetes IPv4:

- `sudo gedit /etc/sysctl.conf`



```
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 4 4 1 7

#####3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#net.ipv6.conf.all.forwarding=1
```

Figura 7.2.2 Archivo `sysctl.conf`

Acto seguido, lanzamos el archivo `host.py` en los cuatro hosts que forman parte de la prueba mediante el siguiente comando:

- `sudo python host.py`

Finalmente, ejecutaremos en el controller el archivo `controller.py`:

- `sudo python controller.py -t 192.168.51.102,192.168.56.103, 192.168.56.104, 192.168.56.105`

Una vez hecho esto, los hosts estarán listos para trabajar. Para comprobar que cada host solo puede comunicarse directamente con los hosts designados en el JSON, enviamos pings desde cada uno de los hosts hacia los demás:


```

user@user-desktop:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.91 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.747 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.973 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.747/1.544/2.914/0.973 ms
user@user-desktop:~$ ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
, pipe 3
user@user-desktop:~$ ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
^C
--- 10.0.0.4 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
, pipe 3

```

Figura 7.2.3 Pings host 1 (sin AODV)

```

user@user-desktop:~$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1.20 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.876 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.752 ms
^C
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.752/0.943/1.201/0.189 ms
user@user-desktop:~$ ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=4.11 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.800 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.905 ms
^C
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.800/1.940/4.117/1.540 ms
user@user-desktop:~$ ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable
^C
--- 10.0.0.4 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
, pipe 3

```

Figura 7.2.4 Pings host 2 (sin AODV)

```

user@user-desktop:~$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
From 10.0.0.3 icmp_seq=3 Destination Host Unreachable
From 10.0.0.3 icmp_seq=4 Destination Host Unreachable
^C
--- 10.0.0.1 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
, pipe 3
user@user-desktop:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.792 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.30 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=1.21 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.792/1.104/1.308/0.224 ms
user@user-desktop:~$ ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=4.83 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.799 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.795 ms
^C
--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.795/2.141/4.830/1.901 ms

```

Figura 7.2.5 Pings host 3 (sin AODV)

```

user@user-desktop:~$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
From 10.0.0.4 icmp_seq=3 Destination Host Unreachable
From 10.0.0.4 icmp_seq=4 Destination Host Unreachable
^C
--- 10.0.0.1 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
, pipe 3
user@user-desktop:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
From 10.0.0.4 icmp_seq=3 Destination Host Unreachable
From 10.0.0.4 icmp_seq=4 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2998ms
, pipe 3
user@user-desktop:~$ ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.50 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=1.19 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=1.33 ms
^C
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.197/1.347/1.509/0.127 ms

```

Figura 7.2.6 Pings host 4 (sin AODV)

Podemos comprobar que, efectivamente, solo existe conectividad entre los hosts que hemos designado en el archive JSON. A continuación, mediante Wireshark, vamos a comprobar el comportamiento del controller cuando le llega un paquete desde un host que tiene conectividad con varios Hosts, concretamente, enviaremos un ping desde el host 2, que tiene conectividad con los hosts 1 y 3, al host 3, que tiene conectividad con los hosts 2 y 4:

1 0.000000	192.168.56.103	192.168.56.101	UDP	Source port: 50000	Destination port: 57915
2 0.000125	192.168.56.101	192.168.56.102	UDP	Source port: 34447	Destination port: 50000
3 0.000167	192.168.56.101	192.168.56.104	UDP	Source port: 48038	Destination port: 50000
4 0.000606	192.168.56.104	192.168.56.101	UDP	Source port: 50000	Destination port: 48038
5 0.000903	192.168.56.101	192.168.56.103	UDP	Source port: 57915	Destination port: 50000
6 0.000984	192.168.56.101	192.168.56.105	UDP	Source port: 34554	Destination port: 50000

Figura 7.2.7 Ping de host 2 a host 3 (Controller)

En esta traza podemos ver que cuando el controller recibe un paquete desde el host 2, inmediatamente lo reenvía tanto al host 1 como al 3, pues son los hosts conectados a él, de igual forma, cuando recibe la respuesta del host 3, la reenvía tanto al host 2 como al host 4.

Desde el punto de vista de la interfaz tun0 del host 1, podemos ver que ha llegado un paquete que no va destinado a él, por lo que lo ignora:

1 0.000000	10.0.0.2	10.0.0.3	ICMP	Echo (ping) request
2 1.001644	10.0.0.2	10.0.0.3	ICMP	Echo (ping) request
3 2.003241	10.0.0.2	10.0.0.3	ICMP	Echo (ping) request

Figura 7.2.8 Ping de host 2 a host 3 (Host 1)

De igual forma, al host 4 llegará un paquete (ping reply) del host 3 para el host 2 que ignorará.

Una vez comprobado que todo funciona correctamente y no existe conectividad cuando no debería, procedemos a utilizar el protocolo AODV para permitir conectividad total entre todos nuestros hosts, para ello, ejecutaremos los siguientes comandos en cada uno de los hosts:

- `sudo modprobe kaodv`
- `sudo aodvd -i tun0 -l -r 2 -w`

Estos comandos activarán la implementación del protocolo AODV realizada por la Universidad de Upsala en nuestros hosts, concretamente, activará el protocolo AODV para la interfaz tun0, activará un log, fijará la actualización de la tabla de routing cada 2 segundos y activará el modo gateway. Una vez activado, los hosts empezarán a enviar mensajes HELLO a sus vecinos directos y a añadirlos en sus tablas de routing. Como ejemplo, veamos el host 1:

```
user@user-desktop:~$ sudo aodvd -i tun0 -l -r 2 -w
15:27:07.944 aodv_socket_init: RAW send socket buffer size set to 262142
15:27:07.944 aodv_socket_init: Receive buffer size set to 262142
15:27:07.944 main: In wait on reboot for 15000 milliseconds. Disable with "-D".
15:27:07.944 hello_start: Starting to send HELLOs!
15:27:22.945 wait_on_reboot_timeout: Wait on reboot over!!
15:27:26.377 rt_table_insert: Inserting 10.0.0.2 (bucket 10) next hop 10.0.0.2
15:27:26.377 nl_send_add route msg: ADD/UPDATE: 10.0.0.2:10.0.0.2 ifindex=3
15:27:26.377 rt_table_insert: New timer for 10.0.0.2, life=2100
15:27:26.377 hello_process: 10.0.0.2 new NEIGHBOR!
```

Figura 7.2.9 AODV host 1

Podemos comprobar que el host 1, que tenía como vecino al host 2, lo añade a su tabla de routing. Haciendo uso de Wireshark podemos observar los mensajes route reply enviados:

2 0.195476	10.0.0.1	255.255.255.255	AODV	Route Reply, D: 10.0.0.1, O: 10.0.0.1 Hcnt=0 DSN=1 Lifetime=2000
3 1.024312	10.0.0.2	255.255.255.255	AODV	Route Reply, D: 10.0.0.2, O: 10.0.0.2 Hcnt=0 DSN=1 Lifetime=2000

Figura 7.2.10 AODV host 1

Por último, comprobaremos que existe conectividad entre el host 1 y el 4 a través de los hosts 2 y 3 gracias a AODV mediante un ping:

```
user@user-desktop:~$ ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=62 time=2.10 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=62 time=3.18 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=62 time=2.48 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=62 time=2.32 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=62 time=2.11 ms
^C
--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 2.109/2.445/3.186/0.400 ms
```

Figura 7.2.11 Ping de host 1 a host 4

Vemos que gracias a la utilización de AODV, ahora los hosts 1 y 4 tienen conectividad entre ellos. Si analizamos la tabla de AODV, vemos que el host 1 ha aprendido que para llegar al host 4 su siguiente salto es a través del host 2:

```
user@user-desktop:~$ sudo aodvd -i tun0 -l -r 2 -w
18:44:20.527 aodv_socket_init: RAW send socket buffer size set to 262142
18:44:20.527 aodv_socket_init: Receive buffer size set to 262142
18:44:20.527 main: In wait on reboot for 15000 milliseconds. Disable with "-D".
18:44:20.527 hello_start: Starting to send HELLOs!
18:44:35.528 wait_on_reboot_timeout: Wait on reboot over!!
18:44:38.361 rt_table_insert: Inserting 10.0.0.2 (bucket 10) next hop 10.0.0.2
18:44:38.361 nl_send_add_route_msg: ADD/UPDATE: 10.0.0.2:10.0.0.2 ifindex=4
18:44:38.361 rt_table_insert: New timer for 10.0.0.2, life=2100
18:44:38.361 hello_process: 10.0.0.2 new NEIGHBOR!
18:45:35.007 nl_kaadv_callback: Got ROUTE REQ: 10.0.0.4 from kernel
18:45:35.007 rreq_create: Assembled RREQ 10.0.0.4
18:45:35.007 log_pkt_fields: rreq->flags: rreq->hopcount=0 rreq->rreq_id=0
18:45:35.007 log_pkt_fields: rreq->dest_addr:10.0.0.4 rreq->dest_seqno=0
18:45:35.007 log_pkt_fields: rreq->orig_addr:10.0.0.1 rreq->orig_seqno=2
18:45:35.007 aodv_socket_send: AODV msg to 255.255.255.255 ttl=2 size=24
18:45:35.007 rreq_route_discovery: Seeking 10.0.0.4 ttl=2
18:45:35.010 aodv_socket_process_packet: Received RREP
18:45:35.010 rrep_process: from 10.0.0.2 about 10.0.0.1->10.0.0.4
18:45:35.010 log_pkt_fields: rrep->flags: rrep->hcnt=2
18:45:35.010 log_pkt_fields: rrep->dest_addr:10.0.0.4 rrep->dest_seqno=1
18:45:35.010 log_pkt_fields: rrep->orig_addr:10.0.0.1 rrep->lifetime=1304
18:45:35.010 rt_table_insert: Inserting 10.0.0.4 (bucket 10) next hop 10.0.0.2
18:45:35.010 nl_send_add_route_msg: ADD/UPDATE: 10.0.0.4:10.0.0.2 ifindex=4
18:45:35.010 rt_table_insert: New timer for 10.0.0.4, life=1304
18:45:48.024 route_expire_timeout: Route 10.0.0.4 DOWN, seqno=1
18:45:48.024 nl_send_del_route_msg: Send DEL ROUTE to kernel: 10.0.0.4
18:45:48.024 rt_table_invalidate: 10.0.0.4 removed in 15000 msec
```

Figura 7.2.12 Tabla de routing de AODV host 1

Por otro lado, el host 2 ha aprendido que para llegar al host 4 tiene que pasar por el host 3:

```
user@user-desktop:~$ sudo aodvd -i tun0 -l -r 2 -w
18:44:22.778 aodv_socket_init: RAW send socket buffer size set to 262142
18:44:22.778 aodv_socket_init: Receive buffer size set to 262142
18:44:22.778 main: In wait on reboot for 15000 milliseconds. Disable with "-D".
18:44:22.778 hello start: Starting to send HELLOs!
18:44:35.033 rt_table_insert: Inserting 10.0.0.1 (bucket 10) next hop 10.0.0.1
18:44:35.033 nl_send_add_route_msg: ADD/UPDATE: 10.0.0.1:10.0.0.1 ifindex=3
18:44:35.033 rt_table_insert: New timer for 10.0.0.1, life=2100
18:44:35.033 hello_process: 10.0.0.1 new NEIGHBOR!
18:44:37.779 wait_on_reboot_timeout: Wait on reboot over!!
18:45:00.602 rt_table_insert: Inserting 10.0.0.3 (bucket 10) next hop 10.0.0.3
18:45:00.602 nl_send_add_route_msg: ADD/UPDATE: 10.0.0.3:10.0.0.3 ifindex=3
18:45:00.602 rt_table_insert: New timer for 10.0.0.3, life=2100
18:45:00.602 hello_process: 10.0.0.3 new NEIGHBOR!
18:45:34.470 rreq_process: ip_src=10.0.0.1 rreq_orig=10.0.0.1 rreq_dest=10.0.0.4
ttl=2
18:45:34.470 rreq_record_insert: Buffering RREQ 10.0.0.1 rreq_id=0 time=5600
18:45:34.470 log_pkt_fields: rreq->flags: rreq->hopcount=0 rreq->rreq_id=0
18:45:34.470 log_pkt_fields: rreq->dest_addr:10.0.0.4 rreq->dest_seqno=0
18:45:34.470 log_pkt_fields: rreq->orig_addr:10.0.0.1 rreq->orig_seqno=2
18:45:34.470 rreq_forward: forwarding RREQ src=10.0.0.1, rreq_id=0
18:45:34.470 aodv_socket_send: AODV msg to 255.255.255.255 ttl=1 size=24
18:45:34.471 aodv_socket_process_packet: Received RREP
18:45:34.471 rrep_process: from 10.0.0.3 about 10.0.0.1->10.0.0.4
18:45:34.471 log_pkt_fields: rrep->flags: rrep->hcnt=1
18:45:34.471 log_pkt_fields: rrep->dest_addr:10.0.0.4 rrep->dest_seqno=1
18:45:34.471 log_pkt_fields: rrep->orig_addr:10.0.0.1 rrep->lifetime=1304
18:45:34.471 rt_table_insert: Inserting 10.0.0.4 (bucket 10) next hop 10.0.0.3
18:45:34.471 nl_send_add_route_msg: ADD/UPDATE: 10.0.0.4:10.0.0.3 ifindex=3
18:45:34.471 rt_table_insert: New timer for 10.0.0.4, life=1304
18:45:34.471 rrep_forward: Forwarding RREP to 10.0.0.1
18:45:34.471 aodv_socket_send: AODV msg to 10.0.0.1 ttl=254 size=20
18:45:34.471 precursor_add: Adding precursor 10.0.0.1 to rte 10.0.0.4
18:45:34.471 precursor_add: Adding precursor 10.0.0.3 to rte 10.0.0.1
18:45:47.486 route_expire_timeout: Route 10.0.0.4 DOWN, seqno=1
18:45:47.486 nl_send_del_route_msg: Send DEL_ROUTE to kernel: 10.0.0.4
18:45:47.486 rt_table_invalidate: 10.0.0.4 removed in 15000 msecs
```

Figura 7.2.13 Tabla de routing de AODV host 2

De igual forma, el host 4 ha aprendido que para llegar al host 1 tiene que pasar por el host 3 y el host 3 para contactar con el host 1 tiene que pasar por el host 2.

Con esto damos por terminadas las pruebas de funcionamiento, que se han resuelto satisfactoriamente. Lamentablemente, debido a no poder acceder a un ordenador lo suficientemente potente como para ejecutar numerosas máquinas virtuales, no se han podido realizar pruebas de performance para averiguar el número máximo de hosts que puede manejar el controller sin que se resienta el rendimiento.

Capítulo 8 – Conclusiones

En este capítulo se recogen las conclusiones del proyecto, así como un apartado dedicado a hablar de futuras ampliaciones al mismo.

8.1 Conclusiones del trabajo

Gracias a este trabajo he consolidado mis conocimientos sobre las redes inalámbricas, el uso y configuración de máquinas virtuales y aprendido el lenguaje de programación Python. Existe una gran cantidad de factores a tener en cuenta al trabajar con redes inalámbricas que hay que adaptar a la hora de intentar virtualizar una red inalámbrica en una cableada, como se ha podido comprobar a lo largo del trabajo.

Hay que destacar la flexibilidad del diseño realizado, puesto que permite utilizar desde dispositivos reales hasta máquinas virtuales para que actúen tanto de host como de controller, como las utilizadas en la implementación. Así mismo, aunque se ha hecho uso de Ubuntu como sistema operativo para poder hacer uso de la implementación de AODV utilizada, adaptando los programas, se podrían utilizar otros S.O. Además, también se podría utilizar otro lenguaje de programación, si bien en este trabajo se ha utilizado Python debido a ser un lenguaje multiplataforma, flexible, simplificado y rápido.

Si bien la implementación realizada no abarca toda la problemática mencionada en el capítulo de análisis, se trata de un prototipo de laboratorio virtual de redes inalámbricas que sirve para ver lo potente que es esta herramienta. Con el diseño utilizado podemos crear nuestro laboratorio de forma escalable, y añadir o quitar funcionalidades modificando solamente el programa Python del controller, lo que permite crear escenarios específicos para cada situación que queramos probar. Por otro lado, al poder funcionar con máquinas virtuales, se muestra como una solución barata y fácil de configurar en un solo ordenador, y permite trabajar con los dispositivos que forman parte de los experimentos en tiempo real, no como en el caso de los simuladores.

Este prototipo nos ha permitido crear una red ad hoc inalámbrica sobre una cableada que, si bien no es perfecta, puesto que no tiene en cuenta factores como la distancia a la que se encuentran los dispositivos para los tiempos de propagación, nos permitiría probar el funcionamiento de diversas aplicaciones en este tipo de redes, así mismo, se podría sustituir la implementación de AODV por otras implementaciones de diversos protocolos, nuevos o ya existentes.

Como conclusiones finales, podemos resaltar la importancia de esta herramienta, puesto que nos proporciona un entorno real de pruebas a bajo coste, lo que conlleva facilitar la evaluación de nuevos dispositivos, protocolos o aplicaciones.

8.2 Futuras ampliaciones

Como se ha mencionado con anterioridad, la implementación llevada a cabo en este trabajo no ha tenido en cuenta aspectos como los modelos de movilidad de dispositivos inalámbricos, los tiempos de propagación o el rango de cobertura de los dispositivos.

Es por ello que una línea de trabajo futura puede ser incluir estos aspectos dentro del programa del controller para conseguir un laboratorio más completo. Otra línea de trabajo posible es la adaptación de los programas a otros sistemas operativos o a diferentes lenguajes de programación.

Si bien con la implementación actual podríamos emular la existencia de distintas redes unidas a un solo controller, otro proyecto podría crear interacción entre distintos controllers, de forma que cada uno controle una red y a su vez actúe de puente entre redes.

Finalmente, otra ampliación importante es la adaptación del controller para que permita acoger cambios en la estructura de la red en tiempo real, ya sea automáticamente si alguno sale de su radio de cobertura debido a su movilidad, o de forma manual pasándole al programa información sobre la estructura de la red cada vez que se produzca un cambio.

Relacionado con esto, se podría hacer uso del proyecto “Desarrollo de una herramienta de simulación para un laboratorio MANET virtual con AODV”, realizado por Ángel Hita Albarracín y dirigido por Gabriel Maciá Fernández y Rafael Alejandro Rodríguez Gómez para la Universidad de Granada. Este proyecto, como se mencionó en el capítulo 5, trata el diseño y desarrollo de una interfaz gráfica de usuario (GUI) que permite crear y posicionar en un mapa un conjunto de máquinas virtuales. Además, implementa y simula las vecindades entre nodos.

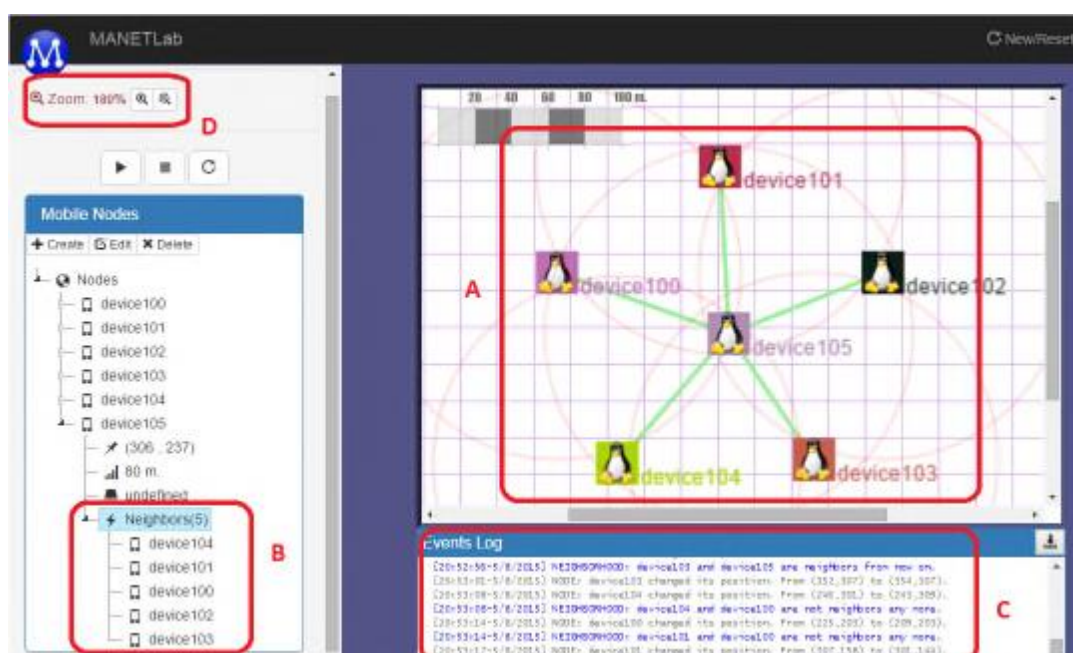


Figura 8.2.1 MANETLab

Haciendo uso de este proyecto, se conseguiría tanto la adaptación en tiempo real de los nodos del laboratorio como una interfaz gráfica para colocarlos, lo cual es mucho más cómodo que rellenar un JSON a mano. Simplemente habría que adaptar los datos de salida proporcionados por este proyecto para que sigan la estructura del JSON y en el controller estar escuchando los cambios de la red que le pase este programa y aplicándolos a su vez a la red.

Referencias

- [1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015-2020 White Paper. Cisco White Papers.
<http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html>
- [2] OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. OMNET++. <https://omnetpp.org/>
- [3] ns-3 is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. ns-3. <https://www.nsnam.org/>
- [4] The Cisco Network Simulator, Router Simulator & Switch Simulator. NetSim.
<http://www.boson.com/netsim-cisco-network-simulator>
- [5] The fastest discrete event-simulation engine for analyzing and designing communication networks. Riverbed Modeler.
<http://www.riverbed.com/gb/products/steelcentral/steelcentral-riverbed-modeler.html>
- [6] Request for Comments dedicado a AODV. RFC 3561.
<https://tools.ietf.org/html/rfc3561>
- [7] VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. VirtualBox. <https://www.virtualbox.org/>
- [8] Parallels Remote Application Server is a leading solution for virtual application and desktop delivery. It provides Windows applications to anyone using any OS or mobile device. Parallels. <http://www.parallels.com/>
- [9] VMware is a global leader in cloud infrastructure and business mobility. VMWare.
<http://www.vmware.com/>
- [10] Mobility Models I. Dr. Tarek R. Sheltami. Wireless Mobile Ad hoc Networks. King Fahd University of Petroleum & Minerals.
<http://faculty.kfupm.edu.sa/coe/tarek/COE549/LectureSlides/6-Mobility%20Models%20I.pps>
- [11] Mobility Models II. Dr. Tarek R. Sheltami. Wireless Mobile Ad hoc Networks. King Fahd University of Petroleum & Minerals.
<http://faculty.kfupm.edu.sa/coe/tarek/COE549/LectureSlides/7-Mobility%20Models%20II.pps>
- [12] Capítulo dedicado al Networking del manual de VirtualBox. Virtual Networking.
<https://www.virtualbox.org/manual/ch06.html>

[13] Understanding TCP over TCP: Effects of TCP Tunneling on End-to-End Throughput and Latency. Osamu Honda, Hiroyuki Ohsaki, Makoto Imase, Mika Ishizuka and Junichi Murayama. Graduate School of Information Science and Technology, Osaka University and NTT Information Sharing Platform Laboratories, NTT Corporation.

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.5815&rep=rep1&type=pdf>

[14] Repositorio de Github de la implementación de AODV de la Universidad de Upsala. AODV-UU. <https://github.com/erimatnor/aodv-uu>

[15] Repositorio de Github del trabajo diseño e implementación de un laboratorio virtual de redes inalámbricas de Adrián Ripoll Casas. Virtual WLAN LAB. <https://github.com/Adrirc8/Virtual-WLAN-Lab>