UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA
DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Aplicación de técnicas de *Deep Learning* para mejorar el enrutamiento en redes comunitarias inalámbricas basadas en OLSR

Autor:

D. Enrique Díez Benito

Tutores:

Dr. D. Miguel Luis Bote Lorenzo

Dr. D. Eduardo Gómez Sánchez

Valladolid, 16 de julio de 2020

TÍTULO:	**Aplicación de técnicas de *Deep Learning* para mejorar el enrutamiento en redes comunitarias inalámbricas basadas en OLSR**
AUTOR:	**D. Enrique Díez Benito**
TUTORES:	**Dr. D. Miguel Luis Bote Lorenzo**
	Dr. D. Eduardo Gómez Sánchez
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Ignacio de Miguel Jiménez**
VOCAL:	**Dr. D. Mario Martínez Zarzuela**
SECRETARIO:	**Dr. D. Federico Simmross Wattenberg**
SUPLENTE 1:	**Dr. D. Juan Pablo Casaseca de la Higuera**
SUPLENTE 2:	**Dr. D. Guillermo Vega Gorgojo**

FECHA:	**16 de julio de 2020**
CALIFICACIÓN:	

Resumen del TFG

Una red comunitaria inalámbrica (*Wireless Community Network*, WCN) es una red inalámbrica en malla creada por un grupo local de personas dando lugar a una infraestructura de red alternativa y autogestionada. Las WCN son redes que crecen y decrecen de forma dinámica y cuyos enlaces se caracterizan por ser asimétricos y escasamente fiables. En este contexto, la selección de rutas adecuadas para el encaminamiento del tráfico es necesaria para ofrecer a sus usuarios acceso a Internet con una buena calidad de servicio.

Las características de estas redes hacen conveniente el uso de protocolos de encaminamiento como OLSR, usando la calidad de enlace como métrica de coste. El uso de técnicas de aprendizaje automático para la predicción del estado futuro de la calidad de enlace puede ser crítico a la hora de mejorar el encaminamiento con rutas en las que se prevé una menor probabilidad de pérdida de paquetes. Trabajos previos han tratado de predecir la calidad con el uso de técnicas de aprendizaje automático pero sin tener en cuenta características clave del funcionamiento de OLSR como el *fish-eye* (ojo de pez).

El objetivo del trabajo es aplicar técnicas de aprendizaje profundo (*Deep Learning*) para predecir la calidad del enlace en redes WCN. Para ello, en primer lugar, se estudia el protocolo OLSR concreto de una WCN, *Funkfeuer Graz*, y se discuten las implicaciones necesarias que se han de tomar en cuenta para la recogida de datos y el futuro entrenamiento de técnicas de aprendizaje automático.

A continuación, se realiza y analiza una captura de tráfico de la red *Funkfeuer Graz*, obtenida a través de una VPN. Tras su análisis, se procede a realizar la predicción con una técnica de aprendizaje profundo, LSTM (*Long Short-Term Memory*), debido a que este tipo de RNN (*Recurrent Neural Network*) ha tenido éxito en la predicción de series temporales en otros ámbitos. Se prueban distintas arquitecturas de LSTM como es el de LSTM con y sin estado, predicción múltiple o *encoder-decoder* (codificador-decodificador).

Se observa que el mecanismo de *fish-eye* de OLSR hace que la información que un nodo tiene de otros sea distinta según su distancia y que este efecto se debe tener en cuenta a la hora de predecir y entrenar. Además, se muestra que el uso de técnicas LSTM, costosas computacionalmente, no mejora de manera significativa al algoritmo de referencia salvo con enlaces cercanos o en la predicción a varios instantes vista.

Palabras Clave

Redes comunitarias, OLSR, predicción calidad de enlace, aprendizaje profundo, LSTM.

Abstract

A Wireless Community Network (WCN) is a wireless mesh network created by a local group of people resulting in an alternative, self-managed network infrastructure. WCNs are networks that grow and decrease dynamically, and their links are characterized by being asymmetric and barely reliable. In this context, the selection of suitable routes for routing traffic is necessary to offer its users a good quality of service.

The characteristics of these networks make it convenient to use routing protocols such as OLSR, using link quality as a cost metric. The use of Machine Learning techniques for predicting the future state of link quality can be critical in improving routing with routes that foresee a lower probability of packet loss. Previous works have attempted to predict link quality with the use of Machine Learning techniques but without taking into account the key features of OLSR behaviour such as the fish-eye.

The objective of this work is to apply Deep Learning techniques to predict link quality in WCN networks. To do this, first, the specific OLSR protocol of a WCN, Funkfeuer Graz, is studied and the implications to be considered for data collection and future training in Machine Learning techniques are discussed.

Next, a traffic capture of the Funkfeuer Graz network, obtained through a VPN, is made and analysed. After analysis, the prediction will be made with a Deep Learning technique, LSTM (Long Short-Term Memory), because this type of RNN (Recurrent Neural Network) has been successful in predicting time series problems. Different LSTM architectures are tested, such as stateless/stateful LSTM, multiple prediction or encoder-decoder.

The observation of the OLSR fish-eye mechanism makes one node has different information from others according to their distance and this effect must be considered when predicting and training. Furthermore, it is shown that the use of computationally expensive LSTM techniques does not significantly improve the reference algorithm except with close links or in multi-step prediction.

Keywords

Community networks, OLSR, link quality prediction, Deep Learning, LSTM.

Agradecimientos

Me gustaría agradecer por la ayuda en la realización de este trabajo a mis dos tutores, Miguel y Eduardo, sin olvidarme de Asen, por el esfuerzo que han realizado a la hora de resolver mis dudas o escucharme en las largas reuniones que hemos tenido durante el desarrollo de este. De igual forma, me gustaría agradecer a Andreas Jakum por su generosidad al darnos acceso a la red de *Funkfeuer* para una nueva captura de datos. También a mis compañeros de clase, amigos en muchos casos, que me han acompañado a lo largo de estos cuatro años y han provocado que el hecho de ir a clase sea más que entretenido.

Gracias.

Índice General

Lista de Figuras

Lista de Tablas

Capítulo 1. Introducción

1.1 Motivación

Una red comunitaria inalámbrica (*Wireless Community Network*, WCN) es una red inalámbrica en malla creada por un grupo local de personas dando lugar a una infraestructura de red alternativa y autogestionada. Esta infraestructura se utiliza habitualmente para: la interacción entre sus usuarios (mensajería, compartición de recursos, etc.) y llevar el acceso a Internet a sitios donde no está presente [1].

Estas redes crean un gran impacto social al proveer a la comunidad la oportunidad de la comunicación a través de Internet [2]. Las redes de esta naturaleza son extremadamente diversas y dinámicas porque están compuestas de nodos descentralizados y se mezclan muchos tipos de tecnologías, protocolos, aplicaciones y servicios [3]. Con el paso del tiempo y la importancia que ha ganado el acceso a Internet en la sociedad, las redes comunitarias inalámbricas han crecido de manera notable en muchos países. Dado que estas redes son gestionadas por los miembros de una comunidad, hay que tener en cuenta su escalabilidad, por lo que estas redes deben tener tecnologías y protocolos que permitan el funcionamiento de la red de la forma más autónoma posible, reduciendo las acciones de gestión y control al mínimo [4].

Las WCN, debido a su gran, descentralizada y heterogénea estructura, plantean grandes desafíos que pueden ser de interés para los investigadores, bien como fuente de inspiración bien como campo de aplicación. Las redes de comunidad son normalmente redes inalámbricas, debido al bajo coste al construir este tipo redes a gran escala [5]. Además, son redes que crecen y decrecen de forma dinámica y cuyos enlaces se caracterizan por ser asimétricos y escasamente fiables, dando lugar a problemas que dificultan la posibilidad de ofrecer a sus usuarios acceso a Internet con una buena calidad de servicio. Uno de los problemas más importantes es el de la selección de rutas para el encaminamiento del tráfico.

Las características de estas redes hacen necesario el uso de protocolos de encaminamiento que usan la calidad de enlace (*link quality*, LQ) como métrica de coste. La selección de enlaces con mayor calidad frente a usar los enlaces con menor número de saltos permite maximizar la tasa de entrega y minimizar la congestión de tráfico [6]. Como consecuencia, el seguimiento de la calidad de enlace recobra vital importancia para obtener una buena calidad de servicio. Además, se ha mostrado que se deben evitar a toda costa los enlaces con mala calidad siempre que sea posible [7] y tan pronto como se pueda [8].

La predicción de LQ puede verse como un problema de predicción de series temporales. Una serie temporal es una secuencia de valores ordenados de una variable tomados en intervalos de tiempo normalmente equiespaciados. Entre las aplicaciones del análisis de las series temporales se encuentran la obtención del conocimiento subyacente de las variables o la realización de modelos para su predicción y monitorización [9]. La predicción de LQ es una propuesta que incrementa las mejoras en el enrutamiento logradas con el seguimiento de LQ. Normalmente, las métricas calculadas en tiempo real no proporcionan suficiente información para detectar la degradación o activación de un enlace en el momento correcto. Por lo tanto, técnicas de predicción podrían ser necesarias para prever cambios futuros de LQ y tomar acciones apropiadas [2]. Se podría, por ejemplo, predecir un enlace que va a sufrir una pérdida de calidad y mejorar el encaminamiento con

otra ruta con mayor calidad, en la que se prevé una menor probabilidad de pérdida de paquetes.

En la literatura es posible encontrar trabajos que proponen distintas aproximaciones de uso de técnicas de aprendizaje automático que ayuden a abordar este problema. En [2] se estudia el uso de aprendizaje automático en lote para la predicción de la calidad de los enlaces con un horizonte temporal de 5 minutos. En [10] y [11] se prueban distintos algoritmos de aprendizaje automático en línea (reduciendo coste computacional), con el mismo horizonte temporal y además se añade un algoritmo de referencia (*baseline*) para poder comparar los resultados. Uno de los problemas de estas dos investigaciones es el sentido práctico que tiene predecir la calidad de enlace con horizontes temporales tan grandes, ya que en 5 minutos pueden haber pasado muchas cosas y más a la velocidad a la que funcionan las redes. Es por eso por lo que en [12] se aplican de nuevo las técnicas de aprendizaje automático en línea de [10] pero con un nuevo conjunto de datos con mayor tasa de muestreo, permitiendo predecir con horizontes temporales más cortos. En ese trabajo, se observa que con alta tasa de muestro de LQ, el algoritmo de referencia basado en predecir la muestra anterior de LQ iguala o supera en muchas ocasiones a las técnicas de aprendizaje automático.

Además, todos esos trabajos han tratado de predecir pero sin tener en cuenta características clave del protocolo de encaminamiento del que se obtuvieron el conjunto de datos, *Optimized Link State Routing* (OLSR), ya que estos estudios toman las muestras de la WCN de *Funkfeuer*[1], que usa una implementación de OLSR, OLSR Daemon[2] (OLSRd). Entre estas características se encuentra el mecanismo *fish-eye*, que hace que la información de los enlaces que obtiene un nodo dependa de la distancia en saltos de los nodos que la generan. Este mecanismo junto con otros propios de OLSR, como el de un tiempo de retención aleatorio de cada paquete, hace que se deba tener mayor cuidado en la obtención de la serie temporal de LQ, crítica para los modelos de predicción posteriores.

Otro campo de estudio en auge es el del aprendizaje profundo (*Deep Learning)* y para el caso de predicciones temporales, está teniendo relevancia el algoritmo *Long short-term memory* (LSTM) [13]. Por ejemplo, en [14] diseñan una arquitectura LSTM que es capaz de predecir el nivel de señal inalámbrica futuro a partir de niveles de señales pasados. Hay que tener en cuenta el incremento de la capacidad computacional durante la última década y la cantidad de datos disponibles. Además, se suma el avance en el campo del aprendizaje automático, proporcionándonos la oportunidad de diseñar modelos que mejoren el rendimiento de predicción de variaciones de calidad de canales inalámbricos [15], [16]. En [15] se identifican muchas oportunidades para el uso del *Deep Learning* en redes inalámbricas y se enfatiza su buen rendimiento. Durante la etapa final de este trabajo, se publicaron nuevos trabajos sobre el uso de aprendizaje profundo y la calidad de enlace. En [17] se hace la predicción de calidad de enlace, en este caso de la relación señal/ruido, entrenado para ello dos modelos LSTM. Un modelo es entrenado con la parte determinística de la señal y otro con la estocástica (descompuestas mediante un algoritmo de transformada *wavelet*), prediciendo conjuntamente la salida determinística y la varianza de la parte estocástica. En [18] se prueban modelos de aprendizaje profundo de redes neuronales recurrentes, entre ellas LSTM, para predecir los datos de LQ usando una captura de la red *Funkfeuer* de Viena. De nuevo, este estudio usa datos muestreados cada 5 minutos con los inconvenientes que conlleva. Además, no se comparan los resultados con un algoritmo básico de referencia. En este trabajo, indican que los modelos de aprendizaje profundo propuestos

[1] https://www.funkfeuer.at/

[2] http://www.olsr.org/mediawiki/index.php/Olsr_Daemon

mejoran a los de las redes neuronales clásicas y remarcan su gran coste computacional respecto a otras técnicas de predicción en línea.

1.2 Objetivos

El presente trabajo tiene como objetivo **estudiar nuevas alternativas para la predicción de la calidad de enlace** en las redes inalámbricas comunitarias mediante el uso de técnicas de *Deep Learning*. Para ello, se pretende **analizar los efectos e implicaciones concretas** que suponen el hecho de predecir la calidad de enlace cuando los datos se obtienen con el protocolo de encaminamiento **OLSR**. A estas características concretas hay que sumarle las particularidades de este protocolo en la red comunitaria inalámbrica de estudio, *Funkfeuer Graz*, que investigaciones anteriores no tuvieron en cuenta.

Por consiguiente, los objetivos principales de este trabajo son:

- **Estudiar** las implicaciones del uso de **OLSR** a la hora de predecir la calidad de enlace.
- **Analizar** la red de estudio ***Funkfeuer Graz***, viendo las particularidades que se derivan de su configuración concreta.
- Obtener acceso a la red *Funkfeuer Graz* a través de una VPN (*Virtual Private Network*), para **realizar una captura de tráfico OLSR y comprobar las conclusiones** a las que se llegaron en los primeros dos puntos. Si fuese necesario, conseguir una nueva captura de datos para futuras investigaciones.
- Por último, el más importante, es el de **estudiar el uso de algoritmos** de *Deep Learning* para la predicción de calidad de enlace. Este trabajo se centrará en el uso de la técnica de **LSTM**, probando distintos modelos y estructuras, comparándolos entre ellos y con un algoritmo de referencia.

1.3 Metodología

La metodología seguida en este Trabajo Fin de Grado es la propuesta por el método de ingeniería [19]. Este método es un enfoque sistemático para alcanzar la solución deseada a un problema. A continuación, se explican cada una de las etapas y cómo se abordan en este trabajo:

1. **Idea**: esta fase surge con un problema, que en un principio está definido de manera pobre y requiere investigación para ver su viabilidad y factibilidad, es decir, si tiene valor buscarle solución y es realizable. En nuestro caso, el problema inicial fue el de que la calidad de enlace no se actualice lo suficientemente rápido para evitar enlaces con mala calidad. Además, está el problema de intentar dar una solución al hecho de que con alta tasa de muestreo, los algoritmos de aprendizaje automático probados en [12] no mejoraban prácticamente al algoritmo básico de referencia, el *baseline*. Este problema es viable porque puede proporcionar un nuevo estado del arte y es factible porque se dispone de las herramientas software y un ordenador con una buena tarjeta gráfica que dispone el grupo de investigación en el que se desarrolla este trabajo.
2. **Concepto**: esta fase trata de generar modelos de todo tipo, matemáticos,

físicos, simulaciones o borradores para cumplir los requisitos marcados. Para esta etapa, hay que tener en cuenta que ya hay trabajos previos y no se necesitan generar cosas nuevas sino adaptarlas al nuevo contexto. También surgen nuevos modelos de arquitecturas LSTM.

3. **Planificación**: en esta etapa se define el plan de implementación, elección de personas, tareas, duración de cada tarea, presupuesto… En este caso al ser un trabajo de Fin de Grado es un trabajo individual y no tiene presupuesto. La planificación se realizó al comienzo del curso académico, donde se establecieron las horas necesarias para la formación para la realización de este trabajo. Fueron 150 horas para realizar cursos de aprendizaje automático, dudas a mis tutores o lectura de artículos científicos que tuvieron lugar en las horas de Prácticas en Empresa (GSIC/EMIC[3]). El resto de las horas fueron marcadas por las propias del TFG y las extras por la obtención de una beca de investigación. La planificación fue la siguiente:

    - El estudio a fondo las características de OLSR, tanto el protocolo estándar como el que se implementa en la red de estudio, OLSRd. Se encuentran las principales implicaciones en la obtención de los valores LQ y cómo deben tenerse en cuenta en la posterior implementación de modelos de predicción.
    - La captura de tráfico OLSR a través de una VPN de la red de estudio, *Funkfeuer Graz*. Se comprueban las hipótesis del estudio anterior y se buscan nuevas consideraciones importantes que puedan surgir. También, se puede usar la captura para futuras investigaciones.
    - Obtener y estudiar los datos de LQ del trabajo de [12], teniendo las consideraciones estudiadas en los dos puntos anteriores.
    - Seleccionar un subconjunto de enlaces para la predicción.

    Después, se pasaría al estudio de los datos y OLSR para implementar distintos modelos LSTM. Para todas esas horas se hace uso de una hoja de cálculo en línea que permite contabilizarlas e indicar la tarea que se hizo en ellas. Gracias a ello, se permite llevar una planificación y poder darse cuenta de tareas que quizás estén llevando un tiempo excesivo.

4. **Diseño**: es la fase donde se detallan los detalles y se establecen las especificaciones. Es aquí donde se diseña cada uno de los experimentos que tiene lugar en el trabajo, se hace de forma iterativa según los resultados de la iteración anterior.

5. **Desarrollo**: es la etapa en la que se genera documentación de ingeniería, esquemas, código o cualquier otra cosa que demuestre la solución al problema planteado. La solución puede ser tangible o intangible como una simulación funcional. Esta etapa ha sido la más larga en el trabajo. La primera fase consiste en realizar las tareas explicadas en la fase de planificación, relacionada con el estudio de protocolos y obtención del tráfico OLSR de la red de estudio.

    Una vez obtenida una selección de enlaces con los que hacer pruebas de modelos de predicción, se realiza de manera iterativa:

[3] https://www.gsic.uva.es/index.php?lang=es

- Diseño del código del modelo LSTM de estudio.
- Depuración de errores.
- Conclusiones de los resultados y en función de estos se realiza la variación de los parámetros o modelo y vuelta a empezar.

6. **Lanzamiento**: en esta fase se produce el lanzamiento del diseño de ingeniería y la documentación para su producción, con un prototipo funcional. Este trabajo no tiene por objetivo hacer un producto final, simplemente pretende explorar nuevas técnicas de estimación que podrían usarse posteriormente en un sistema real. Podría considerarse la entrega de este trabajo como el propio lanzamiento.

Hay que tener en cuenta que en este caso hay etapas que no se llegan a completar debido al contexto del trabajo, que no es el de crear un producto sino el de estudiar una red concreta y explorar nuevos métodos para la predicción de la calidad de enlace.

1.4 Estructura

En cuanto a la organización de este trabajo, en el capítulo 2 se explican las bases para la realización de este trabajo. Se expondrán las características de las redes de estudio, las redes de comunidad inalámbrica y se entrará en el funcionamiento del protocolo de encaminamiento OLSR y de la aplicación OLSRd, que implementa este protocolo en la red de *Funkfeuer*. Además, se tratarán las virtudes de la predicción de LQ.

En el capítulo 3 se habla de la red concreta de la que tenemos datos de calidad de enlace. Tras ver la configuración específica de OLSRd de esa red y haber estudiado su funcionamiento en el capítulo anterior, se enumeran las implicaciones en la captura de datos de LQ que se deben hacer. En este capítulo, se explican los distintos conjuntos de datos de los que se disponen y además, se crea uno nuevo con una captura de tráfico OLSR. Se analiza esa captura y se comprueban las implicaciones anteriores. Por último, se eligen una serie de enlaces con los que evaluar los modelos de predicción.

En el capítulo 4 se hace una introducción al *Deep Learning* y al algoritmo empleado para las predicciones, LSTM. En este capítulo también se habla del software y hardware empleado para los experimentos.

En el capítulo 5 se muestran los resultados de la predicción con el uso de distintos modelos de LSTM para evaluar el rendimiento entre estos modelos y el algoritmo de referencia. A continuación de explicar la métrica de evaluación empleada, se encuentran los mejores parámetros para un modelo LSTM convencional y después, se prueban distintos modelos como LSTM bidireccional o el de predicción múltiple.

Para finalizar, en el capítulo 6 se encuentran las conclusiones extraídas de este trabajo y los trabajos futuros. Además, se añaden dos apéndices, en el primero se explica cómo se consiguió el tráfico OLSR de la red de *Funkfeuer Graz* y el segundo contiene figuras y tablas del capítulo 5.

Capítulo 2. Planteamiento del problema

2.1 Introducción

Como se ha indicado en el capítulo anterior, el objetivo de este trabajo es el de abordar la predicción de calidad de enlace con algoritmos de aprendizaje profundo que no se han probado hasta el momento. Todo ello con el objetivo de mejorar el encaminamiento en las redes de comunidad inalámbricas, partiendo del hecho de que la información de LQ (*link quality*) que llegue a los nodos puede no estar lo suficientemente actualizada para evitar enlaces con mayor probabilidad de pérdida.

En este capítulo es dónde se explicará todo lo relacionado con las características de las redes de estudio, los protocolos que se usan para dar la información de la calidad de enlace, cómo se calcula esta calidad, etc.

En primer lugar, en la sección 2.2 se hablará más en profundidad del tipo de redes en las que puede tener mucha utilidad este trabajo, las redes de comunidad inalámbricas. Otro de los puntos importantes del estudio, es el hecho de cada nodo pueda conocer información de la métrica de coste del resto de enlaces de la red. Esto se consigue gracias al encaminamiento por estado de enlace, que se trata en la sección 2.3. A continuación, se explica el protocolo OLSR y sus características en la sección 2.4. En la siguiente sección, se habla de OLSRd que permite introducir a OLSR la métrica de calidad de enlace y es la implementación concreta que usa la red de comunidad inalámbrica de estudio, *Funkfeuer Graz*[4]. Es en esta implementación concreta de OLSR, dónde podremos ver que hay que tener cuidado a la hora de obtener los datos de LQ para su predicción y que no se habían tenido en cuenta hasta ahora en los estudios anteriores. Aun así, en este capítulo solo se explica su funcionamiento, las implicaciones que se derivan de éste se abordarán en capítulos posteriores. Por último, se hablarán de las ventajas de la predicción de LQ en estas redes y las conclusiones de este capítulo.

2.2 Redes de Comunidad Inalámbricas

Las nuevas tecnologías de la información y telecomunicación tienen un enorme potencial para mejorar la vida de la gente a la hora de ofrecerlas conexión a un mercado global y mejorar el acceso a servicios de comunicación. Sin embargo, aún hay gente que no tiene acceso a ello, con el riesgo de dejar a esas personas atrás en el mundo globalizado actual [20].

Una red de comunidad inalámbrica es una red inalámbrica en malla con un enfoque ascendente (*bottom-up*). En una WCN, un grupo local de personas crean una infraestructura de red comunitaria, alternativa y autogestionada que se usa normalmente para dos motivos: la interacción entre sus usuarios (mensajería, compartición de recursos, etc.) y llevar el acceso a Internet a sitios donde no está presente. Hoy en día, el mercado ofrece equipos de comunicación de bajo coste que pueden ser usados para establecer enlaces inalámbricos en distancias de hasta decenas de kilómetros. Usando un enfoque multisalto, unas pocas

[4] https://graz.funkfeuer.at

conexiones a Internet pueden ser compartidas en una gran área [2], [21].

Las WCN son un modelo emergente para la futura red de Internet, donde comunidades de ciudadanos construyen, operan y son dueñas de una red abierta basada en IP. Estas redes son normalmente administradas por organizaciones sin fines de lucro y que cooperan con socios o sectores locales interesados en mejorar servicios comunitarios, la red local o proporcionar conexiones de voz o acceso a Internet [5].

Las redes comunitarias se expanden sobre los vecindarios gracias al esfuerzo colaborativo de sus habitantes, al establecer nuevos nodos vinculados a otros nodos cercanos. Se suele utilizar la tecnología inalámbrica IEEE 802.11a/b/n asequible y accesible, utilizando equipos de varios fabricantes, con diversos protocolos de enrutamiento dinámico [5]. Un ejemplo es el de la Figura 1. Los nodos inalámbricos transmiten los datos a un pequeño número de nodos de salida con acceso a Internet en zonas donde disponen de conectividad Internet cableada. Si no se dispone de esa conexión se puede usar conectividad por satélite para transmitir los datos, aunque proporciona una menor velocidad de transmisión y mayor latencia [20].

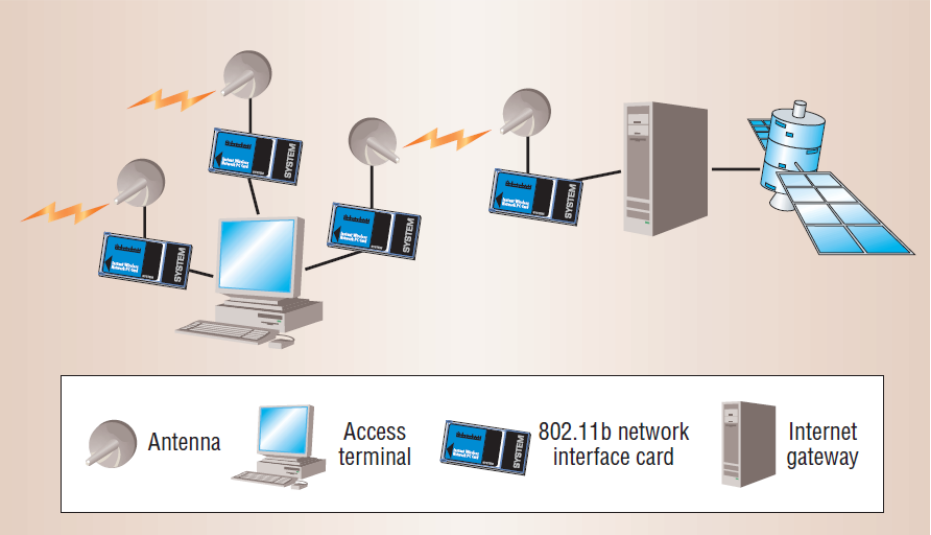


Figura 1. Ejemplo de un nodo inalámbrico con acceso a Internet en una WCN [20]

Para ilustrar la diversidad geográfica y técnica de estas redes, se enumeran una serie de ejemplos. En el sur de Europa, en España se encuentra *Guifi.net*[5] con casi 36.000 nodos activos y más de 38.000 enlaces, haciendo un total de 68.000 km de enlaces totales. Esto la hace la red comunitaria más grande del mundo en términos del número de nodos y área cubierta [22]. En Grecia se encuentra la red *Athens Wireless Metropolitan Network* (AWMN[6]) que cuenta con más de 2.500 nodos [5]. En Europa central, existen redes como *Funkfuer*, compuesta de múltiples redes más pequeñas en ciudades como Graz (de la que se va a hacer el estudio) o Viena y que está conectada con *wlanslovenija*[7] en Eslovenia. En Europa Occidental se encuentra *WirelessBelgie*[8] en Bélgica y *WirelessLeiden* en Holanda. Existen este tipo de redes en todo el mundo, en EEUU hay en San Luis o Portland, también en Sudamérica o en Australia dónde está creciendo en Melbourne la red *Melbourne Wireless*[9].

El funcionamiento de estas redes está basado en el principio de cooperación entre sus

[5] https://guifi.net/es
[6] http://www.awmn.net/
[7] https://wlan-si.net/en/
[8] http://www.wirelessbelgie.be/
[9] https://www.melbournewireless.org.au/

miembros. Estas comunidades suelen tener reglas para su uso, que definen su libertad, apertura y neutralidad. Sin embargo, los diseños e implementaciones actuales de estas redes en malla imponen dificultades y restricciones para lograr la trasmisión correcta de los datos entre cualquier par de nodos de la red. Esto provoca el uso de protocolos específicos de encaminamiento y métrica de coste de tal forma que los nodos de la red puedan aprender e informar constantemente del estado de la red y actualizar sus propias tablas de encaminamiento [23].

Típicamente, el despliegue de estas redes consiste en un sistema autónomo (*Autonomous System*, AS) o varios de ellos, en los cuales operan un protocolo de encaminamiento interno (por ejemplo, OSPF[10] o OLSR) y conectándose con otros AS a través de los protocolos de encaminamiento externo como EGP[11] o BGP[12] [23]. Por las características de estas redes que los protocolos de encaminamiento sean robustos y flexibles es especialmente importante en el contexto de enlaces inalámbricos, donde la métrica debe tener en cuenta la calidad debido a posibles fuertes fluctuaciones. De entre ellos destaca un protocolo de estado de enlace, OLSR.

2.3 Algoritmos de encaminamiento en las WCN

La principal función de la capa de red es la de encaminar paquetes de una máquina origen a una máquina destino. En una comunicación entre nodos, por ejemplo en una de las WCN vistas, será muy común que los paquetes que se envíen tengan que necesitar pasar por múltiples saltos para llegar a su destino. El algoritmo de enrutamiento es la parte de la red responsable de decidir por qué salida deberá ser transmitido un paquete entrante. Hay que tener en cuenta que la decisión debe tomarse cada vez que llegue un paquete porque la mejor ruta para ese paquete puede cambiar con el tiempo [24].

Estos algoritmos se pueden dividir en dos grupos: los no adaptativos y los adaptativos [24]. Los primeros no basan su información de encaminamiento en ninguna medida o estimación de la topología o tráfico actual. Simplemente, al iniciarse el sistema se descargan los destinos de cada nodo. Se le llama también encaminamiento estático y como no responde a fallos, se suele usar en redes en los que la selección de rutas sea clara. Los adaptativos, por el contrario, cambian sus decisiones de encaminamiento para reflejar cambios en la topología, y en ocasiones en el tráfico [24].

Por lo tanto, el propósito de un algoritmo de enrutamiento es simple: dado un conjunto de nodos y sus conexiones, es el de encontrar el mejor camino entre un nodo origen y un nodo destino. Normalmente, el mejor camino es aquel que minimiza el coste [25]. Para entender mejor esto se puede usar un grafo, representado en la Figura 2.

[10] https://tools.ietf.org/html/rfc2178
[11] https://tools.ietf.org/html/rfc827
[12] https://tools.ietf.org/html/rfc1105

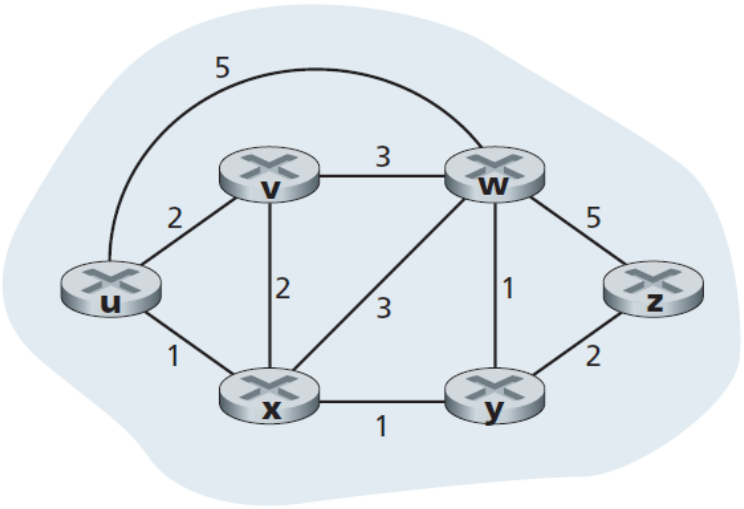


Figura 2. Grafo de un modelo de red de ejemplo [25]

En esta figura, los nodos pueden representar los routers/nodos de una red y las líneas/aristas que los unen son las conexiones físicas entre ellos, pudiendo ser cables o inalámbricas (caso de las WCN). En este grafo, se indica además el valor del coste de cada enlace. Este coste puede reflejar la longitud del enlace físico (ej. enlace transatlántico mayor coste que uno corto), velocidad binaria del enlace o el precio monetario de usar ese enlace. El coste de un camino entre un origen y destino es claro, es la suma de todos los costes de los enlaces por los que se pasa para llegar al destino. En este ejemplo, el coste de los enlaces no es direccional, es el mismo independientemente del sentido, veremos que en nuestro caso con OLSRd no será así y dependerá también del sentido (y el coste será la LQ). Los algoritmos de encaminamiento buscan el camino con menor coste de todos los posibles. Es de notar, que si todos los costes son 1, el de menor coste será el camino más corto (es decir, aquel camino con el menor número de enlaces entre origen y destino) [25].

Otra forma de clasificar estos algoritmos es de acuerdo si son globales o descentralizados [25]:

- **Globales**: calculan el camino de menor coste entre dos destinos usando información y conocimiento completo y global de la red. El algoritmo tiene como entrada todos los nodos y todos los costes de los enlaces. Los algoritmos con información global se suelen llamar algoritmos de **estado de enlace** (*link state*, LS), dado que el algoritmo debe saber el estado de cada enlace en la red.
- **Descentralizados**: el cálculo del camino con menor coste se hace de una forma iterativa y distribuida. Ningún nodo tiene la información completa de la red. En cambio, cada nodo empieza con solo el conocimiento del coste de los nodos a los que se conecta de manera directa. Después, a través de un proceso iterativo de cálculos e intercambio de información con sus vecinos, un nodo aprende gradualmente la ruta de menor coste hacia cada destino. Un algoritmo descentralizado es el llamado **vector de distancias** (*distance vector*, DV), porque cada nodo mantiene un vector de estimaciones de costes (distancias) de todos los nodos de la red.

La convergencia se define como el establecimiento de las mejores rutas en los nodos de la red. Los algoritmos de vector de distancias tienen un grave inconveniente en la práctica: aunque converjan en el estado correcto, lo hacen de forma muy lenta (surgiendo problemas como el del fenómeno de la cuenta hasta el infinito) [24]. En las WCN se prefieren los

algoritmos de estado de enlace en lugar de los de vector de distancias. Estas redes pueden llegar a estar muy densamente conectadas (muchos nodos y muchos enlaces), por lo que el vector de distancias no escala correctamente, interesa usar otras métricas distintas del número de saltos (los algoritmos de vector distancia estándar usan el número de saltos); y, sobre todo, la calidad de los enlaces es muy variable y se necesitan algoritmos que converjan rápido. Los algoritmos de estado de enlace aparte de poseer menor tiempo de convergencia, son más robustos que los de vector de distancias (un fallo o sabotaje en el coste de un nodo en DV puede comprometer a cualquier ruta, mientras que con LS hay mayor independencia entre nodos) [25].

Los protocolos de encaminamiento que utilizan la información global de la red para realizar la tabla son los llamados **protocolos de estado de enlace**. En la actualidad muchos de estos protocolos son usados, por ejemplo, IS-IS (*Intermediate System to Intermediate System*) [26]. Este protocolo fue diseñado para una red pionera llamada *DECnet*, después fue aceptado por el ISO para su uso en los protocolos OSI y modificada para soportar otros protocolos como IP. OSPF (*Open Shortest Path First*) [27] es otro de los principales protocolos de estado de enlace, fue diseñado por la IETF varios años después de IS-IS y adoptó muchas innovaciones diseñadas para IS-IS. Innovaciones como la de la creación de un router asignado en una LAN o el de usar distintas métricas/costes [24].

La idea detrás de un protocolo de estado de enlace es simple y se puede resumir en cinco pasos que debe seguir cada encaminador para hacerlo funcionar [24]:

1. **Descubrir sus vecinos y aprender su dirección de red.** Cuando se enciende un encaminador su primera tarea es la aprender quiénes son sus vecinos. Se suele hacer enviando paquetes especiales, paquetes HELLO. Los que lo reciben envían otro de vuelta con su dirección.
2. **Establecer la distancia o coste de cada uno de sus vecinos.** Este coste se puede establecer de forma automática o puede ser configurarlo el administrador de la red. Por ejemplo, puede ser inversamente proporcional a la velocidad binaria del enlace o distintos costes que se han explicado con anterioridad.
3. **Construir un paquete que contenga la información que ha aprendido.** Una vez que tiene la información, su siguiente paso es el de construir un paquete con todos los datos. Este paquete debe contener la identidad del que lo genera, la edad (número que se decrementa por salto y que al llegar a 1, el siguiente nodo lo descarta), número de secuencia y la lista de vecinos con su coste. Un ejemplo es de la Figura 3.

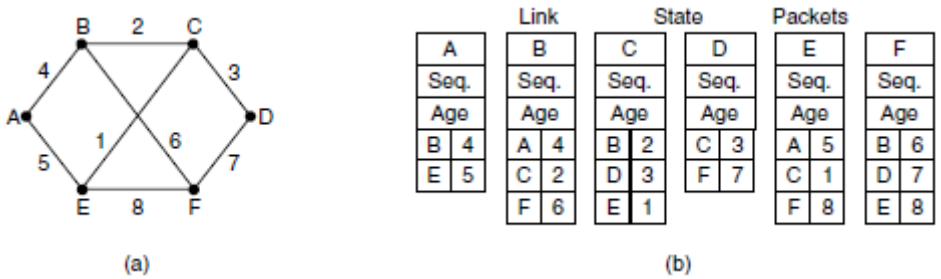


Figura 3. (a) Grafo de una red. (b) Paquetes de estado de enlace para esta red [24]

4. **Enviar ese paquete y recibir esos paquetes de todos los encaminadores.** Es la parte más complicada, todos los encaminadores deben conseguir la información de forma rápida y fiable. La idea fundamental es la de mandar los

mensajes por difusión para que llegue a todos los nodos. Por eso existe el número de secuencia para descartar paquetes viejos o duplicados y el de edad para que ningún paquete se pierda y viva indefinidamente.

5. **Calcular el camino de menor coste para cada encaminador.**

Como consecuencia, todos los encaminadores disponen de la topología de red completa. Y se puede aplicar un algoritmo como *Dijkstra* que permita a cada encaminador calcular la mejor ruta para cada destino. Otro de los protocolos de estado de enlace que se usa mucho en WCN, como ya se ha comentado, es el de OLSR.

2.4 OLSR (Optimized Link State Routing)

El protocolo *Optimized Link State Routing* (OLSR) es un protocolo desarrollado para redes móviles *ad-hoc* (*mobile ad-hoc networks*, MANETs). Es un protocolo de encaminamiento de estado de enlace proactivo que se ha diseñado para grandes redes móviles. Se denomina proactivo debido a que se intercambia información de topología con otros nodos de la red periódicamente. Su principal característica frente a alternativas como OSPF o IS-IS, es la de reducir el tráfico de control (que se propaga por *flooding*, inundación de paquetes), de ahí el *Optimized,* ya que es una optimización de protocolos de estado de enlace. Esta reducción del tráfico se consigue gracias a la selección en cada nodo de la red de un conjunto de nodos vecinos, llamados "*multipoint relays*" (MPR), que actúan como retransmisores de los mensajes de control [28].

OLSR está diseñado para redes cuyo tráfico sea aleatorio y esporádico entre un gran conjunto de nodos. Al ser proactivo, también se usa en escenarios donde la comunicación entre pares origen/destino cambia a lo largo del tiempo, en los que se ha demostrado que el uso de MPR es beneficioso. Cuánto más grande y densa sea la red, la optimización usando MPR es mayor comparando con algoritmos de estado de enlace clásicos. OLSR está diseñado para trabajar de manera distribuida y no necesita de una entidad central. Tampoco requiere de transmisiones fiables, puesto que los mensajes de control se envían periódicamente y se pueden soportar pérdidas razonables [28]. Este protocolo se convirtió en estándar en el RFC 3626, teniendo en cuenta que esta versión optimiza las rutas según el camino más corto, es decir, su coste o métrica es el número de saltos. No se utiliza la calidad de enlace de ninguna forma. En la revisión del estándar OLSRv2 [29], entre las mejoras se encuentra la posibilidad de elegir otra métrica pero no se especifica cómo implementarla.

Como se ha visto en la sección anterior, el estado de enlace se basa en que cada nodo sabe la información completa de la red. Para ello cada nodo debe mandar mensajes de control/topología al resto de nodos sobre la información que conoce. Es aquí donde entra la clave de OLSR el uso de los MPR, que pretenden reducir el tráfico debido a la propagación de información de control de la red, reduciendo mensajes redundantes en una misma región. Cada nodo selecciona un conjunto de nodos con los que tenga enlace simétrico y sean vecinos (nodos a un salto de distancia) que retransmitirán sus mensajes. Este conjunto de nodos seleccionado se llama conjunto MPR de un nodo. Los nodos vecinos de un nodo *N* que no estén en ese conjunto de nodos MPR, recibirán y procesarán los mensajes *broadcast* del nodo *N*, pero no los retrasmitirán [28].

Cada nodo selecciona sus MPR entre los vecinos, eligiéndolos de tal forma que cubra (en número de saltos) todos los nodos que se encuentren a dos saltos de distancia. Es decir, con la selección de MPR se debe asegurar que los mensajes de *broadcast* del nodo origen lleguen a todos los vecinos que se encuentran a dos saltos. Cuantos menos MPR necesite un

nodo para cubrir todos los nodos de dos saltos, más eficiente será el tráfico de control. En la Figura 4 se observa este efecto: a la izquierda los mensajes de *broadcast* generados por el nodo central hacen que lleguen a distintos nodos mensajes duplicados e innecesarios; a la derecha el nodo central elige los MPR (nodos negros) de tal forma que sus mensajes de *broadcast* lleguen a todos los nodos de al menos dos saltos de distancia. Como se ve, los que no son seleccionados como MPR, no retransmiten el mensaje del nodo central.

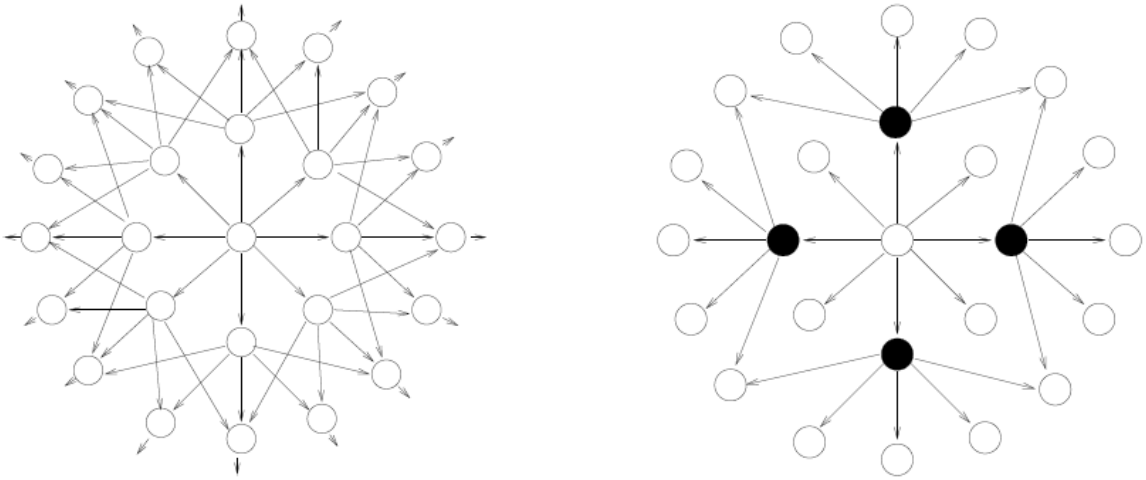


Figura 4. A la izquierda, propagación de mensajes broadcast en una red normal. A la derecha, el nodo central selecciona a los nodos negros como sus MPR en una red con OSLR [30]

Cada nodo debe saber si otro nodo lo ha elegido como MPR para saber si retransmitir sus mensajes de *broadcast* o no. Por lo tanto, cada nodo mantiene información sobre el conjunto de nodos que lo ha seleccionado como MPR. A este conjunto se le llama conjunto *MPR selector*. Un nodo obtiene esta información (de si le han elegido como MPR o para decir quiénes son sus MPR) a través de mensajes HELLO periódicos, que se envían entre los nodos vecinos. Ambos conjuntos pueden cambiar con el tiempo, por ejemplo, si se añade o se desconecta un nodo.

Los mensajes de OLSR pueden ser de tres tipos [28]:

- **HELLO**: se encargan de descubrir los vecinos y declarar los MPR. Su máxima distancia es de un salto por lo que un nodo solo ve los de sus vecinos.
- **TC** (*Topology Control*): son los mensajes de topología, los que se mandan por difusión y deben llegar a todos los nodos para saber la topología completa de la red. Son los que deben retransmitir los MPR.
- **MID**: su función es la de informar de múltiples interfaces y alguna otra información que no es relevante para la obtención futura de calidad de enlace.

Haciendo analogía con la sección anterior, los mensajes HELLO servirían para el paso 1 de los protocolos de estado enlace, el de descubrir lo vecinos y su dirección. Los mensajes que se construyen en el paso 3 son los de TC y el paso 4, que era de la difusión de estos mensajes en OLSR, se hace con los MPR. Por último, el paso 5 es el de calcular la ruta de mejor coste, y se puede hacer uso de *Dijkstra*, por ejemplo. Falta el paso 2, que consiste en calcular el coste con los vecinos. Este paso no es incluido en el estándar de OLSR (porque los costes de los enlaces son los saltos, que no necesitan cálculos al ser 1 para todos ellos), pero si la métrica es distinta del número de saltos, debe llevarse a cabo de alguna manera. Para el caso de escoger LQ como métrica, la próxima sección explicará cómo se completa este paso.

En la Figura 5 se muestra la estructura de un paquete OLSR genérico. La cabecera del paquete está compuesta por dos campos: *Packet Length* que indica la longitud en bytes del paquete entero (incluyendo la cabecera) y *Packet Sequence Number* que es un número incrementado cada vez que se genera un mensaje OLSR.

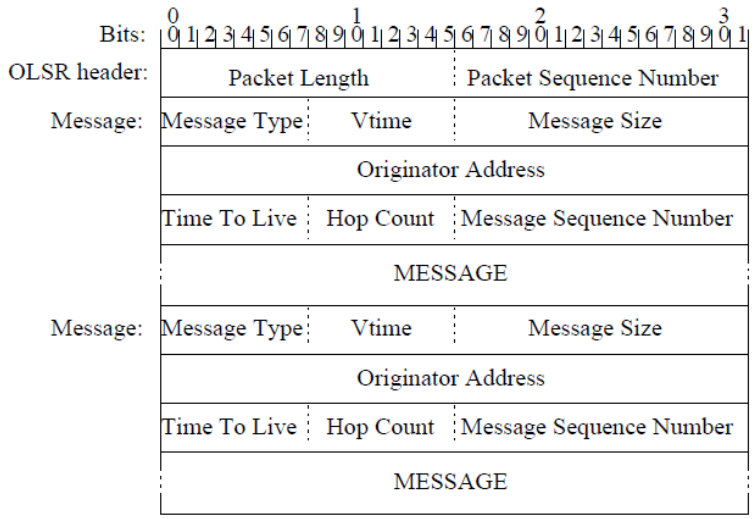


Figura 5. Paquete OLSR genérico [30]

El paquete OLSR puede contener uno o más mensajes OLSR y la cabecera de cada mensaje está formada por [30]:

- *Message type*: número entero que identifica el tipo de mensaje. OLSR usa los tres tipos mencionados pero se pueden ampliar para aplicaciones personalizadas.
- *Vtime*: indica el tiempo en el que es válido ese mensaje.
- *Message Size*: tamaño del mensaje en bytes incluyendo sus cabeceras.
- *Originator Address*: la dirección del nodo que creó el mensaje OLSR original.
- *Time To Live*: máximo número de saltos que el mensaje puede ser retrasmitido, En cada salto se decrementa y al llegar a 0 no se retransmite. Tendrá mucha importancia en el posterior estudio debido al algoritmo *fish-eye*. Pero en el estándar OLSR, los mensajes HELLO tendrán valor 1 y los de TC el máximo 255.
- *Hop Count*: número de veces que un mensaje ha sido retrasmitido.
- *Message Sequence Number*: número incremental que genera el nodo origen cada vez que manda un mensaje OLSR.

El formato de los mensajes de TC y de HELLO se muestran en la Figura 6. Básicamente los mensajes de HELLO contienen su lista de vecinos indicando en *Link Code* si son MPR o no, y el tipo de enlace (si es bidireccional o de otro tipo). Los mensajes TC indican la información de topología del propio nodo, es decir, declaran quiénes son todos sus vecinos de un salto de distancia (incluye a vecinos que no son MPR). De esta forma, al propagar los mensajes TC a toda la red, los demás nodos, sabiendo quién ha generado el mensaje de TC y sus vecinos, pueden construir la topología completa de la red y calcular el camino con menor número de saltos.

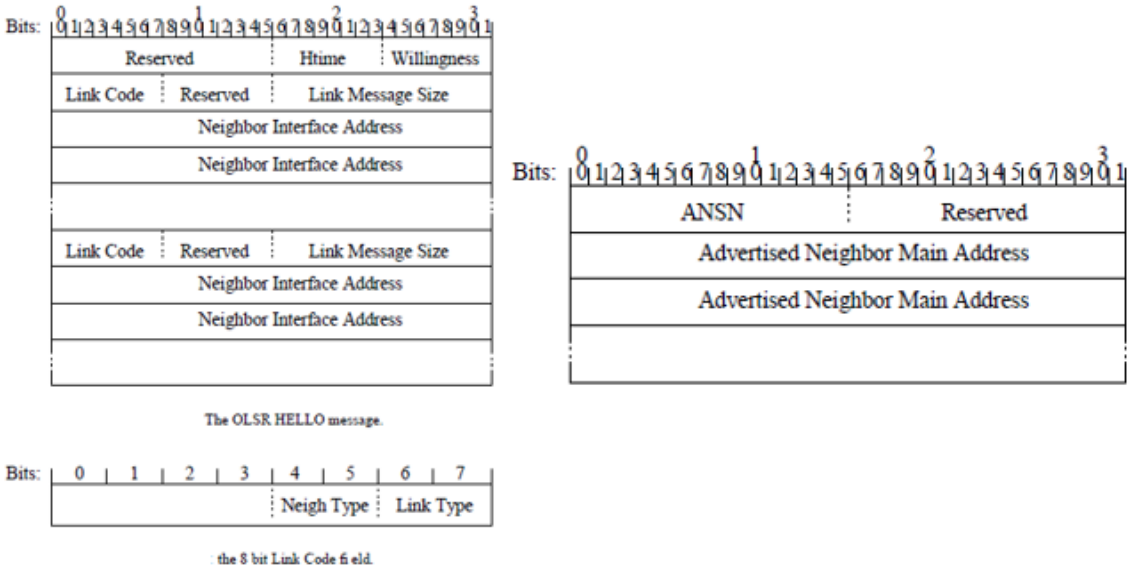


Figura 6. A la izquierda, formato de un mensaje HELLO. A la derecha, formato de un mensaje TC [30]

Otro aspecto importante de OLSR para entender cómo obtener los valores de calidad de enlace es saber cada cuánto se generan estos mensajes y cómo. Según el estándar, los mensajes HELLO y TC se generan periódicamente según los valores los temporizadores HELLO_INTERVAL y TC_INTERVAL, con valores razonables (según el propio estándar) de 2 y 5 segundos, respectivamente. Otra propiedad importante es que los nodos no tienen ningún tipo de sincronismo a la hora de transmitir sus mensajes y, de hecho, éste se debe evitar, ya que puede hacer que se produzcan colisiones y pérdidas continuas en mensajes del mismo tipo OLSR.

Es por esto por lo que, al tiempo entre mensajes OLSR (HELLO_INTERVAL o TC_INTERVAL), se le resta un tiempo *jitter*[13], siendo *jitter* un número aleatorio entre 0 y MAXJITTER. Además, este tiempo *jitter* se debe introducir a la hora de reenviar mensajes TC, de tal forma que, cuando a un nodo le llega un mensaje TC, debe esperar ese tiempo hasta transmitirlo. Durante este tiempo se hace *piggybacking*, es decir, los mensajes OLSR que el nodo tenga que transmitir (HELLO, TC propios o ajenos, etc.) se trasmiten en el mismo paquete OLSR para reducir el tráfico de la red. El estándar indica que el valor de MAXJITTER es del orden de 4 veces menor que el HELLO_INTERVAL.

2.5 OLSRd (Optimized Link State Routing Daemon)

El *Optimized Link State Routing Daemon* (OLSRd[14]) es una implementación del protocolo OLSR, documentado en el RFC 3626 ya explicado. OLSRd empezó como parte de la tesis de máster de Andreas Tønnesen en 2004 [30]. El desarrollo de esta implementación continuó como un proyecto de código abierto y se usó como protocolo de encaminamiento en redes como *Freifunk*[15], *Funkfeuer* (de la que vamos a estudiar la calidad de sus enlaces) y otras WCN. Existen actualmente dos versiones, OLSRd y OLSRd2. En este trabajo

[13] No debe confundirse esta variable, que es un número aleatorio sorteado en cada nodo, con el concepto clásico de *jitter* o fluctuación en el retardo. Obviamente este tiempo aleatorio afectará al retardo, lo cual probablemente explique el nombre elegido por los proponentes de OLSR, pero esta variable no explica toda la fluctuación en el retardo.
[14] http://www.olsr.org
[15] https://freifunk.net/

estudiaremos las características de la primera versión, que es la que usa la red de estudio.

OLSRd está diseñado para ser una implementación extensible y modular. Dispone de una interfaz para *plugins*, permitiendo a más desarrolladores extender sus funcionalidades sin interferir en el código central de OLSR. Esta aplicación también destaca por incluir la métrica de calidad de enlace. Además, si se pone en funcionamiento la aplicación sin usar la calidad de enlace es compatible con las demás versiones que sigan el RFC 3626 [31]. Algunos de los *plugins* disponibles son el *httpInfo* o el *txtInfo*, que son los que se han usado para conseguir la calidad de enlace en cada instante.

De entre todas las funcionalidades, se explicarán las dos más importantes para el trabajo. Una sirve para calcular la LQ de cada enlace y la otra tiene importancia a la hora de obtener los datos para su predicción. Estas dos funcionalidades son la implementación de la métrica ETX y del uso del algoritmo de *fish-eye*, respectivamente.

Empecemos por la métrica de coste. Desde la versión 0.4.8 de OLSRd se incluye la posibilidad de usar la métrica de coste ETX (*Expected Transmission Count*). Su explicación es muy sencilla: cada enlace direccional tiene un coste llamado calidad de enlace. Para calcular esa calidad de enlace hay distintos métodos. OLSRd usa la métrica ETX [32], que se pasa a explicar a continuación.

Primero, hay que hacer que OLSR pueda decir cómo de bueno o malo es un enlace. Un criterio razonable es medirlo según la tasa de pérdidas de paquetes OLSR que recibimos de nuestro vecino. Para ello se aprovecha que los mensajes HELLO de nuestros vecinos se envían periódicamente para determinar la tasa de pérdidas de paquetes con los vecinos. Por ejemplo, si nos llegan 4 mensajes HELLO de nuestro vecino cuando en ese tiempo nos deberían haber llegado 10, significará que se han perdido 6 paquetes y se asigna una tasa de pérdida de pérdida de 6/10=0.6 o del 60%. Es decir la probabilidad de mandar un paquete correctamente es del 0.4. Esta última probabilidad es a la que llamamos calidad de enlace (Link Quality, **LQ**). Hay que notar que la LQ nos da información de la calidad de enlace entre el vecino y nosotros, pero en la dirección del vecino a nosotros mismos. Nos dice cómo de probable un paquete en esa dirección no se perderá [33].

Sin embargo, es igual de importante calcular la calidad de enlace en el sentido opuesto, es decir de nosotros al vecino. Este valor es la probabilidad de enviar correctamente mensajes OLSR a nuestro vecino. Este valor se conoce como **NLQ** (*Neighbor Link Quality*), que nos dice como de bueno es el enlace en sentido de nosotros mismos a nuestro vecino. Desde la vista del vecino será su LQ, después se explicará cómo se obtiene. En la Figura 7, se representa el sentido de LQ y NLQ del nodo *A*. En este caso *B*, tendrá que decir a *A* su LQ, que para *A* será la NLQ.

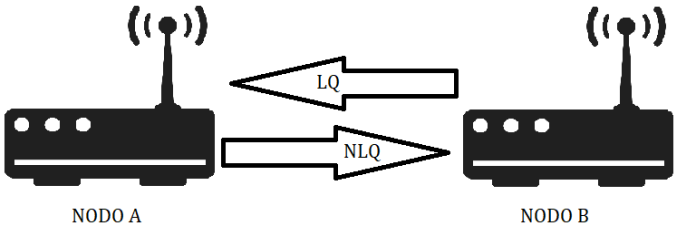


Figura 7. Sentido de LQ y NLQ del enlace desde el punto de vista del nodo A

Estos valores de LQ y NLQ están comprendidos entre 0 y 1. A partir de estos valores se puede calcular la probabilidad de que un paquete se envíe correctamente a nuestro vecino y que éste nos responda. Esta probabilidad sería la probabilidad que le enviemos un paquete

correctamente (NLQ) por la de que el vecino nos responda correctamente (LQ), es decir, NLQ x LQ. La utilidad de combinar ambos valores viene dada por el contexto de redes inalámbricas, ya que cada recepción de un paquete se confirma enviando otro paquete de confirmación. Esta metodología provoca que el emisor tenga que retransmitir un paquete si no le llega la confirmación del receptor. Y la confirmación del receptor puede no llegar porque el mensaje del transmisor o la propia confirmación se ha perdido, siendo la probabilidad de que no ocurra esto la calculada anteriormente.

Con este valor se puede calcular el número de intentos de transmisión que serán necesarios para enviar un paquete a nuestro vecino o nuestro vecino a nosotros, siendo $\frac{1}{LQ \cdot NLQ}$ los intentos. Este valor se conoce como ETX (*Expected Transmission Count*), y será la métrica utilizada por OLSRd para encontrar el camino de menor coste para cada par origen/destino. Será mayor el coste cuanto peor sea la calidad de un enlace y teniendo valor 1 como mínimo [33]. Es decir, si la calidad que ve un nodo tiene un LQ=NLQ=1 (no hay pérdidas en los dos sentidos del enlace), el ETX será de 1, puesto que en teoría solo se necesitará un intento para enviar cada paquete. En cambio, para un enlace malo de LQ=0.3 y NLQ=0.1, el número de intentos de transmisión de cada paquete para que llegue correctamente sería de 33,33. Y OLSR de esta forma, intentará evitar ese enlace debido a su alto coste, cuando con la implementación del RFC en función de los saltos esto no se tendría en cuenta.

Por lo explicado anteriormente, OLSRd cambia el protocolo de OLSR de la sección anterior. El primer cambio es el de los mensajes HELLO, con la creación de los mensajes **LQ HELLO**. Se ha visto que un nodo puede obtener, viendo los mensajes de su vecino, el valor de LQ, pero no sabe cómo de bien le llegan sus transmisiones a su vecino, el valor NLQ. Es por esto por lo que el nodo que genera el LQ HELLO envía su LQ con todos los vecinos. De esa forma un nodo sabe su LQ (la calcula) y la NLQ (se la envían) con cada vecino. En OLSR normal, los HELLO solo contienen la dirección de los vecinos. En OLSRd se incluyen los valores LQ. Por la misma razón que antes, se crean los **LQ TC**, donde antes con los TC normales se enviaban a toda la red los vecinos que se tenía, ahora se le añade el valor LQ y NLQ. La razón es porque para hacer el mapa de topología completo de la red, cada nodo debe saber el coste de cada enlace, necesitando los valores LQ y NLQ para calcularlo.

El otro aspecto importante de OLSRd es el ojo de pez (***fish-eye***), que tendrá implicaciones a la hora de capturar los paquetes TC y predecir los datos de LQ a partir de las capturas. Normalmente OLSR "inunda" toda la red de mensajes TC, pudiendo provocar mucho tráfico en los encaminadores. Para reducir ese tráfico se puede optar por bajar el temporizador de los mensajes TC, a costa de que la red responda peor a cambios en la topología y costes de los enlaces en la tabla de encaminamiento. Una solución de compromiso entre estos dos problemas es el mecanismo de *fish-eye*. Este algoritmo se introdujo en la versión 0.4.10 de OLSRd, y su idea es la de mandar mensajes TC más frecuentemente a nodos cercanos y no enviar cada mensaje TC a toda la red.

Anteriormente se explicó que en OLSR cada mensaje lleva consigo un *Time To Live* (TTL) que especifica el máximo número de saltos que puede viajar por la red. En la versión del RFC, se indicaba que era de 1 para los HELLO y de 255 para los TC, es decir, llegarían a toda la red. El mecanismo de *fish-eye* plantea que se generan los mensajes TC no solo de 255, sino de diferentes TTL, por ejemplo 1, 2, 3, 4, etc. De manera que los mensajes TC que tengan un TTL de 3, solo llegará a los nodos de la red que se encuentren a 3 saltos o menos. Para ello, los mensajes TC se generan con una secuencia de TTL, por ejemplo [255 3 2 1 2 1 1 3 2 1 2 1 1]. En esta secuencia de 13 TTL, los 13 mensajes generados los ven los vecinos de un salto, los de dos saltos solo ven 7 de los 13 y los que están a cuatro o más saltos solo ven 1 de

cada 13 [34].

La razón de la introducción de este mecanismo tiene que ver con el hecho de que en una comunicación inalámbrica tener un enlace saturado puede producir interferencias continuas en el medio, provocando la pérdida de paquetes de otros enlaces cercanos. Pero, reducir intervalos de tiempos entre TC puede ser crítico para el encaminamiento, al perder rapidez en la información de cambios. *Fish-eye* parte de la idea de que a los nodos simplemente les basta tener una decisión razonable para los siguientes dos o tres saltos. Si la ruta está a 8 saltos, por ejemplo, los nodos que están a 5 saltos del nodo que envió el paquete, están más cerca del destino y tienen mejor información sobre el mejor camino. Estos son los principios básicos del encaminamiento jerárquico. Es suficiente con que los nodos tengan una idea vaga de la topología lejana, haciendo que solo unos pocos mensajes TC lleguen a toda la red [34].

2.6 Predicción de calidad de enlace para OLSRd

Este trabajo trata de la predicción del valor de calidad de un enlace, ya que esta secuencia de valores se puede ver como una serie temporal. Este valor sirve para poder tener información de la red y que un nodo pueda encaminar los paquetes por el mejor enlace posible. La forma en que un nodo decide esto, depende de los algoritmos de encaminamiento que usa dicho nodo. En nuestro caso, es necesario que cada nodo conozca todos los demás nodos y además saber la calidad de enlace de cualquier nodo de la red con sus respectivos vecinos (nodos a un salto de distancia). De esta forma, cada nodo puede estimar valores de LQ de todos los enlaces de la red para la actualización de la tabla de encaminamiento. Con la tabla de encaminamiento de cada nodo, con información de los nodos de la red y su métrica de coste (saltos, calidad de enlace, etc.) se realiza la tabla de reenvío que es la que usa un nodo para decidir a qué nodo vecino transmitir un paquete dependiendo de su destino final. Los protocolos de encaminamiento a los que tiene sentido aplicar este trabajo son los que usan los **algoritmos de estado de enlace** y **la calidad de enlace**, que es el caso de OLSRd.

El nodo que ejecuta OLSRd recibe sucesivas actualizaciones del valor de LQ para cada enlace de la red a través de los mensajes LQ TC del resto de nodos. Esto hace que, para cada enlace se disponga de una serie temporal de valores de LQ que se pueden estudiar y aplicar técnicas de predicción. Este y trabajos anteriores de predicción de calidad de enlace predicen el valor LQ, tal y como se ha definido. No es necesario predecir el valor NLQ, porque éste se obtiene del valor LQ de otro enlace, al considerar los pares origen/destino enlaces distintos. Por ejemplo, el enlace de *A*-*B* tendrá sus valores LQ, mientras que el enlace *B*-*A* tendrá también otros valores LQ que serán los NLQ del enlace *A*-*B*. De esta manera, prediciendo los valores de LQ de cada enlace se puede obtener todos los costes ETX de la red, que serán con los cuáles se decidan las mejores rutas.

El seguimiento de la calidad de enlace es lo que usa la aplicación OLSRd para maximizar la tasa de entrega y minimizar la congestión debida al aumento de tráfico por las retransmisiones. La predicción de calidad de enlace se suma al seguimiento de LQ para determinar de antemano que enlaces son más susceptibles de cambiar su comportamiento. Como resultado, la capa de encaminamiento puede hacer mejores decisiones en el momento apropiado [35]. La predicción de LQ, por tanto, incrementa las mejoras en el enrutamiento logradas con el seguimiento de LQ. Normalmente, las métricas calculadas en tiempo real no proporcionan suficiente información para detectar los cambios de un enlace en el momento correcto. Consecuentemente, técnicas de predicción podrían ser necesarias para prever cambios futuros de LQ y tomar acciones apropiadas [2].

Aparte de usar valores futuros de LQ para mejorar el encaminamiento y reducir la probabilidad de pérdida de paquetes, existen otras posibles aplicaciones. En el caso del protocolo OLSR se envían menos mensajes de difusión para reducir el tráfico, con la correspondiente pérdida de redundancia. Este efecto provoca que la pérdida de los mensajes de TC pueda significar no actualizar la LQ (aumentando la probabilidad de pérdida en aquellos enlaces con mala LQ, que serán los de mayor importancia de predicción). Con la predicción de LQ a varios instantes vistas se puede paliar este efecto, pudiendo aparte de predecir a futuro, estimar el valor LQ que te hubiese llegado. Además, prediciendo a varios instantes vista consigues un mayor horizonte temporal que prediciendo el valor LQ siguiente, pudiendo usarse ese conocimiento futuro para evitar antes los posibles malos enlaces. Otra aplicación es la predicción a corto plazo, que provoca la posible estimación del valor LQ verdadero, ya que el valor de calidad de enlace que llega a un nodo no es exactamente un valor actual (ha sufrido retrasos como el tiempo de *jitter* y retransmisiones).

2.7 Conclusiones

En este capítulo se ha mostrado uno de los problemas de las redes comunitarias inalámbricas que es el del encaminamiento. Las WCN son redes que cambian de forma dinámica y sus enlaces se caracterizan por ser escasamente fiables, haciendo que la selección de rutas para el encaminamiento del tráfico sea importante. Como consecuencia de esto, puede ser necesario el uso de protocolos de encaminamiento que usen la calidad de enlace como métrica de coste. Además, con el uso de protocolos de estado de enlace se permite que desde un nodo puedas obtener la información de la topología de la red. Con esos datos, la estimación de LQ es una propuesta que puede incrementar las mejoras en el encaminamiento.

Uno de los protocolos más usados en WCN es OLSR, protocolo de estado de enlace que permite la difusión de topología de red de manera más eficiente a través de la selección de vecinos MPR, para la retransmisión de los mensajes TC. Para incorporar la métrica de LQ, se hace uso de la aplicación OLSRd. Esta aplicación se usa en la red de la cual disponemos de los datos y a través de la modificación de los mensajes HELLO y TC, se incluyen un valor LQ y NLQ para cada enlace entre dos nodos. Algunos efectos del funcionamiento de OLSR y OLSRd, tales como el tiempo aleatorio *jitter* entre mensajes y el mecanismo *fish-eye*, pueden hacer que el tiempo entre llegadas de mensajes TC varíe. Esta variación afecta a las actualizaciones de valores LQ y tiene que ser tomada en cuenta para la obtención de los datos. El estudio de esto último, se realizará en el siguiente capítulo.

Capítulo 3. Conjuntos de datos

3.1 Introducción

En el capítulo anterior se han repasado todos los aspectos importantes de las redes de comunidad inalámbrica y explicado a grandes rasgos el funcionamiento de sus protocolos de encaminamiento. Ahora, se abordará todo lo relacionado con la obtención de los valores de calidad de enlace de la red de estudio. En este capítulo se presentan los conjuntos de datos de los que se dispone y con los que se ha trabajado. Todos los conjuntos de datos se han obtenido de la red de *Funkfeuer*.

Funkfeuer Graz[16], es una red de comunidad inalámbrica experimental y gratuita situada en la ciudad de Graz, Austria. Como se indica en su página web, *Funkfeuer* es una red abierta para todo aquel que esté interesado y dispuesto a contribuir. Es una red no regulada y que intenta eliminar la brecha digital entre clases sociales, proveyendo infraestructura y conocimiento para ello. Este proyecto no comercial, está construido y mantenido por entusiastas de ordenadores con diferentes motivaciones y objetivos. Esta red implanta la idea del DIY (hazlo tú mismo, *do-it-yourself*), cada participante decide cuanto esfuerzo aportar con la ayuda y el *know-how* de otros usuarios. Para unirse a la red solo se necesita un encaminador WLAN o un PC con el programa OLSRd, paciencia, motivación y tener cobertura hacia otro nodo de la red. Haciendo que cada miembro nuevo extienda el área de la red [36]. Entre los principales servicios que se ofrecen actualmente en *Funkfeuer Graz* se encuentran: el acceso a Internet, CAcert[17] (autoridad certificadora que emite certificados gratuitos), VoIP y Email (con el servidor SMTP `smtp.graz.funkfeuer.at`).

En la actualidad existen más de 1000 nodos y el número sigue creciendo [36]. En la Figura 8 se puede ver la localización de los nodos que existen actualmente. Como se ve en la imagen la mayoría se encuentran en Austria, lógicamente cerca de Graz, pero hay un nodo (el #1267) con la localización en la ETSIT de Valladolid llamado *research*. Este nodo es debido a que en el momento de la captura de esta figura se había realizado una conexión con la red para capturar tráfico, el motivo de esta captura se explicará en una sección posterior.

Figura 8. Localización de los nodos de Funkfeuer Graz

[16] https://graz.funkfeuer.at/
[17] http://www.cacert.org/

Los conjuntos de datos de los que se va a hablar en este capítulo constan de un registro temporal de valores de LQ y NLQ que han ido tomando los distintos enlaces de la red vistos por el nodo que captura esos datos en un momento dado. En la realización de este trabajo se ha trabajado con 3 conjuntos: dos que se realizaron para otros trabajos de predicción de enlace y otro que se ha capturado en este. De los realizados anteriormente, uno está formado de datos de LQ con muestras cada 5 minutos y el otro con muestras cada segundo. El conjunto del que se realiza la captura en este trabajo está formado de tráfico OLSR del que se puede obtener el instante de llegada exacto de LQ y conocer su valor. Todos estos conjuntos de datos pueden ser usados para hacer una predicción de calidad de enlace con distintos horizontes temporales.

Para la explicación de estos conjuntos y de las consideraciones que se van a tener a la hora de tomar los datos se dividirá el capítulo en las siguientes secciones. En la sección 3.2 se explicará la configuración del protocolo de OLSR de la red de comunidad inalámbrica de las que se han obtenido todos los datos, *Funkfeuer Graz*. Ahí se verá un ejemplo de su tabla de topología y la configuración de OLSRd exacta que usan sus nodos. En la siguiente sección se mencionará brevemente el conjunto de datos muestreados cada 5 minutos. Esos datos simplemente se usaron en este trabajo como una iniciación a predicción con LSTM y no se reportarán resultados. En la sección 3.4, se explican los datos usados para evaluar las predicciones en este trabajo, teniendo una alta tasa de muestreo de 1 segundo y se explicarán las limitaciones encontradas e implicaciones que se deben tomar para obtener la secuencia de LQ correctamente. En la sección 3.5, se realiza una nueva captura de datos de la red *Funkfeuer*, pero esta vez de tráfico OLSR para trabajos futuros y para ver si las implicaciones de la sección 3.4 se cumplen y entender mejor la actualización de LQ a través de mensajes TC. A continuación, en la sección 3.6 se selecciona un conjunto de enlaces con los que realizar las pruebas de predicción, teniendo en cuenta las conclusiones de las secciones anteriores. Por último, se termina con una sección de conclusiones.

3.2 Configuración del protocolo OLSR en *Funkfeuer Graz*

En esta sección se presenta la configuración del protocolo OLSR de *Funkfeuer Graz*. En esta sección no se entra a valorar cómo afecta esto a los conjuntos de datos; eso se hará más adelante. Antes de seguir con datos más específicos de la WCN de estudio y de cómo se ha configurado OLSRd, se aclararán las preguntas de qué datos concretos se buscan y cómo se obtienen. Los datos que se quieren obtener para estudiar la predicción son las calidades de enlace (LQ) para cada enlace de la red y saber el instante temporal de cuándo se obtuvieron. De modo que para cada enlace de la red (origen/destino) se quiere la secuencia de LQ que ha ido tomando a lo largo del tiempo a través de un muestreo. Esta secuencia de valores no es más que una serie temporal, una sucesión de datos medidos ordenados cronológicamente. Con estos datos se pueden utilizar algoritmos de predicción de series temporales para predecir valores futuros de LQ a partir de valores pasados. Entre otros, se podría enseñar a una red neuronal o algoritmo de aprendizaje automático. Como ya se ha explicado, con los valores LQ para cada enlace, se nos permite saber el coste ETX, puesto que los valores NLQ se obtienen de igual forma al considerar enlaces *A-B* y *B-A* como distintos.

Sabiendo ya qué datos se necesitan, pasamos al cómo obtenerlos. Al ser un protocolo de estado de enlace es aparentemente fácil, porque cada nodo de la red tiene información de la topología completa. Con tan solo conectar un nodo a través de una VPN a la red se puede conseguir la información de todos los enlaces y sus LQ. Aunque no es tan fácil como parece, porque esa información puede estar desactualizada por los intervalos de mensaje TC (y como

se verá más tarde, se empeorará aún más por el *fish-eye)*. Para obtener los conjuntos de datos hasta ahora se habían usado *plugins* de OLSRd como *httpInfo* que permite ver información de OLSRd en tiempo real en el navegador o t*xtInfo*. Los datos que devuelven se pueden clasificar por tablas, por ejemplo de topología o de ruta. En la Figura 9, se puede ver una tabla de topología de OLSRd devuelta en un determinado instante. En ella se indica la dirección de origen del enlace, la de destino y los valores de LQ, NLQ y del coste ETX. Entonces, para obtener las series temporales de estos valores para cada enlace, sólo es necesario ir guardando esos valores de LQ en cada instante.

Table: Topology

Dest. IP	Last hop IP	LQ	NLQ	Cost
10.12.91.23	10.12.3.1	1.000	1.000	1.000
10.12.91.25	10.12.3.1	1.000	1.000	1.000
10.12.0.62	10.12.3.1	0.736	0.827	1.639
10.12.4.81	10.12.3.1	0.713	0.851	1.646
10.12.0.139	10.12.3.1	0.819	0.740	1.646
10.12.2.186	10.12.3.1	0.792	0.972	1.298
10.12.0.235	10.12.3.1	0.705	0.525	2.695
10.12.240.3	10.12.240.1	1.000	1.000	1.000
10.12.240.4	10.12.240.1	1.000	1.000	1.000
10.12.240.7	10.12.240.1	1.000	1.000	1.000
10.12.91.119	10.12.240.1	1.000	1.000	1.000
10.12.4.224	10.12.240.1	1.000	1.000	1.000
10.12.91.185	10.12.4.2	1.000	1.000	1.000

Figura 9. Ejemplo de tabla de topología devuelta por el plugin httpInfo de OLSRd

Explicado esto, pasamos a ver la configuración de OLSRd que usa la red, extraída de su página web [37]. Esta información se encuentra en el Listado 1.

```
DebugLevel 0
IpVersion 4
AllowNoInt yes
Pollrate 0.025
TcRedundancy 2
MprCoverage 7
LinkQualityFishEye 1
LinkQualityWinSize 100
LinkQualityDijkstraLimit 0 10.0
LinkQualityLevel 2
UseHysteresis no
FIBMetric "approx"
ClearScreen yes
LinkQualityAging 0.1
LinkQualityAlgorithm "etx_ff"

RtTable 111
RtTableDefault 112
InterfaceDefaults
{
        HelloInterval 3.0
        HelloValidityTime 125.0
        TcInterval 2.0
        TcValidityTime 500.0
        MidInterval 25.0
        MidValidityTime 500.0
        HnaInterval 10.0
        HnaValidityTime 125.0
}
Interface "tap0"
{
        LinkQualityMult 10.12.11.1 0.2

}
```

Listado 1. Configuración de OLSRd (fichero olsrd.conf) de Funkfeuer Graz

Este fichero de configuración indica cómo debe funcionar OLSRd. A continuación, se indican y explican los parámetros que afectarán de mayor forma a la hora de capturar los datos.

- *TcRedundancy 2:* con esto se hace que el nodo que genera mensajes TC envíe en ellos la lista completa de sus vecinos. Otras opciones serían la de enviar solo el conjunto de MPR o de MPR selectores.
- *LinkQualityFishEye 1*: la línea más importante porque al ser 1, activa el mecanismo de *fish-eye*, haciendo que los mensajes TC cambien su TTL de forma secuencial y que no llegue a todos los nodos de la red.
- *LinkQualityLevel 2*: este parámetro hace que se use OLSR con calidad de enlace, si fuese 0, se usarían solo los saltos y cumpliría el RFC 3626. Con 2 se usa la calidad de enlace para elegir MPR y para todas las rutas, con 1 solo se usaría para elegir los MPR.
- *LinkQualityAlgorithm "etx_ff"*: hace que la calidad de enlace se calcule con los mensajes HELLO y TC, no solo los HELLO como se explicó en el capítulo anterior [32]. Su nombre viene de *ETX Freifunk*, otra WCN.
- *HelloInterval 3.0*: el tiempo entre mensajes HELLO es de 3 segundos.
- *TcInterval 2.0*: el tiempo entre mensajes TC es de 2 segundos.

De su configuración sabemos que el mecanismo *fish-eye* está activado. Para saber la secuencia de TTL que se usa en esta WCN se revisa el código de OLSRd, que es de código abierto. En la Figura 10 se puede ver en la creación de un mensaje TC la secuencia de TTL que se usa, donde MAX_TTL es una constante de valor 255. La secuencia TTL cíclica para *fish-eye* en la red es de 2, 8, 2, 16, 2, 8, 2, 255. Las implicaciones de toda la configuración se expondrán en la sección 3.4, después de haber explicado los conjuntos de datos muestreados cada 5 minutos y cada segundo.

```
static void
create_lq_tc(struct lq_tc_message *lq_tc, struct interface *outif)
{
  struct link_entry *lnk;
  struct neighbor_entry *walker;
  struct tc_mpr_addr *neigh;
  static int ttl_list[] = { 2, 8, 2, 16, 2, 8, 2, MAX_TTL };
```

Figura 10. Secuencia TTL de OLSRd por defecto

3.3 Conjunto de datos muestreados cada 5 minutos

En esta sección se habla de los conjuntos de datos muestreados cada 5 minutos. Con este horizonte temporal se han realizado varios trabajos. En [2] se usó un conjunto de datos obtenidos de la red de *Funkfeuer* gracias a la plataforma de datos abiertos *CONFINE Project*[18]. Se usaron datos muestreados cada 5 minutos durante 7 días, del 28 de abril al 4 de mayo del 2014. Con estos datos se probaron distintas técnicas de aprendizaje automático en lote, tales como SVM (*Support Vector Machine*) o RT (*Regression Trees*). En [10] y [11] se añadió un algoritmo de referencia al trabajo anterior y se probó la eficiencia de algoritmos de

[18] https://confine-project.eu/

aprendizaje automático en línea, con menor coste computacional. Para estos dos últimos trabajos no se encontraban disponibles los datos de [2] por lo que se usó una nueva captura del 15 al 21 de enero de 2016 desde la misma plataforma que antes, *CONFINE Project*. Los datos se pueden encontrar en https://zenodo.org/record/2635054. Este conjunto de datos dispone de ficheros recogidos cada 5 minutos de las tablas de topología en formato .tsv como la mostrada en la Figura 9. Estos datos me sirvieron como iniciación e introducción al manejo de series temporales, extracción y de predicción de LQ con técnicas de aprendizaje automático. Debido a que el muestreo fue muy lento, no se disponían de muchos datos y sirvió de conjunto de datos de prueba ya que se podía entrenar en relativo poco tiempo y bajo coste con mi portátil personal.

Después de hacer pruebas con este conjunto de datos y de otras señales arbitrarias para ganar mayor conocimiento con LSTM, se dispuso de un ordenador del grupo de investigación con tarjeta gráfica dedicada y de mayor procesamiento. No se van a reportar resultados de este conjunto de datos por el motivo de carecer de poca utilidad práctica hacer predicción de valores de LQ tan lejanos. En el mundo telemático, en el que en cualquier instante se puede producir congestión y a las velocidades que trabajan las transmisiones de datos, 5 minutos es demasiado. Ese tiempo es muchas veces superior al tiempo de comunicación entre los nodos más distantes de la red. Siendo más útil predecir valores de LQ futuros pero en tiempos más cercanos. Esto llevó a pensar en la necesidad de un conjunto de datos muestreado con mayor frecuencia. Con este objetivo en [12] se tomaron nuevas medidas de LQ muestreadas cada segundo.

3.4 Conjunto de datos muestreados cada segundo

Como se ha indicado, el muestreo que existía de 5 minutos era demasiado lento. En [12] se pretendía estudiar las técnicas de [10] y [11], pero con datos de *Funkfeuer* con mayor tasa de muestreo. Para ello se creó una VPN (red privada virtual, *Virtual Private Network*) conectado a un nodo de la red y con el *plugin txtinfo* de OLSRd se capturaron tablas de topología como la mostrada en la Figura 9. El conjunto de datos que se usará para comprobar el funcionamiento de técnicas de *Deep Learning* será este, el de muestreo de LQ realizado cada segundo en [12]. Este conjunto de datos contiene 14 días completos, del 13 al 26 de noviembre del 2017 y se encuentra en https://zenodo.org/record/2656385. En el trabajo en el que se obtuvieron estos datos se hace un estudio de la red, considerando variables como el de número de nodos, los enlaces activos cada día, etc. En ese estudio no se encontró una correlación de LQ entre horas del día, día de la semana y demás. Debido a esos resultados, se decide no volver a estudiar el comportamiento del LQ o de los nodos de nuevo.

En las siguientes subsecciones se va a hablar de las limitaciones que tiene esta captura de datos por las características del protocolo OLSR y la configuración de OLSRd que tiene *Funkfeuer*. Después, se explicará cómo se han obtenido los valores LQ de cada enlace a partir de los datos que se disponen. A continuación, se hará un estudio de la actualización de LQ de cada enlace. Para ello se verá si la tasa de actualización de LQ (equivalente a llegada de mensajes TC) se corresponde con la predicha y clasificar los enlaces de algún modo para hacer el remuestreo.

3.4.1 Limitaciones del conjunto de datos e implicaciones para su toma

La principal novedad respecto a otros estudios va a ser respecto como tomar los datos, debidas al funcionamiento tanto de OLSR como OLSRd. En [12], para encontrar un valor óptimo de muestreo (remuestrear a conveniencia partiendo de los datos de 1 segundo), no

se tuvo en cuenta el cómo funciona el protocolo. En ese trabajo para elegir el periodo de muestreo se hizo una FFT a los valores de LQ con el fin de ver su tasa de cambio global y se eligió un periodo de 5 segundos.

Con la configuración que se ha indicado en la sección 3.2, llegamos a un hecho que ninguno de los estudios anteriores de predicción de calidad de enlace en esta red había tenido en cuenta. Es el hecho de que, dependiendo de la distancia de los nodos de la red con el nodo que está capturando datos, la frecuencia de actualización de la calidad de enlace cambia debido al *fish-eye*. Además, hay más efectos que pueden hacer que esta frecuencia cambie.

Estamos en el punto de que queremos capturar las series temporales correspondientes a los valores de LQ de cada enlace. Idealmente, nos gustaría que tuviéramos el valor real para cada enlace en cada momento pero esto no es posible, lógicamente. En la práctica, podría aspirarse obtener valores de LQ de cada enlace cada 2 segundos, ya que es el tiempo que los nodos mandan un mensaje de TC. Pero esto tampoco es así, debido a que, como hemos visto, OLSR tiene un mecanismo que hace que el intervalo entre TC no sea exacto al haber un tiempo *jitter* aleatorio a la hora de generar y de reenviar. Aun así, esto último no debería ser un problema al ser un valor bajo y el tiempo entre llegadas de TC debería ser más o menos constante. El verdadero problema viene por el mecanismo *fish-eye*, que hace que los nodos alejados nos actualicen su valor de LQ con menor frecuencia, ya que muchos de sus TTL serán inferiores a nuestra distancia haciendo que nos lleguen a menor ritmo. Los estudios anteriores usaban los datos de LQ muestreados de igual forma para todos los enlaces, cuando dependiendo del enlace, esos valores LQ podrían estar repetidos (en el caso del muestro de 5 minutos no tendría efecto, pero tiene los problemas ya comentados). Es decir, esos estudios podrían indicar que en un enlace se mantiene la misma LQ en unos instantes de tiempo cuando en realidad lo que ocurría era que no se estaba actualizando el valor al llegar los TC más tarde.

En la sección 3.2 se observó que la secuencia TTL cíclica para *fish-eye* en la red es de 2, 8, 2, 16, 2, 8, 2, 255. Sabiendo que los mensajes TC se generan cada 2 segundos, llegamos a que:

- Vecinos de 1 y 2 saltos: el nodo desde el cual se capturan los valores LQ ve los 8 mensajes y por lo tanto, cada 2 segundos se debería actualizar la LQ.
- Vecinos entre 3 y 8 saltos: - x - x - x – x ('x' indica que llega el mensaje TC, '-' que no, de la secuencia de 8 TTL). De 8 ve 4, llega un mensaje TC cada 4 segundos. Se actualiza el valor de LQ cada 4 segundos.
- Vecinos de entre 9 y 16 saltos: - - - x - - - x. De 8 ve 2, llega un mensaje TC cada 8 segundos.
- Vecinos de +16 saltos (no debería haber si está bien configurada la red): ve 1 de cada 8. Llega un mensaje cada 16 segundos.

Por lo tanto, dependiendo de la distancia de los nodos que informan de un enlace, la frecuencia de actualización de LQ varía entre 2, 4 u 8 segundos. Pero además, hay que sumarle dos dificultades más. La primera es que el número de saltos de un nodo puede variar con el tiempo y en una WCN es muy habitual que se desactiven o añadan nodos haciendo que cambie el número de saltos y por lo tanto, la frecuencia de actualización de LQ. El otro es debido a que un enlace es informado por dos nodos (los que conforman el enlace) y ambos informan del valor LQ y NLQ que conocen. Por ejemplo, supongamos que el nodo A informa del LQ_A y NLQ_A que tiene con B: el mensaje TC de A actualizará la LQ de A-B y la LQ de B-A

(con el valor de NLQ_A). De la misma forma, el mensaje TC de B, actualizará la LQ de A-B (con el NLQ_B) y la LQ de B-A (que será el LQ_B). Esto implica que la información de cada enlace de la red se va a actualizar con dos mensajes TC de origen distinto, haciendo que el desfase entre estos pueda cambiar la frecuencia de actualización de LQ. El desfase puede ir de 0 hasta el intervalo de generación de TC (recordemos que no hay sincronización a la hora de generarse). Si es 0 o el máximo, significa que los mensajes de TC de ambos orígenes llegan a la vez y se mantiene igual la frecuencia de actualización. En cambio, si se encuentra en el punto medio, hará que el intervalo entre llegadas de actualización de LQ sea la mitad. Y todo esto último, es teniendo en cuenta que los dos nodos que forman el enlace están a los saltos necesarios para que compartan intervalos de TC. Por ejemplo, lo anterior no se cumple si un enlace está formado por un nodo a 8 saltos y otro a 9, ya que los TC del de 8 llegarán cada 4 segundos y el de 9 cada 8 segundos.

En resumen. las consideraciones que se deben tomar para obtener los valores LQ de un enlace son:

- Los valores de LQ de un enlace se actualizan dependiendo de la distancia de los nodos que lo forman respecto al nodo que captura, por culpa del *fish-eye*, pudiendo ser de 2, 4, 8 o 16 segundos.
- Estos valores entre llegadas no podrían cumplirse debido al hecho de que OLSR tiene un tiempo *jitter* aleatorio, que modifica la generación y mantiene un tiempo los mensajes antes de reenviar. También puede afectar el *piggybacking*.
- Los nodos pueden cambiar de distancia en saltos debido a las características dinámicas de estas redes, pudiendo modificar también las llegadas de TC y sus correspondientes actualizaciones en el valor de LQ.
- Tal y como se capturan los datos (ver Figura 9), puede haber inconsistencias en los tiempos de llegada entre mensajes TC. Un mismo valor de LQ se actualiza de dos nodos distintos que pueden llegar a distinto ritmo, disminuyendo el tiempo entre actualizaciones de LQ. Además, los nodos pueden estar a distintos saltos, variando el ritmo de distintas formas.

3.4.2 Obtención de LQ de cada enlace

Esta subsección explica cómo se obtuvieron los datos de LQ por enlace a partir de las tablas de topología. El primer paso es el obtener en un único fichero de texto la información de un enlace, con nombre IPOrigen+IPDestino, y de los valores de LQ que tenía cada segundo. Para ello se disponía de los ficheros generados con OLSRd como el de la Figura 9, tablas de topología. Estas tablas se guardaron en la máquina que tenía la gráfica dedicada del grupo de investigación. Cada tabla tiene en el nombre del fichero el instante temporal en el que se capturó. En la Figura 11 se ven las 10 primeras tablas muestreadas cada segundo, y de igual forma se muestra un ejemplo de su contenido.

```
(base) ediezb@ares:/srv/funkfeuer_data/171112-171127/topologia$ ls | head -n 10
topo_12-11-2017-20-00-20
topo_12-11-2017-20-00-21
topo_12-11-2017-20-00-22
topo_12-11-2017-20-00-23
topo_12-11-2017-20-00-24
topo_12-11-2017-20-00-25
topo_12-11-2017-20-00-26
topo_12-11-2017-20-00-27
topo_12-11-2017-20-00-28
topo_12-11-2017-20-00-29
(base) ediezb@ares:/srv/funkfeuer_data/171112-171127/topologia$ cat topo_12-11-2017-20-00-20
Table: Topology
Dest. IP        Last hop IP     LQ      NLQ     Cost
10.12.4.204     192.168.1.1     1.000   1.000   1.000
10.12.91.23     10.12.3.1       1.000   1.000   1.000
10.12.91.25     10.12.3.1       1.000   1.000   1.000
10.12.0.62      10.12.3.1       0.804   0.819   1.518
10.12.4.81      10.12.3.1       0.772   0.709   1.823
10.12.0.139     10.12.3.1       0.619   0.940   1.714
10.12.2.186     10.12.3.1       0.831   0.968   1.241
10.12.0.235     10.12.3.1       0.709   0.215   6.531
10.12.240.3     10.12.240.1     1.000   1.000   1.000
10.12.240.4     10.12.240.1     1.000   1.000   1.000
10.12.240.7     10.12.240.1     1.000   1.000   1.000
```

Figura 11. Arriba: datos con las tablas de topología cada segundo. Abajo: un ejemplo del contenido de estas tablas

En dos semanas hay muchos segundos, y ese directorio tenía 1.250.550 ficheros en total, representando cada tabla un segundo. Se realiza una serie de programas en Python para convertir esos 85 GB de datos de tablas, en una serie de ficheros de LQ por enlace. A partir de la lectura de todos estos archivos, se crean ficheros en formato *arff* (tipo de fichero que permite añadir atributos a los datos, como el tiempo a cada LQ). Cada fichero pertenece a un enlace *ipOrigen*to*ipDestino*, cambiando "." de IPs por "_" y en ellos se guarda el *timestamp* en formato "yyyy-MM-dd HH:mm:ss" y el valor LQ. En la Figura 12 se observa una serie de esos ficheros por enlace y un ejemplo de lo que contiene uno de estos ficheros, el valor LQ y el tiempo en el que se obtuvo. Los ficheros Python que se crearon para esta conversión se pueden encontrar en el GitHub de este trabajo: https://github.com/gsic-emic/tfg_2019_diez.

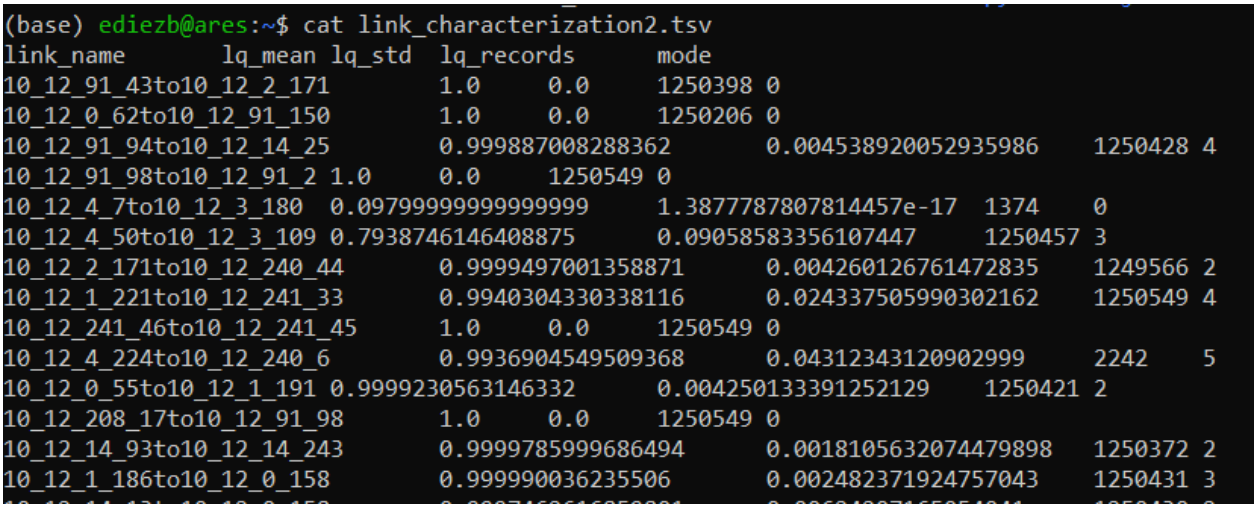
```
10_12_14_25to10_12_2_98.arff        10_12_3_180to10_12_91_130.arff      192_168_1_1to10_12_4_204.arff
10_12_14_25to10_12_3_250.arff       10_12_3_180to10_12_91_134.arff      2_1_109to10_12_2_254.arff
10_12_14_25to10_12_34_89.arff       10_12_3_180to10_12_91_49.arff       2_1_155to10_12_240_254.arff
10_12_14_25to10_12_91_29.arff       10_12_3_180to10_12_91_63.arff       2_14_140to10_12_14_254.arff
10_12_14_25to10_12_91_4.arff        10_12_3_199to10_12_3_180.arff       2_91_136to10_12_2_254.arff
10_12_14_25to10_12_91_67.arff       10_12_3_199to10_12_4_161.arff       40to10_12_14_254.arff
10_12_14_25to10_12_91_87.arff       10_12_3_199to10_12_4_221.arff       4_140to10_12_14_254.arff
10_12_14_25to10_12_91_94.arff       10_12_3_199to10_12_4_7.arff         55to10_12_240_254.arff
10_12_14_26to10_12_14_74.arff       10_12_3_1to10_12_0_139.arff         5to10_12_240_254.arff
10_12_14_26to10_12_14_91.arff       10_12_3_1to10_12_0_235.arff         7to10_12_3_250.arff
10_12_14_26to10_12_14_92.arff       10_12_3_1to10_12_0_62.arff          91_136to10_12_2_254.arff
10_12_1_44to10_12_14_140.arff       10_12_3_1to10_12_14_136.arff
(base) ediezb@ares:~/LQsPorEnlaces$ cat 10_12_0_103to10_12_0_9.arff | head -n 15
@relation 10_12_0_103to10_12_0_9

@attribute timestamp date "yyyy-MM-dd HH:mm:ss"
@attribute link_quality numeric

@data
"2017-11-12 20:00:20",1.000
"2017-11-12 20:00:21",1.000
"2017-11-12 20:00:22",1.000
"2017-11-12 20:00:23",1.000
"2017-11-12 20:00:24",1.000
"2017-11-12 20:00:25",1.000
"2017-11-12 20:00:26",1.000
"2017-11-12 20:00:27",1.000
"2017-11-12 20:00:28",1.000
```

Figura 12. Arriba: lista de ficheros con los LQ por enlace. Abajo: ejemplo de contenido de uno de esos ficheros .arff

Por último, se hizo un fichero .tsv que contiene el nombre de enlace de todos los ficheros e información de ellos. De esta manera, no es necesario leer continuamente el más de 1 millón de muestras LQ para saber la información de cada enlace. En la Figura 13 se puede ver su inicio. Se indica su nombre, la media LQ, su desviación típica, el número de muestras y la moda de cambio que veremos en la siguiente sección. De modo que, este archivo puede servir de índice para filtrar enlaces o caracterizarlos sin tener que procesar todos los datos de nuevo.



```
(base) ediezb@ares:~$ cat link_characterization2.tsv
link_name       lq_mean lq_std  lq_records      mode
10_12_91_43to10_12_2_171        1.0     0.0     1250398 0
10_12_0_62to10_12_91_150        1.0     0.0     1250206 0
10_12_91_94to10_12_14_25        0.999887008288362       0.004538920052935986    1250428 4
10_12_91_98to10_12_91_2 1.0     0.0     1250549 0
10_12_4_7to10_12_3_180  0.09799999999999999     1.3877787807814457e-17  1374    0
10_12_4_50to10_12_3_109 0.7938746146408875      0.09058583356107447     1250457 3
10_12_2_171to10_12_240_44       0.9999497001358871      0.004260126761472835    1249566 2
10_12_1_221to10_12_241_33       0.9940304330338116      0.024337505990302162    1250549 4
10_12_241_46to10_12_241_45      1.0     0.0     1250549 0
10_12_4_224to10_12_240_6        0.9936904549509368      0.04312343120902999     2242    5
10_12_0_55to10_12_1_191 0.9999230563146332      0.004250133391252129    1250421 2
10_12_208_17to10_12_91_98       1.0     0.0     1250549 0
10_12_14_93to10_12_14_243       0.9999785999686494      0.0018105632074479898   1250372 2
10_12_1_186to10_12_0_158        0.999990036235506       0.002482371924757043    1250431 3
```

Figura 13. Fichero de caracterización de todos los enlaces

3.4.3 Estudio de frecuencia de cambio y distancia en saltos

Una vez obtenido un fichero para cada enlace con sus valores LQ y sus instantes temporales, se llega a la duda de qué periodo de muestreo usar. El periodo de muestreo óptimo será aquel intervalo en el que llegan mensajes de TC y se actualiza el valor de LQ. Pero ese valor, en principio, no lo podemos saber porque solo disponemos del valor LQ y del tiempo en el que se obtuvo, sin saber, en caso de haber 2 valores iguales consecutivos, si ha llegado un nuevo mensaje TC con el mismo valor LQ o si este valor no se ha actualizado, proviniendo de un mensaje TC anterior. Este valor de actualización no es tan difícil de calcular si nos damos cuenta de que si cambia el valor de LQ en un enlace, es que en ese tiempo ha llegado como mínimo un TC. Y la duración mínima que haya entre cambios en un enlace va a indicar la frecuencia de llegadas de TC (si se produjese a intervalos regulares), y como consecuencia el tiempo de actualización de LQ de ese enlace. Si disponemos de ese tiempo para cada enlace, el entrenamiento y predicción deberían ser consistentes con ello. Por ejemplo, si se observa que en un enlace el mínimo tiempo que hay entre cambios de LQ es de 4 segundos, el conjunto de datos que se usará es un submuestreo del fichero .arff de ese enlace (reteniendo solo con valores LQ cada 4 segundos).

Antes de seguir con este estudio de cambios de valores en la LQ, hay que fijarse también en la distancia a la que se encuentran respecto del nodo que capturó esos datos. Como vimos en la 3.4.1, los valores de LQ de un enlace se actualizan de distinta manera dependiendo de esa distancia, debido al *fish-eye*, y además, los nodos pueden cambiar su distancia en saltos debido a las características dinámicas de estas redes. Para poder saber la distancia de los nodos, por suerte, en esta captura se guardaron en otro directorio también las tablas de rutas de las que podemos obtener esta información. En la Figura 14, se muestran los 10 primeros ficheros de estas tablas junto con un ejemplo del contenido.

```
(base) ediezb@ares:/srv/funkfeuer_data/171112-171127/rutas$ ls | head -n 10
route_12-11-2017-20-00-42
route_12-11-2017-20-00-43
route_12-11-2017-20-00-44
route_12-11-2017-20-00-45
route_12-11-2017-20-00-46
route_12-11-2017-20-00-47
route_12-11-2017-20-00-48
route_12-11-2017-20-00-49
route_12-11-2017-20-00-50
route_12-11-2017-20-00-51
(base) ediezb@ares:/srv/funkfeuer_data/171112-171127/rutas$ cat route_12-11-2017-20-00-42 | head -n 15
Table: Routes
Destination     Gateway IP      Metric  ETX     Interface
0.0.0.0/0       10.12.11.1      2       2.000   tap0
10.12.0.2/32    10.12.11.1      8       8.143   tap0
10.12.0.9/32    10.12.11.1      6       6.113   tap0
10.12.0.10/32   10.12.11.1      2       2.000   tap0
10.12.0.38/32   10.12.11.1      12      13.961  tap0
10.12.0.47/32   10.12.11.1      12      13.961  tap0
10.12.0.52/32   10.12.11.1      4       4.000   tap0
10.12.0.53/32   10.12.11.1      4       4.000   tap0
10.12.0.55/32   10.12.11.1      4       4.000   tap0
10.12.0.61/32   10.12.11.1      6       8.399   tap0
10.12.0.62/32   10.12.11.1      6       8.399   tap0
10.12.0.80/32   10.12.11.1      11      12.068  tap0
10.12.0.93/32   10.12.11.1      7       11.547  tap0
```

Figura 14. Arriba: datos con las tablas de ruta cada segundo. Abajo: un ejemplo del contenido de estas tablas

Esta tabla muestra por columnas el nodo/subred al que se hace referencia, el nodo para el reenvío (solo tenemos un vecino, el 10.12.11.1 va a ser al que vayan todos nuestros paquetes), la métrica que indica los saltos, el coste ETX (suma de todos los costes ETX hasta ese nodo) y la interfaz de salida que es la de la VPN, `tap0`. Merece mención el hecho de que en la imagen se ve como el coste ETX es como mínimo el número de saltos, porque por cada enlace el mejor ETX es 1. Si hay algún enlace con ETX mayor a 1, el coste ya es superior al número de saltos.

Con estas tablas se hace un procesado similar al de LQ y se crea esta vez un fichero por nodo (no enlace) indicando el número de saltos y los cambios en ese número que sufre a lo largo del tiempo. De esta forma se puede tener caracterizado cualquier nodo para saber a cuántos saltos estaba en un momento determinado. En estos ficheros se observa que hay nodos muy inestables que cambian frecuentemente entre 2 números de saltos, otros que no cambian u otros que aunque cambien están el 99% del tiempo a la misma distancia. Como consecuencia, el estudio de LQ para los nodos que cambien mucho de saltos va a ser muy problemático por el hecho de que cambiarán de igual forma el intervalo entre TC. En cambio, los nodos estables o que cambien solo a distancias del mismo rango de alcance de TTL, se podrá estudiar su comportamiento de manera más clara.

Ahora pasamos al cálculo del periodo de muestreo óptimo para cada enlace. Para calcularlos se realiza un programa para que calcule la duración de tiempos entre cambios de LQ de cada enlace y poder hacer su histograma. La moda del tiempo entre los cambios (el valor que más se repite) de LQ será el que se use para muestrear cada enlace. Primero, se muestra el histograma de tiempos de cambios entre LQ. Para ello se eligió una serie de enlaces con una gran desviación típica (para ver los cambios) y número de muestras. Para cada uno de ellos también se observó la distancia a la que estuvieron los dos nodos que formaban ese enlace durante las dos semanas. El primer enlace que se muestra es el que va de la IP de origen 10.12.2.103 a la de destino 10.12.4.160, enlace 10.12.2.103/10.12.4.160. Los nodos que conforman este enlace están las dos semanas entre 11 y 12 saltos de distancia, lo que significaría que el TC llegase cada 8 segundos. El resultado se muestra en la Figura 15. Se puede ver que no hay valores en 0, puesto que el mínimo valor para un cambio de LQ es de 1 segundo al muestrear a esa tasa. También es lógico pensar que tiene que haber valores de cambio en torno a un número y sus múltiplos, al poder perderse un mensaje TC o que el cambio sea tras dos, tres o más actualizaciones. En la Figura 15 se ve que la moda es 7, muy

cercano a los 8 segundos que se esperaban. Que no sea exacto puede deberse a los redondeos de tiempo o por los otros efectos comentados en la sección 3.4.1. Además, se observan perfectamente los múltiplos de 7 que son 14, 21, 28 y que son valores posibles para un cambio de LQ por lo comentado. En concreto los valores alrededor de los múltiplos deben ser consecuencia directa del tiempo *jitter* aleatorio que hace variar los intervalos entre TC.

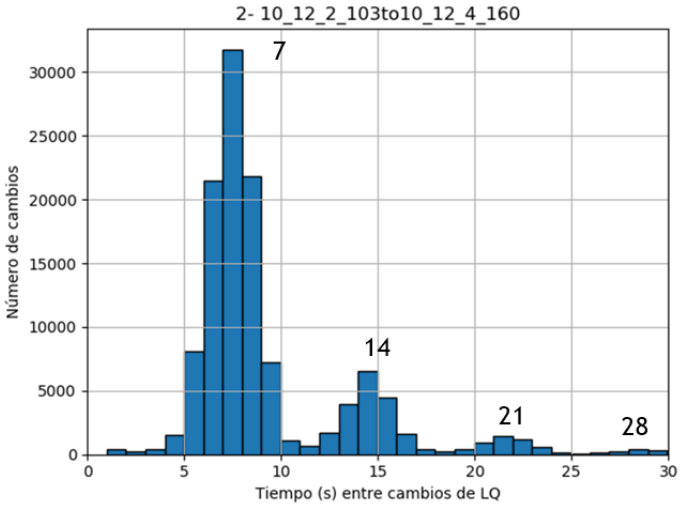


Figura 15. Histograma entre cambios de LQ de 10_12_2_103to10_12_4_160

Con el objetivo de ver otro tipo de enlaces más cercanos se elige el enlace 10.12.2.241/10.12.240.60, cuyos nodos están a 4 y 5 saltos durante todo el tiempo. A estas distancias se espera que la tasa de actualización sea de 4s. En la Figura 16, se ve que la moda, en efecto, es 4. Se observan los múltiplos pero esta vez se encuentran también los del 7 pero como 7 es casi múltiplo de 4 no sería un problema muestrear cada 4 segundos.

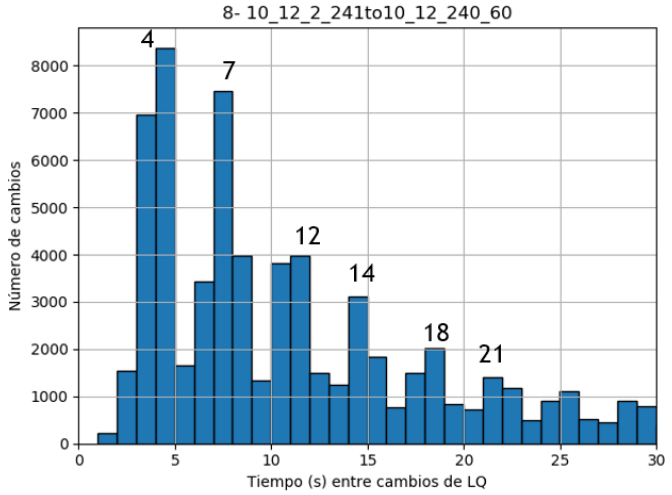


Figura 16. Histograma entre cambios de LQ de 10_12_2_241to10_12_240_60

Aparte de estos enlaces se observaron otros que de forma general compartían esa apariencia, debidos a los efectos mencionados en la sección 3.4.1. Por último, se hace uso del fichero de caracterización que se explicó en la subsección anterior que contenía la moda de los cambios y se realiza un histograma con el resultado. Hay que aclarar, que este no es un histograma agregado de todos los enlaces, sino que es el histograma de las modas de cada enlace. Además, los enlaces que han tenido un valor LQ constante a lo largo de toda la captura y tendrían moda infinita (no hay cambios, tiempo entre cambios infinito) se pondrán con moda 0, puesto que es un valor que por sí, ningún enlace podría tomar. En la Figura 17 se

encuentra el resultado. Se observa que más de 700 enlaces no han tenido ningún cambio y por lo tanto tienen moda 0.

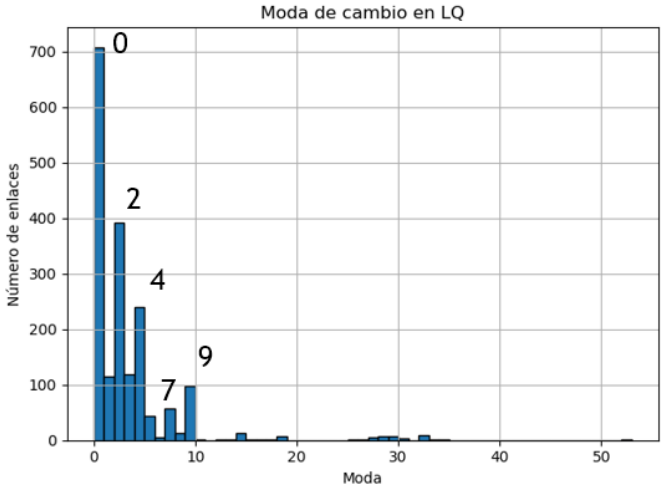


Figura 17. Histograma de las modas de los tiempos de cambio de LQ

De esta figura se pueden sacar las siguientes conclusiones. Se observan que hay dos modas muy repetidas que son la de 2 y 4 segundos. Y corresponden justamente con las dos zonas predichas en la sección 3.4.1. Llegadas de TC de 2s para nodos a 1 o 2 saltos y 4s para los de entre 3 y 8. Para los enlaces de más de 8 saltos se llegaba a que la actualización sería cada 8s y no se observa ese valor. En cambio en 7s y 9s hay bastantes enlaces por lo que es de esperar que estos dos valores tengan relación con nodos lejanos. Como vimos, hay muchos factores que influyen en este valor. También, se puede ver modas de 1s, cosa que es bastante improbable que ocurra porque los TC son cada 2 segundos, salvo debidos al efecto comentado de recibir doble TC desfasado. Pero lo más probable es que se deba a que nodos muy inestables y de los que hay muy pocas muestras, se han ido detectando cambios entre LQ consecutivos. Pero no porque haya variado el valor en un segundo, sino que entre esos valores de LQ han podido pasar minutos u horas al perderse o no haber LQ intermedios.

Hemos visto la ardua tarea que ha sido decidir el periodo entre actualizaciones LQ con los datos obtenidos a través del *plugin httpInfo* de OLSRd. Se ha necesitado entender de forma completa el protocolo OLSR, las particularidades de OLSRd con *fish-eye*, estudiar los histogramas de cambios por enlace, los saltos a los que está el nodo, etc. Todo esto se podría haber evitado si se hubiese dispuesto del tráfico OLSR intercambiado en vez de solo la tabla con LQ en cada momento. Con la tabla de topología no se puede saber si valores iguales de LQ se deben a que no ha llegado un TC y se mantiene el valor o que ha llegado con el mismo valor y se mantiene. Una solución es la de capturar el tráfico OLSR, obteniendo los valores LQ directamente de los mensajes TC. De esta forma, se sabe en todo momento cuándo llega información de LQ y a qué ritmo, también puedes conocer la distancia de los nodos que te envían información de LQ. Un punto negativo, es la complejidad de obtener los LQ de cada paquete que llega, pero una vez programado un *script* para obtenerlos siempre se podría usar el mismo.

3.5 Captura de un nuevo conjunto de datos

Todos los motivos anteriores llevan a realizar una nueva captura de datos en la red *Funkfeuer Graz*, esta vez de tráfico OLSR. Para ello, se necesitó pedir permiso a algún administrador de la red para que nos dejara conectarnos a través de una VPN. Los pasos que

siguieron fueron los siguiente: configurar la VPN, configurar las OLSRd (e interfaces) y capturar el tráfico OLSR recibido. Todos los pasos están explicados en el Apéndice A. Se obtuvo todo el tráfico OLSR del 20 de mayo a las 12:49:15, hasta el 22 de mayo a las 22:18:27 de 2020, pudiéndose encontrar el tráfico en https://doi.org/10.5281/zenodo.3906818. Con estos datos se hará en la primera subsección un análisis a algunos de sus paquetes y, después, se estudiará el tiempo entre TC que se observa de manera real en la red, para verificar si las hipótesis que se han hecho en secciones previas son ciertas.

3.5.1 Análisis de algunos paquetes. LQ HELLO y LQ TC

En esta sección comprobaremos el detalle de algunos paquetes capturados a través de la aplicación Wireshark[19]. Antes de nada, veremos el contenido exacto de los mensajes HELLO y TC que se usan en la red. Esto es debido a que, al contrario que en el RFC 3626 de OLSR, en la documentación OLSRd no se especifica el contenido y forma de los nuevos mensajes LQ HELLO y LQ TC.

Empezando por los mensajes LQ HELLO, en nuestra conexión VPN solo tenemos un vecino a un salto, que es el nodo al que se conectan todas las VPN a la red. Debido a que los mensajes HELLO solo llegan a los vecinos adyacentes al tener un TTL 1, los únicos mensajes de este tipo que se observan son los de nuestra máquina 10.12.11.33 y la interfaz de nuestro vecino 10.12.11.1. En la Figura 18, se ve arriba cómo solo hay mensajes de nuestra interfaz la .33 y la del vecino .1, y las direcciones de destino son .255 de *broadcast* al mandarse así todos los mensajes de OLSR (los mensajes se reenvían si se eres MPR y si el TTL es mayor que 1). En esta figura se observa el formato de un LQ HELLO generado por nuestra máquina. Se comprueba que el valor de TTL es 1, la generación es de cada 3 segundos y se indica nuestra lista de vecinos. En este caso, solo consta de un único vecino que además, lógicamente, se indica como MPR y para que sepa éste la NLQ respecto a nosotros, se envía la LQ (y la NLQ). La LQ se envía como un número entero de 0 a 255 (un byte), que se escala de 0 a 1. El formato de los LQ HELLO es consistente con lo estudiado de OLSRd.

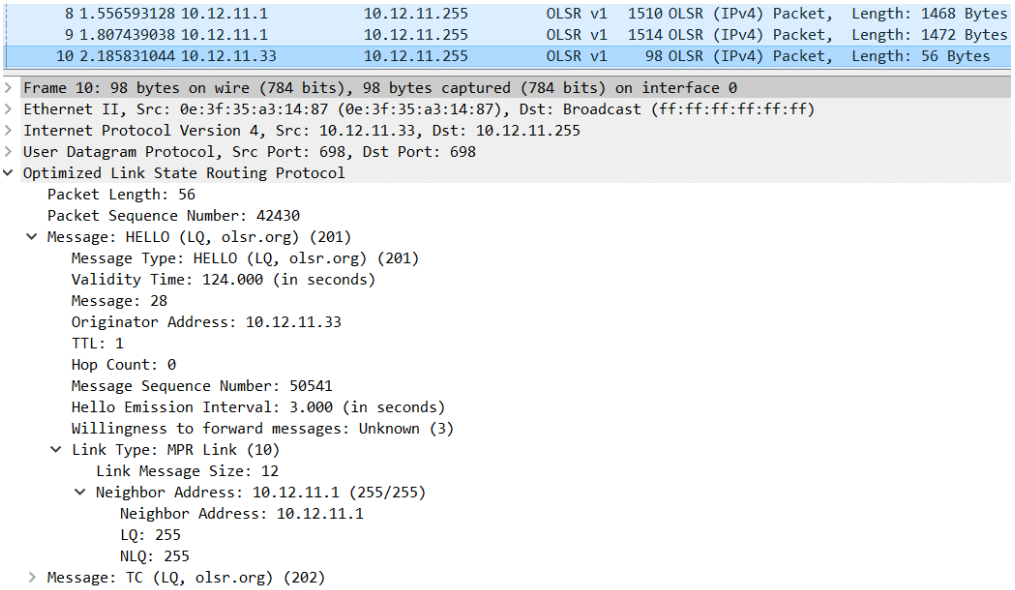


Figura 18. Formato de un mensaje LQ HELLO

Pasemos ahora a ver los mensajes de TC. Los mensajes de TC que envía nuestra máquina solo contienen la información de LQ y NLQ de nuestro vecino. En la Figura 18 se ve

[19] https://www.wireshark.org/

que se mandó el TC junto al HELLO (en un mismo paquete al hacer *piggybacking*). En cambio, de 10.12.11.1 nos llegan continuos paquetes OLSR que contienen múltiples mensajes OLSR de todos los nodos de la red. Algunos mensajes son de tipo LQ HELLO (solo generados por el propio nodo, nuestro único vecino) y de tipo LQ TC (del propio nodo y del resto, la gran mayoría). También hay mensajes OLSR de tipo MID o HNA pero que no se han estudiado en este trabajo porque no influyen en el cálculo o llegadas de LQ.

En la Figura 19, se muestran dos mensajes TC de un mismo paquete OLSR que nos envió nuestro vecino, juntos por el *piggybacking* de OLSR. Lo primero que se va a ver es su formato, para cada mensaje LQ TC se indica el generador del mensaje inicial, el TTL y *Hop Count* con el que llega y la lista de vecinos que tiene con cada LQ y NLQ. El hecho de que indique LQ y NLQ es importante porque confirma que con esos dos datos, se actualizaría la LQ de dos enlaces distintos por lo explicado en 3.4.1. En la Figura 19, se produce el efecto comentado; el primer mensaje TC lo envía 10.12.5.109 e informa del LQ y NLQ con su único vecino 10.12.5.47. Pero, en ese mismo paquete, otro mensaje TC de 10.12.5.47 informa del LQ y NLQ de todos sus vecinos, entre ellos el 10.12.5.109. En este caso la actualización de LQ de los dos enlaces formados por esos nodos tendrán desfase 0 y por tanto será igual al intervalo entre TC y seguirá el ritmo predicho. Sin embargo, si hubiesen estado desfasados se cambiaría la tasa de actualizaciones.

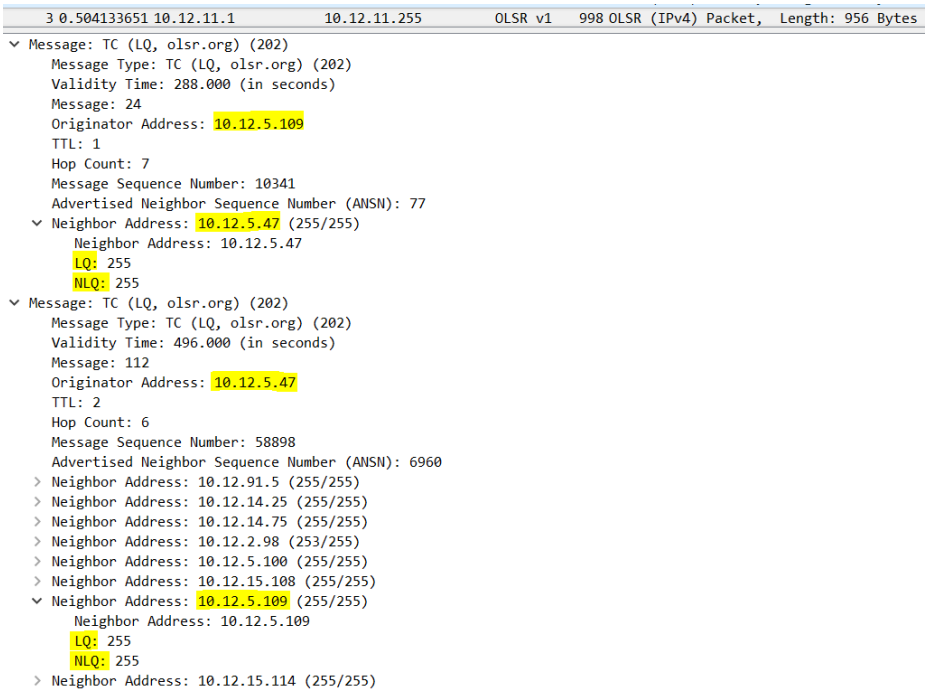


Figura 19. Formato de los mensajes LQ TC. Ejemplo de dos mensajes LQ TC en el mismo paquete OLSR

3.5.2 Análisis de las llegadas de TC

Por último, se hará un estudio del efecto que tiene el *jitter* de OLSR sobre las llegadas de TC (para ver si es posible considerarlo despreciable) y si se cumple lo esperado por *fish-eye*. Para ello, es de utilidad el valor de cuenta de saltos o *Hop Count* con el que podemos saber a qué distancia está de nosotros el nodo generador de mensaje TC y el de la suma de TTL + *Hop Count* que nos da el TTL con el que se generó inicialmente el mensaje. Por ejemplo, en la Figura 19 podemos saber que el nodo terminado en 109 estaba a 7 saltos, el de 47 a 6 saltos y que ambos TC se generaron con unos TTL de 8.

En la Figura 20, se aplican una serie de filtros en Wireshark para ver solo los mensajes OLSR generados por el nodo 10.12.91.103 que se encuentra a 5 saltos. A 5 saltos se estimó

que por *fish-eye* deberían llegar los mensajes TC de la secuencia inicial de TTL de [2, 8, 2, 16, 2, 8, 2, 255], solo los [8, 16, 8, 255] y cada 4 segundos. Los TTL generados por el nodo de la Figura 20 para los primeros 8 mensajes de TC son 8, 16, 8, 255, 8 ,16, 255 por lo que el funcionamiento de *fish-eye* es evidente. En cuanto a los 4 segundos entre mensaje, se observa que por el efecto del *jitter*, que restaba tiempo entre generación, los tiempos son más cortos de forma general. Los hay de 3s de diferencia, algunos del orden de 2s y otros muy cercanos a los 4s que se esperaban. Hay que tener en cuenta que alguno de 2s es debido a que hay algún paquete MID que ha llegado y que Wireshark no ha filtrado, siendo los valores más habituales entre 3 y 4 segundos.

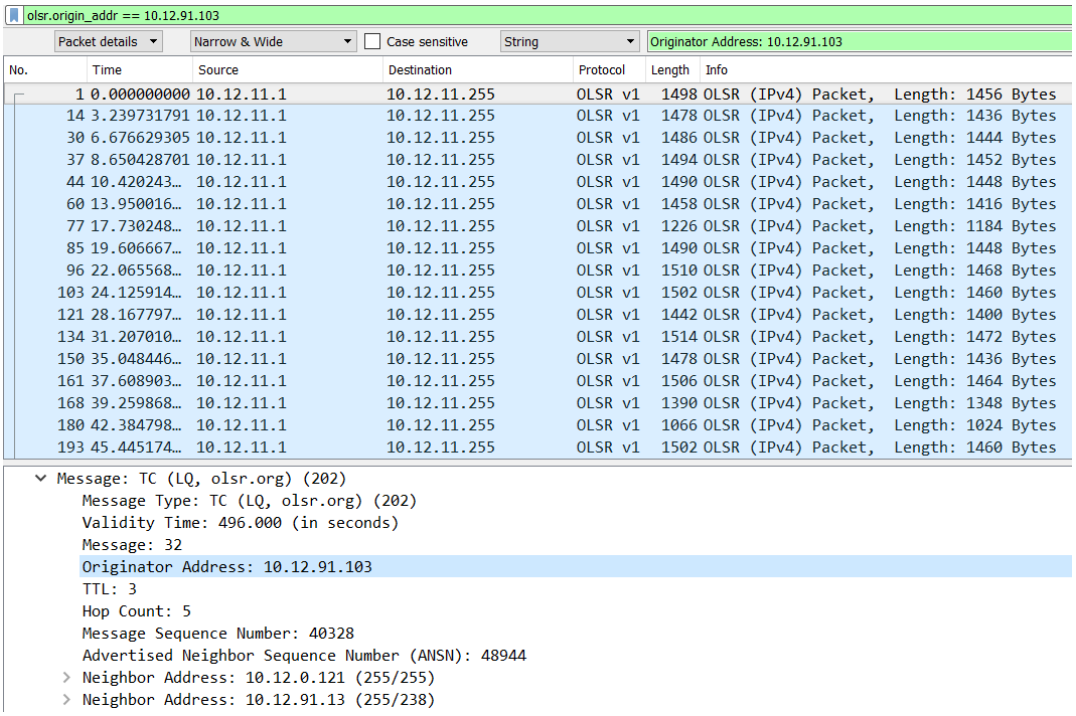


Figura 20. Llegada de mensajes TC de 10.12.91.103

Se observan más patrones de llegadas y se ve que cumplen con lo esperado. En algunos nodos se ha visto que se producen pérdidas de TC y debe ser el gran motivo por el que en los histogramas aparezcan múltiplos en torno al intervalo de TC. Estas capturas nos permiten confirmar que las implicaciones planteadas en la actualización de LQ por el funcionamiento de OLSR y OLSRd son acertadas. Y por tanto, la solución que se ha tomado a la hora de obtener los datos de LQ puede aproximarse, pese a no ser perfecta.

La intención de esta captura es la de usarse para trabajos futuros y para comprobar que las suposiciones que hacemos sobre la llegada de TC y funcionamiento de OLSRd son ciertas. No se usarán estos datos para las pruebas al tener solo dos días de tráfico y al habernos concedido permiso para la captura con el trabajo demasiado avanzado. Para poder convertirse en un conjunto de datos usable para los experimentos es necesario un procesamiento previo. Este procesamiento consiste en leer los ficheros pcap y conseguir de cada paquete OLSR los mensajes TC. De cada mensaje TC obtener el nodo inicial que lo generó y guardar los valores LQ y NLQ que tiene con cada uno de sus vecinos. Con estos valores ya se obtiene el valor LQ de tantos enlaces como vecinos tenía ese nodo. El tiempo asociado a ese LQ será el de la llegada del paquete OLSR, con la ventaja de que sabes el instante exacto de llegada, sin provenir del muestreo de una tabla. Además, a diferencia de las tablas, puedes guardar por cada valor LQ el TTL inicial o a cuántos saltos estaba el nodo origen en el momento de la transmisión. Datos útiles para el estudio de la secuencia de LQ, incluso para

predecir saltos.

3.6 Conjunto de datos empleados en los experimentos

Esta última sección relativa a la toma de los datos va a tratar sobre la elección del conjunto final de enlaces con los que hacer las pruebas, partiendo del conjunto de datos explicado en 3.4. Se usa éste, y no el de la nueva captura, por lo ya comentado (debido a que, se dispuso de los datos con el desarrollo de este trabajo muy avanzado y se necesita un procesamiento que aún no se ha realizado). Además, el número de muestras para entrenamientos de *Deep Learning* iba a ser muy limitado con 2 días, en vez de los 14 días que se tiene del otro conjunto de datos. Para la elección del conjunto de datos final se va a hacer uso del fichero con la información de todos los enlaces realizada en la sección 3.4.2 (ver Figura 13). De esos enlaces hay que elegir un subconjunto para hacer las pruebas, básicamente por 4 motivos:

1. Hay enlaces que no varían o lo hacen en muy pocas ocasiones en todos sus valores de LQ. No tiene sentido predecir estos enlaces porque la predicción sería trivial y en un contexto práctico no se realizaría.
2. Hay algunos errores en la toma de datos que hacen que a algunos enlaces les falte parte de la dirección IP. Esos enlaces se tienen que descartar.
3. Algunos de los enlaces solo estuvieron activos una parte del tiempo durante esas dos semanas, generando pocos datos para entrenamiento y test. Por debajo de un cierto número de muestras de LQ (que más tarde se discutirá) se descartarán los enlaces.
4. El entrenamiento de técnicas de *Deep Learning*, en concreto con LSTM, es generalmente muy lento solo se probará el rendimiento con unos pocos enlaces representativos de toda la red.

Para la decisión del conjunto de enlaces primero se observan algunas características de todos los enlaces. En la sección 3.4 en la Figura 17 se mostró el histograma de las modas de los tiempos de cambio de LQ. En la Figura 21 se enseña el histograma con las medias y desviación típica de los valores LQ del conjunto entero de los enlaces.

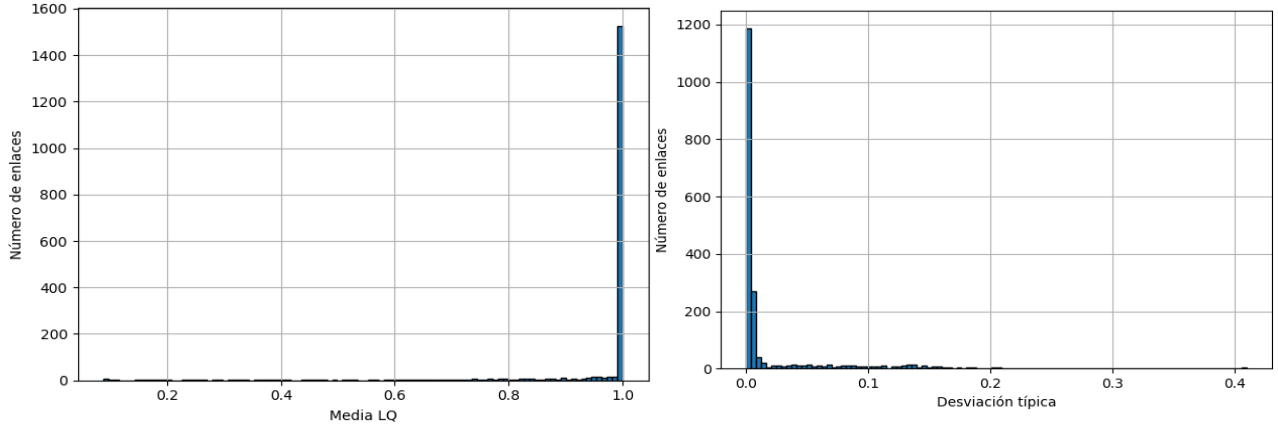


Figura 21. Histograma de las medias y desviación típica de LQ del conjunto de enlaces

De los 1859 enlaces que hay, unos 1200 tienen desviación típica muy cercana a 0 y la gran mayoría tiene valores medios muy altos de LQ. Pese a ser las WCN redes mucho más dinámicas respecto a otras se observa que tienen valores muy cercanos a 1 mucha parte del

tiempo, al final las bajadas en la calidad de LQ son ocasionales y no tienen mucho peso observable en estas gráficas. Además, hay que tener en cuenta que son las medias y desviaciones con el muestreo de un segundo. Hay muchos de esos valores que son repeticiones de LQ no reales (habrá que remuestrear cada 2s, 4s, …) y la repetición de esos valores disminuye la desviación típica.

En la Figura 22 se muestra el histograma según el número de muestras de LQ que tienen los enlaces. En este histograma destacan 3 cosas. La primera, la más notoria es la de que la mayoría de enlaces (más de 1400) tienen casi todos los valores LQ. Estos enlaces se corresponderán con los más estables que poseen un LQ de 1 la mayor parte del tiempo. La segunda es que hay un conjunto en torno a 900.000 muestras que corresponden a un conjunto de enlaces de los cualares se dejaron de recibir LQ a la vez al 11º día a la misma hora. Por último, hay un grupo de enlaces que tienen muy pocas muestras. Esos son los enlaces que se capturaron con un error puntual en alguna IP y que, consecuentemente, solo tienen 1 o 2 muestras.

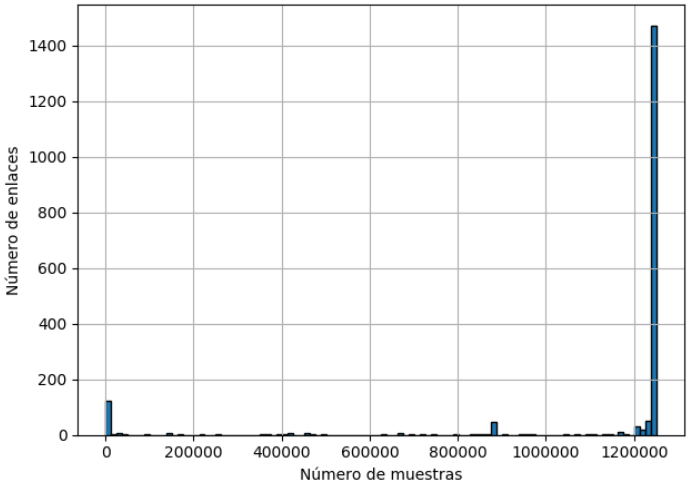


Figura 22. Histograma del número de muestras de los enlaces

Para filtrar los enlaces, se limitan el número de muestras mínimas a 900.000 muestras para que haya cierta igualdad el número de datos para los entrenamientos. Se toma este valor y no un número mayor porque los enlaces con mala LQ, que son los que nos interesan, son más propensos a perder sus TC (por la propia definición de LQ) y es normal que falte información de LQ, provocando menores números de muestras. El otro criterio de corte es el de la desviación típica mínima, de tal manera que se consigue eliminar los enlaces que no varíen lo suficiente para que tenga sentido hacer una predicción. De nada sirve hacer un sistema que estime siempre el mismo valor. En la Tabla 1 se muestra cuántos enlaces quedan con el criterio de muestras y distintas desviaciones típicas.

Desviación típica mínima	**Número de enlaces**
0.005	405
0.01	246
0.02	224
0.05	**167**
0.1	98

Tabla 1. Número de enlaces en función de la desviación típica mínima

Se decide por la desviación típica mínima de 0.05 que es un valor que deja enlaces bastante variables y con un número suficiente para que haya diversidad de ellos para su estudio. Con estas condiciones se obtienen los nuevos histogramas de medias de LQ y de desviación típica. En la Figura 23 y Figura 24, se muestran los histogramas y los diagramas de cajas de cada uno de ellos.

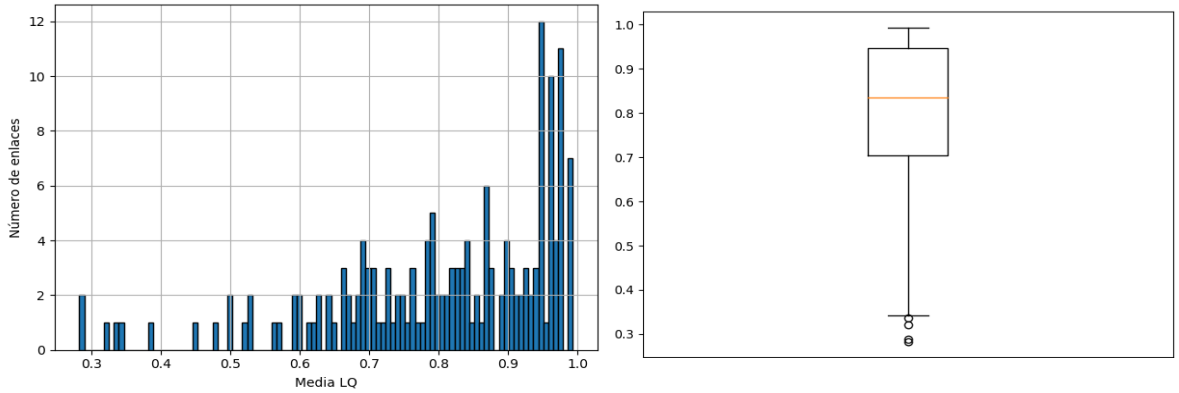


Figura 23. Histograma y diagrama de caja de las medias de LQ de los enlaces filtrados

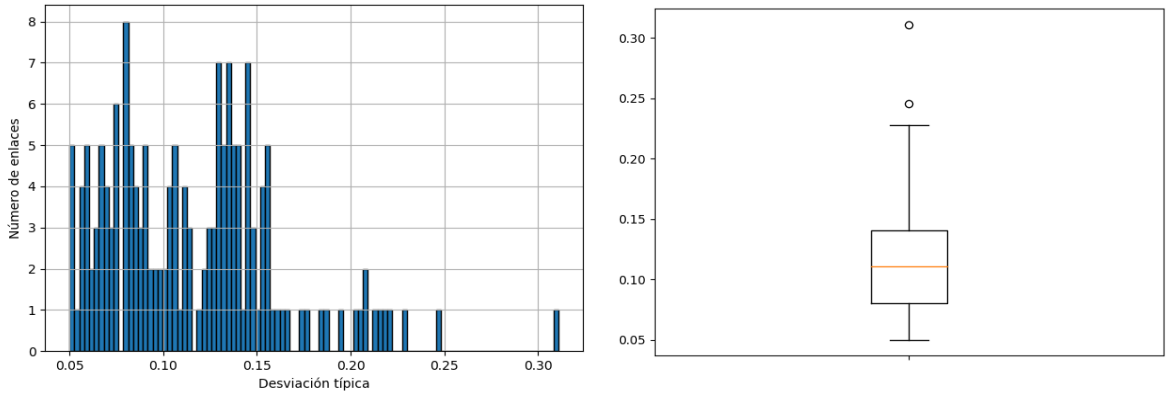


Figura 24. Histograma y diagrama de caja de las desviaciones típicas de LQ de los enlaces filtrados

En este nuevo conjunto de enlaces se observa que las medias de LQ se han distribuido bastante más que antes, aun así la mediana es cercana al 0.8. En cuanto a la desviación típica, se observa que el histograma se ha truncado respecto al de la Figura 21 en los 0.05. Lo mismo ocurre con el histograma del número de muestras en la Figura 25.

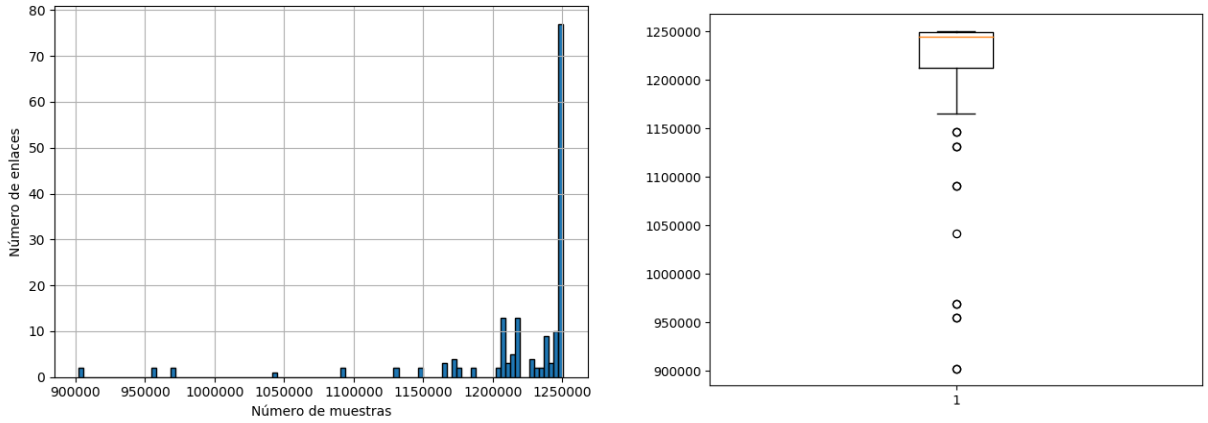


Figura 25. Histograma y diagrama de caja del número de muestras de LQ de los enlaces filtrados

En cuanto a la moda de cambios en LQ, se representa su histograma y diagrama de caja en la Figura 26. Es el histograma más importante y en él, vemos que destacan los cambios en 2 segundos, en 4 segundos (los mayoritarios) y los de 7/9 segundos.

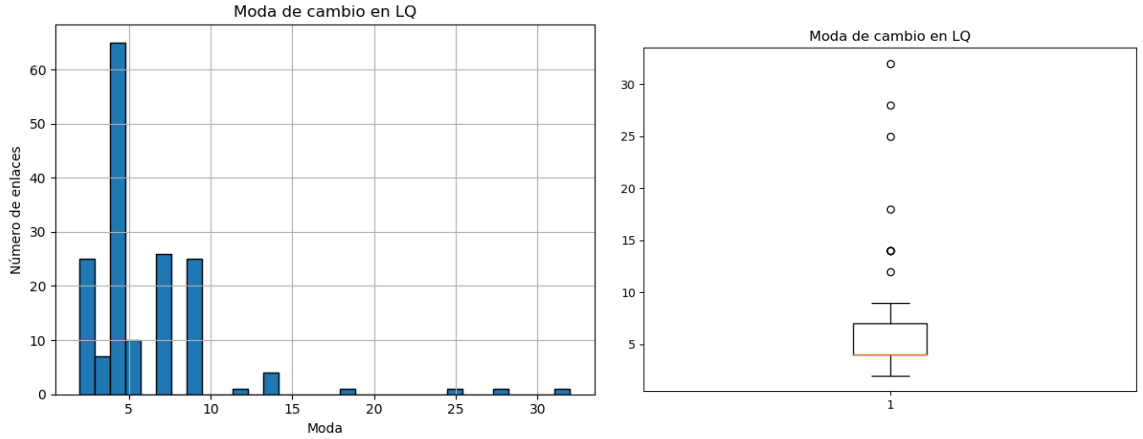


Figura 26. Histograma y diagrama de caja de las modas de cambios de LQ de los enlaces filtrados

La elección final elegida para las pruebas va a ser la de los 5 enlaces con mayor desviación típica de las 3 modas mayoritarias. La elección de solo estos enlaces se debe a que con este número se pretende poder caracterizar el resto, sin las limitaciones en tiempo que llevaría entrenar los 167 enlaces para cada prueba que se haga porque se observará que LSTM tarda bastante en realizar cada entrenamiento. Al ser los más variantes se pondrá a prueba al máximo la predicción de los modelos que se propongan. Consecuentemente, se van a hacer las pruebas con los 5 enlaces más variantes de los que tengan una actualización de LQ de 2 segundos, de 4 segundos y de 7 segundos. Los tres grupos de 5 enlaces van a representar a los enlaces cercanos, de media distancia[20] y lejanos, en concordancia con los efectos de *fish-eye*. Para hacer el entrenamiento y predicción cada enlace se va remuestrear según la moda de su grupo. De esta forma, los datos de LQ del 1º grupo son muestras de LQ tomadas cada 2s, las del 2º tomadas cada 4s y las del 3º cada 7s. Con estos grupos diferenciados de enlaces se podrá ver el rendimiento de las predicciones en función de la lejanía de los enlaces.

A continuación, se muestran las tablas de los 3 grupos de enlaces a los que se va a estudiar su rendimiento. Para cada uno de ellos se ha visto la distancia en saltos de cada nodo

[20] A los enlaces de media distancia se les llamará enlaces medianos

que lo forma. La distancia se ha puesto acorde el valor mínimo y el máximo de saltos de cualquiera de los nodos, ignorando valores de saltos que se mantuvieron muy poco tiempo.

Enlaces Cercanos/Moda 2s	Media LQ	Desviación típica	Distancia en saltos
10.12.3.11/10.12.3.8	0.3207	0.1754	4/5
10.12.2.98/10.12.240.50	0.8428	0.1479	4/5
10.12.208.22/10.12.208.29	0.9592	0.1457	2/3
10.12.208.22/10.12.208.19	0.9592	0.1455	2/3
10.12.208.22/10.12.208.18	0.9591	0.1454	2/3

Tabla 2. Caracterización de los enlaces cercanos

Enlaces Medianos/Moda 4s	Media LQ	Desviación típica	Distancia en saltos
10.12.4.221/10.12.3.199	0.6598	0.2209	9/11
10.12.91.186/10.12.91.185	0.8688	0.2184	8/12
10.12.2.241/10.12.240.60	0.6386	0.2008	4/5
10.12.208.22/10.12.208.28	0.9379	0.1879	2/4
10.12.2.40/10.12.4.45	0.7575	0.1739	6/7

Tabla 3. Caracterización de los enlaces medianos

Enlaces Lejanos/Moda 7s	Media LQ	Desviación típica	Distancia en saltos
10.12.3.199/10.12.4.221	0.5909	0.3109	9/14
10.12.2.103/10.12.4.160	0.6812	0.2458	11/12
10.12.4.56/10.12.4.160	0.7920	0.2277	10/12
10.12.3.8/10.12.1.48	0.5266	0.2078	8/10
10.12.4.160/10.12.2.103	0.7300	0.1957	11/12

Tabla 4. Caracterización de los enlaces lejanos

Tres de los enlaces cercanos de la Tabla 2, sí que están a 2 saltos y por lo tanto es lógica la llegada de TC cada 2 segundos. Y de los otros dos enlaces que están a 4 saltos de distancia es probable que esté ocurriendo la llegada de TC desfasados dando lugar a la actualización cada 2 segundos también (llegada cada 4s de TC de los dos nodos generadores de TC que forman el enlace, pero las transmisiones pueden estar desfasadas 2s, provocando actualización de LQ cada 2s). Con los de media distancia algo parecido: se estimó que para la llegada de TC cada 4s los nodos debían estar entre 3 y 8 saltos. En cambio, solo tres enlaces

lo cumplen y hay dos con saltos mayores por el mismo posible motivo anterior. Con los lejanos, que se estimó llegadas cada 8s (menores por el *jitter*), sí se cumple que todos los enlaces están a esa distancia o más. Hay que tener en cuenta las limitaciones de este conjunto de datos final para el estudio. La primera es que es un número muy reducido de enlaces y los resultados pueden no ser del todo generales, y la segunda es la de muestrearlos según la moda de la frecuencia de cambios LQ, que como hemos visto, no es fija al haber valores alrededor de ésta.

3.7 Conclusiones

En trabajos previos de predicción de LQ se han obtenido datos de LQ a través de tablas de topología muestreadas en la red de *Funkfeuer*. Estos trabajos empezaron con la obtención de estas tablas con muestras cada 5 minutos. Al ser un tiempo excesivo en un contexto práctico de predicción para la actualización de las tablas de encaminamiento, se realizó una nueva captura de tablas de topología muestreadas cada segundo. El problema de estas tablas viene por el hecho de no saber el tiempo verdadero de actualización de cada LQ, crítico a la hora de entrenar modelos de aprendizaje automático. Este tiempo es importante porque, tras ver la configuración de OLSRd de la red, ésta tenía activado el mecanismo de *fish-eye*. Este mecanismo, sumado a otros efectos del protocolo como el del tiempo de *jitter*, los cambios de saltos o la actualización de LQ por dos nodos, hace que el tiempo entre actualizaciones de LQ pueda variar según el enlace.

Como consecuencia del *fish-eye*, la frecuencia de llegadas de TC, que contienen los LQ, cambia según la distancia en saltos de los nodos que informan de los LQ de un enlace. Los nodos cercanos informan de los valores LQ con mayor frecuencia que los nodos lejanos. Con los datos muestreados cada segundo se observó con histogramas que este efecto existía, con tiempos entre cambios de LQ cercanos a los esperados y con valores variables en torno a ese tiempo por consecuencia de efectos como el *jitter* de OLSR. Además, se realizó una captura de tráfico OLSR en la que tras su análisis, se cumplían los efectos comentados. Esta captura no se usará para los experimentos debido a que cuenta con solo dos días de datos y que se logró realizar con el trabajo muy avanzado, sin poder procesar los datos de LQ. Para los experimentos se usarán enlaces del muestreo de un segundo realizado anteriormente (de dos semanas de duración), de tal forma que se emplearán los 5 enlaces con mayor desviación típica de las 3 modas de tiempos entre cambios mayoritarias. Pese a la limitación del número de enlaces, se espera poder caracterizar los enlaces según la distancia que se encuentren y por tanto, su frecuencia de actualización de LQ.

Capítulo 4. Introducción a LSTM y herramientas empleadas

4.1 Introducción

Uno de los objetivos de este Trabajo Fin de Grado es el de emplear algoritmos de aprendizaje profundo (DL, *Deep Learning*) para la predicción de LQ. A continuación se tratará de explicar brevemente que se entiende por *Deep Learning*. El *Deep Learning* se encuentra dentro del campo del aprendizaje automático (ML, *Machine Learning*) y en el que ambos se engloban en el campo de la Inteligencia Artificial (IA), como muestra la Figura 27 [38].

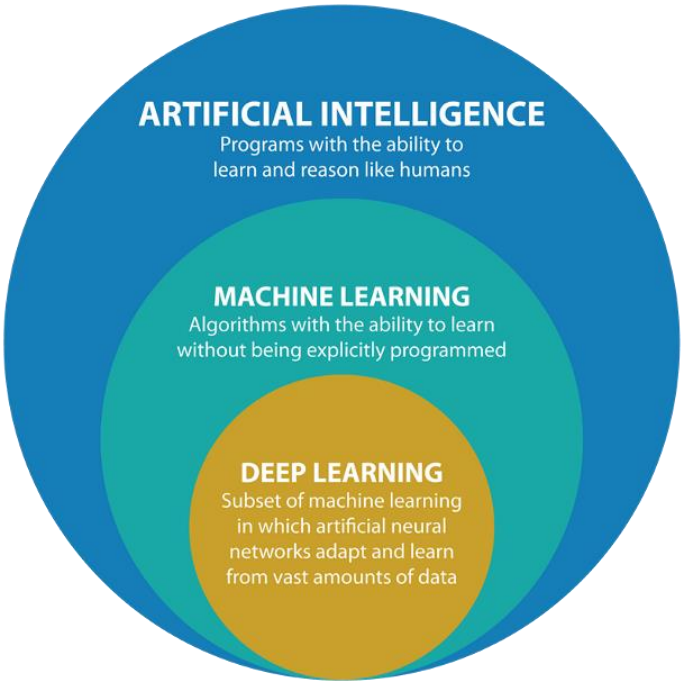


Figura 27. Clasificación del Deep Learning dentro de la Inteligencia Artificial [39]

La Inteligencia Artificial tiene muchas definiciones. La más certera tecnológicamente hablando, es la que la define como la rama de la informática que trata el estudio del comportamiento inteligente en los ordenadores. Este comportamiento guarda similitudes con capacidades humanas como el de aprender de la experiencia, razonar, tomar decisiones o corregirse [38]. El *Machine Learning* es un subconjunto de técnicas de la Inteligencia Artificial, nacido de la necesidad de que los sistemas de IA fuesen capaces de conseguir su propio conocimiento, extrayendo patrones y aprendiendo de los propios datos [38]. Los algoritmos de *Machine Learning* permitieron a los ordenadores afrontar problemas relacionados con el mundo real y tomar decisiones que parecen subjetivas. El rendimiento de estos algoritmos depende mucho de las "representación" de los datos, siendo cada tipo de información una característica (*feature*). El problema viene de que en algoritmos simples de ML no está clara la extracción de características sencillas a partir de datos más complejos, pudiendo ser éstos una imagen formada por miles de píxeles, por ejemplo [40].

Las técnicas convencionales de ML estaban muy limitadas en su habilidad para

procesar datos naturales en su forma "cruda" (por ejemplo, píxeles de imágenes u ondas de sonido). Durante mucho tiempo se necesitaron técnicas muy complicadas para transformar los datos crudos en una representación interna para el algoritmo de ML, normalmente un detector o clasificador de entradas [41]. El *Deep Learning* soluciona este problema permitiendo al sistema construir conceptos complejos a partir de conceptos más simples. Por ejemplo, el de representar el concepto de una imagen de una persona, a partir de combinar conceptos más simples como contornos o esquinas, a su vez compuestos por bordes [40]. Las técnicas de *Deep Learning* suelen necesitar muchos más datos para entrenarse que las técnicas clásicas de ML.

Todas estas técnicas se basan en el entrenamiento, pudiendo ser de los siguientes tipos: aprendizaje supervisado, no supervisado o por refuerzo [42]. Este trabajo hará uso de una técnica de aprendizaje supervisado que además es el más común de los aprendizajes de ML, independientemente sean de DL o no. Este aprendizaje se basa en entrenar a un modelo a partir de un conjunto de datos de los cuales se conocen tanto las entradas como las salidas. Sabiendo las salidas correctas para unas entradas dadas, durante el entrenamiento trata de ajustar una serie de pesos que minimicen el error entre su salida y la salida verdadera. El error está definido mediante una función de coste, al que se debe buscar el mínimo con la existencia de diferentes técnicas.

En este capítulo, en la sección 4.2 se hablará de las redes neuronales que existen dentro del aprendizaje profundo. Seguidamente, en la sección 4.3 se explica LSTM, el algoritmo de redes neuronales recurrentes que se evaluará en este trabajo y en la 4.4 se detallará el uso de LSTM para secuencias temporales de LQ. En las dos siguientes secciones se indicarán las herramientas software con las que se han implementado los modelos LSTM, así como el equipo hardware que se ha usado. Por último, en la sección 4.7 se extraen las conclusiones de este capítulo.

4.2 Redes neuronales en aprendizaje profundo

Una de las técnicas más importantes tanto de *Machine Learning* como de *Deep Learning* es la de las redes neuronales (NN, *Neural Networks*). Las redes neuronales son imitaciones simples de cómo funcionan nuestras neuronas o nuestro cerebro y con el aumento de procesamiento en los ordenadores actuales han tenido una mayor importancia [42]. Estas redes se basan en una serie de unidades logísticas llamadas neuronas o neuronas artificiales. En la Figura 28 se indica su funcionamiento con un diagrama. Se basan en que a los valores de unas entradas se les multiplica por unos pesos, esos valores se suman entre sí (más un valor de sesgo o *bias*) y produce una salida tras pasar una función de activación.

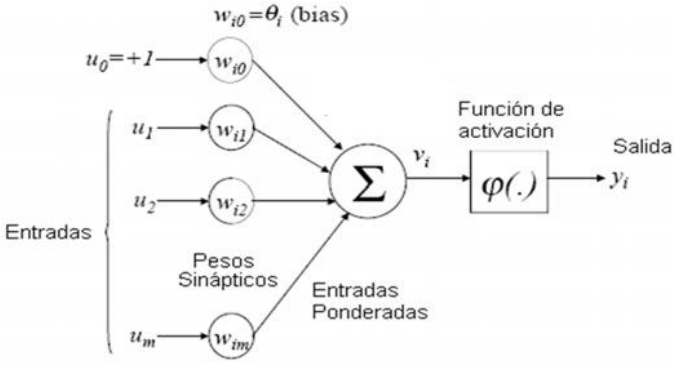


Figura 28. Diagrama del funcionamiento de una neurona [43]

La función de activación sirve para poder dotar de no linealidad a una red neuronal [42]. De esta manera, las neuronas producen una transformación no lineal de las entradas ponderadas. Hay distintos tipos de funciones de activación, como la sigmoide que acota los valores entre 0 y 1 o la tangente hiperbólica de -1 a 1. Las hay también no acotadas como la *ReLU* (*Rectified Lineal Unit*) o *Leaky ReLU*, o la *softmax* para representaciones en forma de probabilidades.

Estas neuronas se unen en capas formando la red neuronal completa (ver Figura 29). En esta red las entradas forman la capa de entrada y la salida o salidas formarían la capa de salida. Todas las neuronas dispuestas entre estas dos capas forman las capas ocultas o intermedias [42]. Como se puede ver en esta figura, la diferencia para considerar a una NN como técnica de aprendizaje profundo depende del número de capas y neuronas que tenga.

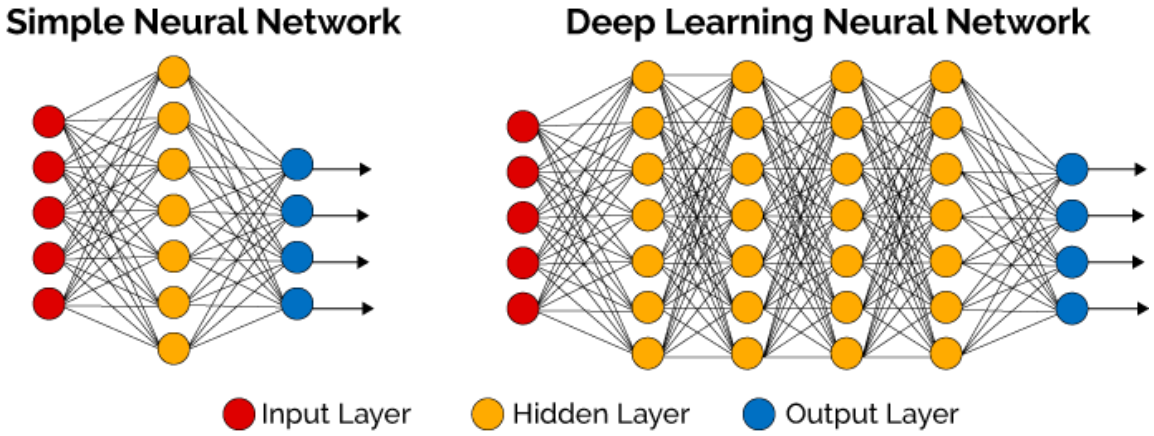


Figura 29. A la izquierda una red neuronal simple. A la derecha una red neuronal con más capas ocultas de DL [44]

Existen distintos tipos de redes neuronales, como el **perceptrón multicapa** (MLP, *Multilayer Perceptron*), las convolucionales o las recurrentes. El **MPL** es el que hemos explicado hasta ahora y el mayor ejemplo de *Deep Learning* cuando existen muchas capas o neuronas, sirviendo de base para el resto. Para ajustar los pesos de todas las neuronas es necesario calcular un vector de gradientes que, para cada peso, indique la cantidad de error que se incrementaría o decrementaría si el peso variase una pequeña cantidad. Después, el vector de pesos se ajusta en la dirección opuesta al gradiente. Lo que se busca es minimizar la función de coste, que representa el error entre las salidas de la red y las verdaderas. En la práctica, las redes neuronales usan el descenso del gradiente estocástico (SGD, *Stochastic Gradient Descent*). Está técnica computa la función de coste con solo unas pocas entradas, en vez de todo el conjunto de datos (muy grande en *Deep Learning*). Este método hace que poco a poco con muestras pequeñas se acerque al mínimo de la función. Para calcular estos gradientes se hace uso de la propagación hacia atrás (*backpropagation*). Este mecanismo no es nada más ni nada menos que la aplicación continua de la regla de la cadena de las derivadas. Su nombre viene por el hecho de que, ahora, el error "se propaga" de las últimas capas a las primeras [41].

Otro tipo de redes son las **redes neuronales convolucionales** (CNN, *Convolutional Neural Networks*). Son redes neuronales especializadas para procesar datos con topología cuadriculada. Algunos ejemplos son el de series temporales, vistas como cuadriculas 1D, tomando muestras en intervalos temporales fijos o el de datos de imágenes, vistas como cuadriculas 2D de píxeles. Han tenido mucho éxito en aplicaciones prácticas y su nombre viene por el hecho de que utilizan la operación matemática de la convolución. Estas redes son como las redes neuronales simples en las que se usa una convolución en lugar de una

multiplicación matricial en al menos una de sus capas [40].

El último de los tipos y el que más nos interesa para este trabajo es el de las **redes neuronales recurrentes** (RNN, *Recurrent Neural Networks*). Estas redes se especializan en los datos secuenciales como el reconocimiento de la voz, generación de música, análisis de secuencias de ADN, traducción de idiomas, etc. Las RNN procesan una secuencia de entrada en un determinado momento, manteniendo en sus neuronas ocultas un vector de estado con información de todos los elementos pasados de la secuencia. Considerando las salidas de las neuronas de un tiempo discreto distinto como salidas de otras neuronas de una red multicapa profunda, se puede aplicar un tipo de *backpropagation* para entrenar estas redes [41]. En los otros tipos de redes, el tiempo entre salidas y entradas en los que ocurría un evento no tenía significado, la única forma era codificar las entradas para que de algún modo la representación de los datos incluyese datos pasados. Comparándola con una MLP, los pesos de una RNN están compartidos a través de distintas instancias de la red neuronal, cada una asociada con un tiempo diferente [40].

Las RNN deben tener un tipo especial de neuronas. Estas neuronas reciben información de la entrada actual a la red (como en las MLP) y también valores de las neuronas ocultas del estado anterior de la red. Siendo la entrada en el instante t, $\mathbf{x}^{(t)}$ y las salidas de la capa oculta en ese instante, $\mathbf{h}^{(t)}$. La salida $\hat{\mathbf{y}}^{(\mathbf{t})}$ en el instante t será función del valor del vector $\mathbf{h}^{(t)}$, que se calcula según la fórmula (1), donde se aplican los pesos W^{hx} a la entrada actual y los pesos W^{hh} al estado interno del instante pasado, se les suma con el vector de *bias* y aplica una función *f* de activación. Debido a esto, la salida $\hat{\mathbf{y}}^{(t)}$, definida en la fórmula (2) puede depender de otros valores de entrada pasados ($\mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \ldots$)[45].

$$\mathbf{h}^{(t)} = f(W^{hx}\boldsymbol{x}^{(t)} + W^{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h) \qquad (1)$$

$$\hat{\mathbf{y}}^{(t)} = f(W^{yh}\mathbf{h}^{(t)} + \mathbf{b}_y) \qquad (2)$$

Desde este punto de vista, las neuronas recurrentes pueden ser vistas como neuronas con una salida recursiva hacia ella misma y otras neuronas. Y a su vez, se puede interpretar como la unión de salidas de otras neuronas de una red multicapa. A esta vista se conoce como *unfold*ed, es decir, desenrolladas temporalmente. Estas distintas formas de ver a las neuronas recurrentes se pueden apreciar en la Figura 30.

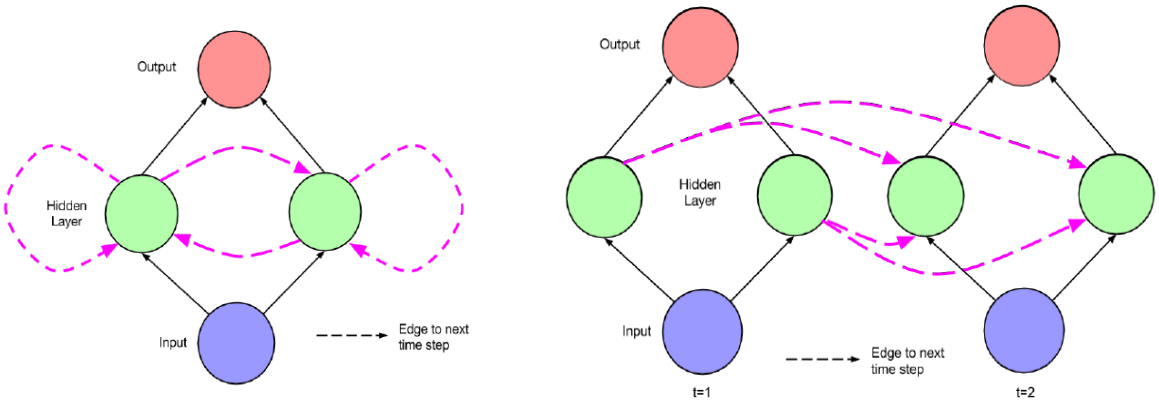


Figura 30. A la izquierda, grafo de una neurona recurrente con realimentación. A la derecha, grafo unfolded o desenrollado temporalmente [45]

Del grafo desenrollado en tiempo, automáticamente se puede pensar en la idea de *backpropagation*, en este caso haciendo *forward* en el tiempo para calcular las salidas y pérdidas y *backward* en el tiempo para el cálculo de gradientes. De la misma forma que se

usa el algoritmo de *backpropagation,* se puede hacer el algoritmo *Backpropagation Through Time* (BPTT) [40]. Aun así, el entrenamiento de estas redes no es fácil por dos problemas conocidos al realizar el BPTT, el del desvanecimiento del gradiente (*vanishing gradient*) y el del gradiente explosivo (*exploding gradient*). Estos efectos se generan para instantes temporales grandes, donde las contribuciones a la entrada harán que los valores exploten a valores muy grandes o tiendan a cero, debido a que los pesos de las neuronas ocultas son siempre iguales. Esto provoca que el error de la derivada respecto a la entrada sea enorme o desaparezca [45]. Una solución es la de [46] con el uso del *Truncanted Backpropagation Through Time* (TBPTT), que se basa en limitar un cierto número de instantes temporales la propagación del error. Otra solución es la propuesta por [13] con el uso de *Long Short-Term Memory* (LSTM) que es la técnica usada en este trabajo.

Otro tipo de redes neuronales recurrentes son las bidireccionales (BRNN, *Bidirectional RNN*), estas redes no solo toman información del pasado sino también de valores futuros. Se usan sobre todo para secuencias dónde la salida pueda depender, no solo de los valores pasados, sino de la secuencia completa. Entre sus usos, destacan el de técnicas de reconocimiento de voz por los fonemas o en las traducciones de idiomas. Se pueden implementar con el uso de dos RNN con sentidos contrarios. En esta arquitectura, hay dos capas ocultas en las que las recurrencias se hacen en dos sentidos, como se muestra en la Figura 30.

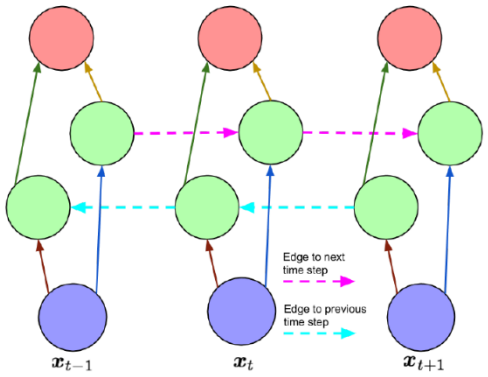


Figura 31. Grafo unfolded de las neuronas de una BRNN [45]

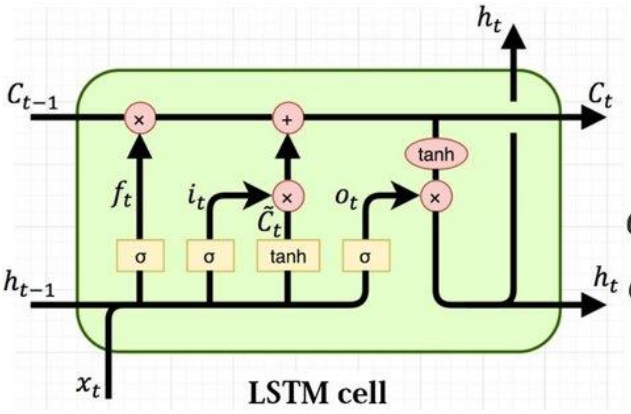
4.3 LSTM (Long Short-Term Memory)

Las redes neuronales recurrentes están especializas en el tratamiento de secuencias temporales por lo que pueden llegar a ser idóneas para la predicción de valores de LQ. Asimismo, las RNN sufren de lo que se conoce como memoria a corto plazo. Esto significa que no transportan bien la información asociada a las primeras entradas hacia las últimas; es una de las consecuencias de uno de los problemas comentados, el del desvanecimiento del gradiente: al hacer cero el gradiente, no se contribuye al aprendizaje del modelo.

El algoritmo de *Long Short-Term Memory* (LSTM) propuesto por [13] es un tipo de RNN que corrige el problema de la memoria a corto plazo. Este algoritmo se basa en el uso de puertas para permitir o denegar el flujo de información. Otro algoritmo más simple basado en puertas es el *Gated Recurrent Unit* (GRU). Con el uso de estas puertas, se consigue que aprendan que datos de la secuencia son importantes mantener u olvidar. Esto provoca que los datos importantes tengan peso en predicciones, aunque éstos sean muy lejanos. Las redes LSTM han demostrado ser más efectivas que las RNN convencionales, convirtiéndolas en el estado del arte en muchas disciplinas, por ejemplo, en traducción con el uso del *encoder-*

decoder que se explicará después [41].

Este modelo crea una red neural recurrente usando celdas LSTM *o memory cells*, en vez de las neuronas recurrentes vistas hasta ahora. La compleja estructura de una celda LSTM y sus fórmulas se muestran en la Figura 32.



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figura 32. Estructura de una celda LSTM con las fórmulas asociadas [47]

La idea de funcionamiento se encuentra en el concepto de *Cell state* (la C de la figura) y sus numerosas puertas. Este estado puede considerarse la memoria de la red y la que permite transferir la información importante a otras etapas. A medida que avanza en las instancias temporales de la red, la información se añade o elimina con las puertas asociadas. De la Figura 32, se puede diferenciar las siguientes puertas [45]:

- ***Forget gate***: es la puerta que decide qué información se mantiene o no. A la información de entrada junto a la salida del estado oculto anterior, se le aplican unos pesos y se usa la función de activación sigmoide. Tras ello, el valor f_t entre 0 (olvidar) o 1 (mantener todo), se multiplica al *Cell state*.
- ***Input gate***: esta puerta es la que puede añadir información al *Cell state*. Para ello se aplica una sigmoide a la entrada y estado previo, para decidir lo importante (de 0 a 1) que es esa nueva entrada. Al multiplicarlo por la salida de la tangente hiperbólica se genera lo que se va a añadir de información a la *Cell State*.
- ***Output gate***: finalmente, el valor producido por la celda es el valor de la tangente hiperbólica del *Cell state* actualizado ($\boldsymbol{C}_t$) multiplicado por el valor del output *gate,* $\boldsymbol{o}_t$. El $\boldsymbol{o}_t$ consiste, de nuevo, en la aplicación de la sigmoide a la suma de la entrada y el estado previo con unos pesos.

Uno de los usos de LSTM es el de usarse para un modelo *encoder-decoder*, también llamado *sequence-to-sequence* o *autoencoder LSTM*. La filosofía es la siguiente: se usa un codificador (*encoder*) RNN que lee la entrada y genera una salida, esta salida entra en un decodificador (*decoder*) RNN que genera la secuencia de salida. El estado oculto final del *encoder* suele ser una variable de tamaño fijo que representa el resumen semántico de la secuencia de entrada y con el cual se va a determinar su salida con el *decoder*. Este modelo se usa en muchas aplicaciones prácticas, tales como la de la traducción automática o reconocimiento de voz. Se suele llamar a la entrada el "contexto" y lo que se busca es una representación del contexto como un vector o conjunto de vectores que resuman la secuencia de entrada [40]. Un ejemplo de la filosofía seguida se muestra en la Figura 33.

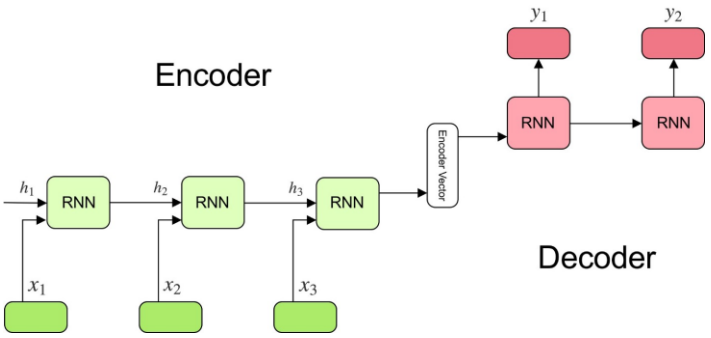


Figura 33. Ejemplo de diagrama unfolded de un encoder-decoder [48]

4.4 LSTM para la predicción de LQ

Existen diversas formas de emplear LSTM para la predicción de secuencias temporales de valores de LQ. Se puede usar un *encoder-decoder LSTM*, un modelo de una capa oculta con celdas LSTM o incluir varias neuronas de salida a LSTM para la predicción a varios instantes vista, etc. De la misma manera, estos modelos se pueden entrenar para cada enlace o para varios, y constar tanto de una única entrada y salida como de varias entradas y salidas de valores LQ de distintos enlaces. Muchos de los modelos anteriormente comentados serán evaluados en el siguiente capítulo, pero antes de nada conviene reforzar la idea del funcionamiento de LSTM con un ejemplo. Un modelo LSTM con una entrada (*feature*) de valores LQ de un enlace y de una salida con el valor LQ siguiente, predice de la siguiente manera: tras una serie de entradas LQ (por su única entrada) predice la LQ siguiente. Por ejemplo, entra un LQ, este valor cambia el estado interno de las celdas de la red, después se introduce por la entrada el siguiente valor LQ y con este último valor y el estado interno se realiza una predicción. Si la predicción se hace después de *N* valores LQ, la ventana temporal es de tamaño *N*. La ventana temporal indica los valores que se van a usar para una predicción de LQ, esta ventana se desplaza una unidad con cada predicción en el conjunto de datos.

La filosofía principal usada en este trabajo va a ser la de crear un modelo LSTM para cada uno de los 15 enlaces individuales, con el fin de que sea capaz de aprender según las características concretas de cada enlace. Es decir, cada nueva arquitectura de LSTM planteada será evaluada para cada enlace, mostrando los resultados de los enlaces agrupados según si son cercanos, medianos o lejanos. Consecuentemente, habrá un modelo LSTM entrenado para cada uno de los enlaces de la red y dependiendo de la arquitectura concreta se podrán crear modelos que predigan a un instante vista o a varios. La justificación de este planteamiento es que, dependiendo del enlace, se esperan comportamientos distintos. Algunos pueden variar con un determinado patrón que poco o nada pueda parecerse al de otro enlace. Con el entrenamiento de un modelo a partir de todos los enlaces habría problemas en los enlaces más particulares, ya que las redes neuronales tienden a aprender lo común en detrimento de lo específico. Además, si hay enlaces que varían de forma similar, los modelos LSTM por enlace funcionarían también correctamente para ellos. Sin embargo, también se entrenará un LSTM que tendrá varias salidas y entradas con la finalidad de predecir la LQ de varios enlaces simultáneamente.

A la hora de entrenar un modelo, se van a remuestrear los valores de LQ obtenidos del conjunto de datos muestreados cada segundo según la frecuencia de actualización (cada 2, 4 o 7 segundos). Esto hará que la predicción LSTM del siguiente instante vista de una secuencia dada sea el valor LQ futuro que se esperaría en 2 segundos en el caso de los enlaces

cercanos, en 4 segundos para los medianos y en 7 segundos para los lejanos. Tras obtener los LQ remuestreados según la tasa de frecuencia de LQ del enlace, se usan la mitad de los datos para el conjunto de entrenamiento y la otra mitad para el test. Esto se traduce en que se tomarán datos de unos 7 días para el entrenamiento y de otros 7 días para la evaluación del error de predicción. Para el entrenamiento y test se va a usar una ventana temporal, de tal forma que para una predicción de valor de LQ se van a usar el mismo número de valores pasados de LQ que el tamaño de la ventana. Por ejemplo, en una secuencia de 20 valores LQ y una ventana de tamaño 7, el conjunto de entrenamiento/test serían 13 vectores de 7 valores LQ asociado cada uno con su salida verdadera. Las posiciones de los valores LQ de este ejemplo se muestran en la Figura 34, arriba, los vectores que formarían el conjunto de entradas al modelo y abajo, el vector con la posición de los LQ de las salidas esperadas.

```
[[ 1.,  2.,  3.,  4.,  5.,  6.,  7.],
 [ 2.,  3.,  4.,  5.,  6.,  7.,  8.],
 [ 3.,  4.,  5.,  6.,  7.,  8.,  9.],
 [ 4.,  5.,  6.,  7.,  8.,  9., 10.],
 [ 5.,  6.,  7.,  8.,  9., 10., 11.],
 [ 6.,  7.,  8.,  9., 10., 11., 12.],
 [ 7.,  8.,  9., 10., 11., 12., 13.],
 [ 8.,  9., 10., 11., 12., 13., 14.],
 [ 9., 10., 11., 12., 13., 14., 15.],
 [10., 11., 12., 13., 14., 15., 16.],
 [11., 12., 13., 14., 15., 16., 17.],
 [12., 13., 14., 15., 16., 17., 18.],
 [13., 14., 15., 16., 17., 18., 19.]]),
[ 8.,  9., 10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.]
```

Figura 34. Ejemplo de una ventana temporal de 7 elementos con una secuencia de 20 valores

4.5 Software empleado

Para implementar LSTM se usó Python y sus paquetes especializados para aprendizaje automático y profundo haciendo uso de la GPU. A continuación, se explica el software empleado, desde el lenguaje de programación hasta el entorno de desarrollado usado en el trabajo.

Python

El lenguaje de programación empleado en este Trabajo Fin de Grado, tanto para el procesamiento de datos como el uso de técnicas *Deep Learning,* fue Python. Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, datos dinámicos de muy alto nivel, así como clases. A pesar de ser un lenguaje orientado a objetos, soporta otros tipos de paradigmas de programación como la programación imperativa o funcional. Diseñado por Guido van Rossum, Python combina una notable potencia con una sintaxis muy clara. Tiene una licencia de código abierto pudiendo ejecutarse en variantes *Unix*, como *Linux* o *MacOS*, y en *Windows* [49].

El hecho de decantarse por Python para este trabajo viene dado por su amplio uso en la computación científica y numérica, al contar con bibliotecas como *NumPy*, *SciPy* o *Pandas* y otras específicas para el ML. La versión de Python usada en este trabajo fue la 3.7.6.

Keras

Para los modelos *Deep Learning* desarrollados en este trabajo se usó la API de *Deep Learning* de *Keras*. *Keras*[21] es una API para la creación de redes neuronales de código abierto

[21] https://keras.io/

en Python, permitiendo crear de una forma más sencilla modelos de redes que con *TensorFlow*. *TensorFlow*[22], desarrollada por Google, es una biblioteca, también de código abierto, basada en computación de grafos flujos de datos y ampliamente utilizada en el mundo del aprendizaje automático por su gran eficiencia. *TensorFlow* te permite construir la arquitectura de redes neuronales a nivel de operaciones. *Keras* es una API de alto nivel que te permite construir una red neuronal por capas, siendo capaz de ejecutarse sobre *TensorFlow*. Se puede decir que *Keras* es la interfaz de alto nivel de *TensorFlow*, pasando a estar integrada en uno de sus módulos, `tensorflow.keras`. Esto aporta una interfaz sencilla, a nivel de capas, de todas las opciones que aporta *TensorFlow*, haciendo que la creación de modelos sea más fácil y rápida de aprender.

Otra biblioteca para aprendizaje automático usada en el trabajo es *Scikit-learn*[23]. Esta última no se usó para la creación de modelos de redes, pero se usaron algunas de sus funciones para evaluar el rendimiento de esos modelos. Las versiones usadas para estos tres componentes se encuentran en la Figura 35.

Anaconda

Para la gestión de versiones y paquetes de Python se hizo uso de *Anaconda*[24], la versión 5.3.1. *Anaconda* es una plataforma de distribución de Python de código abierto ampliamente usada para realizar ciencia de datos o aprendizaje automático en una máquina. Se pueden encontrar e instalar más de 7500 paquetes Python en su repositorio en la nube, a través del comando `conda-install`. Se basa en los entornos (*environments*) que pueden ser mantenidos, actualizados y ejecutados de forma separada sin interferencia entre ellos [50]. También permite hacer duplicados de entornos de forma cómoda.

Está aplicación puede usarse con una interfaz gráfica o a través de comandos. Para este trabajo se creó un entorno con los paquetes Python específicos para hacer uso de la GPU (y no de la CPU) en técnicas de aprendizaje automático. En la Figura 35, se muestra parte de los paquetes instalados y sus versiones con la salida del comando `conda list -n gpuQuique`. Se han incluido solo algunos paquetes importantes para la creación de modelos de aprendizaje profundo, como K*eras* o *TensorFlow*.

```
# packages in environment at /home/ediezb/.conda/envs/gpuQuique:
# Name                    Version                   Build   Channel
_tflow_select             2.1.0                       gpu
keras                     2.2.4                         0   anaconda
keras-base                2.2.4                    py37_0
matplotlib                3.1.1            py37h5429711_0
numpy                     1.17.4           py37hc1035e2_0
pandas                    0.25.3           py37he6710b0_0
python                    3.7.6                h0371630_2
scikit-learn              0.22             py37hd81dba3_0
scipy                     1.3.2            py37h7c811a0_0
tensorflow                2.0.0          gpu_py37h768510d_0    anaconda
tensorflow-base           2.0.0          gpu_py37h0ec5d1f_0    anaconda
tensorflow-estimator      2.0.1                    pypi_0    pypi
tensorflow-gpu            2.0.0                    pypi_0    pypi
```

Figura 35. Paquetes Python importantes instalados en el entorno de Anaconda

PyCharm

El entorno de desarrollo usado en este trabajo es la versión de *PyCharm* 2019.3.3 (*Professional Edition*). *PyCharm*[25] es un IDE de programación para Python desarrollado por la empresa *JetBrains*. El uso de este entorno estuvo motivado por sus numerosas opciones

[22] https://www.tensorflow.org/
[23] https://scikit-learn.org/
[24] https://www.anaconda.com/
[25] https://www.jetbrains.com/es-es/pycharm/

de depuración, gestión de proyectos y consola interactiva con soporte de *Anaconda*. También se hizo uso de utilidades que tiene con bibliotecas como *NumPy* y *matplotlib*. Aun con todas estas funcionalidades, la principal fue la del uso del desarrollo remoto que dispone la versión profesional (obtenida gratuitamente por ser alumno). Con ello se pudo realizar desde un portátil personal una conexión SSH al servidor que guardaba los datos LQ y el entorno de *Anaconda*, ejecutándose el código generado en el servidor. Además, desde el propio IDE se pueden realizar transferencias SFTP entre el ordenador local y el servidor que ejecutaba el código. Una captura de este entorno de desarrollo con el proyecto usado en este trabajo se muestra en la Figura 36.

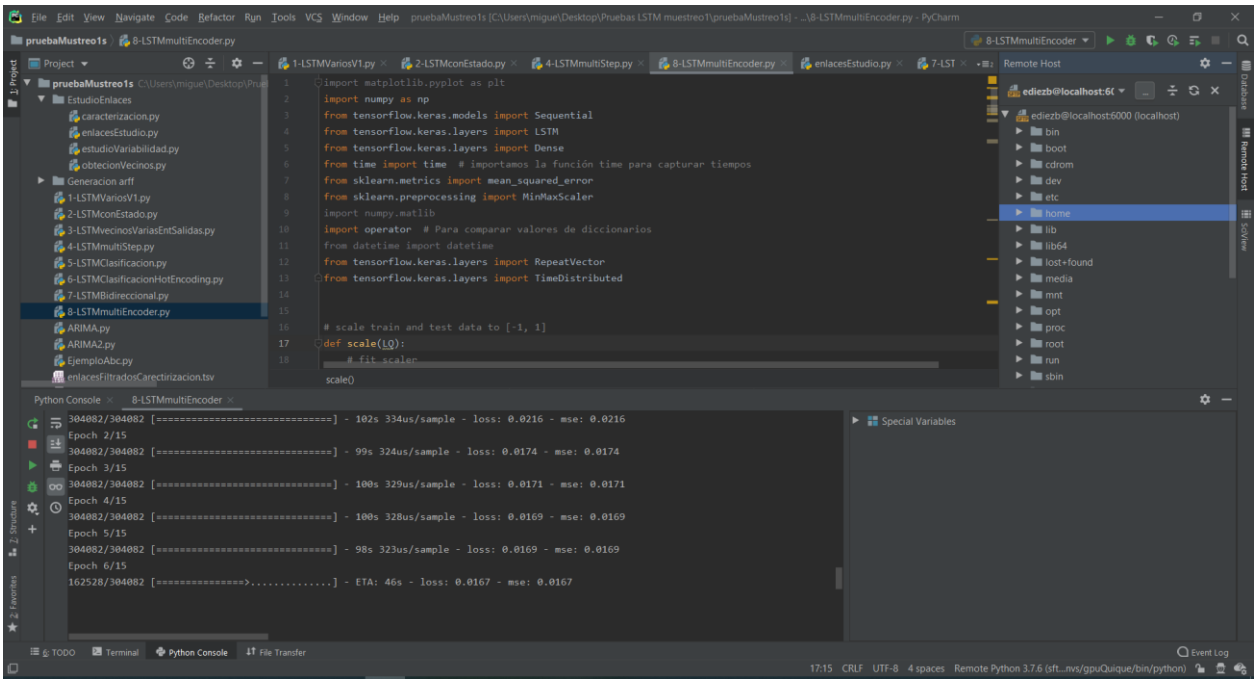


Figura 36. Captura del proyecto PyCharm que se ha usado para este trabajo

4.6 Hardware empleado

Todo el software comentado ha tenido que funcionar en un equipo hardware, del que se indican sus características a continuación. Para las pruebas de predicción LSTM se ha hecho uso de un ordenador que disponía el grupo de investigación de la Universidad de Valladolid con el que se realizó este trabajo, GSIC/EMIC[26]. Pudiéndose conectar a éste a través de una conexión SSH. En la Figura 37 se encuentran las características hardware del ordenador empleado.

[26] https://www.gsic.uva.es/

HPE PROLIANT DL380 GEN 10	1,00
HPe Intel Xeon Silver 4210 caché de 13,75 M 2,20Gz	1,00
HPE 32GB 2RX4 PC4-2933Y-R SMART KIT	3,00
HPE 800W Flex Slot Platinum Hot Plug	1,00
HPE 1.8TB SAS 2.5" 10K SFF SC 512e DS HDD	2,00
HP HPE DL38X GEN10 HIGH PERF FAN	1,00
HPE DL380 GEN10 HIGH PERF HEATSINK	1,00
GRAFICA NVIDIA SERVER RTX 5000 16GB DDR6	2,00
(2) Intel Xeon S4210 10 core	
128 GB (4 x 32GB) PC4-2933Y DDR4 RDIMM	
Smart Array P408i-a SR	
(2)HPE 1.8TB 12G 10k rpm HPL SAS SFF (2.5in)	
(2) GPU NVIDIA RTX5000	
HPE DL38x Gen10 High Performance Fan Kit	
Doble fuente 800W Flex Slot Platinum Hot Plug Low	
HPE Ethernet 10Gb 2P 530T Adapter	

Figura 37. Hardware empleado para los entrenamientos de técnicas LSTM

Este ordenador destaca por sus dos tarjetas gráficas *Quadro RTX 5000*[27]. Esto hace que al emplear software especializado para la paralelización en el entrenamiento de técnicas *Deep Learning*, se consiga una alta eficiencia en tiempo respecto a usar la CPU. También se hizo uso de su capacidad de casi 600 GB para poder almacenar todos los datos de muestreos de tablas LQ y realizar su procesamiento. Hay que tener en cuenta, en contraste con el entorno de investigación, que el equipamiento de las WCN suele ser barato y con poco procesamiento computacional. Al ser OLSR un protocolo de estado de enlace, cada nodo debería construir modelos para estimar la LQ de todos los enlaces (o al menos de aquellos de los que se considerase oportuno tener una predicción). Por lo que sería inviable que cada nodo tuviese un hardware equivalente al del entorno de experimentación. Una solución podría ser que en el entrenamiento se optara por un enfoque centralizado. Por ejemplo, en el que un ordenador con buena capacidad computacional capture datos de LQ desde uno o varios nodos (para mitigar efectos de *fish-eye*), y pudiese compartir a los demás nodos los pesos finales obtenidos del entrenamiento de LSTM para cada enlace. El coste de aplicar los pesos y las funciones de activación para obtener salidas de redes LSTM es mucho menor y realizable con el equipamiento barato de estas redes.

4.7 Conclusiones

A lo largo de este capítulo se han explicado las bases del aprendizaje automático. Dentro de este campo de estudio se encuentran las técnicas de aprendizaje profundo, que en los últimos tiempos han adquirido mayor importancia debido al incremento en el procesamiento y capacidad de los ordenadores. Estas técnicas de *Deep Learning* basadas en redes neuronales se han convertido en el estado de arte en muchas aplicaciones prácticas de nuestro día a día. El tipo de redes neuronales que más interés tienen para el trabajo son las recurrentes, especializadas en el tratamiento de secuencias temporales y por tanto, adecuadas para la predicción de secuencias de LQ.

El problema de las redes neuronales recurrentes clásicas es la falta de memoria a largo plazo, debido a efectos en el entrenamiento con técnicas de BPTT como el del

[27] https://www.nvidia.com/es-es/design-visualization/quadro/rtx-5000/

desvanecimiento del gradiente. LSTM es un tipo de RNN que soluciona este problema, gracias al uso de celdas que permiten mantener u olvidar información con el uso de puertas. Es este el algoritmo que se va a usar para la predicción de LQ en este trabajo, a causa de los buenos resultados que ha tenido en otras aplicaciones o estudios. Para implementar estos modelos LSTM se va a usar *Keras*, la API de alto nivel de aprendizaje automático en Python. Además, la versión de *Keras* para tarjetas gráficas va a permitir aprovechar las dos GPU de alta gama de las que se dispone, con el fin de disminuir el alto tiempo de entrenamiento que necesitan los modelos de aprendizaje profundo.

Capítulo 5. Predicción de la calidad de enlace con LSTM

5.1 Introducción

Las WCN son redes que cambian de forma dinámica y sus enlaces son poco fiables al estar compuestas por nodos descentralizados. La selección de enlaces con mayor calidad permite maximizar la tasa de entrega y minimizar la congestión de tráfico. Como consecuencia, el seguimiento de la calidad de enlace es de vital importancia para dotar a los usuarios de estas redes de una buena calidad de servicio [2], [6]. Con la estimación de LQ se pueden producir mejoras en el enrutamiento respecto a usar el seguimiento de LQ proporcionado por el protocolo de encaminamiento, al poder preverse cambios futuros para tomar acciones apropiadas [2].

La predicción de la calidad de enlace se puede ver como un problema de predicción de series temporales. Asimismo, como se ha visto en el capítulo anterior, las redes neuronales recurrentes son redes especializadas en el tratamiento de secuencias temporales. De las distintas redes recurrentes destacan las redes LSTM por su capacidad de poder retener información de valores pasados gracias al uso de puertas para descartar o mantener los datos de su entrada. Por tanto, LSTM puede ser una red adecuada para el problema de predicción de LQ.

En este capítulo se mostrarán los resultados de la predicción de calidad de enlace con el uso de distintos modelos de LSTM a fin de evaluar el rendimiento entre estos modelos y un algoritmo de referencia (*baseline*). Para los resultados se usarán los enlaces que representan los 5 enlaces más variables para cada tipo de enlace según su frecuencia de actualización de LQ. A estos tipos de enlace se les ha llamado cercanos, medianos y lejanos por su relación entre la distancia en saltos y frecuencia de LQ, siendo ésta de 2, 4 y 7 segundos, respectivamente. La metodología empleada es la comentada en el capítulo anterior, usando ventanas temporales, mitad de datos para el entrenamiento y mitad para el test, etc.

El resto de este capítulo se estructura como sigue: en la sección 5.2 se introducirá el algoritmo de referencia empleado en este trabajo. En la siguiente sección se discutirán las métricas usadas en otros trabajos y el porqué de la elección del RMSE en éste. Las primeras pruebas tienen lugar en la sección 5.4, donde se estudia y ajusta una serie de parámetros de LSTM para usarlos de referencia en el resto de los modelos, usándose en esa sección solo los enlaces cercanos. A continuación, en la sección 5.5 se muestran los resultados con los mejores parámetros de LSTM obtenidos en la sección previa para todos los enlaces, añadiendo un modelo LSTM con estado. En la sección 5.6 se evalúa el rendimiento de un modelo LSTM bidireccional. Después, en la 5.7 se propone un modelo LSTM que sea capaz de predecir valores LQ de varios enlaces a la vez que puedan tener características similares o que ayuden a la predicción. Seguidamente, en la sección 5.8 se examinan los resultados de dos modelos LSTM que predicen a varios instantes vista, entre ellos un *autoencoder LSTM*. En última instancia, se termina con una sección de conclusiones referentes a los resultados obtenidos.

5.2 Algoritmo de referencia

El algoritmo de referencia servirá para la comparación entre los nuevos modelos de predicción usando LSTM y lo que se está usando ahora mismo en el protocolo. Teniendo en cuenta que OLSRd toma los valores de LQ actuales para el cálculo de los costes de las rutas, se puede considerar que se está haciendo una predicción del valor LQ como el último valor recibido. Esto traducido a un algoritmo de referencia, al que llamamos *baseline*, se trata de un estimador que dice que el valor de la siguiente muestra en una secuencia LQ va a ser el último valor de esa secuencia. Lo que se ha indicado en este trabajo es que ese valor no tiene que ser el valor óptimo o verdadero, y estimándolo en función de muestras pasadas se podría obtener un valor más cercano al verdadero o valores futuros, ayudando a mejorar el encaminamiento.

Además, este algoritmo de referencia demostró tener un rendimiento superior o muy cercano a otras técnicas de aprendizaje automático cuando se usó como comparación con el conjunto de datos de muestreos de LQ cada 5 minutos en trabajos como [10] y [11]. De igual forma, en [12] con el conjunto de datos de LQ cada segundo se probó que numerosos algoritmos de aprendizaje automático en línea no mejoraban, al menos de manera significativa, al algoritmo de referencia. Debido a estos hechos y su muy bajo coste computacional, se comparará las predicciones de LSTM con este algoritmo de referencia. En el caso de que la predicción sea de varios instantes temporales, el *baseline* predecirá para cada uno de esos instantes el mismo valor, siendo éste el último valor de la secuencia conocido.

5.3 Métrica de evaluación

Se va a realizar un conjunto de predicciones sobre unas secuencias temporales de LQ y se necesita una métrica de evaluación para medir el rendimiento de los distintos modelos LSTM y del propio *baseline*. Al ser un tipo de regresión, algunas de las métricas existentes son: el error absoluto medio (MAE, *Mean Absolute Error*), el error cuadrático medio (MSE, *Mean Square Error*) y la raíz del error cuadrático medio (RMSE, *Root Mean Square Error*). Siendo y_n el valor verdadero de la predicción enésima y $\hat{y}_n$ su predicción para un conjunto de *N* predicciones, estas métricas se calculan según las fórmulas (3), (4) y (5). Otra medida posible del error en la predicción es la del error relativo medio (MRE, Mean *Relative Error*), en la fórmula (6).

$$MAE = \frac{1}{N}\sum_{n=1}^{N} |y_n - \hat{y}_n| \qquad (3)$$

$$MSE = \frac{1}{N}\sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \qquad (4)$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (y_n - \hat{y}_n)^2} \qquad (5)$$

$$MRE = \frac{1}{N}\sum_{n=1}^{N} \frac{|y_n - \hat{y}_n|}{y_n} \qquad (6)$$

En lo referido a estas métricas de evaluación, el MAE asigna el mismo peso a todos los errores, el módulo del error absoluto. El MSE, por su definición, asigna más peso a los errores grandes y menos a los pequeños, debido al cuadrado del error. El RMSE es muy similar al MSE, ya que comparten máximos y mínimos al ser la raíz cuadrada una función creciente para números positivos. Esta raíz hace que la escala de los errores sea similar a la escala de las variables de partida, al deshacer el cuadrado y tener las mismas unidades que las

entradas. En cambio, el MRE tiene en cuenta la escala del valor verdadero al estar éste en el denominador del valor absoluto del error, aumentando el peso del error a medida que se hace más pequeño el valor verdadero.

En cuanto a trabajos de predicción de enlace, en [2] indican que los algoritmos que usaron presentaron el mismo comportamiento tanto para el MAE como para el RMSE, usando el MAE como métrica final. De igual forma, en los trabajos de [11] y [12] se usó el MAE para presentar sus resultados. En [14] hacen uso del RMSE, MAE y el MRE, mientras que [18] reporta el MAE, MSE y RMSE.

Para este trabajo se va a utilizar la métrica del RMSE y no el MAE, debido a que el MAE da el mismo peso al error de todas las diferencias, haciendo que el *baseline*, cuando el valor de LQ sea el mismo (cuando vale mucho tiempo 1, por ejemplo), tenga errores nulos. En cambio, para LSTM aunque la predicción sea el valor anterior, van a existir errores muy pequeños no nulos que tienen efecto en la media, sobre todo si hay muchos valores iguales de LQ. Además, a favor del RMSE se encuentra que éste asignará, como se ha comentado, más peso a los errores grandes y menos a los pequeños. Este criterio de selección se basa en que al final no es tan interesante el tener el valor exacto de LQ, sino el que se predigan cambios fuertes de LQ que puedan afectar al encaminamiento. De nada sirve tener un MAE bajo con el algoritmo de referencia si cuando ha habido cambios fuertes de LQ puntuales no detectados, estos tengan un peso bajo en comparación a errores nulos de LQ iguales. En contraposición, con el RMSE, esos errores que comete el algoritmo de referencia o LSTM de grandes cambios se ven penalizados. El MSE podría usarse también, pero al tratarse del cuadrado del RMSE, el resultado sería totalmente equivalente con la mejora del RMSE de dar el resultado en las mismas unidades que las de las entradas. El MRE no se usa porque no tiene sentido asignar medidas relativas al error en este problema, aumentaría el peso de los errores de LQ cercanos a 0 en vez de considerar los errores por igual en todo el rango (lo más lógico para la predicción de LQ). Por ejemplo, el error relativo de una predicción con error absoluto de 0.2 y un valor verdadero del 0.1 es 2, y con el mismo error absoluto, pero valor verdadero de 0.2 es 1, la mitad.

5.4 Estudio del número de épocas, escalado, pasos y capas LSTM

En esta sección se buscan los mejores parámetros de una red LSTM que puedan servir de punto de partida para el resto de los modelos propuestos. Un modelo LSTM simple puede ser una sola capa oculta de celdas LSTM y una capa de salida con neuronas normales que realice la predicción. Algunos de los parámetros que se tendrán en cuenta en un modelo LSTM creado con *Keras*, son los siguientes:

- **Número de capas**: con *Keras* puedes crear todas las capas que quieras de distintos tipos. En este caso se considerará una o varias capas LSTM ocultas y una de salida "normal" (del tipo *Dense()* en *Keras*).
- **Número de celdas LSTM**: para cada una de esas capas ocultas de LSTM, se indica el número de celdas LSTM de las que están compuestas.
- **Número de características (*features*)**: se debe indicar el número de características que tiene el modelo, es decir, de entradas. En la mayoría de los casos será de solo una, un único valor LQ.
- **Número de pasos temporales (*steps*)**: con este número se indica el tamaño de la

ventana temporal ya explicada. Son el número de valores LQ pasados que se tienen en cuenta para la predicción.

- **Número de épocas**: indica el número de veces que se pasa en el entrenamiento por el conjunto de datos completo.
- **Tamaño del lote (*batch size*)**: el número de entradas que se toman para realizar una optimización de la función de coste. Se usa el valor por defecto de *Keras*, 32.
- **Con estado (*Stateful*)**: es una variable *booleana*, indica con `True` que es una capa LSTM con estado o con `False` si es una sin estado. Si es con estado, el estado interno de las celdas LSTM tras un lote (*batch*), será usado como estado inicial del siguiente lote. Tiene consecuencias en el entrenamiento y la predicción. Se usarán, en su mayor parte, LSTM con este parámetro por defecto, sin estado.
- **Optimizador**: indica el algoritmo que se usa para encontrar el mínimo de la función de coste. Para todos los modelos se va a usar el algoritmo de *Adam*, un tipo de descenso del gradiente estocástico muy usado para modelos secuenciales. Se han hecho pruebas no presentes en este trabajo con otros optimizadores, pero éste es el que mejor resultado obtenía.
- **Función de coste (*loss*)**: función de coste que se pretende minimizar. En todos los modelos se minimizará el MSE.
- **Neuronas de la capa de salida**: indica cuántos valores de salida tendremos. En la mayoría de las modelos se usará una neurona, representando la predicción del valor LQ siguiente.

Para buscar los mejores valores de estos parámetros se usan solo los enlaces cercanos de frecuencia de cambio de LQ cada 2 segundos. El motivo de hacer el estudio solo con los cercanos es el del coste computacional de las numerosas pruebas que se necesitan hacer, esperando que el ajuste sea el mismo para el resto de los enlaces. En un principio, se realizarán unas pruebas para ver el número de épocas necesarias y, si escalar los datos LQ, mejora los resultados. Acto seguido, se estudiarán el número de *steps* y, por último, si la incorporación de celdas o capas mejora la predicción. En esta sección, aunque se muestren los valores de RMSE del *baseline*, no se compararán los resultados con el modelo LSTM porque se hará en la siguiente sección con el mejor modelo encontrado.

Para el estudio de las épocas se hace primero un modelo LSTM con una capa oculta de 100 celdas LSTM, 10 *steps*, datos sin escalar y 125 épocas. Los resultados del RMSE del *baseline* y del modelo LSTM, así como el tiempo medio de entrenamiento de LSTM de esos enlaces se muestran en la Tabla 5.

Enlaces cercanos moda 2	*Baseline* (RMSE)	LSTM con capa de 100 celdas(RMSE)
10.12.3.11/10.12.3.8	0.019016	**0.019000**
10.12.2.98/10.12.240.50	0.015542	**0.015099**
10.12.208.22/10.12.208.29	0.077841	**0.053104**
10.12.208.22/10.12.208.19	0.077472	**0.050912**
10.12.208.22/10.12.208.18	0.077466	**0.051391**
Media RMSE	0.053467	**0.037901**
Tiempo medio de entrenamiento (s)		8360.12 s

Tabla 5. RMSE del baseline y LSTM con 125 épocas

Para saber cuántas épocas son suficientes se usan 10.000 datos de validación, siendo éstos los 10.000 primeros datos de test. En la Figura 38 se muestra el coste MSE del entrenamiento y de la validación según el número de épocas para dos enlaces. Se muestra el MSE, y no la métrica de evaluación RMSE, porque el MSE es la función de coste que se está optimizando. Aun así, los mínimos y máximos para ambas son los mismos. Las gráficas de los otros 3 enlaces se pueden encontrar en el Apéndice B. Se observa que a partir de 10 épocas tanto el coste MSE de entrenamiento como el de validación bajan muy poco, menos de 0.0003. De hecho, para estos dos enlaces, las épocas cercanas a 125 se produce un ligero aumento en el coste de validación, indicando que puede haber sobreajuste (*overfitting*) en los datos de entrenamiento. En las demás figuras del Apéndice B se observa que el valor permanece constante o se reduce muy poco. Además, se han necesitado más de dos horas en el entrenamiento por enlace, aunque para este modelo también se ha sumado el tiempo de cálculo del coste de la validación. Debido al alto coste computacional que conlleva tener muchas épocas, la poca mejora y el riesgo de *overfitting* **se decide que el resto de los modelos del trabajo tengan 15 épocas**.

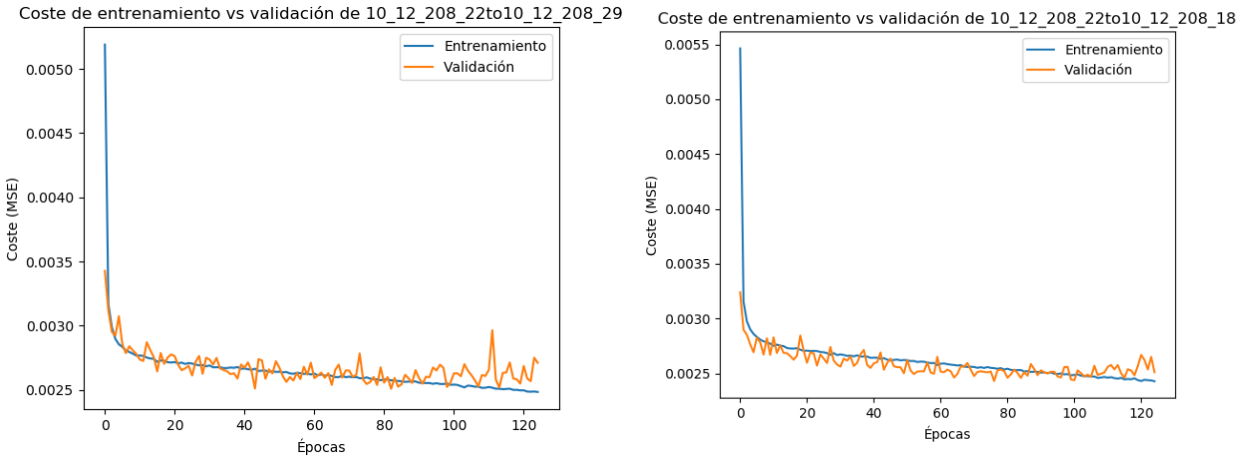


Figura 38. MSE del entrenamiento y datos de validación de dos enlaces según el número de épocas

Con la misma configuración que antes, pero con solo 15 épocas se evalúa el modelo escalando los datos de LQ de cada enlace. Aunque el valor del enlace pueda tener solo valores entre 0 y 1, los enlaces pueden llegar a no tomar el rango de esos valores y puede ser una buena opción escalarlos. En la Tabla 6, se muestran los resultados con los datos sin escalar, escalados entre (0,1) y entre (-1,1).

Enlaces cercanos moda 2	*Baseline*	LSTM sin escalar	LSTM escalado de (0,1)	LSTM escalado de (-1,1)
10.12.3.11/10.12.3.8	0.019016	0.019072	**0.018986**	0.018987
10.12.2.98/10.12.240.50	0.015542	0.015006	0.015026	**0.014906**
10.12.208.22/10.12.208.29	0.077841	0.052026	0.052509	**0.051464**
10.12.208.22/10.12.208.19	0.077472	0.051886	0.052066	**0.051577**
10.12.208.22/10.12.208.18	0.077466	0.051861	0.051756	**0.051611**
Media RMSE	0.053467	0.037970	0.038069	**0.037709**
Tiempo medio de entrenamiento (s)		988.33 s	993.12 s	993.61 s

Tabla 6. RMSE de baseline y LSTM comparando con datos escalados y sin escalar

El primer comentario es que la media de RMSE de LSTM sin escalar de 15 épocas es ligeramente mejor que la anterior con 125 épocas, demostrando que el número de épocas no es insuficiente y reduciendo el entrenamiento por enlace a unos 15 minutos. Se obtiene un tiempo medio de entrenamiento similar para los tres tipos de datos. En cuanto al RMSE, se consigue una mejora para todos los enlaces escalando los datos de -1 a 1. Debido a esto, **en todos los modelos se escalarán los datos LQ de los enlaces de -1 a 1**. Aunque la diferencia puede ser pequeña y no se haya hecho un estudio estadístico, en el desarrollo del trabajo se observó que el escalado de -1 a 1 mejoraba al algoritmo de referencias en todas las ocasiones. Hay que mencionar que el error RMSE sigue calculándose con los valores LQ una vez deshecho el escalado.

El siguiente estudio es sobre los *steps* a considerar para el modelo LSTM (los anteriores modelos se realizaron con 10). Para ello se evalúa el mismo modelo LSTM, escalando los datos de -1 a 1 para diferentes pasos. Las tablas con todos los resultados de RMSE para los enlaces cercanos se encuentran el Apéndice B. En la Figura 39 se muestra la gráfica de los RMSE medios según se varía en *steps* y en la Figura 40 se muestra el tiempo de entrenamiento medio para cada *step*.

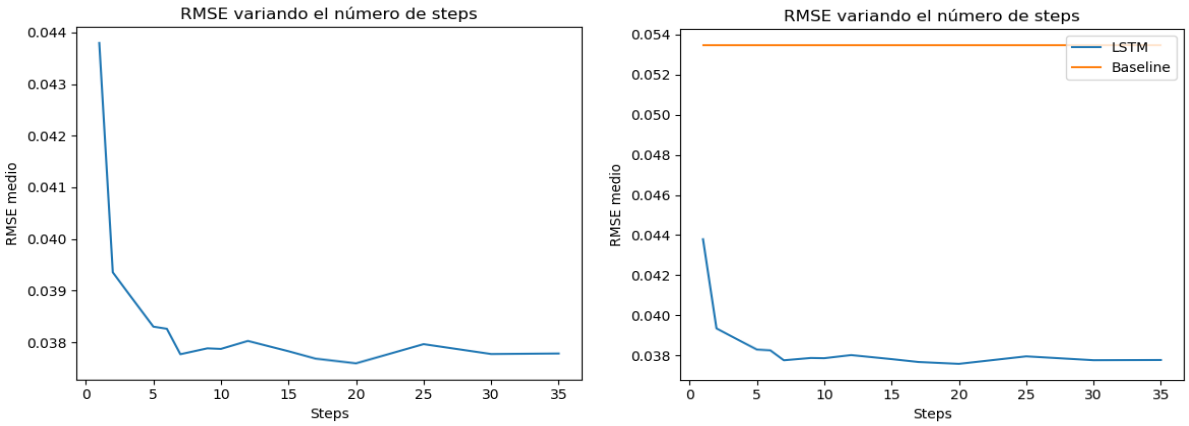


Figura 39. A la izquierda, RMSE medio de los enlaces cercanos con LSTM variando los steps. A la derecha, se añade el Baseline

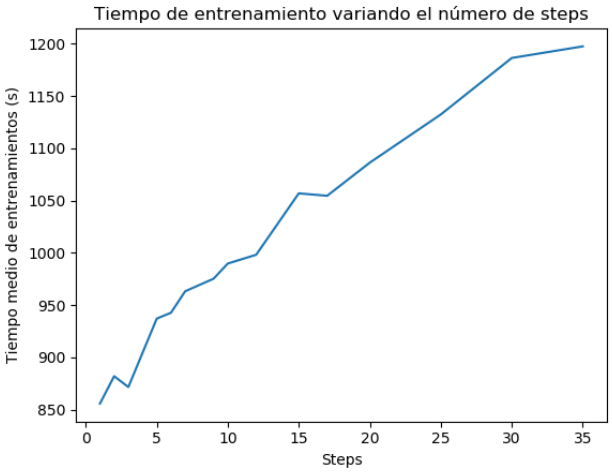


Figura 40. Tiempo medio de entrenamiento en segundos de LSTM con los enlaces cercanos en función de los steps

Se observa que a medida que se aumentan el tamaño de los valores LQ pasados para hacer una predicción, el RMSE baja asintóticamente hasta cierto valor. Es lo que cabría esperar, ya que la mayor influencia la tienen los valores más recientes. Además, el aumento de *steps* lleva consigo un tiempo de entrenamiento mayor, casi de forma lineal, como se aprecia en la segunda figura. Hay dos mínimos con 7 *steps* y con 20, se elegirá **20 *steps* para los modelos LSTM del resto de secciones**. Pese al que de 7 sea más eficiente, con 20 nos aseguramos de que el modelo pueda aprender dependencias de secuencias de LQ más largas si hiciese falta.

Ahora, se va a ver la influencia en rendimiento y tiempo de entrenamiento del número de celdas LSTM de la única capa oculta de LSTM de este modelo. Para ello, se muestran los resultados de RMSE con distintas celdas en la Tabla 7. En este caso las características del modelo LSTM son iguales que la anterior pero con *steps* de 7, dado que arrojaba buenos resultados y se ahorra tiempo de entrenamiento.

Enlaces cercanos moda 2	*Baseline*	LSTM 2 celdas	LSTM 10 celdas	LSTM 25 celdas	LSTM 50 celdas	LSTM 100 celdas	LSTM 200 celdas
10.12.3.11/10.12.3.8	0.0190	0.0190	**0.0189**	0.0190	0.0190	0.0190	0.0190
10.12.2.98/10.12.240.50	0.0152	0.0150	0.0149	**0.0149**	0.0149	0.0149	0.0149
10.12.208.22/10.12.208.29	0.0778	0.0535	0.0520	0.0516	0.0519	0.0518	**0.0513**
10.12.208.22/10.12.208.19	0.0774	0.0533	0.0518	0.0518	0.0517	**0.0515**	0.0515
10.12.208.22/10.12.208.18	0.0774	0.0537	0.0525	0.0521	**0.0516**	0.0525	0.0520
Media RMSE	0.0534	0.0389	0.0380	0.0379	0.0378	0.0379	**0.0377**
Tiempo medio de entrenamiento (s)		950.93	980.66	963.36	984.19	946.14	958.60

Tabla 7. RMSE de baseline y LSTM con 7 steps según el número de celdas LSTM en la capa oculta

Como se ve en la Tabla, con solo 2 celdas LSTM se obtienen valores cercanos de RMSE de números de celdas más altos. Con 200 celdas se arroja el mejor valor pero no es una mejora muy significativa respecto al resto. Además, debido a la paralelización que hace la GPU en el entrenamiento, el tiempo medio de entrenamiento es parecido para todos los números de celdas. **Por este hecho, se elegirán capas LSTM que consten entre 50 y 200 celdas LSTM** al tener este rango de celdas los valores más bajos. Teniendo en cuenta de nuevo, que las diferencias son mínimas tanto en rendimiento como en tiempo de entrenamiento y podría ser necesario un estudio estadístico.

En relación con el estudio anterior, se prueban modelos LSTM con 2 capas ocultas con 50 celdas cada una, con 2 capas de 100, con 3 capas de 100 y un modelo con capas híbridas formada de una capa de 100 celdas LSTM y otra con 100 neuronas normales. Los resultados se encuentran en la Tabla 8. Aunque no se incluya, se observó si era necesario aumentar las 15 épocas al tener más capas, pero no fue así. El hecho de incluir otra capa aumenta el tiempo de entrenamiento en casi un 60%, mientras que en la híbrida con dos capas solo aumenta en torno al 5%. Esto muestra claramente el coste de entrenamiento de una red recurrente, en especial una LSTM. En el caso de 3 capas el incremento fue del casi 110%. Salvo en el caso de la LSTM híbrida, el resto de RMSE han disminuido con el aumento de capas. Para la próxima sección se usará la **de 2 capas de 100 celdas** al presentar mejores resultados que el resto.

Enlaces cercanos moda 2	*Baseline*	LSTM 2 capas de 50 celdas	LSTM 2 capas de 100 celdas	LSTM 3 capas de 100 celdas	LSTM 2 capas, híbridas
10.12.3.11/10.12.3.8	**0.019016**	0.01921	0.019023	0.019107	0.019490
10.12.2.98/10.12.240.50	0.015542	0.015015	0.015056	0.015204	**0.014967**
10.12.208.22/10.12.208.29	0.077841	0.051424	0.051269	**0.051265**	0.051502
10.12.208.22/10.12.208.19	0.077472	0.051265	**0.051054**	0.05148	0.051576
10.12.208.22/10.12.208.18	0.077466	0.050986	0.051183	**0.050893**	0.052011
Media RMSE	0.053467	0.037579	**0.037517**	0.037589	0.037909
Tiempo medio de entrenamiento (s)		1523.33 s	1473.94 s	1985.75 s	1002. 79 s

Tabla 8. RMSE de baseline y modelos LSTM con 7 steps variando capas y celdas

Los resultados de los mejores parámetros pueden no serlos para otros tipos de enlaces o modelos. Aun así, en el desarrollo de este trabajo se realizaron experimentos similares con los enlaces lejanos, no recogidos en este documento, llegando a parámetros prácticamente idénticos. Los parámetros de esta sección servirán de base para el resto. Lo ideal para modelos posteriores sería tomarlos como punto de partida y buscar mejores parámetros a partir de ellos. Sin embargo, este nuevo barrido de parámetros para cada modelo no se hará por el coste que conlleva el entrenamiento y la cantidad de tablas necesarias para cada barrido y tipo de enlace. Además, a la hora de la elección de los parámetros hubiese sido conveniente hacer test estadísticos para saber si las diferencias medias son estadísticamente significativas. Esta evaluación no se realizó, de nuevo, por el coste de entrenamiento de los enlaces para cada parámetro y el tiempo que se tuvo para el desarrollo de este trabajo.

5.5 Predicción con un modelo LSTM por enlace

En esta sección se probará el modelo LSTM con los mejores parámetros encontrados gracias al estudio realizado en la sección previa. A partir de ahora, todos los resultados se harán para los tres tipos de enlaces y no solo los cercanos. Como recordatorio los parámetros del modelo LSTM que se va a probar son el de 2 capas ocultas LSTM con 100 celdas LSTM cada una, 20 *steps*, 15 épocas y escalando los datos LQ de cada enlace entre -1 y 1. Se probará también un modelo LSTM con estado. El LSTM con estado tendrá la misma estructura mencionada anteriormente, pero las capas 2 capas LSTM tendrán el atributo *stateful* como verdadero. Como consecuencia, LSTM aprenderá de la secuencia entera y no solo a mapear una secuencia fija en una salida.

De esta forma, se quiere dejar al modelo aprender las dependencias de la secuencia entera, en vez de definirlas manualmente con la ventana fija. Esto se puede hacer en *Keras* haciendo capas con estado y reiniciando manualmente el estado interno de la red al final de cada época (de hecho, es como las redes LSTM se esperan que se usen) [51]. Esto implica que se debe cambiar la configuración de la red, las épocas se deben hacer individualmente con un bucle *for* y reiniciando con la función de *Keras* `reset_states()` al terminar cada época.

Además, las secuencias de entrada (los lotes o *batchs*) no se deben reordenar aleatoriamente, sino que deben seguir la secuencia inicial, teniendo que indicar en el entrenamiento este efecto con el parámetro *shuffle*.

Por último, *Keras* obliga en redes con estado a que el número de muestras sea múltiplo del tamaño del lote. Para cumplir este efecto en LSTM con estado, se reducirán las muestras necesarias del conjunto de LQ de entrenamiento de cada enlace para hacerlas múltiplo de 32, el mismo tamaño de lote que los modelos LSTM sin estado. Esta reducción de 32 muestras como máximo de los enlaces no tiene un efecto significativo en el total de muestras de entrenamiento al haber cientos de miles y, además, el conjunto de test se queda inalterado. Los resultados para los enlaces cercanos, medianos y lejanos se encuentran en la Tabla 9, Tabla 10 y Tabla 11, respectivamente.

Enlaces Cercanos Moda 2	***Baseline***	**LSTM Sin estado**	**LSTM Con estado**
10.12.3.11/10.12.3.8	**0.019016**	0.019216	0.021053
10.12.2.98/10.12.240.50	0.015542	**0.014645**	0.015027
10.12.208.22/10.12.208.29	0.077841	**0.051128**	0.053056
10.12.208.22/10.12.208.19	0.077472	**0.050540**	0.051628
10.12.208.22/10.12.208.18	0.077466	**0.051220**	0.051385
Media RMSE	0.053467	**0.037349**	0.0384298
Tiempo medio de entrenamiento (s)		1719. 23 s	1620.70 s

Tabla 9. RMSE de baseline y LSTM con los mejores parámetros de los enlaces cercanos

Enlaces medianos moda 4	***Baseline***	**LSTM Sin estado**	**LSTM Con estado**
10.12.4.221/10.12.3.199	0.026723	**0.026303**	0.026855
10.12.91.186/10.12.91.185	**0.029943**	0.032273	0.031172
10.12.2.241/10.12.240.60	0.027386	**0.026299**	0.028938
10.12.208.22/10.12.208.28	0.109640	**0.070507**	0.109326
10.12.2.40/10.12.4.45	0.067945	**0.062776**	0.067651
Media RMSE	0.052327	**0.043631**	0.052788
Tiempo medio de entrenamiento (s)		849 s	804.23 s

Tabla 10. RMSE de baseline y LSTM con los mejores parámetros de los enlaces medianos

Enlaces lejanos Moda 7	*Baseline*	LSTM Sin estado	LSTM Con estado
10.12.3.199/10.12.4.221	**0.040402**	0.040506	0.043501
10.12.2.103/10.12.4.160	0.034596	**0.033368**	0.034850
10.12.4.56/10.12.4.160	0.013246	**0.012921**	0.025235
10.12.3.8/10.12.1.48	0.023780	**0.023534**	0.025503
10.12.4.160/10.12.2.103	0.026901	**0.025458**	0.026571
Media RMSE	0.027784	**0.027157**	0.031132
Tiempo medio de entrenamiento (s)		463.03 s	410.34 s

Tabla 11. RMSE de baseline y LSTM con los mejores parámetros de los enlaces lejanos

El primer comentario sobre los resultados es un efecto que siempre se va a observar en el resto de las modelos del trabajo, siendo éste, la diferencia de tiempo de entrenamiento medio por enlace entre los 3 tipos de enlace. El tiempo de entrenamiento de los medianos es la mitad que el de los cercanos y casi el doble que el de los lejanos. La razón es muy simple: es debido a que hay el doble datos de LQ para entrenar los enlaces cercanos que a los medianos porque uno se obtiene de muestrear en un mismo periodo de tiempo cada 2 segundos y otro cada 4 segundos, y para los lejanos de muestrear cada 7 segundos, que es cercano al doble que el de los medianos. También se estudió si 15 épocas eran suficientes con los otros enlaces al haber menos datos, viendo que sí era apto. Además, LSTM con estado empeora los resultados de LSTM sin estado con un tiempo de entrenamiento medio parecido.

En cuanto a la comparación de LSTM sin estado con los resultados del *baseline*, se hará para cada tipo de enlace. Para los **cercanos** salvo el primer enlace más variable, que presenta prácticamente el mismo RMSE, el resto de los enlaces presentan una gran mejoría. El RMSE medio respecto al *baseline* se reduce en un **30.15%**. La gran mejoría viene motivada por la predicción de los 3 últimos enlaces de este tipo que comparten nodo de origen y, por lo que se ve en su caracterización en la Tabla 2, son muy parecidos. Para los **medianos** ocurre algo parecido al primer caso, uno tiene un RMSE superior al *baseline* y en el resto existe mejora. Aun así, la mejora es mucho menor que con los cercanos, solo se reduce el RMSE medio respecto al *baseline* un **16.62%**, viniendo está mejora motivada en gran medida por el cuarto enlace de este tipo. Con los **lejanos** pasa lo mismo que los cercanos, el primero iguala prácticamente al *baseline* y el resto mejora, siendo esta mejora en el RMSE medio de tan solo el **2.26%**. Se observan con estos resultados que a medida que se usan enlaces cuya frecuencia de actualización es menor, el rendimiento empeora. Las posibles causas pueden ser de que se disponen de menos datos con los medianos y lejanos, que LSTM solo puede aprender con datos de calidad de enlace con mayor tasa de refresco o que el conjunto de enlaces que se han tomado no son del todo representativos. Aun así, siguiendo la filosofía del encaminamiento jerárquico, la misma que la del *fish-eye*, es más útil saber los costes de los enlaces cercanos que los lejanos, ya que a los nodos simplemente les puede bastar tener una decisión razonable para los siguientes dos o tres saltos.

En la Figura 41, se muestra la secuencia completa de test de los valores LQ del enlace mediano 10.12.2.241/10.12.240.60: en azul se encuentra la secuencia real, en naranja la del

baseline y en verde la de LSTM. Los 20 primeros valores de LQ para LSTM son 0 porque necesita de esos valores para hacer la primera predicción. Se observa que la secuencia de LQ es muy ruidosa y que LSTM no llega a los valores más altos y bajos (color naranja en esas zonas porque la secuencia verdadera, azul, está superpuesta en la imagen por el *baseline*). Para ver que ocurre se amplía la secuencia en la Figura 42 para poder distinguir la secuencia verdadera del resto. Se ve como el *baseline* es la secuencia original desplazada a la derecha y como LSTM parece seguir también a la secuencia LQ pero no llega a la predicción óptima. Es la razón de que para este enlace los dos RMSE sean tan parejos, LSTM parece predecir el valor anterior de LQ en gran medida.

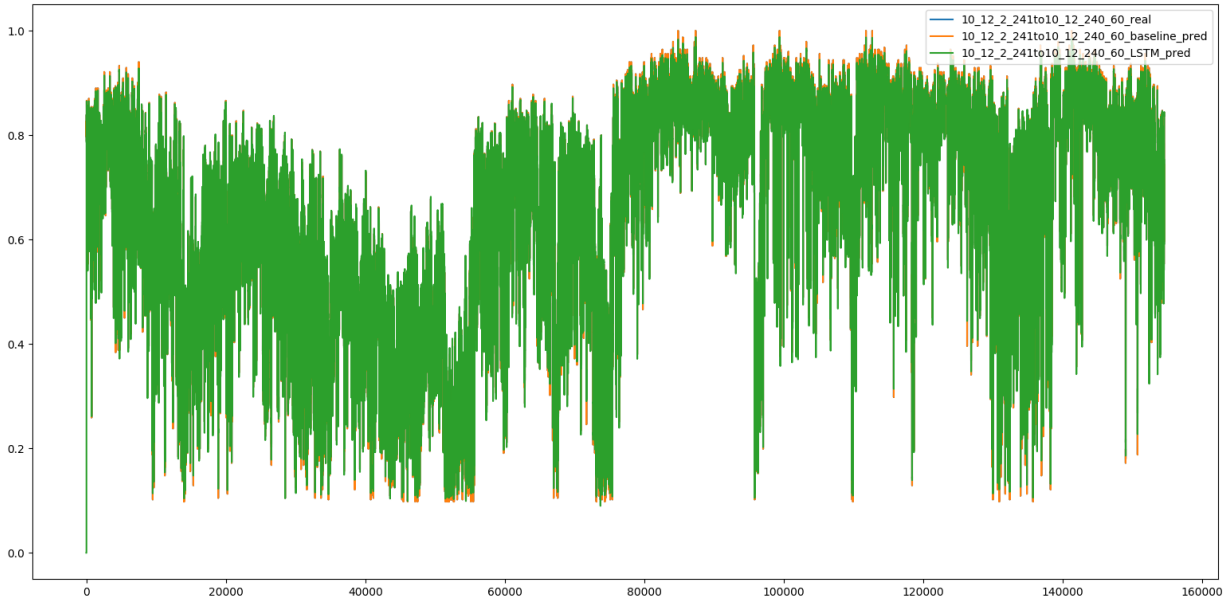


Figura 41. Secuencia completa de valores LQ de test del enlace 10.12.2.241/10.12.240.60, en azul el real, en naranja la predicción del baseline y en verde la de LSTM

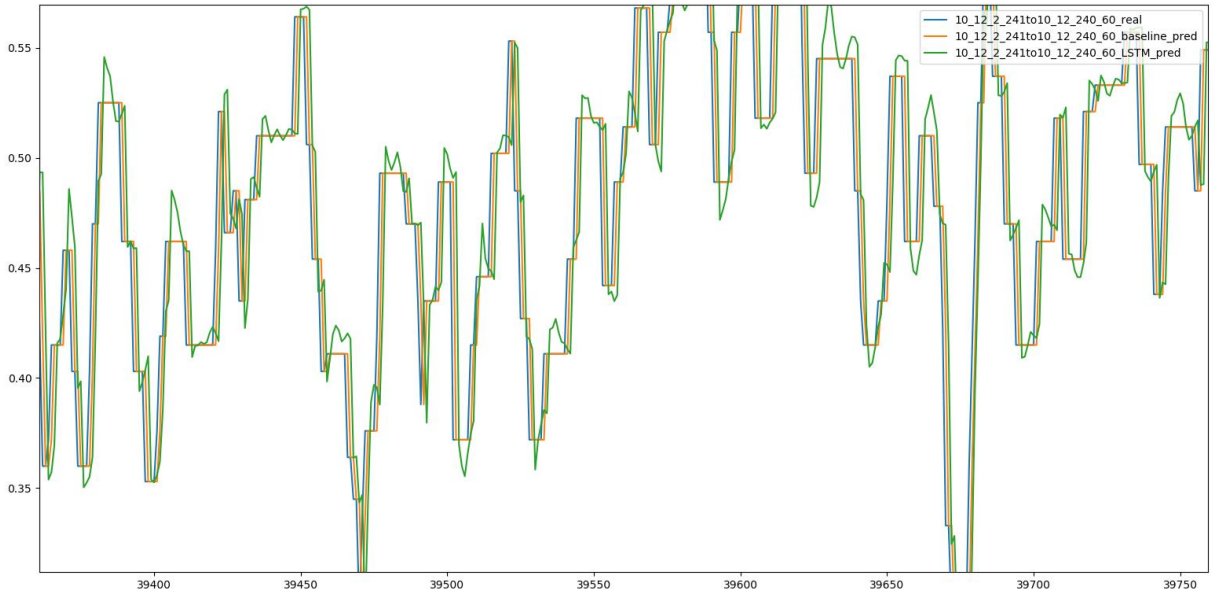


Figura 42. Secuencia ampliada de valores LQ de test del enlace 10.12.2.241/10.12.240.60, en azul el real, en naranja la predicción del baseline y en verde la de LSTM

5.6 Predicción con un modelo LSTM bidireccional por enlace

En esta sección se probará un tipo de LSTM cuya filosofía es la de una red neuronal recurrente estudiada en el capítulo anterior, las redes recurrentes bidireccionales. Haciendo una red LSTM bidireccional se dejará aprender al modelo LSTM con secuencias de entrada de datos pasados y futuros. Para estas pruebas se usarán todos los parámetros del modelo LSTM de la sección anterior, salvo el de que tenga dos capas ocultas. En este caso se hará el modelo con una única capa oculta de 100 celdas LSTM bidireccionales. La razón de esto es por lo explicado en el capítulo 4, las neuronas bidireccionales ya están formadas básicamente de dos RNN de sentidos opuestos. Los resultados del modelo LSTM bidireccional de RMSE en comparación con el *baseline* y los valores de la sección anterior se muestran en la Tabla 12, Tabla 13 y Tabla 14.

Enlaces cercanos moda 2	***Baseline***	**LSTM Sin estado**	**LSTM Bidireccional**
10.12.3.11/10.12.3.8	0.019016	0.019216	**0.018989**
10.12.2.98/10.12.240.50	0.015542	**0.014645**	0.014727
10.12.208.22/10.12.208.29	0.077841	**0.051128**	0.051488
10.12.208.22/10.12.208.19	0.077472	**0.050540**	0.051974
10.12.208.22/10.12.208.18	0.077466	**0.051220**	0.051588
Media RMSE	0.053467	**0.037349**	0.037753
Tiempo medio de entrenamiento (s)		1719. 23 s	1583.78 s

Tabla 12. RMSE de baseline, LSTM con los mejores parámetros y LSTM bidireccional de los enlaces cercanos

Enlaces medianos moda 4	***Baseline***	**LSTM Sin estado**	**LSTM Bidireccional**
10.12.4.221/10.12.3.199	0.026723	0.026303	**0.026179**
10.12.91.186/10.12.91.185	0.029943	0.032273	**0.029923**
10.12.2.241/10.12.240.60	0.027386	**0.026299**	0.026482
10.12.208.22/10.12.208.28	0.109640	**0.070507**	0.070537
10.12.2.40/10.12.4.45	0.067945	**0.062776**	0.062888
Media RMSE	0.052327	0.043631	**0.043201**
Tiempo medio de entrenamiento (s)		849 s	793.51 s

Tabla 13. RMSE de baseline, LSTM con los mejores parámetros y LSTM bidireccional de los enlaces medianos

Enlaces lejanos moda 7	*Baseline*	LSTM Sin estado	LSTM Bidireccional
10.12.3.199/10.12.4.221	0.040402	0.040506	**0.039733**
10.12.2.103/10.12.4.160	0.034596	**0.033368**	0.033479
10.12.4.56/10.12.4.160	0.013246	0.012921	**0.012619**
10.12.3.8/10.12.1.48	0.023780	0.023534	**0.023526**
10.12.4.160/10.12.2.103	0.026901	**0.025458**	0.025591
Media RMSE	0.027784	0.027157	**0.026989**
Tiempo medio de entrenamiento (s)		463.035 s	430.85 s

Tabla 14. RMSE de baseline, LSTM con los mejores parámetros y LSTM bidireccional de los enlaces lejanos

En cuanto a los tiempos de entrenamiento medio por enlace, éstos son inferiores en el bidireccional pero muy similares teniendo en cuenta que LSTM bidireccional solo tiene una capa oculta de 100 celdas LSTM bidireccionales. Esto pone en manifiesto que, a la hora de entrenar una capa bidireccional es casi como entrenar un modelo de dos capas, una para cada sentido de la secuencia. Los resultados de RMSE muestran valores muy parecidos a los obtenidos con el modelo de la sección anterior. Dentro de la igualdad en los resultados, el modelo bidireccional mejora los RMSE de los enlaces que el LSTM clásico no venció al *baseline*, bajando el RMSE medio para los enlaces medianos y lejanos, pero sin ser significativo al ser muy poca la diferencia entre ambos modelos LSTM.

5.7 Un modelo LSTM para varios enlaces

En esta sección se probará un modelo LSTM con una filosofía distinta a las anteriores. Hasta este momento, se creaba un modelo LSTM para cada enlace que tenía una única entrada (y salida) en la que se introducía la secuencia de LQ de ese enlace. Ahora, se quiere probar a hacer un único modelo LSTM que tenga un número *N* de entradas y *N* salidas, representando cada entrada y salida un enlace, con lo cual, tras 20 instancias temporales (es decir, *steps*) donde se meta un valor LQ de un enlace distinto a cada una de las *N* entradas, se generarán *N* salidas con el valor LQ siguiente predicho para cada enlace. De esta forma, se espera que el modelo LSTM pueda llegar a aprender dependencias entre las secuencias de enlaces de entrada que puedan ayudar a la predicción. Por ejemplo, que un enlace sufra una caída de LQ pueda hacer que otro enlace cercano la sufra también, si LSTM consigue aprender estas dependencias podría mejorar la predicción. En la subsección 5.7.1 se aborda el problema de qué enlaces elegir para la predicción conjunto y en la 5.7.2 se muestran los resultados.

5.7.1 Nuevo subconjunto de enlaces

Un criterio razonable para entrenar varios enlaces es el de coger enlaces vecinos, es decir, que compartan alguna de las direcciones IP de origen o destino. Esta forma de seleccionar este conjunto de enlaces para el mismo modelo LSTM tiene varias ventajas. La

primera es que para la predicción de LSTM ha quedado claro que los datos dependen del número de saltos del enlace en cuestión, así que es necesario que la frecuencia de cambio de LQ de esos enlaces sea la misma. Los enlaces vecinos tendrán un número de saltos parecido y, por lo tanto, misma (o parecida) frecuencia de cambios de LQ. La segunda razón es que es lógico pensar que las mayores dependencias se den entre enlaces vecinos, maximizando el hecho de que LSTM pueda usar relaciones entre ellos para mejorar su predicción. Otra opción es la de usar enlaces cualesquiera de misma frecuencia de cambio y entrenar el modelo con la esperanza de que haya dependencias, y de no haberlas, aprenda a diferenciar cada enlace con su salida.

Para elegir los vecinos con los que se hace la prueba, se crea un fichero Python que indica las direcciones IP que más se repiten como origen o destino de la lista de los enlaces filtrados de 3.6. De los 167 enlaces filtrados, las IP más repetidas son las que se muestran en la Figura 43, a su derecha se indica el número de enlaces que comparten esa IP y, consecuentemente, el número de enlaces vecinos.

```
Enlace: 10_12_208_22 Veces que sale: 24
Enlace: 10_12_208_28 Veces que sale: 19
Enlace: 10_12_3_180 Veces que sale: 10
Enlace: 10_12_4_119 Veces que sale: 9
Enlace: 10_12_3_1 Veces que sale: 8
Enlace: 10_12_3_109 Veces que sale: 8
Enlace: 10_12_4_160 Veces que sale: 6
Enlace: 10_12_4_56 Veces que sale: 5
Enlace: 10_12_1_86 Veces que sale: 5
Enlace: 10_12_4_50 Veces que sale: 5
```

Figura 43. IP más repetidas junto con el número de enlaces en las que sale

Se eligen como enlaces de prueba los enlaces cercanos vecinos de **10.12.208.22**, ya que de los 24 enlaces vecinos que tiene, solo 3 no tienen frecuencia de cambio de 2 segundos y por lo tanto, está formado por 21 enlaces cercanos. Del mismo modo, se usarán los enlaces medianos vecinos de **10.12.3.180**, ya que de 10 enlaces, 9 tienen frecuencia de cambio cada 4 segundos. En último lugar, para los enlaces lejanos se usarán los vecinos de **10.12.4.119** que cuenta con 6 enlaces vecinos con frecuencia de cambio de 7 segundos. Serán estos los enlaces con los que se comprobará el rendimiento de este modelo.

5.7.2 Resultados

El modelo LSTM será como el de la sección 5.5, pero con tantas entradas y salidas como enlaces tenga cada conjunto de vecinos y con 40 épocas, al aumentar el conjunto de datos. De igual manera, se realiza un escalado a las entradas de LQ. Los resultados se calculan como el RMSE medio, después de haber calculado el RMSE de cada enlace del conjunto de vecinos al separar la salida de LSTM. En los resultados además del *baseline* y del nuevo modelo para varios enlaces, se muestra la media entrenando cada enlace con el modelo LSTM sin estado de la sección 5.5. En este caso el tiempo de entrenamiento no es medio, es el tiempo de entrenamiento del único modelo para todos los enlaces y el acumulado para el modelo de LSTM por enlace. En la Tabla 15 se muestran los resultados con el RMSE medio del *baseline*, LSTM para varios enlaces y LSTM sin estado de los enlaces que se han escogido en la subsección previa.

Conjunto de enlaces	*Baseline*	LSTM para varios enlaces	Tiempo de entrenamiento LSTM para varios enlaces	LSTM sin estado por enlace	Tiempo de entrenamiento acumulado de LSTM por enlace
*Enlaces cercanos vecinos de **10.12.208.22***	0.059324	**0.044363**	4572.33 s	0.045145	36692.31 s
*Enlaces medianos vecinos de **10.12.3.180***	0.013898	0.014168	2427.19 s	**0.013636**	8139.51 s
*Enlaces lejanos vecinos de **10.12.4.119***	0.015880	0.016273	1354.97 s	**0.014921**	3109.86 s

Tabla 15. RMSE medio del baseline, LSTM para varios enlaces y LSTM sin estado por enlace para unos enlaces vecinos cercanos, medianos y lejanos

La principal ventaja de este modelo es el tiempo de entrenamiento, cada época del modelo con los enlaces cercanos vecinos tarda unos 2 minutos en completarse. Esto quiere decir que para 15 épocas hubiera tardado unos 30 minutos para entrenar 21 enlaces y, en cambio, con el modelo de LSTM sin estado con 21 enlaces a unos 28 minutos de entrenamiento para cada uno, ha tardado más de 10 horas. Con los otros dos grupos de enlaces sucede lo mismo. Esta gran diferencia de tiempos viene motivada por la paralelización en el entrenamiento, tardando prácticamente lo mismo en entrenar 1 entrada o 21 entradas a la vez. Por desgracia, la mejora solo queda ahí, ya que el RMSE de LSTM respecto al *baseline* solo es mejor para los enlaces cercanos. Aun así, para los enlaces cercanos LSTM para varios enlaces mejoró al modelo por enlace y reduciendo el tiempo de manera notable. De nuevo, LSTM funciona mejor con los enlaces cercanos que con los medianos o lejanos, tanto para el nuevo modelo como el modelo por enlace.

5.8 Modelo LSTM de predicción a varios instantes vista

Los modelos planteados hasta ahora tenían una única salida que informaba del valor LQ a un instante vista de un enlace. Una predicción de una secuencia también puede hacerse a varios instantes vistas, es decir, predecir no solo el siguiente valor, sino una secuencia de valores futuros. Estos problemas de predicción se les llama predicción de series temporales multisalto (*multi-step*). En LSTM este tipo de predicción se puede hacer con el uso de un LSTM con salida vectorial o con un *encoder-decoder LSTM*. El primero es simplemente añadir una o más neuronas normales a la capa de salida, el segundo sigue la filosofía explicada en el capítulo 4.

Para los conjuntos de datos de entrenamiento y test se sigue usando la ventana temporal, pero cada entrada va a tener varias predicciones, una para cada instante temporal futuro. La razón de predecir a varios instantes vista tiene dos argumentos claros. Uno puede ser que para el encaminamiento se pueda necesitar información del valor futuro de LQ no solo de 2 segundos vista sino de 4, 6 segundos o incluso más. La otra razón es la de que se pierden muchos mensajes TC y con el uso de esta predicción se puede estimar el valor futuro

sin necesidad de que te llegue el mensaje TC con el nuevo valor LQ. En la sección 5.8.1 se probará el modelo siguiendo la primera metodología y en el 5.8.2 con el uso de un *encoder-decoder LSTM.*

5.8.1 LSTM con varias salidas

La configuración de este modelo LSTM va a ser el de la mejor configuración de LSTM clásica de la sección 5.5, cambiando el número de capas a una única capa LSTM con 100 celdas y añadiendo más neuronas de salida. Hay que tener en cuenta que la predicción del *baseline* va a ser el último valor LQ para todos los instantes vistas, cambiando el error para cada número de salidas. El resultado de RMSE para la predicción a 2, 4 y 6 instantes vista para cada grupo de enlaces se encuentran en la Tabla 16, Tabla 17 y Tabla 18.

Enlaces cercanos moda 2	***Baseline* 2 salidas**	**LSTM 2 salidas**	***Baseline* 4 salidas**	**LSTM 4 salidas**	***Baseline* 6 salidas**	**LSTM 6 salidas**
10.12.3.11/10.12.3.8	**0.023290**	**0.023290**	0.030071	**0.029940**	0.035573	**0.035422**
10.12.2.98/10.12.240.50	0.019682	**0.017837**	0.027432	**0.023542**	0.034713	**0.029218**
10.12.208.22/10.12.208.29	0.095335	**0.056900**	0.121529	**0.067511**	0.141794	**0.079389**
10.12.208.22/10.12.208.19	0.094913	**0.057851**	0.120995	**0.067650**	0.141143	**0.079463**
10.12.208.22/10.12.208.18	0.094994	**0.057784**	0.121157	**0.067888**	0.141357	**0.079718**
Media RMSE	0.065643	**0.042732**	0.084236	**0.051306**	0.098916	**0.060641**
Tiempo medio de entrenamiento (s)		1131.55 s		1114.14 s		1119.69 s

Tabla 16. RMSE de baseline y LSTM con 2, 4 y 6 instantes vista de los enlaces cercanos

Enlaces medianos moda 4	***Baseline* 2 salidas**	**LSTM 2 salidas**	***Baseline* 4 salidas**	**LSTM 4 salidas**	***Baseline* 6 salidas**	**LSTM 6 salidas**
10.12.4.221/10.12.3.199	0.032989	**0.031859**	0.043161	**0.040654**	0.051609	**0.048393**
10.12.91.186/10.12.91.185	**0.037674**	0.038632	**0.051211**	0.051774	0.062771	**0.061418**
10.12.2.241/10.12.240.60	0.034625	**0.032737**	0.047609	**0.044156**	0.058932	**0.053780**
10.12.208.22/10.12.208.28	0.132007	**0.079718**	0.163680	**0.102888**	0.180452	**0.116765**
10.12.2.40/10.12.4.45	0.078444	**0.071579**	0.094332	**0.084349**	0.107762	**0.094939**
Media RMSE	0.063147	**0.050904**	0.079998	**0.064764**	0.092305	**0.075058**
Tiempo medio de entrenamiento (s)		542 s		552.22 s		548 s

Tabla 17. RMSE de baseline y LSTM con 2, 4 y 6 instantes vista de los enlaces medianos

Enlaces lejanos moda 7	*Baseline* 2 salidas	LSTM 2 salidas	*Baseline* 4 salidas	LSTM 4 salidas	*Baseline* 6 salidas	LSTM 6 salidas
10.12.3.199/10.12.4.221	0.048989	**0.048599**	0.062045	**0.057689**	0.071951	**0.067784**
10.12.2.103/10.12.4.160	0.043316	**0.041131**	0.058342	**0.054670**	0.071248	**0.066563**
10.12.4.56/10.12.4.160	0.016374	**0.015252**	0.021471	**0.019325**	0.025723	**0.022629**
10.12.3.8/10.12.1.48	0.029617	**0.028913**	0.039947	**0.037591**	0.048754	**0.045931**
10.12.4.160/10.12.2.103	0.034707	**0.032500**	0.048689	**0.044854**	0.061034	**0.056922**
Media RMSE	0.034601	**0.033278**	0.04609868	**0.042825**	0.055741	**0.051965**
Tiempo medio de entrenamiento (s)		303.07 s		305.18 s		303.21 s

Tabla 18. RMSE de baseline y LSTM con 2, 4 y 6 instantes vista de los enlaces lejanos

Un efecto que se observa es que el tiempo de entrenamiento medio por enlace no se ve afectado al subir el número de instantes vistas, ya que al final solo se añade una neurona más al conjunto del entrenamiento. Como consecuencia, puede ser útil hacer siempre predicciones a varios instantes vistas porque no hay prácticamente coste adicional, pero sí puede tener ventajas, las comentadas al comienzo de esta sección. En cuanto al RMSE, al aumentar los instantes vista, empeoran tanto los resultados del *baseline* como los de LSTM, como es lógico. Aun así, a medida que se avanza en instantes vista, la mejora de LSTM respecto al *baseline* es superior. Comparando con los valores de la sección 5.5 o 5.6, el % de mejora de LSTM respecto al *baseline* crece para todos los enlaces. Llegando a un descenso de RMSE de LSTM respecto al *baseline* del 18.68% a 6 instantes vistas para los enlaces medianos y del 38.69% para los cercanos. La mejora de los enlaces lejanos es menor, pero se ve aumentada también.

5.8.2 Modelo *encoder-decoder LSTM*

Para el modelo *encoder-decoder LSTM*, se van a usar de igual forma 20 *steps*, 15 épocas y se escalan los datos LQ de cada enlace entre -1 y1. El *encoder* y *decoder* van a constar de una capa de 100 celdas LSTM cada uno, conectando la salida fija del *encoder* a la entrada del *decoder*. Además, con este modelo de RNN, aunque esté orientado a predecir una secuencia a partir de otra secuencia, también se puede predecir una única salida. Se presentan los resultados del RMSE de este modelo con los instantes vista anteriores y la predicción a un único instante vista en la Tabla 19, Tabla 20 y Tabla 21.

Enlaces cercanos moda 2	*Autoencoder LSTM* 1 salida	*Autoencoder LSTM* 2 salidas	*Autoencoder LSTM* 4 salidas	*Autoencoder LSTM* 6 salidas
10.12.3.11/10.12.3.8	0.019113	0.023525	0.030449	0.035331
10.12.2.98/10.12.240.50	0.014613	0.017904	0.023808	0.028914
10.12.208.22/10.12.208.29	0.051268	0.057072	0.067476	0.079503
10.12.208.22/10.12.208.19	0.051286	0.056824	0.067544	0.079256
10.12.208.22/10.12.208.18	0.051259	0.057129	0.068172	0.079485
Media RMSE	0.037507	**0.042490**	0.051489	**0.060497**
Tiempo medio de entrenamiento (s)	1510.52 s	1696.62 s	1990.22 s	2292.23 s

Tabla 19. RMSE de encoder-decoder LSTM con 1, 2, 4 y 6 instantes vista de los enlaces cercanos

Enlaces medianos moda 4	*Autoencoder LSTM* 1 salida	*Autoencoder LSTM* 2 salidas	*Autoencoder LSTM* 4 salidas	*Autoencoder LSTM* 6 salidas
10.12.4.221/10.12.3.199	0.026483	0.032118	0.041112	0.047997
10.12.91.186/10.12.91.185	0.030977	0.094641	0.053655	0.060280
10.12.2.241/10.12.240.60	0.026777	0.032653	0.046285	0.054350
10.12.208.22/10.12.208.28	0.070273	0.079534	0.102588	0.116539
10.12.2.40/10.12.4.45	0.062866	0.071089	0.084160	0.094482
Media RMSE	0.043475	0.062007	0.065559	**0.074729**
Tiempo medio de entrenamiento (s)	753.90 s	830,17 s	1005.33 s	1128.96 s

Tabla 20. RMSE de encoder-decoder LSTM con 1, 2, 4 y 6 instantes vista de los enlaces medianos

Enlaces lejanos moda 7	*Autoencoder LSTM* **1 salida**	*Autoencoder LSTM* **2 salidas**	*Autoencoder LSTM* **4 salidas**	*Autoencoder LSTM* **6 salidas**
10.12.3.199/10.12.4.221	0.040077	0.048531	0.058378	0.066326
10.12.2.103/10.12.4.160	0.033670	0.041565	0.054537	0.069949
10.12.4.56/10.12.4.160	0.012752	0.015460	0.019359	0.024298
10.12.3.8/10.12.1.48	0.023144	0.029153	0.037790	0.045827
10.12.4.160/10.12.2.103	0.026436	0.032048	0.045549	0.057672
Media RMSE	0.027215	0.033351	0.043122	0.052814
Tiempo medio de entrenamiento (s)	405.90 s	470.42 s	549.84 s	600.10 s

Tabla 21. RMSE de encoder-decoder LSTM con 1, 2, 4 y 6 instantes vista de los enlaces lejanos

Se observa que ahora los instantes vistas sí que modifican el tiempo de entrenamiento, debido a que el propio funcionamiento de este modelo cambia dependiendo de la longitud de la secuencia de salida. Los tiempos son mayores que con el otro modelo de LSTM *multi-step*, a causa de que los *encoder-decoder* requieren entrenar a dos redes LSTM, una para codificar y otra para decodificar. Los resultados a un instante vista son parecidos a los del LSTM clásico de la sección 5.5 pero sin llegar a mejorarlos. A varios instantes vista, de nuevo, se consiguen resultados muy parecidos a los de LSTM con varias salidas, mejorando en algunos enlaces concretos pero sin grandes diferencias entre ellos. Debido a que necesita un mayor tiempo de entrenamiento, hace al primer modelo de esta sección mejor opción.

5.9 Conclusiones

En este capítulo se han presentado los resultados de aplicar LSTM a la predicción de LQ para unos enlaces representativos del conjunto de datos descrito en el capítulo 3. Para la selección de los mejores parámetros de LSTM se tuvieron en cuenta solo los enlaces cercanos y hay que tener en cuenta que estos parámetros pueden diferir en otros tipos de enlaces, aunque se espera que no lo hagan de manera muy significativa. Con las pruebas de evaluación de RMSE para LSTM se ha visto que el número de celdas o de capas no genera grandes cambios en los resultados. También se observa que al escalar los datos de LQ de cada enlace entre -1 y 1, mejora a los resultados de predicciones sin escalar. Igualmente, con un número de épocas pequeño se consiguen valores muy similares que con otros mayores, siendo importante por el alto tiempo de entrenamiento en comparación con otros algoritmos. Una de las limitaciones fue el hecho de no comparar los resultados con tests estadísticos debido a las limitaciones en el tiempo de desarrollo de este trabajo.

Por otra parte, se ha comprobado, como en otros estudios, el gran rendimiento que presenta el algoritmo de referencia, igualando o mejorando en algunos casos los algoritmos complejos de LSTM. La mayor mejora se dio con los modelos de LSTM sin estado y bidireccional en los enlaces cercanos con un 30.15% de reducción de RMSE, existiendo dudas sobre si este resultado es generalizable, por el hecho de que la mejora es debida a 3 enlaces con características muy similares. Además, se observa que a medida que se usan enlaces con

frecuencia de actualización menor, el rendimiento empeora. Este resultado puede ser debido a que hay menos datos para el entrenamiento, a que la predicción se ve afectada por la menor tasa de muestreo o a la selección de parámetros inicial. Hubiese sido deseable hacer una nueva búsqueda de parámetros para cada modelo y tipo de enlace, pero entre los pocos cambios en resultados que se preveían y el tiempo necesario para hacerlo, no se realizó. En este capítulo, se vio que el uso de un único modelo LSTM para varios enlaces tiene una gran ventaja en la reducción del tiempo de entrenamiento, pero, desafortunadamente, no hay mejora significativa respecto al *baseline* o al LSTM por enlace. Por el contrario, se ha visto que en el uso de predicción a varios instantes vista existe mejora, viéndose incrementada al aumentar los instantes vista, siendo en estos casos en los que más sentido tiene sustituir al *baseline* por algoritmos más complejos. De los modelos LSTM probados para la predicción múltiple, el modelo clásico LSTM con 2 salidas mejora al *encoder-decoder LSTM* propuesto, con resultados similares y menor tiempo de entrenamiento. El uso de LSTM teniendo en cuenta el coste computacional, no es tan claro, habría que hacer estudios sobre la mejora a nivel de encaminamiento que producen valores de RMSE menores que los del *baseline*. En todo caso, su uso estaría justificado para la predicción de enlaces cercanos o a varios instantes vista, ya que del resto la mejora respecto al algoritmo de referencias es nula o no muy significativa.

Capítulo 6. Conclusiones

El objetivo de este trabajo era el de explorar técnicas de aprendizaje profundo para la predicción de calidad de enlace en redes inalámbricas comunitarias, con el fin de que la predicción de valores LQ futuros pudiese mejorar el encaminamiento, al proporcionar información de una posible degradación o mejora de calidad y poder evitar rutas donde haya mayor probabilidad de pérdida. Con predicciones de LQ a corto plazo se podría predecir el verdadero LQ que tiene el enlace y no el que llega a un nodo tras las retransmisiones y tiempos de espera. En cambio, con la predicción a largo plazo o varios instantes vista se podría usar el conocimiento futuro para evitar lo más rápidamente posible los futuribles malos enlaces que surgieran.

Para empezar, se hizo un estudio completo relativo a la LQ de uno de los protocolos más usados en redes comunitarias inalámbricas, OLSR y del funcionamiento de la aplicación que implementaba este protocolo, OLSRd. A partir de su estudio, se llegó a conclusiones como que el tiempo de actualización de valores de LQ, dependía de las llegadas de mensajes TC o que el tiempo de generación de estos mensajes dependía de componentes aleatorias. También se comprobó que el mecanismo de *fish-eye* implementado en la red tiene un impacto en los datos de calidad de enlace recogidos con una tasa de muestreo alta. Este mecanismo no se había tenido en cuenta en trabajos anteriores de predicción con datos de *Funkfeuer Graz*. La activación de este mecanismo implica que la actualización de los valores LQ depende de la distancia de los nodos que proporcionan esa información. Este efecto, sumado a otros como el que la actualización de un LQ pueda llegar de dos nodos, o que los nodos cambien la distancia, hacen que obtener la tasa regular de actualización de LQ a partir de los datos que se tenían hasta ahora sea muy complicado. Debido a esto, se realizó una captura tráfico OLSR completo y no solo los valores de LQ muestreados de tablas de los que hacen uso trabajos anteriores. Este tráfico sirvió para corroborar las hipótesis que se hicieron sobre la actualización de LQ y para que pudiese usarse como un conjunto de datos para posteriores trabajos.

A continuación, se obtuvieron los datos de LQ por enlace de un conjunto de datos que tenía muestras de tablas obtenidas con OLSRd de cada segundo durante 14 días. A partir de estas muestras se hizo un estudio y filtrado de los enlaces y se eligió un conjunto de 15 enlaces para hacer las pruebas de predicción. Esta elección se hizo en conjuntos de los 5 enlaces más variantes con frecuencias de actualización aproximadas de LQ cada 2, 4 y 7 segundos. Al hacer la elección así, se pretendía observar las diferencias de rendimiento que presentaban los predictores de LQ de los distintos enlaces, según la distancia de éstos al nodo que hace la predicción (cercana, mediana, lejana). El bajo número de enlaces es debido al gran tiempo necesario para el entrenamiento de los modelos de predicción, y es una de las limitaciones de este estudio.

Por último, se comprobaron distintos modelos de LSTM con un algoritmo de referencia (*baseline*). La elección de este algoritmo de referencia fue debida a que igualaba o superaba a muchos otros algoritmos de *Machine Learning*, tanto de aprendizaje automático en lote como de aprendizaje en línea que se probaron en otros trabajos. Tras buscar los parámetros óptimos de un modelo LSTM por enlace, se midió el RMSE para los enlaces seleccionados. La elección de los parámetros se realizó con los enlaces cercanos y sin test estadísticos, debido a las limitaciones de tiempo en el desarrollo del Trabajo Fin de Grado.

Se observó el gran coste computacional que conlleva entrenar a este tipo de RNN y los buenos resultados que obtiene el *baseline*. Los resultados para el modelo sin estado de LSTM para los enlaces cercanos fueron relativamente buenos con una reducción del RMSE en torno al 30% respecto al *baseline*. En cambio, para los lejanos y medianos esas reducciones eran bastante menores, habiendo enlaces en los que incluso era mejor la predicción del *baseline*. Los resultados con un LSTM bidireccional fueron muy similares al LSTM sin estado y los de LSTM con estado fueron peores que los otros dos modelos. Con el objetivo de que LSTM pudiera aprender dependencias entre enlaces se hizo un modelo con varias entradas y salidas para predecir valores de LQ de enlaces vecinos a la vez. Los resultados fueron positivos en coste computacional (el entrenamiento de una red LSTM con muchos enlaces de entrada y salida, tardaba casi lo mismo que el entrenamiento de un único enlace), pero los resultados de RMSE fueron muy similares a los del *baseline* y los de LSTM sin estado por enlace. Después, se probó a predecir a varios instantes vista, con un modelo LSTM con varias salidas y con un *autoencoder LSTM*. Presentaron resultados similares pero tenía menor tiempo de entrenamiento el primero, y además se vio que aumentar las neuronas de salida no aumentaba prácticamente el tiempo respecto a entrenar con solo una.

Los resultados respecto al *baseline* fueron mejores a medida que se aumentaban los instantes vista y podrían ser útiles en caso de perder mensajes TC (y el correspondiente LQ) o que se necesitarán mayores intervalos temporales de LQ para la mejora en el encaminamiento. En cualquier caso, estos resultados también fueron mejores para los cercanos y medianos que para los lejanos.

Tras estos resultados, parece que el uso de LSTM no mejora en gran medida al *baseline*, salvo para algunos enlaces cercanos o para predicción a varios instantes vista. Su uso puede estar justificado para la predicción de solo enlaces cercanos, siguiendo la filosofía de *fish-eye* o el encaminamiento jerárquico, de que lo importante es saber encaminar los paquetes en sus primeros saltos y ya los siguientes nodos conocerán mejor los próximos saltos. De tal manera, que el coste de entrenamiento no sería tan grande (solo enlaces cercanos) y es donde mejor RMSE se obtenía respecto al *baseline*. Aun así, para justificar el uso de LSTM los resultados tendrían que haber sido mejores porque el coste computacional de estas RNN es enorme y el del *baseline* es nulo. Además, para el entrenamiento de este modelo se ha necesitado la recogida de 7 días de datos, sin saber la degradación que puede sufrir con el paso del tiempo.

6.1 Trabajo futuro

La primera continuación de este trabajo es obvia: partir del tráfico OLSR capturado y descrito en la sección 3.5, obtener los valores de LQ y poder usar los datos de LQ teniendo claro cuál es su tasa de llegada o a cuántos saltos se encuentra el nodo que se envió, etc. De esta manera, se partiría de un conjunto de datos sin ambigüedades y podrían hacerse nuevas pruebas de predicción. De hecho, se podría generar un modelo de predicción teniendo en cuenta no solo los valores de LQ, sino añadiendo también la distancia en saltos de los nodos de ese enlace, pudiendo incluso predecir estos saltos. De esta forma, se puede hacer un modelo que sea capaz de predecir valores teniendo en cuenta la tasa de llegadas de LQ a través de los saltos a los que se encuentra el enlace. Otro de los caminos a seguir es el de, sabiendo las implicaciones del *fish-eye* (el hecho de que hay que tratar los enlaces de distinta forma, variación de la tasa de actualización de LQ, etc.), probar de nuevo técnicas de aprendizaje en línea. Son mucho menos costosas computacionalmente y se podría ver si con esta consideración, consigue superar al algoritmo de referencia. También se podría estudiar

si añadir características extras como la hora del día o día de la semana puede mejorar la predicción. Habría también que desarrollar más algunos aspectos de la metodología de este trabajo. Por ejemplo, asegurarse de que la mejora de predicción para enlaces cercanos respecto a otros más lejanos se cumple de manera general (se probó con solo 5 enlaces de cada tipo) o el de hacer test estadísticos a los resultados. Se podría hacer de igual forma, un estudio de modelos que entrenen con todos los enlaces a la vez y buscar nuevos parámetros para la predicción por la ventaja que se ha visto de que el tiempo de entrenamiento es mucho menor que la suma de hacerlos por separado.

Si se quiere predecir la LQ para cada enlace obviando el efecto de *fish-eye* en la captura, se podría capturar tráfico de varios puntos de la red simultáneamente de tal forma que se conozca (a efectos de investigación) un valor de LQ actualizado de cualquier enlace. Así, se podrían entrenar modelos de aprendizaje automático para, en un nodo determinado, tener con alta frecuencia estimaciones de la LQ de un enlace lejano a partir de las medidas infrecuentes que le van llegando.

Otra oportunidad de investigación pasaría por cambiar el parámetro a predecir: en vez de calcular las LQ, calcular los costes extremo a extremo. Es decir, estimar el coste, suma de ETX, con cada nodo de la red. Al final, los valores LQ se predicen para después calcular los valores ETX para el encaminamiento, pudiendo comprobar si predecir el coste por destino mejora al cálculo de ETX a partir de la predicción de LQ.

Por último, un estudio relevante para afirmar (o descartar) la importancia de estas investigaciones requeriría ver hasta qué punto puede ayudar al encaminamiento el usar predicciones de calidad de enlace, en lugar del último valor conocido. Se podría usar un simulador con el que se mediría este efecto, variando el porcentaje de pérdida de paquetes para poder comprobar cuál es el mejor horizonte temporal para la predicción (tener en cuenta valores de LQ futuros a 5, 10 o 15 segundos vista, etc.). De esta forma, se podría medir si mejoras pequeñas de RMSE o MAE respecto al *baseline* son significativas o no, y si merece la pena el coste computacional de estos algoritmos.

Referencias

[1] L. Maccari y R. Lo Cigno, «A week in the life of three large Wireless Community Networks», *Ad Hoc Networks*, vol. 24, n.º PB, pp. 175-190, 2015, doi: 10.1016/j.adhoc.2014.07.016.

[2] P. Millan *et al.*, «Time series analysis to predict link quality of wireless community networks», *Computer Networks*, vol. 93, pp. 342-358, 2015, doi: 10.1016/j.comnet.2015.07.021.

[3] J. Avonts, B. Braem, y C. Blondia, «A questionnaire based examination of community networks», *International Conference on Wireless and Mobile Computing, Networking and Communications*, n.º 978, pp. 8-15, 2013, doi: 10.1109/WiMOB.2013.6673333.

[4] A. Neumann, E. López, y L. Navarro, «Evaluation of mesh routing protocols for wireless community networks», *Computer Networks*, vol. 93, pp. 308-323, 2015, doi: 10.1016/j.comnet.2015.07.018.

[5] B. Braem *et al.*, «A case for research with and on community networks», *Computer Communication Review*, vol. 43, n.º 3, pp. 68-73, 2013, doi: 10.1145/2500098.2500108.

[6] C. E. Koksal y H. Balakrishnan, «Quality-Aware Routing Metrics for Time-Varying», vol. 24, n.º 11, pp. 1984-1994, 2006.

[7] S. Vural, D. Wei, y K. Moessner, «Survey of experimental evaluation studies for wireless mesh network deployments in urban areas towards ubiquitous internet», *IEEE Communications Surveys and Tutorials*, vol. 15, n.º 1, pp. 223-239, 2013, doi: 10.1109/SURV.2012.021312.00018.

[8] J. Rodríguez-Covili, S. F. Ochoa, J. A. Pino, R. Messeguer, E. Medina, y D. Royo, «A communication infrastructure to ease the development of mobile collaborative applications», *Journal of Network and Computer Applications*, vol. 34, n.º 6, pp. 1883-1893, 2011, doi: 10.1016/j.jnca.2010.12.014.

[9] J. Prins, «Introduction to Time series Analysis (Engineering Statistics Handbook)», 2013. https://doi.org/10.18434/M32189 (accedido jun. 14, 2020).

[10] C. Mediavilla-Pastor, «Predicción de calidad de enlace en redes comunitarias inalámbricas basadas en OLSR», Trabajo Fin de Grado, Universidad de Valladolid, 2017.

[11] M. L. Bote-Lorenzo, E. Gómez-Sánchez, C. Mediavilla-Pastor, y J. I. Asensio-Pérez, «Online machine learning algorithms to predict link quality in community wireless mesh networks», *Computer Networks*, vol. 132, pp. 68-80, 2018, doi: 10.1016/j.comnet.2018.01.005.

[12] C. Mediavilla-Pastor, «Predicción de calidad de enlace en redes comunitarias

inalámbricas basadas en OLSR a partir de datos obtenidos con una frecuencia alta de muestreo», Trabajo Fin de Máster, Universidad de Valladolid, 2019.

[13] S. Hochreiter y J. Schmidhuber, «Long Short-Term Memory», *Neural Computation*, vol. 9, n.º 8, pp. 1735-1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[14] A. Kulkarni, A. Seetharam, A. Ramesh, y J. D. Herath, «DeepChannel: Wireless channel quality prediction using deep learning», *IEEE Transactions on Vehicular Technology*, vol. 69, n.º 1, pp. 443-456, 2020, doi: 10.1109/TVT.2019.2949954.

[15] Q. Mao, F. Hu, y Q. Hao, «Deep learning for intelligent wireless networks: A comprehensive survey», *IEEE Communications Surveys and Tutorials*, vol. 20, n.º 4, pp. 2595-2621, 2018, doi: 10.1109/COMST.2018.2846401.

[16] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. C. Chen, y L. Hanzo, «Machine Learning Paradigms for Next-Generation Wireless Networks», *IEEE Wireless Communications*, vol. 24, n.º 2, pp. 98-105, 2017, doi: 10.1109/MWC.2016.1500356WC.

[17] W. Sun *et al.*, «LSTM based link quality confidence interval boundary prediction for wireless communication in smart grid», *Computing*, 2020, doi: 10.1007/s00607-020-00816-7.

[18] M. Abdel-Nasser, K. Mahmoud, O. A. Omer, M. Lehtonen, y D. Puig, «Link quality prediction in wireless community networks using deep recurrent neural networks», *Alexandria Engineering Journal*, 2020, doi: 10.1016/j.aej.2020.05.037.

[19] R. Lasser, «Engineering Method | Electrical and Computer Engineering Design Handbook.», 2020. https://sites.tufts.edu/eeseniordesignhandbook/2013/engineering-method/ (accedido may 31, 2020).

[20] S. Jain y D. P. Agrawal, «Wireless community networks», *Computer*, vol. 36, n.º 8, pp. 90-92, 2003, doi: 10.1109/MC.2003.1220588.

[21] L. Maccari, «An analysis of the Ninux wireless community network», *International Conference on Wireless and Mobile Computing, Networking and Communications*, pp. 1-7, 2013, doi: 10.1109/WiMOB.2013.6673332.

[22] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, y L. Navarro, «A technological overview of the guifi.net community network», *Computer Networks*, vol. 93, pp. 260-278, 2015, doi: 10.1016/j.comnet.2015.09.023.

[23] A. Neumann, E. López, L. Cerdà-Alabern, y L. Navarro, «Securely-entrusted multi-topology routing for community networks», *2016 12th Annual Conference on Wireless On-Demand Network Systems and Services, WONS 2016 - Conference Proceedings*, vol. 2016-Janua, pp. 73-80, 2016.

[24] D. J. Tanenbaum, Andrew S. Wetherall, *Computer Networks*, 5.ª ed. Boston: Pearson, 2010.

[25] J. Kurose y K. Ross, *Computer Networking - A top-down approach*, 6.ª ed. Boston: Pearson, 2013.

[26] D. Oran, «OSI IS-IS Intra-domain Routing Protocol», RFC Editor, 1990. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc1142.txt.

[27] J. Moy, «OSPF Version 2», RFC Editor, 1997. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc2178.txt.

[28] T. Clausen y P. Jacquet, «Optimized Link State Routing Protocol (OLSR)», RFC Editor, 2003. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc3626.txt.

[29] T. Clausen, C. Dearlove, P. Jacquet, y U. Herberg, «The Optimized Link State Routing Protocol Version 2», RFC Editor, 2014. [En línea]. Disponible en: http://www.rfc-editor.org/rfc/rfc7181.txt.

[30] A. Tønnesen, «Implementing and extending the Optimized Link State Routing Protocol», University of Oslo, 2004.

[31] A. Tønnesen, T. Lopatic, y A. Kaplan, «OLSR/olsrd», 2007. https://github.com/OLSR/olsrd/blob/master/unmaintained/README (accedido jun. 19, 2020).

[32] A. Tønnesen, T. Lopatic, y A. Kaplan, «olsrd/README-Olsr-Extensions», 2007. https://github.com/OLSR/olsrd/blob/master/README-Olsr-Extensions (accedido jun. 19, 2020).

[33] T. Lopatic, «olsrd Link Quality Extensions», 2004. http://www.olsr.org/docs/README-Link-Quality.html (accedido jun. 20, 2020).

[34] C. Elektra, «olsrd/README-Link-Quality-Fish-Eye.txt at master · OLSR/olsrd», 2005. https://github.com/OLSR/olsrd/blob/master/unmaintained/README-Link-Quality-Fish-Eye.txt (accedido jun. 19, 2020).

[35] P. Millan *et al.*, «Tracking and predicting link quality in wireless community networks», *International Conference on Wireless and Mobile Computing, Networking and Communications*, pp. 239-244, 2014, doi: 10.1109/WiMOB.2014.6962177.

[36] «Funkfeuer Graz». https://www.ffgraz.net/en/about.html (accedido jun. 21, 2020).

[37] M. Mayrhofer, «FunkInselDebian - FunkFeuer Graz», 2017. https://wiki.graz.funkfeuer.at/FunkInselDebian (accedido jun. 21, 2020).

[38] D. Jakhar y I. Kaur, «Artificial intelligence, machine learning and deep learning: definitions and differences», *Clinical and Experimental Dermatology*, vol. 45, n.º 1, pp. 131-132, 2020, doi: 10.1111/ced.14029.

[39] B. Harikrishna, «Deep Learning - Data Driven Investor - Medium», 2018. https://medium.com/datadriveninvestor/deep-learning-2025e8c4a50 (accedido jun. 20, 2020).

[40] I. Goodfellow, Y. Bengio, y A. Courville, *Deep Learning*. MIT Press, 2016.

[41] Y. Lecun, Y. Bengio, y G. Hinton, «Deep learning», *Nature*, vol. 521, n.º 7553, pp. 436-444, 2015, doi: 10.1038/nature14539.

[42] A. Ng, «Machine Learning. Stanford University». https://www.coursera.org/learn/machine-learning (accedido dic. 10, 2020).

[43] J. C. Jaimes, O. E. Gualdrón, y J. Díaz, «Desarrollo de un modelo de regresión con redes neuronales artificiales para estimar la resistencia rotórica de un motor de un motor de inducción», 2017.

[44] «Aprendizaje profundo aplicado al control adaptativo predictivo en UAVs», 2019. https://www.embention.com/es/news/aprendizaje-profundo-y-control-adaptativo-predictivo/ (accedido jun. 20, 2020).

[45] Z. C. Lipton, J. Berkowitz, y C. Elkan, «A Critical Review of Recurrent Neural Networks for Sequence Learning», pp. 1-38, 2015, [En línea]. Disponible en: http://arxiv.org/abs/1506.00019.

[46] R. J. Williams y D. Zipser, «A Learning Algorithm for Continually Running Fully Recurrent Neural Networks», *Neural Computation*, vol. 1, n.º 2, pp. 270-280, 1989, doi: 10.1162/neco.1989.1.2.270.

[47] S. Varsamopoulos, K. Bertels, y C. Almudever, «Designing neural network based decoders for surface codes», Delft University of Technology, 2018.

[48] S. Kostadinov, «Understanding Encoder-Decoder Sequence to Sequence Model», 2019. https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346.

[49] «Welcome to Python.org», 2020. https://www.python.org/doc/ (accedido jun. 26, 2020).

[50] «The World's Most Popular Data Science Platform. Anaconda», 2020. https://www.anaconda.com/ (accedido jun. 26, 2020).

[51] J. Brownlee y M. L. Mastery, *Deep Learning with Python: Develop Deep Learning Models on Theano and TensorFlow Using Keras*. Machine Learning Mastery, 2017.

Apéndice A

A.1 Creación de la VPN a *Funkfeuer Graz*

El primer paso fue el de conseguir permiso para la captura de tráfico a través de una VPN (red privada virtual, *Virtual Private Network*) a algún administrador de *Funkfeuer Graz*. Tras un email solicitándolo a una lista de correos que indicaban en su página web, se nos proporcionó una guía en alemán [37], una serie de ficheros y la IP 10.12.11.33. La guía estaba bastante desactualizada, pero con la información que venía ahí y de [12] (hubo que hacer algún cambio) se consiguió hacer la conexión. La configuración se hizo en un ordenador con Ubuntu.

Los ficheros que me pasaron fueron un certificado raíz SSL/TLS "ca.ctr", el certificado con mis iniciales "edbvpn.crt" y la clave privada "edbvpn.key". El primer paso fue el de instalar la aplicación *openvpn* con `sudo apt-get install openvpn`.

Después, se copian los tres ficheros anteriores en `/etc/openvpn/` y se modifica `client.conf`, en la misma ruta con las líneas del Listado 2.

```
dev tap0

proto udp
remote tun.graz.funkfeuer.at 1194
resolv-retry infinite
nobind

persist-key
persist-tun

daemon ovpn-ff-tunnel
cd /etc/openvpn
chroot /etc/openvpn
user nobody
group nogroup

ca "ca.crt"
cert "edbvpn.crt"
key "edbvpn.key"

ns-cert-type server

cipher none


verb 3
```

Listado 2. Contenido de client.conf de openvpn

En estas líneas se indica que somos un cliente de una VPN, la interfaz del túnel se llamará tap0, indicamos el túnel de la red de *Funkfeuer* al que nos conectamos y se indica el nombre de los ficheros para la certificación. Además, se crea en el mismo directorio, un directorio temporal llamado `tmp`. Una vez hecho esto se inicia la aplicación *openvpn*.

Con la orden `ifconfig` y `netstar -rn` se comprueba que se ha creado correctamente la interfaz tap0. En la Figura 44 se puede ver cómo se ha generado la interfaz asociada al

túnel que nos conecta con la WCN y que la IP de la red que se nos ha asignado es la 10.12.11.33. En la Figura 45, se muestra la tabla de reenvío y como para ir a la red de Funkfeuer se usa la interfaz tap0. Nuestro único vecino a un salto de la red es 10.12.11.1.

```
tap0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.12.11.33  netmask 255.255.255.0  broadcast 10.12.11.255
        inet6 fe80::6832:51ff:fe61:2608  prefixlen 64  scopeid 0x20<link>
        ether 6a:32:51:61:26:08  txqueuelen 100  (Ethernet)
        RX packets 2  bytes 84 (84.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 70  bytes 7468 (7.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figura 44. Salida del comando ifconfig tras iniciar openvpn

```
Tabla de rutas IP del núcleo
Destino         Pasarela        Genmask         Indic   MSS Ventana irtt Interfaz
0.0.0.0         10.0.2.2        0.0.0.0         UG        0 0          0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U         0 0          0 enp0s3
10.12.11.0      0.0.0.0         255.255.255.0   U         0 0          0 tap0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 enp0s3
```

Figura 45. Salida del comando netstat -rn

A.2 Configuración de OLSRd

El siguiente paso es el de configurar OLSR en la interfaz tap0. Para ello se usa la aplicación de OLSRd, cuyo primer paso es instalarla con `sudo apt-get install olsrd`. Una vez instalada se modifica el fichero de `/etc/olsrd/olsrd.conf` con el contenido del Listado 3.

```
DebugLevel 0
IpVersion 4
AllowNoInt yes
Pollrate 0.025
TcRedundancy 2
MprCoverage 7
LinkQualityFishEye 1
LinkQualityLevel 2
UseHysteresis no
FIBMetric "approx"
ClearScreen yes
LinkQualityAging 0.1
LinkQualityAlgorithm "etx_ff"

RtTable 111
RtTableDefault 112

Interface "tap0"
{
        HelloInterval 3.0
        HelloValidityTime 125.0
        TcInterval 2.0
        TcValidityTime 500.0
        MidInterval 25.0
        MidValidityTime 500.0
        HnaInterval 10.0
        HnaValidityTime 125.0
}

LoadPlugin "olsrd_txtinfo.so.0.1"
{
  PlParam     "port"   "9100"
```

```
  PlParam     "accept" "0.0.0.0"
}


LoadPlugin "olsrd_jsoninfo.so.0.0"
{
  PlParam     "port"   "9101"
  PlParam     "accept" "0.0.0.0"
}

LoadPlugin "olsrd_dot_draw.so.0.3"
{
PlParam "accept" "0.0.0.0"
PlParam "port" "2004"
}
```

Listado 3. Configuración de /etc/olsrd/olsrd.conf

La explicación de esta configuración se dio a grandes rasgos en la sección 3.2. Es básicamente la misma configuración pero añadiendo unos *plugins* de OLSRd, como *txtinfo* que permite ver las tablas de información de OLSRd en *localhost* en los puestos indicados. Con esos dos *plugins* es como se pudo capturar los datos de la sección 3.3 y 3.4. También se añade otro *plugin* para dibujar los grafos de la red, lo cual se consiguió pero al ser una red tan sumamente grande no se puede ver de forma completa para incluirlo en este trabajo.

El siguiente paso es de configurar la política de encaminamiento. Para ello se crea el fichero `/etc/network/policyrouting.sh` con la configuración del Listado 4.

```
#! /bin/sh
#
# Policy-Routing Script

## zu konfigurieren
OLSR_IFACES="tap0 enp0s3"

case "$1" in
  up)
        ip rule add lookup olsr pref 20000
        ip rule add lookup main pref 30000
        ip rule del pref 32766

        PREF=32000
        for iface in $OLSR_IFACES; do
                ip rule add iif $iface pref $PREF lookup olsr-default
                PREF=$((PREF+1))
        done
    ;;
  down)
        PREF=32000
        for iface in $OLSR_IFACES; do
                ip rule del iif $iface pref $PREF lookup olsr-default
                PREF=$((PREF+1))
        done

        ip rule add lookup main pref 32766
        ip rule del pref 30000
        ip rule del pref 20000
    ;;
  *)
    echo "Usage: $0 {up|down}"
    exit 1
    ;;
```

```
esac

exit 0
```

Listado 4. Configuración de /etc/network/policyrouting.sh

Se vincula la interfaz de la VPN tap0, con la real de mi ordenador enp0s3. Después, se siguen los pasos de la guía de instalación de [37]. Hay que poner permisos de ejecución al último fichero con `chmod +x /etc/network/policyrouting.sh`. Luego, hay que añadir las tablas de encamamiento de OLSR a `/etc/iprtoute2/rt_tables` con:

```
111 olsr
112 olsr-default
```

También se debe modifica el fichero `/etc/network/interfaces`, añadiendo:

```
up /etc/network/policyrouting.sh up
down /etc/network/policyrouting.sh down
```

Por último, se pone en marcha OLSRd con el comando `sudo olsrd -d 0`, de esta forma se activa en segundo plano. Tras estos pasos comenzará la comunicación de mensajes OLSR entre nuestro ordenador y el resto de la red. Pudiendo capturar el tráfico generado en la comunicación.

A.3 Captura de tráfico

Para la captura de tráfico se usó una máquina virtual Linux con dirección pública a la que se puso a capturar tráfico durante un par de días, del 20 de mayo al 22 de mayo de 2020. Se accedía con SSH con doble salto usando `ssh -XC -J`.

Para ello se usó la herramienta de `tcpdump`, que se usa para analizar y capturar tráfico a tiempo real. Entre las opciones que se usaron en el comando final se encuentran:

- "-i": se indica a la interfaz de red que se va a analizar con `tcpdump`. En este caso es la conectada a la VPN que es tap0.
- "-C": se indica el tamaño máximo a la hora de guardar paquetes capturados en un archivo. Si se usa junto -w y mientras se guardan paquetes supera la unidad de tamaño marcada, se cierra el archivo actual y se abre uno nuevo con el mismo nombre y un número incremental. Se especifica en MB (1.000.000 bytes).
- "-w": para que escriba los paquetes que llegan en el fichero que es especifican en vez de simplemente mostrarlos por la consola.
- "port": realiza un filtro para que solo los paquetes que lleguen al puerto indicado sean escritos. OLSR usa el puerto UDP 698 para enviar y recibir los mensajes OLSR.

Además, para que siguiese activo el comando aunque se cerrase la sesión SSH a la máquina virtual se añadió el comando `nohup`. Usándose, finalmente, la siguiente línea:

```
nohup sudo tcpdump -i tap0 -C 100 -w /home/ediezb/capturaTrafico/captura port
698 &
```

En la Figura 46, se muestran los ficheros `.pcap` generados y en la Figura 47 la salida de texto cuando se lee con `tcpdump`. Se ven que los archivos pesan en torno a 100MB y que contienen todos los mensajes OLSR que se llevaron a cabo. En la Figura 48 se muestra la salida normal de `tcpdump`. Los datos capturados se encuentran en https://doi.org/10.5281/zenodo.3906818.

```
ediezb@mv-enrique:~/capturaTrafico$ ls -la
total 876856
drwxrwxr-x  2 ediezb ediezb      4096 May 22 15:57 .
drwxr-xr-x 10 ediezb ediezb      4096 Jun 11 09:24 ..
-rw-r--r--  1 root   root   100000986 May 20 19:07 captura
-rw-r--r--  1 root   root   100001192 May 21 01:24 captura1
-rw-r--r--  1 root   root   100000506 May 21 07:41 captura2
-rw-r--r--  1 root   root   100000312 May 21 14:02 captura3
-rw-r--r--  1 root   root   100001264 May 21 20:32 captura4
-rw-r--r--  1 root   root   100000982 May 22 03:01 captura5
-rw-r--r--  1 root   root   100000112 May 22 09:29 captura6
-rw-r--r--  1 root   root   100000278 May 22 15:57 captura7
-rw-r--r--  1 root   root    97837056 May 22 22:18 captura8
-rw-------  1 ediezb ediezb        83 May 20 12:49 nohup.out
```

Figura 46. Ficheros .pcap capturados con tcpdump

```
ediezb@mv-enrique:~/capturaTrafico$ sudo tcpdump -r captura2 | head -n 10
reading from file captura2, link-type EN10MB (Ethernet)
01:24:47.213243 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd04, length 1472
01:24:47.514583 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd05, length 1380
01:24:47.816242 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd06, length 1376
01:24:48.418059 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd07, length 1448
01:24:48.816064 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd08, length 1328
01:24:48.816394 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd09, length 920
01:24:49.046510 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd0a, length 1448
01:24:49.321836 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd0b, length 1448
01:24:49.748971 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd0c, length 1472
01:24:50.072111 IP 10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0xcd0d, length 1312
```

Figura 47. Contenido de un fichero .pcap leído con tcpdump

```
            public.2public.rb25.ffgraz.net/32
10:06:17.439717 IP (tos 0xc0, ttl 64, id 6921, offset 0, flags [DF], proto UDP (17), length 168)
    10.12.11.1.698 > 10.12.11.255.698: OLSRv4, seq 0x18d1, length 140
        Hello-LQ Message (0xc9), originator 10.12.11.1, ttl 1, hop 0
          vtime 124.000s, msg-seq 0x67b3, length 72
          hello-time 3.000s, MPR willingness 3
            link-type Symmetric, neighbor-type Symmetric, len 36
              neighbor 10.12.11.19, link-quality 100.00%, neighbor-link-quality 100.00%
              neighbor 10.12.11.20, link-quality 100.00%, neighbor-link-quality 100.00%
              neighbor 10.12.11.25, link-quality 100.00%, neighbor-link-quality 32.55%
              neighbor 10.12.11.16, link-quality 100.00%, neighbor-link-quality 27.45%
            link-type Symmetric, neighbor-type Symmetric-MPR, len 20
              neighbor 10.12.11.31, link-quality 100.00%, neighbor-link-quality 100.00%
              neighbor 10.12.11.26, link-quality 100.00%, neighbor-link-quality 100.00%
        TC-LQ Message (0xca), originator 10.12.11.1, ttl 8, hop 0
          vtime 496.000s, msg-seq 0x67b4, length 64
            advertised neighbor seq 0x284a
              neighbor 10.12.11.16, link-quality 100.00%, neighbor-link-quality 27.45%
              neighbor 10.12.11.19, link-quality 100.00%, neighbor-link-quality 100.00%
              neighbor 10.12.11.20, link-quality 100.00%, neighbor-link-quality 100.00%
              neighbor 10.12.11.25, link-quality 100.00%, neighbor-link-quality 32.55%
              neighbor 10.12.11.26, link-quality 100.00%, neighbor-link-quality 100.00%
              neighbor 10.12.0.158, link-quality 100.00%, neighbor-link-quality 100.00%
```

Figura 48. Salida impresa en línea de comandos de tcpdump de un mensaje de nuestro vecino

Apéndice B

B.1 Figuras y tablas extras de la sección 5.4

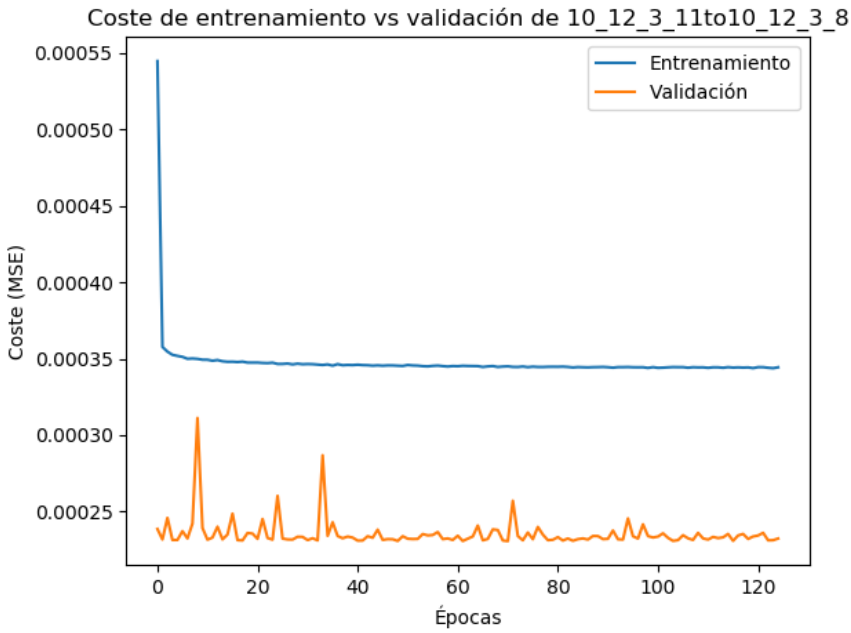


Figura 49. MSE del entrenamiento y datos de validación de 10.12.3.11/10.12.3.8 según el número de épocas

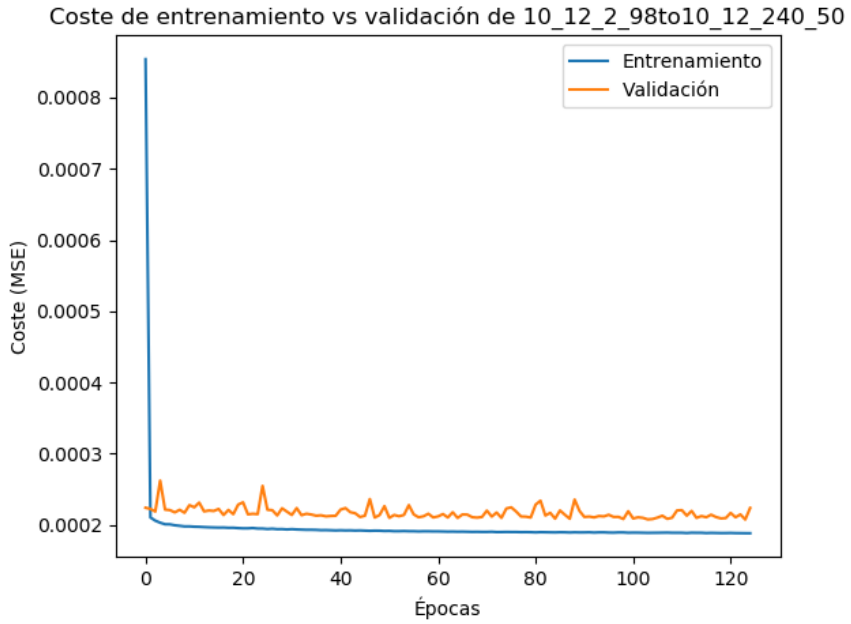


Figura 50. MSE del entrenamiento y datos de validación de 10.12.2.98/10.12.240.50 según el número de épocas

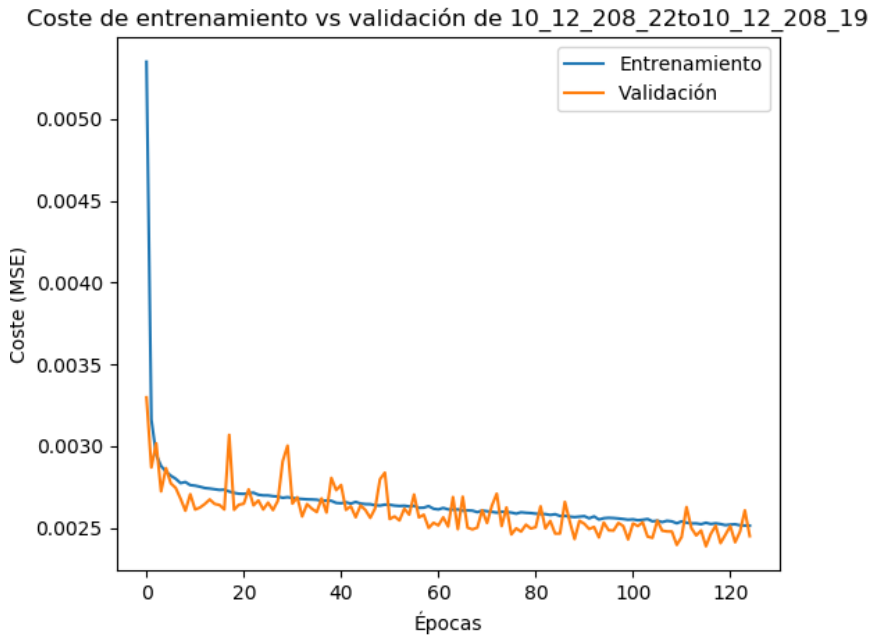


Figura 51. MSE del entrenamiento y datos de 10.12.208.22/10.12.208.19 validación de según el número de épocas

Enlaces cercanos moda 2	***Baseline***	**LSTM 1 step**	**LSTM 2 step**	**LSTM 3 step**	**LSTM 5 step**	**LSTM 6 step**
10.12.3.11/10.12.3.8	0.019016	0.019084	**0.018984**	0.019257	0.019037	0.019148
10.12.2.98/10.12.240.50	0.015542	0.015930	0.015549	0.015452	**0.015048**	0.015270
10.12.208.22/10.12.208.29	0.077841	0.061276	0.054208	0.053302	0.052512	**0.051858**
10.12.208.22/10.12.208.19	0.077472	0.061228	0.054026	0.053373	**0.052380**	0.052650
10.12.208.22/10.12.208.18	0.077466	0.061469	0.054010	0.053626	0.052543	**0.052387**
Media RMSE	0.053467	0.043797	0.039355	0.039002	0.038303	**0.038262**
Tiempo medio de entrenamiento (s)		855.72 s	881.93 s	871.55 s	937.06 s	942.65 s

Tabla 22. RMSE de baseline y LSTM según los steps (I)

Enlaces cercanos moda 2	*Baseline*	LSTM 7 step	LSTM 9 step	LSTM 10 step	LSTM 12 step	LSTM 15 step
10.12.3.11/10.12.3.8	0.019016	**0.018975**	0.019092	0.019445	0.019332	0.019009
10.12.2.98/10.12.240.50	0.015542	0.015146	0.015303	0.014931	0.014918	**0.014861**
10.12.208.22/10.12.208.29	0.077841	0.051510	0.051617	**0.051488**	0.052516	0.052026
10.12.208.22/10.12.208.19	0.077472	**0.051241**	0.051768	0.051422	0.051915	0.051485
10.12.208.22/10.12.208.18	0.077466	0.051974	0.051638	0.052075	**0.051449**	0.051757
Media RMSE	0.053467	**0.037769**	0.037883	0.037872	0.038026	0.037827
Tiempo medio de entrenamiento (s)		963.22 s	975.25 s	989.78 s	998.15 s	1056.95 s

Tabla 23. RMSE de baseline y LSTM según los steps (II)

Enlaces cercanos moda 2	*Baseline*	LSTM 17 step	LSTM 20 step	LSTM 25 step	LSTM 30 step	LSTM 35 step
10.12.3.11/10.12.3.8	0.019016	0.019188	**0.018966**	0.019138	0.019045	0.018996
10.12.2.98/10.12.240.50	0.015542	0.014601	0.014798	0.014646	0.014601	**0.014520**
10.12.208.22/10.12.208.29	0.077841	**0.051380**	0.051464	0.052297	0.051652	0.051607
10.12.208.22/10.12.208.19	0.077472	0.051651	0.051388	0.051414	0.052194	**0.051374**
10.12.208.22/10.12.208.18	0.077466	0.051603	**0.051346**	0.052324	0.051372	0.052421
Media RMSE	0.053467	0.037684	**0.037592**	0.037964	0.037773	0.037783
Tiempo medio de entrenamiento (s)		1054.64 s	1086.33 s	1132.45 s	1186.55 s	1197.65 s

Tabla 24. RMSE de baseline y LSTM según los steps (III)