

Universidad de Alcalá

Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

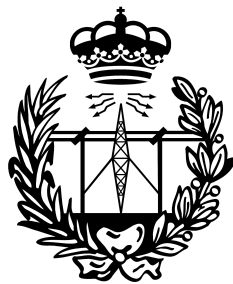
Diseño e implementación de protocolo
de control escalable en redes IoT para
entornos 6G

ESCUELA POLITECNICA
SUPERIOR

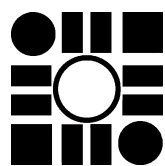
Autor: David Carrascal Acebron

Tutor: Elisa Rojas Sánchez

2022



Máster Universitario en Ingeniería de Telecomunicación



Universidad
de Alcalá

Madrid, 24 de octubre de 2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

**Diseño e implementación de protocolo
de control escalable en redes IoT para entornos 6G**

Autor: David Carrascal Acebron

Tutor: Elisa Rojas Sánchez

Tribunal:

Presidente: Juan Antonio Carral Pelayo

Vocal 1º: José Manuel Rodríguez Ascáriz

Vocal 2º: Elisa Rojas Sánchez

*A mis hermanas, Natalia y Violeta,
quienes día a día, por oscura que sea la noche,
arrojan luz y esperanza a mi vida.*

Agradecimientos

Quiero empezar agradeciendo y reconociendo a mi tutora, Elisa Rojas, sin la cual este trabajo no habría sido posible. Quien desde segundo de carrera creyó en mi, y aun día de hoy, sigue apostando día a día en mis capacidades, incluso cuando ni yo mismo soy capaz de verlas. Su destreza y conocimiento, su apoyo incondicional y carisma, su maestría y pasión por lo que hace, y a lo que se dedica, han hecho que etapa, tras etapa académica siga aprendiendo y disfrutando como el primer día. Este trabajo ha sido financiado por subvenciones de la Comunidad de Madrid a través de los proyectos TAPIR-CM (S2018/TCS-4496) y MistLETOE-CM (CM/JIN/2021-006), y por el proyecto ONENESS (PID2020-116361RA-I00) del Ministerio de Ciencia e Innovación de España.

También me gustaría agradecer a mi familia, por su cariño, comprensión e inspiración en estos meses que han sido tan duros para mi. Y que decir de mis amigos, a los *Caye de Calle*, a los *C de Chill*, a mis estimados *Pueblerinos*, como no, a Pablo y Olga, a mis queridas Noci, a mi señor abuelo de confianza, Bobby, a la señorita Laura de Diego, mis compis de la uni y toda la gente nueva que ha llegado a mi vida durante estos meses, a todos vosotros, gracias por las risas y los buenos momentos que hemos compartido juntos. Gracias de verdad.

No puedo terminar sin agradecer a toda la gente del Laboratorio LE34, quienes me alentan a seguir por este arduo camino de la investigación y quienes con sus consejos y experiencias han ido conformando al ingeniero que soy a día de hoy.

Sinceramente, mil gracias a todos.

Resumen

En este Trabajo Final de Máster (TFM) se presenta el diseño e implementación de un protocolo de control escalable de redes Internet of Things (IoT) para entornos Software-Defined Networking (SDN) en la nueva generación de redes móviles, the sixth generation of mobile technologies (6G). Dicho protocolo de control seguirá un paradigma de control de tipo *in-band*, con el cual se dotará de conectividad a los nodos de la red con el ente de control, empleando el plano de datos para la transmisión de información de control.

En aras de completar el proyecto, se ha partido por analizar las necesidades y características de las distintas tecnologías que se emplearán en ejecución del objetivos predefinidos y así discernir aquellas herramientas necesarias para la implementación del protocolo control. Una vez seleccionadas las herramientas, se estudiarán a fondo para realizar una implementación lo optimizada en la medida de lo posible. Este proyecto concluirá con la validación mediante emulación del protocolo desarrollado para comprobar el correcto funcionamiento del mismo en distintos casos de uso.

Palabras clave: 6G; IoT; SDN; Control in-band; Plano de control

Abstract

In this Master's Thesis (TFM) we present the design and implementation of a scalable control protocol for Internet of Things (IoT) networks for Software-Defined Networking (SDN) environments in the new generation of mobile networks, the sixth generation of mobile technologies (6G). This control protocol will be based on an *in-band* control paradigm, which will provide connectivity between the network nodes and the control entity, using the data plane for the transmission of control information.

In order to fulfil the project, we have started by analysing the requirements and characteristics of the different technologies that will be used in the execution of the predefined objectives and thus be able to determine the tools necessary for the implementation of the control protocol. Once the tools have been selected, they will be studied in depth in order to carry out an optimised implementation as far as possible. This project will conclude with the validation the developed protocol by means of emulation to check its correct operation in different use cases.

Keywords: 6G; IoT; SDN; In-band control; Control plane

“No hay ningún viento favorable para el que no sabe a que puerto se dirige”

Arthur Schopenhauer.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

En este primer capítulo, se desea presentar de manera concisa los aspectos más relevantes del TFM, como son, las redes de dispositivos IoT, la llegada de los entornos 6G, y la tecnología habilitadora en dichos entornos, el SDN. Se explorarán las necesidades actuales de las redes de sensores IoT, se indagará la postulada nueva generación de redes móviles, 6G, y se verá donde entrará las redes SDN, y qué mejoras deberán hacerse para hacer frente a las necesidades imperantes de las próximas redes de sensores.

Se establecerán objetivos claros para el TFM y se describirá detalladamente cómo se planea llevarlos a cabo cada uno de ellos. Estos objetivos ayudarán a al diseño y desarrollo de un nuevo protocolo de comunicación de control escalable para redes de sensores en un ámbito de red SDN. De forma adicional, se presentará la estructura general del TFM, describiendo de manera breve los temas que se abordarán en cada capítulo. Por último, se indicarán las contribuciones realizadas en revistas científicas de alto impacto de este proyecto.

1.1 El Internet de las Cosas y la red 6G

Los recientes avances en las comunicaciones móviles junto a la mejora de las capacidades tecnológicas de los elementos hardware han llevado al IoT a un punto álgido, donde, a día de hoy, interconecta billones de objetos entre sí con comunicaciones Machine to machine (M2M) tanto en entornos particulares, como en entornos industriales [?]. Se puede afirmar que sin lugar a dudas el IoT es parte del hoy y el mañana de Internet, ha revolucionado la forma en se interactúa con el mundo que nos rodea, permitiendo conectar dispositivos entre sí de forma completamente autónoma a través de la red, proveyendo al humano de entornos inteligentes y adaptativos a las necesidades de la sociedad.

Sin embargo, el aumento exponencial de los dispositivos IoT conectados a las redes de comunicaciones móviles ha generado nuevas necesidades en términos de capacidad, rendimiento, latencia y eficiencia de las redes que deben ser solventadas. Las tecnologías móviles the fifth generation of mobile technologies (5G) ya se han propuesto y desplegado comercialmente para dar soporte a las necesidades de las redes IoT y sus aplicaciones. Esta tecnología habilitadora daba solución a las necesidades preliminares del IoT haciendo uso de las

funcionalidades que traía consigo, como por ejemplo, enhanced Mobile BroadBand (eMBB), massive Machine-Type Communication (mMTC), Ultra-Reliable and Low-Latency Communication (URLLC) [?]. Dichas funcionalidades proveían a los ecosistemas IoT de servicios de alto ancho de banda, baja latencia y optimización del consumo, siendo esta última muy importante para los dispositivos IoT. No obstante, con la rápida proliferación de nuevos sensores, y con ello, el aumento de las redes IoT según se puede apreciar en la figura ??, los requisitos técnicos necesarios se han visto aumentados para poder seguir manteniendo entornos M2M tal cual se conocían, completamente autónomos, dinámicos e inteligentes.

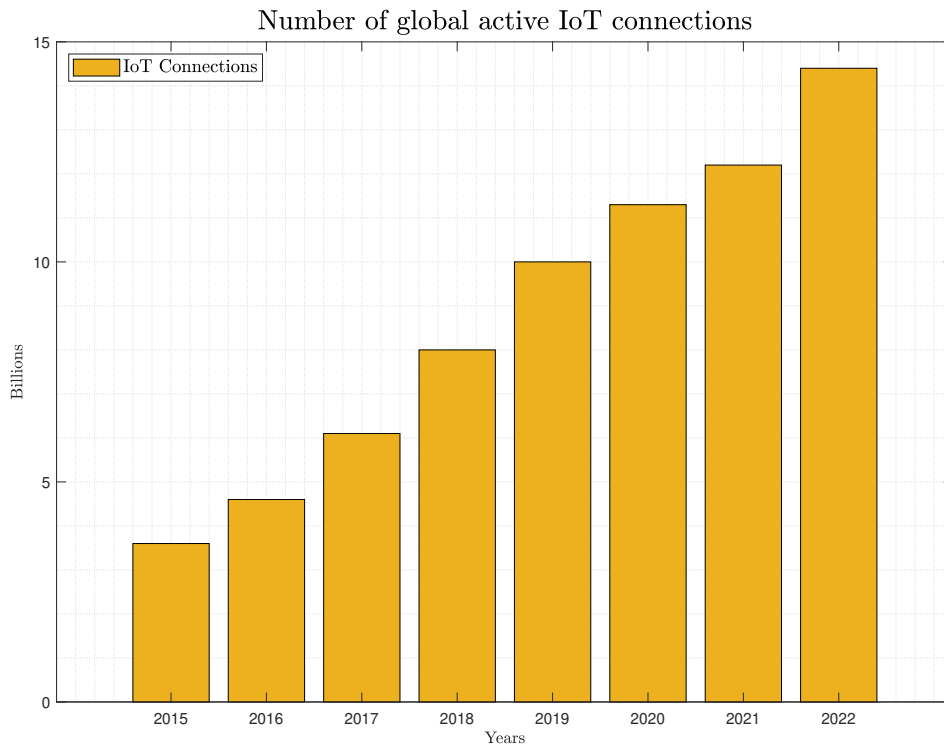


Figura 1.1: Estudio de las conexiones IoT máximas simultáneas a nivel global [?]

Por ello, se necesita una tecnología más avanzada que pueda satisfacer las futuras demandas de las redes IoT, y la tecnología 6G se postula como una solución a todos los nuevos retos planteados.

Parte de la introducción la podemos meter con la survey [?] cuentan la necesidad del IoT y como se postula el 6G para ayudar con estas necesidades.

En europa se está impulsando la carrera por el 6G haciendo un plan de acción dividido en 4 streams con líneas diferenciadas de acción [?]. Actual nos encontramos en el stream B del roadmap [?] el cual tiene como objetivo BLABLA BLA ..

Entre todos los proyectos seleccionados podemos encontrar los más conocidos como por ejemplo: - Hexa-X proyecto liderado por Nokia financiado por Horizon 2020 [?] - 6Genesis por finlandia [?] - ADROIT6G que empieza este mismo enero [?] - 6GTandem que empieza este pasado 26 de enero para mejorar los sistemas distribuidos duales en frecuencia MIMO para apps de 6G [?]

Saliendo de Europa podemos ver los siguientes proyectos: - U.S FCC libera espectro en la banda de los THz en US para hacer pruebas de concepto para el 6G [?]. - Corea del Sur ha planeado lanzar un proyecto piloto de 6G para el 2026 [?]

Las primeras redes 6G indican que podrán ser desplegadas en 2028 pero que su comercialización y la llegada a las personas de a pie no llegará hasta el 2030 [?].

1.2 Redes SDN

1.3 Objetivos

El objetivo principal de este proyecto es conseguir el desarrollo de un protocolo de control in-band eficiente para la integración de la tecnología IoT en las futuras redes de 6G. Como ya hemos introducido, con la llegada del IoT la dimensión de las redes va a crecer exponencialmente. Por lo consiguiente, la complejidad de la administración de dichas redes va a suponer un gran desafío. Los dispositivos IoT se podrán beneficiar de la integración con las redes SDN en 6G, ya que estas les reportará la flexibilidad y programabilidad requerida para una correcta gestión y administración de cada elemento de la red. Por ello, se pueden resumir los objetivos del proyecto en los siguientes puntos:

- Analizar el estado del arte y necesidades actuales de IoT en 6G.
- Diseño de un protocolo in-band eficiente para IoT integrado con SDN.
- Emulación mediante plataformas como Contiki-NG, Mininet y ONOS.
- Implementación y despliegue en hardware real (tarjetas Raspberry Pi, o remotas IoT-LAB)

1.4 Estructura del TFM

1.5 Contribuciones

Este TFM ha proporcionado significativas contribuciones a la comunidad científica, incluyendo dos publicaciones en revistas de alto impacto indexadas en el JCR (2 Q2), un tercer trabajo en revisión (Q2). A continuación, se presentan estas contribuciones en resumen.

Artículos de revistas científicas de alto impacto.

1. Rojas, E., Hosseini, H., Gomez, C., **Carrascal, D.** and Cotrim, J.R., 2021. Outperforming RPL with scalable routing based on meaningful MAC addressing. *Ad Hoc Networks*, 114, p.102433. (JCR **Q2**)
 2. Alvarez-Horcajo, J., Martinez-Yelmo, I., Rojas, E., Carral, J.A. and **Carrascal, D.**, 2022. ieHDDP: An Integrated Solution for Topology Discovery and Automatic In-Band Control Channel Establishment for Hybrid SDN Environments. *Symmetry*, 14(4), p.756. (JCR **Q2**)
 3. **Carrascal, D.**, Rojas, E., Lopez-Pajares, D., Alvarez-Horcajo, J. and Carral, J.A., 2023. A Comprehensive Survey of In-Band Control in SDN: Challenges and Opportunities. *Electronics*, under review. (JCR **Q2**)
-

A Anexo I - Pliego de condiciones

Este anexo se ha añadido siguiendo la recomendación oficial de la Universidad de Alcalá (UAH) sobre Trabajo Final de Grado (TFG)s. En él se indicarán las condiciones materiales de las distintas máquinas donde se ha desarrollado el proyecto. Por último, se resumirán tanto las limitaciones de *hardware*, como las especificaciones del *software* instalado en las máquinas virtuales donde se llevó a cabo los distintos casos de uso.

A.1 Condiciones materiales y equipos

A continuación, se muestran todos los materiales que se han utilizado en el proyecto. Únicamente se indicarán las características de los materiales vitales para el desarrollo del TFG. De este modo, se pretende que queden recogidos todas las especificaciones técnicas con las se han realizado las pruebas y validaciones del desarrollo.

A.1.1 Especificaciones Máquina A

- Procesador: Intel(R) Core(TM) i7-4790 CPU @ 3.60Ghz
- Memoria: 16GB DDR4 (2 slots de 8GB)
- Gráfica: GeForce GTX 970 Windforce - 4GB
- Sistema operativo: Windows 10 - v1909 (Compilación 18363.836)

A.1.2 Especificaciones Máquina B

- Procesador: Intel Core i7-8750H, Hexa Core 2.2GHz-4.1GHz
- Memoria: 16GB DDR4, 2400MHz
- Gráfica: GeForce GTX0150Ti-4GB GDDR5
- Sistema operativo: Windows 10 - v1903 (Compilación 18362.836)

A.1.3 Especificaciones máquinas virtuales

Todas las máquinas virtuales se han ejecutado sobre la máquina B (??), y se ha conectado a ellas vía SSH con la máquina A (??). Las máquinas virtuales se desplegaron sobre el hipervisor **VirtualBox**¹, haciendo uso de su versión **v6.0.14 r133895 (Qt5.6.2)**. Dado que había dos entornos de desarrollo muy definidos, como son el entorno P4 y el entorno eXpress Data Path (XDP), se crearon dos máquinas virtuales pudiendo así aislar posibles conflictos de dependencias. Las dependencias de cada máquina virtual para poder replicar los casos de uso, se pueden instalar haciendo uso de los scripts de instalación suministrados en el repositorio del TFG², pudiendo cualquier interesado replicar los escenarios sin mayor complicación.

A.1.3.1 Máquina virtual entorno P4

- Procesador: Intel Core i7-8750H, 2 cores 2.2GHz-4.1GHz
- Memoria: 8GB DDR4, 2400MHz
- Sistema operativo: Ubuntu 16.04.6 LTS x86_64
- Kernel: 4.15.0-88-generic
- Configuración de red: Modo bridge, conectado a interfaz Intel(R) Wireless-AC 9560

¹<https://www.virtualbox.org/>

²<https://github.com/davidcawork/TFG>

```
n0obie@n0obie-VirtualBox:~$ neofetch
.-/+00ssss00+/- .
`:+ssssssssssssssssss+:`
-+ssssssssssssssssssyyssss+-
.o0ssssssssssssssssssdMMMMNyssso.
/ssssssssssshdmmNNmyNMMMMhssssss/
+ssssssssshmydMMMMMMNdddyssssss+
/ssssssshNMMMyhhyyyhNMMMMhssssss/
.sssssssdMMMNhssssssssshNMMMdssssss.
+ssssshhhyNMMNysssssssssyNMMMyssssss+
ossyNMMMNyMMhssssssssshmmhssssssso
ossyNMMMNyMMhssssssssshmmhssssssso
+ssssshhhyNMMNysssssssssyNMMMyssssss+
.sssssssdMMMNhssssssssshNMMMdssssss.
/ssssssshNMMMyhhyyyhdNMMMMhssssss/
+ssssssssdmydMMMMMMNdddyssssss+
/ssssssssshdmmNNmyNMMMMhssssss/
.o0ssssssssssssssssssdMMMMNyssso.
-+ssssssssssssssssssyyssss+-
`:+ssssssssssssssssss+:`
.-/+00ssss00+/- .

n0obie@n0obie-VirtualBox
-----
OS: Ubuntu 16.04.6 LTS x86_64
Host: VirtualBox 1.2
Kernel: 4.15.0-88-generic
Uptime: 6 mins
Packages: 1953 (dpkg)
Shell: bash 4.3.48
Resolution: 1920x950
DE: Unity
WM: Compiz
WM Theme: Ambiance
Theme: Ambiance [GTK2/3]
Icons: ubuntu-mono-dark [GTK2/3]
Terminal: gnome-terminal
CPU: Intel i7-8750H (2) @ 2.207GHz
GPU: 00:02.0 VMware SVGA II Adapter
Memory: 827MiB / 7976MiB
```

Figura A.1: Especificaciones de la máquina virtual P4

A.1.3.2 Máquina virtual entorno XDP

- Procesador: Intel Core i7-8750H, 6 cores 2.2GHz-4.1GHz
- Memoria: 4GB DDR4, 2400MHz
- Sistema operativo: Ubuntu 18.04.3 LTS x86_64
- Kernel: 5.3.0-40-generic
- Configuración de red: Modo bridge, conectado a interfaz Intel(R) Wireless-AC 9560

```

n0obie@n0obie-VirtualBox:~$ neofetch
      .-/+oossssoo+/-.
      `:+ssssssssssssssss+:`
      -+ssssssssssssssssyyssss+-
      .ossssssssssssssssdMMMMyssso.
      /ssssssssshdmmNNmmyNMMMhsssss/
      +ssssssshmydMMMMMMNdddyssssss+
      /ssssssshNMMMyhhyyyhmmMMNhssssss/
      .sssssssdMMMNhsssssssshNMMMdssssss.
      +ssshhhyNMMNysssssssssyNMMMyssssss+
      ossyNMMMNyMMhssssssssssshmmhssssssso
      ossyNMMMNyMMhssssssssssshmmhssssssso
      +ssshhhyNMMNysssssssssyNMMMyssssss+
      .sssssssdMMMNhsssssssshNMMMdssssss.
      /ssssssshNMMMyhhyyyhdNMMNhssssss/
      +sssssssdmydMMMMMMNdddyssssss+
      /ssssssshdmmNNmmyNMMMhsssss/
      .ossssssssssssssssdMMMMyssso.
      -+ssssssssssssssssyyssss+-
      `:+ssssssssssssssss+:`
      .-/+oossssoo+/-.

n0obie@n0obie-VirtualBox
-----
OS: Ubuntu 18.04.3 LTS x86_64
Host: VirtualBox 1.2
Kernel: 5.3.0-40-generic
Uptime: 6 hours, 38 mins
Packages: 1921
Shell: bash 4.4.20
Resolution: 1920x950
DE: GNOME 3.28.4
WM: GNOME Shell
WM Theme: Adwaita
Theme: Ambiance [GTK2/3]
Icons: Ubuntu-mono-dark [GTK2/3]
Terminal: gnome-terminal
CPU: Intel i7-8750H (6) @ 2.207GHz
GPU: VMware SVGA II Adapter
Memory: 1174MiB / 3935MiB

```

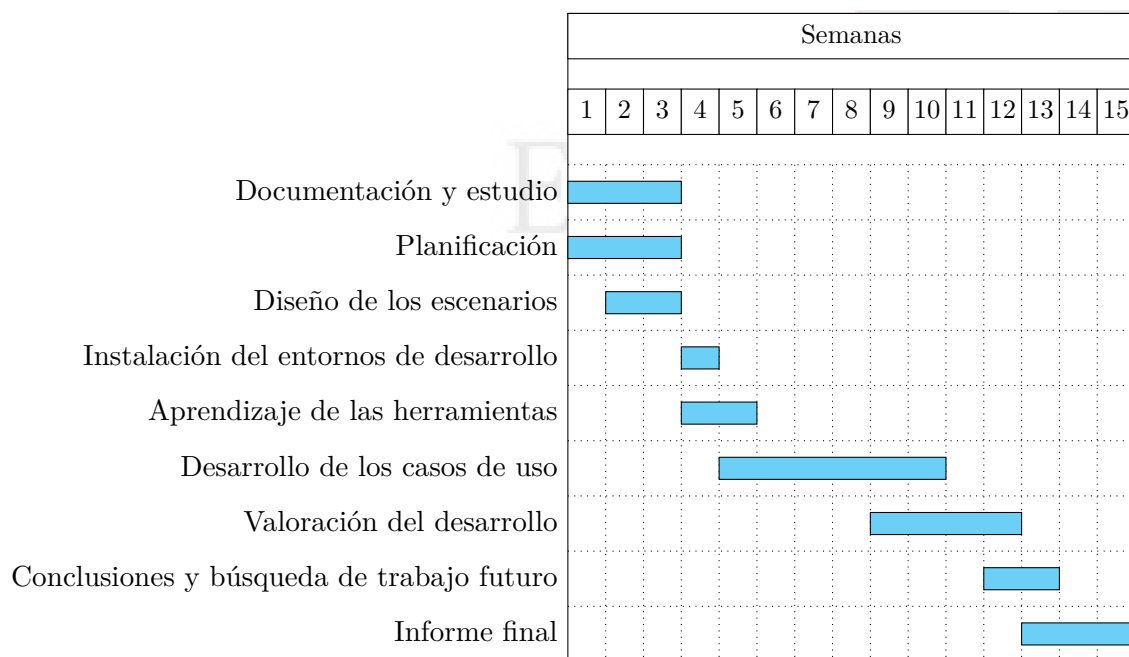
Figura A.2: Especificaciones de la máquina virtual XDP

B Anexo II - Presupuesto

En este anexo se expondrá de una forma detallada el presupuesto del proyecto. Para que este presupuesto sea lo más próximo a la realidad, se hará un breve análisis sobre la duración del proyecto. De esta forma, se podrán calcular la mano de obra con mayor exactitud.

B.1 Duración del proyecto

Con el propósito de obtener el número de horas de trabajo por semana del proyecto en **promedio**, se va a realizar un diagrama de Gantt. De este modo se podrá apreciar la distribución de tareas a lo largo del TFG y así ser capaces de obtener un número de horas de trabajo aproximado por semana.



Número de horas totales	Horas por semana	Horas diarias
425h	≈ 28h	≈ 4.2h

Tabla B.1: Promedio de horas de trabajo

B.2 Costes del proyecto

El cálculo de los costes del proyecto se va a realizar diferenciando previamente por *Hardware*, *Software* y mano de obra. De esta manera, se pretende que los costes se desglosen aportando claridad sobre la cuantía total.

Producto (IVA incluido)	Valor (€)
Ordenador portátil Lenovo Legion	1349,00
Ordenador de sobremesa	1450,00
Pantalla Lenovo L27i	129,99
Pantalla Benq 21"	79,89
Periféricos	150,00
Infraestructura de Red (PLCs y Router Livebox)	70,00

Tabla B.2: Presupuesto desglosado del Hardware

Las licencias de software generalmente se venden por años, o por meses. Por ello, se ha calculado el precio equivalente asociado a la duración del TFG.

Producto (IVA incluido)	Valor (€)
Microsoft Office	300,00
Adobe Photoshop y Adobe Premiere Pro	241,96

Tabla B.3: Presupuesto desglosado del Software

Se han tomado de referencia los honorarios de un ingeniero junior, los cuales corresponden a 20€ la hora. Los costes del *hardware* y *software* se han agregado como un único elemento, añadiéndolo al presupuesto con el valor total del desglose de los productos indicados.

Descripción (IVA incluido)	Unidades	Coste unitario (€)	Coste total (€)
Material Hardware	1	3228,89	3228,89
Material Software	1	541,96	541,96
Mano de obra	425	20,00	8500,00
Costes fijos (Luz, Internet)	4	76,00	304,00
TOTAL			12.574,855 €

Tabla B.4: Presupuesto total con IVA

C Anexo III - Manuales de usuario e Instalación

En este anexo se incluirán todos los manuales de usuario e instalación sobre aquellas herramientas que se crean necesarias para el desarrollo y comprobación de funcionamiento del TFG. De forma adicional, se comentará cómo funcionan los scripts de instalación generados para que cualquier persona interesada en replicar los distintos casos de uso, tenga un fácil acceso a ellos.

C.1 Instalación de dependencias de los casos de uso

La motivación de esta sección es plasmar en un punto como hacer uso de las herramientas que se han dejado desarrolladas para la instalación de las dependencias de los casos de uso. Como ya se indicó en el Pliego de condiciones, al tener dos entornos de trabajo muy diferenciados se iban a crear dos máquinas virtuales (??, ??) para conseguir aislar todo posible conflicto de dependencias. A continuación, se indicará como instalar las dependencias asociadas a cada entorno.

C.1.1 Instalación de dependencias máquina XDP

La tecnología XDP, al ser desarrollada propiamente en el Kernel de Linux, no necesitará de muchas dependencias para trabajar con ella. Todas las dependencias inducidas vienen por la necesidad de ciertos compiladores para establecer todo el proceso de compilación de un programa XDP, desde su C restringido hasta su forma de bytecode. Para instalar dichas dependencias se necesitará haber descargado el repositorio de este TFG en local. Esto se puede realizar según se indica en el bloque ??.

Código C.1: Descarga del repositorio del TFG

```
1  # En caso de no tener "git" instalado lo podemos hacer de la siguiente forma
2  sudo apt install -y git
3
4
5  # Una vez que está instalado git, haremos un "clone" del repositorio
6  git clone https://github.com/davidcawork/TFG.git
```

Una vez descargado el repositorio se debería encontrar un directorio llamado TFG en el directorio donde se haya ejecutado dicho comando. El siguiente paso para instalar las dependencias, será movernos hasta el directorio de los casos de uso XDP y lanzar el script de instalación con permisos de super-usuario según se indica en el bloque ??.

Código C.2: Instalación de dependencias XDP

```
1 # Nos movemos al directorio de los casos de uso XDP
2 cd TFG/src/use_cases/xdp/
3
4 # Lanzamos el script de instalación con permisos de super usuario
5 sudo ./install.sh
```

Este script de instalación añadirá los siguientes paquetes e inicializará el submódulo de la librería libbpf:

- Paquetes necesarios para el proceso de compilación de programas XDP: `clang` `llvm` `libelf-dev` `gcc-multilib`
- Paquete necesario para tener todos los archivos de cabecera del Kernel que se tenga instalado: `linux-tools-$(uname -r)`
- Paquete necesario en caso de querer hacer debug vía excepciones con la herramienta perf: `linux-tools-generic`

Por último, se quiere comentar el hecho de que es muy recomendable tener una versión superior a la v4.12.0 de iproute2 ya que en versiones anteriores no se da soporte para XDP. En Ubuntu 18.04 ya viene por defecto una versión compatible con XDP por lo que no será necesario actualizarla, más información en el punto ??.

C.1.2 Instalación de dependencias máquina P4

El entorno de trabajo P4 es bastante áspero y complicado, ya que se requieren de numerosas dependencias para poder empezar a trabajar con la tecnología P4. Por ello, para la instalación del entorno de P4 se ha dejado un script de instalación en el directorio de los casos de uso P4, bajo la carpeta `vm` con el nombre de `install.sh`. En el repositorio oficial, hay un método de instalación similar pero enfocado a un aprovisionamiento de Vagrant¹.

El equipo de *p4lang* monta una máquina virtual personalizada que al parecer del autor de este TFG es demasiado *User Friendly* ya que deja poco margen de maniobra para hacer una instalación más perfilada a un entorno de desarrollo real. Por ello, se ha tenido que

¹<https://www.vagrantup.com/>

desarrollar un script propio para su instalación. Esta nueva vía de instalación fue ofrecida en forma de pull-request al equipo *p4lang* se puede consultar [aquí](#).

En primer lugar, se debe descargar el repositorio de este TFG. Si no lo ha hecho aún puede consultarlo en el bloque **??**. Acto seguido, se deberá navegar hasta el directorio de los casos de uso P4 y lanzar el script como se indica en el bloque **??**.

Código C.3: Instalación de dependencias P4

```
1  # Nos movemos al directorio de los casos de uso XDP
2  cd TFG/src/use_cases/p4/
3
4  # Lanzamos el script de instalación con permisos de super usuario
5  sudo ./vm/install.sh -q
```

Este script de instalación añadirá los siguientes paquetes y herramientas necesarias para el desarrollo en P4:

- Paquetes necesarios que son dependencias de las herramientas principales de P4env.
- Herramientas del P4env: P4C PI P4Runtime
- Paquetes necesarios para la prueba de P4: Mininet BMV2 gRPC Protobuf

C.2 Herramienta *iproute2*

Se ha querido añadir esta sección, ya que la herramienta *iproute2* va a ser fundamental a la hora de cargar los programas XDP en el Kernel, consultar interfaces, o verificar en que *Network namespace* se encuentra el usuario. Por todo lo anterior, la herramienta *iproute2* será una de las piezas claves para gestión de las *Network Namespaces*, y la verificación de los casos de uso.

C.2.1 ¿Qué es *iproute2*?

Iproute2 es un paquete utilitario de herramientas para la gestión del *Networking* en los sistemas Linux. Además, se encuentra ya en la mayoría de las distribuciones actuales. Sus desarrolladores principales son Alexey Kuznetsov y Stephen Hemminger, aunque hoy en día es un proyecto opensource donde cientos de personas contribuyen activamente en el repositorio².

Actualmente, la versión más reciente de la herramienta es v5.2.0. Dicha versión será la que se utilizará en Ubuntu 18.04. El conjunto de utilidades que ofrece *iproute2* está

²<https://github.com/shemminger/iproute2>

pensado para la sustitución de herramientas que se recogen en el paquete de **net-tools**, como por ejemplo a **ifconfig**, **route**, **netstat**, **arp**, etc. En la tabla ?? se pueden apreciar las herramientas de net-tools equivalentes en iproute2.

net-tools	iproute2
arp	ip neigh
ifconfig	ip link
ifconfig -a	ip addr
iptunnel	ip tunnel
route	ip route

Tabla C.1: Comparativa de herramientas Iproute2 con paquete net-tools

C.2.2 ¿Por qué necesitamos iproute2?

Cuando se está trabajando con los programas XDP y se quiere comprobar su funcionamiento, se debe compilarlos. Esto se hará con los compiladores LLVM³ más clang⁴, como ya se comentaba en el estado del arte. Este proceso de compilación convertirá el código de los programas XDP, en un *bytecode* Berkeley Packet Filter (BPF), y más tarde, se almacenará este *bytecode* en un fichero de tipo Executable and Linkable Format (ELF). Una vez compilados, se tendrá que anclarlos en el Kernel, y es en este punto es donde entrará iproute2, ya que tiene un cargador ELF (generalmente se trabajará con extensiones del tipo *.o).

Además, la herramienta iproute2 permite al usuario comprobar si una interfaz tiene cargado un programa XDP. Arrojando en dicho caso, el identificador del programa XDP, que tiene anclado la interfaz y si este programa está cargado de una forma nativa o de una forma genérica. Al final de la esta sección, se indicará cómo hacer esta comprobación.

C.2.3 Estudio de compatibilidad de la herramienta iproute2 en Ubuntu

Al trabajar con esta herramienta para cargar programas XDP, se necesita la versión que soporte el cargador ficheros ELF. Si usted tiene la versión de iproute2 que viene instalada por defecto en Ubuntu 16.04, le indicamos que aún no da soporte a XDP. Inicialmente se buscó información relativa a partir de que versión se daba soporte a XDP tanto en Ubuntu 16.04, como en Ubuntu 18.04. Como no se encontró información precisa sobre ello, se ha realizado un estudio de la compatibilidad de iproute2 a través de Ubuntu 16.04 y Ubuntu 18.04.

³<https://llvm.org/>

⁴<https://clang.llvm.org/>

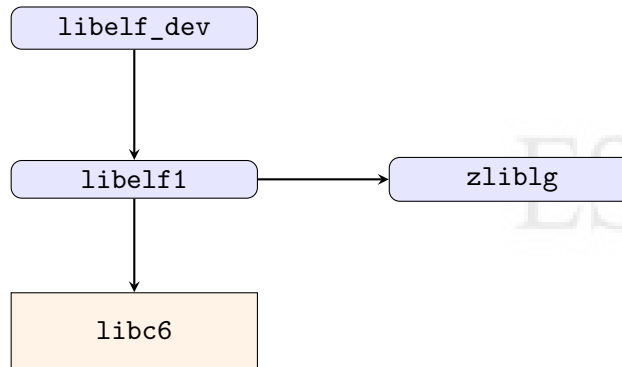


Figura C.1: Ramificación de dependencias de Iproute2.

Este estudio de compatibilidad se llevó a cabo descargando cada versión de iproute2, compilándola, e instalándola en nuestra máquina. Por último, para verificar si dicha versión daba soporte a XDP, se comprobaba si un programa XDP genérico que se sabía que funcionaba, cargaba o no, y si éste mostraba estadísticas sobre su carga. Más adelante, se indicará cómo compilar e instalar una versión en particular de iproute2.

Como se puede apreciar en la siguiente tabla ??, en Ubuntu 16.04 a partir de la versión v4.14.0 no existe compatibilidad. Esto es debido a que requiere librerías de enlazado extensible de formato (No ELF Support). Para resolver este requerimiento se debería añadir una versión más reciente de la librería **libelf_dev**. Se puede agregar dicha librería, pero al hacerlo aparecerán dependencias que se van ramificando una a una llegando a librerías más sensibles para nuestro sistema como **libc6**, por lo que se decidió no comprobar el funcionamiento añadiendo las nuevas librerías requeridas para no comprometer el sistema.

Versiones IProute2	v4.9.0	v4.10.0	v4.11.0	v4.12.0	v4.13.0	v4.14.0	v4.15.0	v4.16.0	v4.17.0	v4.18.0	v4.20.0	v5.1.0	v5.2.0
Ubuntu 16.04	No XDP supp	No XDP supp	No XDP supp	Si	Si	No	No	No	No	No	No	No	No
Ubuntu 18.04	-	-	-	-	-	-	Si	Si	Si	Si	Si	Si	Si

Tabla C.2: Estudio de compatibilidad de la herramienta Iproute2

C.2.4 Compilación e instalación de iproute2

El proceso es prácticamente análogo tanto en Ubuntu 16.04 como en Ubuntu 18.04, salvo por una única diferencia que se indicará más adelante. Ahora se mostrarán los pasos necesarios para la compilación e instalación de una versión, en concreto de la herramienta iproute2.

- En primer lugar, se necesitará de instalar los paquetes necesarios para la configuración previa a la compilación.
 - **bison**, es un herramienta generadora de analizadores sintácticos de propósito general.
 - **flex**, es una herramienta para generar programas que reconocen patrones léxicos en el texto.
 - **libmnl-dev**, es una librería de espacio de usuario orientada a los desarrolladores de Netlink. Netlink⁵ es una interfaz entre espacio de usuario y espacio de Kernel vía sockets.
 - **libdb5.3-dev**, éste es un paquete de desarrollo que contiene los archivos de cabecera y librerías estáticas necesarias para la BBDD de Berkley (*Key/Value*).
 - Se entiende que se tiene el paquete **wget**. En caso de no tenerlo, solo se deberá añadir para poder descargar la herramienta.

Código C.4: Instalación de las dependencias de Iproute2

```
1 sudo apt-get install bison flex libmnl-dev libdb5.3-dev
```

- En segundo lugar, se debe descargar el comprimido de la herramienta iproute2. Al haber varios paquetes, se descargará aquel cuya versión sea con la que se quiere trabajar. Podemos descargarlas desde aquí: kernel.org.

Código C.5: Obtención del source de Iproute2

```
1 wget -c http://ftp.iiij.ad.jp/pub/linux/kernel/linux/utils/net/iproute2/iproute2-4.15.0.tar.gz
```

- En tercer lugar, se debe descomprimir el comprimido de la herramienta. Acto seguido, se procederá a configurarla, compilarla e instalarla.

⁵<https://www.man7.org/linux/man-pages/man7/netlink.7.html>

Código C.6: Compilación e instalación de Iproute2

```
1 # Se descomprime y se entra al directorio
2 tar -xvfz $(tar).tar.gz && cd $tar
3
4 # Se configura
5 ./configura
6
7 # Se compila e instala, para añadir el nuevo binario en el path
8 sudo make
9 sudo make install
```

C.2.4.1 Diferencias con Ubuntu 18.04

La única diferencia en el proceso de instalación de la herramienta de iproute2 en Ubuntu 18.04, es añadir un paquete extra antes de proceder a configurar, compilar e instalar. El paquete extra es **pkg-config**; de no añadirlo fallará al lanzar el script de configuración y hacer el build.

Código C.7: Instalación de las dependencias de Iproute2 - Ubuntu 18.04

```
1 sudo apt-get install bison flex libmnl-dev libdb5.3-dev pkg-config
```

C.2.5 Comandos útiles con iproute2

A continuación, se indican los comandos más frecuentes con la herramienta iproute2. Todos ellos han sido utilizados en el proceso de desarrollo del proyecto y en el proceso de verificación de los distintos casos de uso. Por ello, se considera que este apartado puede ser de gran utilidad para el lector que nunca ha trabajado con esta herramienta.

Código C.8: Comandos útiles con iproute2

```
1 # Listar interfaces y ver direcciones asignadas
2 ip addr show
3
4 # Poner/Quitar dirección a una interfaz
5 ip addr add {IP} dev {interfaz}
6 ip addr del {IP} dev {interfaz}
7
8 # Levantar/deshabilitar una interfaz
9 ip link set {interfaz} up/down
10
11 # Listar rutas
12 ip route list
13
14 # Obtener ruta para una determinada dirección IP
15 ip route get {IP}
16
17 # Listar Network namespace con nombre
18 ip netns list
```


C.3 Herramienta tcpdump

La motivación de añadir esta sección ha sido la de tener un punto de encuentro para las personas que nunca han utilizado tcpdump, ya que a lo largo de todas las secciones del proyecto, se hará uso de esta herramienta para verificar si los casos de uso realmente funcionan según lo esperado.

C.3.1 ¿Qué es tcpdump?

Tcpdump es un analizador de tráfico para inspeccionar los paquetes entrantes y salientes de una interfaz. La peculiaridad de esta herramienta es que funciona por línea de comandos, y tiene soporte en la mayoría de sistemas UNIX⁶, como por ejemplo Linux, macOS y OpenWrt. La herramienta está escrita en lenguaje C por lo que tiene un gran rendimiento y hace uso de libpcap⁷ como vía para interceptar los paquetes.

La herramienta fue escrita en el año 1988 por trabajadores de los laboratorios de Berkeley. Actualmente, cuenta con una gran comunidad de desarrolladores a su espalda en su repositorio oficial⁸ sacando nuevas actualizaciones de forma periódica (última versión v4.9.3).

C.3.2 ¿Por qué necesitamos tcpdump?

Hoy en día, es un hecho que en la mayoría de los casos no se suele desarrollar en una misma máquina. Se suele utilizar contenedores o máquina virtuales con el propósito de tener acotado el escenario de desarrollo. Por ello, se suele trabajar la mayoría de veces de forma remota, conectándose a la máquina/contenedor haciendo uso de ssh⁹.

Esto implica numerosas ventajas, pero también complicaciones. Si una persona no sabe configurar un *X Server* con el cual ejecutar aplicaciones gráficas de forma remota, no podría correr por ejemplo Wireshark. En este punto entra tcpdump, el cual no requiere de ningún tipo de configuración extra para poder ser ejecutado de forma remota. Esto añadido al hecho de su rápida puesta en marcha, con respecto a otros *sniffers* como Wireshark, han convertido a tcpdump en una herramienta fundamental en los procesos de verificación de los casos de uso.

⁶Unix es un sistema operativo desarrollado en 1969 por un grupo de empleados de los laboratorios Bell

⁷<https://github.com/the-tcpdump-group/libpcap>

⁸<https://github.com/the-tcpdump-group/tcpdump>

⁹<https://www.ssh.com/ssh/>

C.3.3 Instalación de tcpdump

Como ya se comentaba en la introducción, esta herramienta tiene un gran soporte entre los sistemas UNIX, por lo que generalmente suele encontrarse ya instalado en la mayoría de distribuciones Linux. De no tenerla instalada, siempre se podrá instalar de la siguiente forma ??.

Código C.9: Instalación de Tcpdump

```
1 sudo apt install tcpdump
```

C.3.4 Comandos útiles con tcpdump

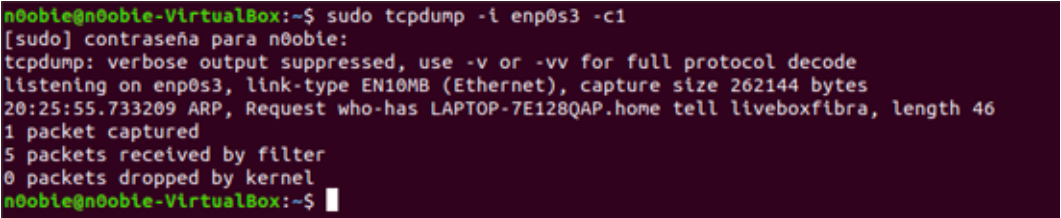
A continuación, se indican los comandos más frecuentes con la herramienta tcpdump. Todos ellos han sido utilizados en su mayoría en el proceso de verificación de los distintos casos de uso. Por lo tanto, se considera que este apartado puede ser de gran utilidad para el lector que nunca ha trabajado con esta herramienta. Además, se recomienda al lector consultar su *man-page*¹⁰ donde podrá encontrar información más detallada sobre el uso básico de tcpdump.

Código C.10: Comandos útiles con Tcpdump

```
1 # Indicar sobre que Interfaz se quiere escuchar
2 tcpdump -i {Interfaz}
3
4 # Almacenar la captura a un archivo para su posterior análisis
5 tcpdump -w fichero.pcap -i {Interfaz}
6
7 # Leer captura desde un archivo
8 tcpdump -r fichero.pcap
9
10 # Filtrar por puerto
11 tcpdump -i {Interfaz} port {Puerto}
12
13 # Filtrar por dirección IP destino/origen
14 tcpdump -i {Interfaz} dst/src {IP}
15
16 #Filtrar por protocolo
17 tcpdump -i {Interfaz} {protocolo}
18
19 # Listar interfaces disponibles
20 tcpdump -D
21
22 # Limitar el número de paquetes a escuchar
```

¹⁰<https://linux.die.net/man/8/tcpdump>

```
23 tcpdump -i {Interfaz} -c {Número de paquetes}
```



```
n0obie@n0obie-VirtualBox:~$ sudo tcpdump -i enp0s3 -c1
[sudo] contraseña para n0obie:
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
20:25:55.733209 ARP, Request who-has LAPTOP-7E128QAP.home tell liveboxfibra, length 46
1 packet captured
5 packets received by filter
0 packets dropped by kernel
n0obie@n0obie-VirtualBox:~$
```

Figura C.2: Interfaz CLI de Tcpcdump

