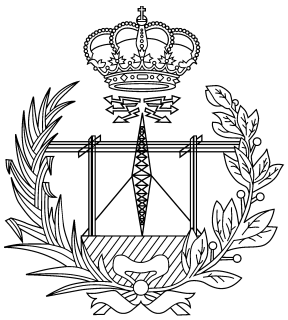
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Trabajo Fin de Máster

ANÁLISIS DEL RENDIMIENTO DEL PROTOCOLO QUIC EN COMUNICACIONES SATELITALES

(Analysis of QUIC performance in Satellite Communications)

Para acceder al Título de

Máster Universitario en Ingeniería de Telecomunicación

Autor: Cristina Aurora Hervella Baturone

Septiembre- 2022

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE MÁSTER

Realizado por: Cristina Aurora Hervella Baturone
Director del TFM: Ramón Agüero Calvo y Luis Francisco Diez Fernández
Título: "Análisis del rendimiento del protocolo QUIC en Comunicaciones Satelitales"
Title: "Analysis of QUIC performance in Satellite Communications"

Presentado a examen el día: 28 de septiembre de 2022

para acceder al Título de

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Composición del tribunal:
Presidente (Apellidos, Nombre): Arce Diego, José Luis
Secretario (Apellidos, Nombre): García Arranz, Marta
Vocal (Apellidos, Nombre): Basterrechea Verdeja, José

Este Tribunal ha resulto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFM
(solo si es distinto del Secretario)

V^{o} B^{o} del Subdirector

Trabajo Fin de Máster N^{o}
(a asignar por Secretaría)

Resumen

Hoy en día, se buscan tecnologías de comunicación que posean, entre otras propiedades, alta cobertura, velocidades (tasa binaria) elevadas y baja latencia. No obstante, parece que conseguir estas características a la vez es un auténtico reto. Si se desea cobertura, las Redes No-Terrestres destacan (sobre todo utilizando satélites geoestacionarios), pero al encontrarse tan alejados de la Tierra sus retardos son muy elevados. Si se desea alta velocidad (o, en otras palabras, baja latencia), las nuevas generaciones de redes móviles parecen las ganadoras –como el famoso 5G o incluso *Beyond 5G*. En cambio, su cobertura puede estar limitada en ciertas zonas, requiriendo un gran despliegue de estructuras (e implicando grandes costes). Es por ello por lo que en este trabajo se centra en la unión de ambas tecnologías, valiéndose de satélites de órbita baja (LEO) usados como elementos de la red de acceso radio. Sin embargo, los enlaces utilizados en dichos escenarios se caracterizan por ser inestables, lo que puede tener efectos negativos en los protocolos usados por los servicios, especialmente en los de transporte. Este trabajo analiza el rendimiento del protocolo QUIC para reducir el impacto de estas inestabilidades en la conexión, resultando en mejoras respecto al protocolo por defecto, TCP. Por ende, este proyecto tiene como objetivo no solo estudiar cuál es el comportamiento de diferentes protocolos de transporte sobre redes de satélites realistas (simuladas), sino también analizar en profundidad cómo funciona el protocolo QUIC y cuál es su rendimiento en entornos inestables. Asimismo, y para finalizar, se han implementado modificaciones en la gestión de flujos a fin de mejorar su rendimiento, los cuales se analizan y comparan con la solución aplicada por defecto en la implementación de QUIC adoptada.

Palabras clave: modelo LMS; protocolo QUIC; protocolo TCP; Redes No-Terrestres; satélites LEO;

Abstract

Nowadays, communication technologies with high coverage, high bit rate and low latency, among other properties, are sought after. But achieving these characteristics at the same time seems to be a major challenge. When coverage is sought, Non-Terrestrial Networks are the preferred solution (especially the use of geostationary satellites). However, because they are so far away from the Earth, the delays introduced are way too significant. On the other hand, if high speed and low latency are desired, the new generations of mobile networks seem to be the best option –5G or even Beyond 5G. Nevertheless, their coverage may be limited in certain areas, resulting in the need for a large deployment of structures (and entailing high costs). Therefore, this work focuses on the linking of both technologies, using low Earth orbit (LEO) satellites as elements of the radio access network. However, the links used in such scenarios are characterized by instability, which can have negative effects on the protocols used by the services, especially transport ones. This work analyses the performance of the QUIC protocol to reduce the impact of these instabilities on the connection, resulting in improvements over the default protocol, TCP. Hence, this project aims not only to study the behavior of different transport protocols over realistic (simulated) satellite networks, but also to analyze in depth how the QUIC protocol works and performs in unstable environments. To conclude, modifications have been implemented in the flow management of QUIC in order to improve its performance, which are analyzed and compared with the one applied by default in the adopted QUIC implementation.

Keywords: LEO satellite; LMS model; Non-Terrestrial Networks; QUIC; TCP.

Índice general

Índice de figuras **7**

Índice de tablas **9**

Índice de acrónimos **10**

1 Introducción **12**
1.1. Motivación 12
1.2. Objetivos 13
1.3. Estructura del documento 14

2 Estado del arte **15**
2.1. Servicios *Fog/Cloud* 15
2.1.1. Red de acceso 18
2.1.2. Interacción con el tráfico actual de Internet 19
2.2. Protocolos de Transporte 19
2.2.1. Evolución 20
2.2.2. Protocolo QUIC 22
2.3. Algoritmos de *scheduling* implementados 25
2.3.1. Round-Robin 25
2.3.2. Fair Queuing 26
2.3.3. Weighted Fair Queuing 28
2.3.4. Proposal Queuing 29

3 Diseño, desarrollo e implementación **32**
3.1. Entorno 32
3.2. Canal del enlace 34
3.3. Tráfico de datos 36
3.3.1. Ficheros de resultados 37
3.4. Modelo de servicio: Scheduler 38
3.4.1. Round-Robin 42
3.4.2. Fair Queuing 43

3.4.3. Weighted Fair Queuing 45
3.4.4. Proposal Queuing 47

4 Comparación de rendimiento y resultados **49**
4.1. Comportamiento y comparativa de los protocolos de transporte QUIC y TCP 49
4.1.1. Enlace LMS 50
4.1.2. Comunicación LEO extremo a extremo 55
4.2. Análisis del rendimiento del Scheduler con diferentes estrategias 57
4.2.1. Comportamiento de los algoritmos en un enlace controlado 58
4.2.2. Rendimiento de los algoritmos en un enlace LMS 63

5 Conclusiones **71**
5.1. Trabajo futuro 72

Bibliografía **74**

Índice de figuras

2.1. Arquitectura de los servicios *Fog/Cloud*. Basado en [3]. 16
2.2. Línea temporal de los protocolos de transporte (estandarizaciones). 20
2.3. Arquitectura *Transmission Control Protocol* (TCP) vs *Quick UDP Internet Connections* (QUIC) [15]. 23
2.4. Formato del *Stream Frame*. Basado en [16]. 23
2.5. Diagrama del algoritmo *Round-Robin* (RR). 26
2.6. Diagrama del algoritmo *Fair Queuing* (FQ). 27
2.7. Diagrama del algoritmo *Weighted Fair Queuing* (WFQ). Se representan las prioridades de cada *stream* en porcentaje. 29
2.8. Diagrama del algoritmo *Proposal Queuing*. Se representan las penalizaciones de cada *stream*. 30

3.1. Diagrama del entorno en donde se hará el pertinente banco de pruebas. Integra las herramientas NS-3 y contenedores Docker. 33
3.2. Modelo de cadena de Markov para los canales satelitales *Uplink* (UL)/*Downlink* (DL). . . . 34
3.3. Diagrama de flujo: creación del paquete QUIC. 39
3.4. Arquitectura de los *buffers* necesarios para el envío de datos en la aplicación implementada. . 41
3.5. Diagrama de flujo: *Scheduler* usando el algoritmo RR. 42
3.6. Diagrama de flujo: *Scheduler* usando el algoritmo FQ. 44
3.7. Diagrama de flujo: *Scheduler* usando el algoritmo WFQ. 46
3.8. Diagrama de flujo: *Scheduler* usando el algoritmo *Proposal Queuing*. 48

4.1. Ejemplo de la evolución del tamaño del *buffer*, ventana de congestión y del retardo experimentado en un canal *Land Mobile-Satellite* (LMS) con *buffer* infinito. 50
4.2. Ejemplo de la evolución del tamaño del *buffer*, ventana de congestión y del retardo experimentado en un canal LMS con un *buffer* de capacidad máxima igual a 7 paquetes. 52
4.3. Retardo experimentado por el tráfico TCP y QUIC en un único enlace LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha). 53
4.4. *Boxplot* o diagrama de caja del retardo experimentado por el tráfico TCP y QUIC en un único enlace LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha). 54
4.5. *Boxplot* o diagrama de caja del retardo experimentado por el tráfico TCP y QUIC en una conexión *End to End* (E2E) que comprende 2 enlaces LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha). 55

4.6. Desviación estándar relativa (RSD) del retardo experimentado por el tráfico TCP y QUIC en una conexión E2E que comprende 2 enlaces LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha). 56
4.7. *Boxplot* o diagrama de caja del retardo experimentado por el tráfico TCP y QUIC en la banda *Ka* en una conexión E2E para diferentes tasas de tráfico. La longitud del *buffer* se fija en 15 paquetes. 57
4.8. Desviación estándar relativa (RSD) del retardo experimentado por el tráfico TCP y QUIC en la banda *Ka* en una conexión E2E para diferentes tasas de tráfico. La longitud del *buffer* se fija en 15 paquetes. 57
4.9. Modelo de un enlace controlado, basado en la cadena de Markov de LMS. 58
4.10. Evolución del *buffer* de la cola de los diferentes *streams* a lo largo del tiempo, utilizando diferentes algoritmos de *scheduling* y un enlace de control. 59
4.11. Evolución de los *buffer* al utilizar los algoritmos de *scheduling Round-Robin* y *Proposal*, en un canal controlado. Se representan los valores de los dos *streams* implementados, el primero de ellos con una línea sólida y el segundo con guiones punteados. 61
4.12. Retardo y *throughput* experimentado por los diferentes algoritmos de *scheduling* implementados, en un canal controlado. 62
4.13. Evolución del *buffer* de la cola de los diferentes *streams* a lo largo del tiempo, utilizando tanto diferentes algoritmos de *scheduling* como tasas binarias de envío, en un enlace LMS. 65
4.14. *Throughput* experimentado tanto por diferentes algoritmos de *scheduling* como tasas binarias de envío, en un canal LMS. 67
4.15. Retardo experimentado por los diferentes algoritmos de *scheduling* implementados, en un único enlace LMS. 68
4.16. *Boxplot* o diagrama de caja del retardo experimentado por los diferentes algoritmos de *scheduling* implementados, en un único enlace LMS. Se incluye el valor medio de cada realización con un marcador circular. 70

Índice de tablas

2.1. Tipos de `Stream ID` [16]-Tabla 1. 24
2.2. Codificación de las cabeceras de longitud variable [16]-Tabla 4. 25

3.1. Configuración de los canales en las diferentes bandas. 35
3.2. Configuración de la capa de aplicación. 37

4.1. Configuración de la capa de aplicación, *buffer* y del canal de control. 58
4.2. Configuración de la capa de aplicación y *buffer*. 64

Índice de acrónimos

3GPP	*3rd Generation Partnership Project*.
5G	Quinta Generación.
B5G	*Beyond 5G*.
CSMA	*Carrier Sense Multiple Access*.
DCCP	*Datagram Congestion Control Protocol*.
DCs	*Data Centers*.
DL	*Downlink*.
E2E	*End to End*.
FIFO	*First In, First Out*.
FQ	*Fair Queuing*.
GEO	*Geoestacionary Equatorial Orbit*.
GIT	Grupo de Ingeniería Telemática.
HOL	*Head-of-line*.
HTTP	*Hypertext Transfer Protocol*.
IP	*Internet Protocol*.
ISL	*Inter-Satellite Link*.
LEO	*Low Earth Orbit*.
LMS	*Land Mobile-Satellite*.
LoS	*Line of Sight*.
MEO	*Medium Earth Orbit*.

MPTCP *Multipath TCP*.

NTN *Non-Terrestrial Networks*.

OSI *Open Systems Interconnection*.

QoE *Quality of Experience*.
QoS *Quality of Service*.
QUIC *Quick UDP Internet Connections*.

RFC *Request for Comments*.
RR *Round-Robin*.
RSD *Relative Standard Deviation*.
RTT *Round-Trip Time*.

SCTP *Stream Control Transmission Protocol*.

TCP *Transmission Control Protocol*.

UAVs *Unmanned Aerial Vehicles*.
UDP *User Datagram Protocol*.
UL *Uplink*.

WFQ *Weighted Fair Queuing*.

Capítulo 1

Introducción

1.1. Motivación

Los avances tecnológicos de las últimas décadas han marcado el rumbo de la evolución en la sociedad actual. Se habla de una época extremadamente digital, en la que se busca estar conectados la mayor parte del tiempo con el mayor número de personas. Es por ello que la tecnología Quinta Generación (5G), especialmente en redes celulares, cobra una gran relevancia hoy en día, ofreciendo elevadas tasas binarias con baja latencia a un amplio número de usuarios simultáneamente.

Sin embargo, alcanzar los requisitos de calidad de servicio deseables hace necesario asegurar un cobertura ubicua, incluso donde no existe con tecnologías anteriores. Por ende, se necesita una gran inversión para poder hacer frente a su despliegue, implicando que una variedad de países no puedan permitirse hacer frente al mismo por razones económicas o geográficas. Este problema derivó en el estudio de diferentes alternativas en la red de acceso que ofreciesen el área de cobertura del que carecen las tecnologías anteriores. Una de estas alternativas, y en la que se va a centrar este trabajo, son las Redes No-Terrestres o más conocidas por su término anglosajón, *Non-Terrestrial Networks* (NTN).

Aunque la adopción de nuevas tecnologías en la red de acceso, ya sean interfaces radio y arquitecturas, puede mejorar las capacidades potenciales de la red, para que este aumento de rendimiento se traslade a una mejora en la *Quality of Service* (QoS) es necesario que el resto de protocolos y soluciones sean capaces de aprovecharlo. Entre los muchos aspectos a tener en cuenta, uno de los más relevantes es el de las soluciones a nivel de transporte, que cubren comunicaciones extremo a extremo o *End to End* (E2E). Esto lleva a que cada vez que aparecen nuevas tecnologías de acceso sea necesario re-evaluar la adaptación y rendimiento de los protocolos de transporte y sus algoritmos, lo que tradicionalmente se ha traducido en el estudio de *Transmission Control Protocol* (TCP).

Sin embargo, dadas las limitaciones de TCP y las características de los servicios y aplicaciones, mayoritariamente web, en los últimos años han aparecido protocolos alternativos. En concreto, en esta última década se ha propuesto y especificado el protocolo *Quick UDP Internet Connections* (QUIC), con el apoyo de los principales actores de servicios de Internet (Google, Microsoft, Meta, Apple, etc.). Este protocolo hereda ideas de soluciones anteriores y se presenta como una alternativa viable a TCP para la mejora de la QoS.

En base a lo anterior, resulta necesario estudiar el rendimiento tanto de TCP como de QUIC en las nuevas redes de acceso en general, y en las NTN en particular. Entre las funcionalidades que diferencia a QUIC de TCP se encuentra la capacidad de gestionar múltiples flujos de tráfico de manera independiente para una misma conexión, *multi-streaming*. Esta característica permite diferenciar tráfico de una misma aplicación (p.e. diferentes objetos de una página web) y gestionarlo de forma adecuada. En este trabajo se realizará la comparativa de prestaciones entre TCP y QUIC en entornos NTN. Concretamente, mediante un entorno de evaluación se estudiará el comportamiento de ambas alternativas en redes de acceso basadas en satélites *Low Earth Orbit* (LEO). Asimismo, se analizará el impacto del *multi-streaming* y se propondrán mejoras sobre la solución de *scheduling* de la implementación de QUIC adoptada.

1.2. Objetivos

Tras conocer la motivación de este trabajo, el siguiente punto es describir los objetivos concretos del mismo. Como se ha mencionado, el objetivo principal se centra en conocer cómo funciona y qué ventajas ofrece el protocolo QUIC, sobre todo en el ámbito de las comunicaciones satelitales. Tales escenarios destacan por sus enlaces inestables, razón por la cual el protocolo TCP no resulta el más adecuado. El siguiente objetivo del trabajo consiste en mejorar el rendimiento de dichas comunicaciones, implementando para ello diferentes algoritmos de *scheduling*. Además, siguiendo la misma línea de pensamiento, otro objetivo fundamental reside en la validación de un algoritmo propuesto, el cual asegura una estabilización de colas.

Sin embargo, existen otras metas las cuales se han querido cumplir en el desarrollo de este trabajo. A continuación, se indican todas ellas:

- Familiarizarse con un entorno de emulación, incluyendo la utilización de herramientas para el manejo de diferentes contenedores (Docker).
- Diseñar y desarrollar una aplicación cliente/servidor capaz de enviar información utilizando diferentes protocolos de transporte; en concreto, QUIC y TCP.
- Analizar el protocolo QUIC en el marco de las comunicaciones satelitales.
- Analizar la comparativa del protocolo TCP y el protocolo QUIC en el marco de las comunicaciones satelitales.
- Familiarizarse y estudiar el funcionamiento del *scheduler* aplicado por defecto en la implementación de QUIC utilizada.
- Implementar diferentes algoritmos de *scheduling* para el protocolo QUIC.
- Analizar el rendimiento de los diferentes algoritmos de *scheduling*, haciendo hincapié al propuesto por el grupo.

1.3. Estructura del documento

En el siguiente apartado se desea representar de qué manera se estructura este documento. Se divide en un total de cinco capítulos, siendo el primero de carácter introductorio, definiendo la motivación y los objetivos del trabajo.

En el Capítulo 2 se procede a explicar los fundamentos teóricos más relevantes en el contexto de este trabajo. En concreto, se estudiarán las ventajas de la utilización de los servicios basados en contenedores, así como las características de la red de acceso, y su interacción con el tráfico de Internet, en escenarios satelitales. Tras ello, se hará una revisión de los protocolos de transporte hasta el día de hoy, centrándose en el protocolo QUIC. Para finalizar, se expondrán las características más relevantes de los distintos algoritmos de *scheduling* que se analizarán más adelante, encargados de la distribución de la información en el protocolo ya mencionado.

Una vez se ha presentado el contexto en el que se enmarca el proyecto, el Capítulo 3 muestra la implementación necesaria para realizar el posterior análisis. Para ello, se necesita conocer el entorno donde se trabajará, definiendo los enlaces simulados y las diferentes herramientas utilizadas para llevarlo a cabo, así como los datos obtenidos. Una vez familiarizados con dicho proceso, se mostrarán las características principales de los escenarios a estudiar. Además, se procede a explicar el entorno de desarrollo del *scheduler*, con la idea de entender cómo funciona y cuáles han sido los cambios pertinentes para implementar los algoritmos previamente definidos.

En el Capítulo 4 se expondrán todos los resultados obtenidos de las configuraciones anteriormente explicadas. Para poder realizar un buen análisis del protocolo QUIC, se estudiarán tanto sus características concretas como en comparación con el protocolo TCP en diferentes escenarios, siempre en el contexto de las comunicaciones satelitales. Por último, se analizará el rendimiento de los diferentes algoritmos de *scheduling* para dicho protocolo.

Finalmente, en el Capítulo 5 se presentarán las conclusiones obtenidas tras la realización del trabajo. Además, también se mostrarán las diferentes líneas de trabajo futuro que aparecen a partir del mismo.

Capítulo 2

Estado del arte

Antes de comenzar con la descripción del sistema de emulación de comunicaciones es necesario definir una serie de aspectos teóricos, los cuales ayudarán a contextualizar el trabajo realizado. Este capítulo abarca, entre otros conceptos de relevancia, los servicios *Fog/Cloud* que constituyen la mayoría de servicios de Internet. Estas plataformas resultan la base del proyecto, por lo que se procede a explicar en qué consisten y cuáles son sus características más relevantes (y diferencias entre ellas), así como los usos más importantes actualmente. Asimismo, se estudiará la importancia de utilizar una red de acceso más novedosa, como son las *Non-Terrestrial Networks*, así como la interacción de dichas redes en el tráfico actual de Internet.

Una vez comprendido qué son estos servicios y el por qué de su uso, el siguiente apartado amplía los conocimientos sobre la capa de transporte, en concreto, alguno de los protocolos más relevantes. Para ello, se hará un revisión de los mismos a través del tiempo, incluyendo desde TCP, pasando por *User Datagram Protocol* (UDP), hasta QUIC. El foco se centrará en los protocolos TCP y QUIC, ya que se desea realizar una comparativa de rendimiento de los mismos. Sobre todo se enfoca en el último de ellos, puesto que es el más novedoso y menos conocido, estudiándose cuáles son sus características más relevantes y qué ventajas ofrece respecto a otras alternativas.

Por último, se analizarán los distintos algoritmos de *scheduling* que se han implementado. La planificación resulta un proceso clave cuando se habla de servicios en nube, ya que según el algoritmo elegido la eficiencia puede disminuir o mejorar enormemente. En este trabajo se manejan cuatro algoritmos: (1) el elegido por defecto en la implementación de QUIC utilizada, basado en el algoritmo *Round-Robin* (RR); (2) el algoritmo *Fair Queuing* (FQ); (3) el algoritmo *Weighted Fair Queuing* (WFQ); y (4) un algoritmo de estabilización de colas propuesto en el marco de este trabajo.

2.1. Servicios *Fog/Cloud*

Los servicios basados en la computación en la nube, o más conocidos como servicios *Cloud*, se han convertido progresivamente en imprescindibles –como en la gestión de centros de datos o *Data Centers* (DCs). Esto es así ya que son los encargados de la provisión de infraestructuras, plataformas y aplicaciones totalmente escalables, así como centradas en la red (se accede mediante Internet) y abstractas (independientes del *hardware*). En otras palabras, son capaces de ofrecer una gran capacidad computacional a cada servicio sin degradar su rendimiento individual. Otra característica muy relevante es

la flexibilidad, prácticamente inmediata, que otorga a sus consumidores. Puesto que el aprovisionamiento de los recursos es muy elevado, se puede ampliar su consumo dependiendo de los requisitos instantáneos que necesite; es decir, los recursos ya no son persistentes, sino dinámicos. Por último, se debe mencionar el modelo que ofrece esta plataforma, basado en la facturación por uso o *pay-as-you-go*. Por ende, no solo es llamativo por su amplia variedad de recursos ofrecidos, sino que además supone una reducción de coste al pagar únicamente por su uso (tiempo) [1].

Todo ello es posible gracias a la utilización de recursos de computación y almacenamientos virtualizados, así como tecnologías web modernas. La virtualización, por una parte, es necesaria para poder dotar el uso compartido de los recursos, mejorando su eficiencia. Esta funcionalidad da flexibilidad al sistema al mismo tiempo que reduce los costes económicos –un sistema *hardware* puede servir para varios entornos *software*. Del mismo modo, otorga simplicidad, permitiendo replicar de manera sencilla funcionalidades necesarias en diferentes servicios. Asimismo, esta simplificación del entorno permite al programador centrarse en su tarea, sin tener en cuenta otros factores. Respecto al apartado del uso de las tecnologías web, resultan imprescindibles puesto que mantienen una buena comunicación entre servicios [2].

Estas características convierten a los servicios *Cloud* en esenciales hoy en día. Sin embargo, su utilización masiva al cabo de los años permitió ver algunos inconvenientes. Ciertas aplicaciones parecían no funcionar eficazmente, sobre todo aquellas que eran sensibles a latencias, tales como los servicios en tiempo real. Para dar respuesta a sus requisitos se han desarrollado arquitecturas de cómputo más cercanas al usuario final, lo que se ha dado en llamar *Fog Computing*. La Figura 2.1 muestra las arquitecturas de los servicios *Fog/Cloud*, ordenadas en función de su lejanía con el usuario. Estas plataformas son independientes entre sí, aunque pueden existir dependencias si fuese necesario, de forma que es el consumidor quien elige qué tecnología o tecnologías son más convenientes. Además, cuanto más alejado esté del usuario (*Cloud)* más lenta será la respuesta (mayor latencia), aunque más recursos computacionales pueden estar disponibles.

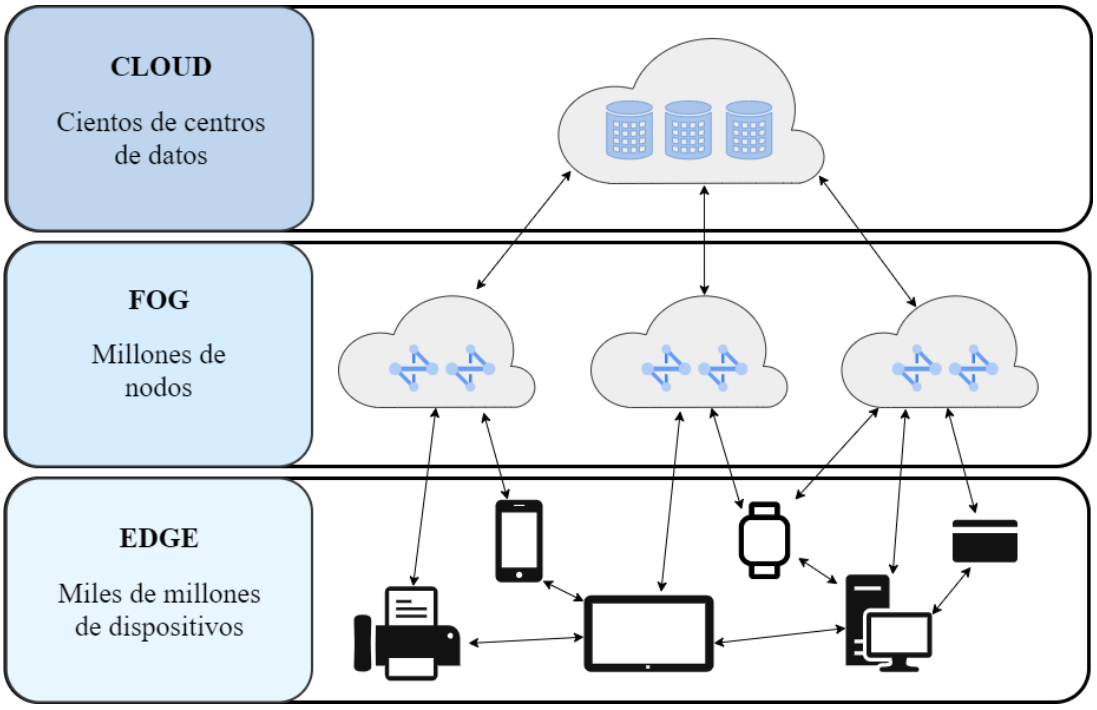


Figura 2.1: Arquitectura de los servicios *Fog/Cloud*. Basado en [3].

El *Fog Computing* nace como una extensión de los servicios en la nube, diseñado para ser utilizado (normalmente, pero no exclusivamente) al borde (*edge*) de la red, por lo que aparece el término de *Edge-computing*. Las características más relevantes se muestran a continuación [4]:

- Bajas latencias y conocimiento o consciencia de la ubicación (*location awareness*). Gracias a estas propiedades, son capaces de dar servicio a aplicaciones de tiempo real.
- Amplia distribución geográfica, en contraposición a los servicios *Cloud* –los cuales son centralizados. Al tener puntos de acceso distribuidos (grandes cantidades de nodos), pueden otorgar señales de alta calidad a equipos en movimiento (por ejemplo, vehículos).
- Compatibilidad con técnicas de movilidad. Esta funcionalidad está muy ligada a las anteriores, permitiendo la comunicación directa con dispositivos móviles.
- Heterogeneidad en sus nodos, capaces de ser utilizados en todo tipo de entornos.
- Interoperabilidad y coordinación entre diferentes proveedores, es decir, otorgando independencia entre los propios servicios o con el *hardware*.

Las capacidades que brindan estas tecnologías no solo son útiles *per se*, sino que resultan en herramientas esenciales para la Quinta Generación de redes móviles.

El 5G requiere características muy específicas, necesarias para dar respuesta a las necesidades de los nuevos servicios en términos de tasa y latencia. Estos seis requisitos serían: (1) mejorar la capacidad –x1000; (2) aumentar la velocidad de datos entre 10 y 100 veces; (3) reducir latencia extremo a extremo, E2E, hasta cinco milisegundos; (4) multiplicar la conectividad masiva de dispositivos entre 10 y 100; (5) tener un coste sostenible y (6) dar un abastecimiento constante en la calidad de experiencia o *Quality of Experience* (QoE) [5]. Incluso se está empezando a trabajar en la siguiente generación de comunicaciones móviles, conocida como *Beyond 5G* (B5G). Esta tecnología tiene como objetivo ofrecer servicios con una alta calidad y latencias extremadamente bajas (para aplicaciones de telepresencia), además de mejorar aún más la capacidad masiva y la conectividad, soportando miles de millones de dispositivos [6].

Por ende, se entiende que las generaciones futuras de tecnologías de comunicaciones inalámbricas (y prácticamente ya presentes) necesitan una infraestructura de red de alta disponibilidad. Este es el principal motivo para utilizar las plataformas anteriores. Los servicios *Fog* aprovechan su infraestructura distribuida, con la ayuda de todos los dispositivos existentes entre el usuario y la nube (como servidores, conmutadores o incluso teléfonos inteligentes), para formar un entorno totalmente conectado y entrelazado. De esta forma, se habilita la cooperación de dispositivos, trabajando conjuntamente bajo las órdenes de un gestor que les controla de la manera más eficaz. En otras palabras, se aprovechan todos los recursos posibles garantizando así latencias ultra-bajas, incluso cuando se trabaja en aplicaciones que requieren altas capacidades [7].

La unión de estas tecnologías es provechosa, como ya se ha comentado. Sin embargo, es un sector muy amplio y novedoso, en el que aún no se han explorado todas sus funcionalidades. Es por ello que en este trabajo se han estudiado distintas mejoras a aplicar, centrándose principalmente en la red de acceso y su interacción con el tráfico de Internet.

2.1.1. Red de acceso

Como ya se ha comentado, las tecnologías de 5G y B5G tienen unos requisitos muy estrictos, sobre todo a la hora de ofrecer conectividad. Esta no solo tiene que ser masiva (teniendo en cuenta la creciente demanda de datos), sino que debe ser global y fiable. Para conseguir estos objetivos se han implementado redes heterogéneas, es decir, redes inalámbricas que utilizan diferentes tecnologías de acceso. Sin embargo, se ha comprobado que aún hay ciertos aspectos a resolver o mejorar.

Uno de los mayores problemas de dicho despliegue es la cobertura de red, causado por una falta de puntos de acceso. Este es uno de los motivos por lo cual un 37.5% de la población mundial no tiene acceso a Internet [8]. Este problema global fue el detonante de buscar nuevas opciones que garantizaran una mayor cobertura, entre las que destacan el uso de las *Non-Terrestrial Networks* (NTN) o Redes No-Terrrestres.

El uso de las redes NTN para ofrecer conectividad está en auge. Cada vez hay más estudios e investigaciones en este campo, e incluso los organismos de estandarización están empezando a considerarlos como habilitadores en las redes B5G [9] [10]. La cobertura que ofrecen es su principal ventaja, ya que permiten prestar servicios a zonas remotas. Asimismo, son capaces de dotar despliegues rápidos en cuestión de recursos de comunicación en determinadas zonas, utilizando para ello *Unmanned Aerial Vehicles* (UAVs) o vehículos aéreos no tripulados –un aspecto muy relevante en el sector sanitario. Otro aspecto positivo se explora en [11], donde se muestra que al combinar las redes satelitales con los servicios *Fog/Cloud* se minimiza el consumo energético global de los usuarios en tierra.

Dentro de las redes de acceso basadas en satélites existen diferencias en función de las distintas órbitas satelitales: (1) la órbita geoestacioanria o *Geoestacionary Equatorial Orbit* (GEO); (2) las órbitas medias o *Medium Earth Orbit* (MEO); y (3) las órbitas bajas o LEO. La diferencia entre ellas es la distancia hasta la superficie de la Tierra, de mayor a menor. Por lo tanto, los satélites en la órbita GEO están a $\approx$36000 *km* sobre el nivel del mar, mientras que los encontrados en órbitas MEO están entre los 10000-12000 *km*. Los más cercanos a la Tierra son los satélites en órbitas LEO, con distancias inferiores a los 1000 *km* [12]. Cuanto menor es esta distancia, menor es la latencia, puesto que se reduce el tiempo de propagación de las señales. Es por ello que en el contexto de comunicaciones 5G, se potencia el uso de los satélites en órbitas bajas.

Sin embargo, a pesar de sus ventajas (baja latencia y coberturas muy elevadas, prácticamente globales), también hay que tener en cuenta un inconveniente relacionado con el uso de este tipo de satélites. Al orbitar tan cerca de la Tierra, implica que estos objetos se mueven a velocidades muy elevadas (entorno a 8 *km/s*) [12]. Además, esta cercanía también afecta a la huella del haz sobre la Tierra, la cual es pequeña (sobre los 100-1000 *km*). Por ambos motivos, se necesita una constelación de satélites bien definida para poder ofrecer un servicio apropiado, lo cual supone grandes inversiones en infraestructura [13]. Sin embargo, actualmente muchas empresas parecen dispuestas a realizar estas inversiones, viendo la relevancia de esta unión de tecnologías, desplegando una gran variedad de constelaciones. Entre ellas se encuentran SpaceX, OneWeb, Telesat o Amazon –en [14] se comparan las diferentes constelaciones de cada empresa, analizando el número de satélites desplegado y su rendimiento.

2.1.2. Interacción con el tráfico actual de Internet

Como se ha comentado en el apartado anterior, al estudiar un caso en el que existe una multitud de satélites que se quieren comunicar entre sí para poder dar conectividad y que orbitan a grandes velocidades, se puede entender por qué estas arquitecturas destacan por sus enlaces inestables. Estas inestabilidades provocan caídas en las tasas binarias cuando no encuentran una línea de visión o *Line of Sight* (LoS) clara entre los dos extremos de un enlace, bien entre dos satélites, debido a la curvatura de la propia Tierra, o entre estos y las estaciones terrenas, debido a obstáculos. En este último caso, enlaces *Land Mobile-Satellite* (LMS), surgen dos posibilidades: cuando hay visión pero está ensombrecida, conocido como *Mid Shadowing*; o cuando se está en penumbra y la tasa cae de manera notable, *Deep Shadowing*.

Es bien conocido que el rendimiento de los protocolos de transporte tradicionales, concretamente TCP, sobre redes inalámbricas no es óptimo, debido a las dificultades de adaptación del control de congestión y los efectos derivados. El tiempo total en el que TCP es capaz de enviar toda la información es elevado, especialmente en conexiones 'cortas' –tráfico *Hypertext Transfer Protocol* (HTTP). Los algoritmos de control de congestión y la operativa en conjunto del protocolo no están diseñados para los escenarios actuales [15], lo cual resulta especialmente necesario en un ámbito tan cambiante como son las telecomunicaciones.

Este hecho parece ir en aumento con la llegada de las nuevas generaciones móviles, provocando la llegada de nuevos protocolos. Entre ellos destaca el protocolo QUIC, propuesto por Google, y el cual se ha especificado recientemente [16]. QUIC elimina la sobrecarga de datos en los múltiples inicios de conexión de TCP (el característico *3-way-handshake* o saludo a tres vías), y puede reducir la latencia, gracias a la multiplexación de flujos de datos sobre una misma conexión, que se realiza sobre el protocolo UDP.

Este protocolo está muy ligado a los servicios *Fog/Cloud*, ya que se estudia la posibilidad de utilizar HTTP sobre QUIC (HTTP/3) en dichos entornos –en [17] se hace un estudio más profundo en este sentido. Asimismo, este protocolo ofrece ventajas en cuestión de seguridad (garantiza la recepción de paquetes) y en tiempo, puesto que dicha entrega se produce en un menor tiempo. Además, parece ser capaz de dar dichas características incluso si las condiciones de red o la ventana de congestión inicial varían –tal y como se puede ver en [18]. Todo ello es extremadamente útil cuando se habla de entornos satelitales, por lo que se plantea un estudio más detallado del comportamiento de dicho protocolo.

Así, el objetivo de este trabajo es extraer conclusiones sobre el rendimiento de QUIC en redes NTN (satelitales), aprovechando el uso de una metodología novedosa, la cual permite imitar el comportamiento de los enlaces inalámbricos satelitales.

2.2. Protocolos de Transporte

Tras entender el entorno en el que se va a trabajar, así como las motivaciones de su uso, se procede a analizar con mayor detalle los distintos protocolos de transporte que existen. Para ello se hará una revisión temporal de los mismos, comenzando en los años 70 con el protocolo TCP, con el objetivo de entender los motivos que llevaron a su creación. Asimismo, se destacarán las características más relevantes de cada uno, así como las diferencias entre sí. También se mostrará el estado que tienen a día de hoy las diferentes opciones a nivel de transporte –no todas tienen el mismo peso, algunas han caído en el desuso.

Dado que este trabajo se centra en la comparativa de QUIC con TCP, se explicará en detalle la operativa de QUIC y cuáles son sus potenciales ventajas con respecto a TCP.

2.2.1. Evolución

El mundo de las telecomunicaciones es muy cambiante, además, hay muchos campos abiertos a la hora de realizar investigaciones. Esta sección se centra en protocolos de transporte más importantes, mostrados en la Figura 2.2. En dicha figura se muestra en una línea temporal el año de aparición de la especificación, *Request for Comments* (RFC), de cada protocolo de transporte.

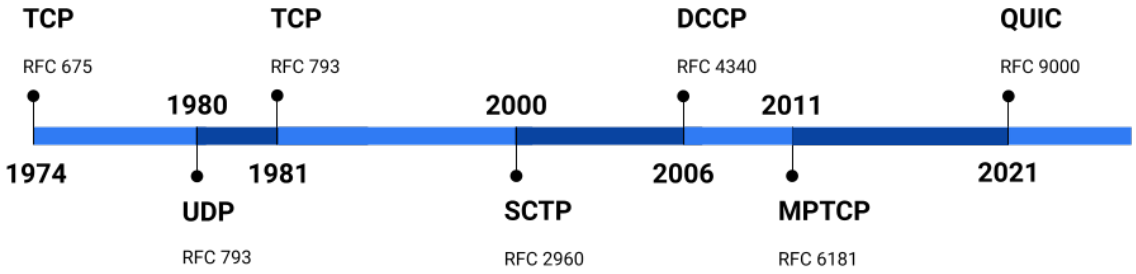


Figura 2.2: Línea temporal de los protocolos de transporte (estandarizaciones).

El primer protocolo diseñado y especificado fue *Transmission Control Protocol* (TCP), remontándose al año 1974. Sin embargo, las especificaciones redactadas en el RFC 675 en dicho año parecían no ser del todo funcionales, por lo que en 1981 se publicó una nueva especificación de este protocolo, añadiendo una versión (`v4`), relacionada mayormente con TCP/IP [19]. En este documento, conocido como RFC 793 o STD-7 (*Standard*), se define TCP como un protocolo totalmente fiable, orientado a la conexión y de extremo a extremo (E2E). Su investigación comenzó al ver la importancia de las comunicaciones telemáticas en todos los ámbitos: militares, gubernamentales y civiles. Aunque sobre todo se centraba en el primer sector, puesto que la característica principal de este protocolo es garantizar la recepción de la información, sin importar que el canal sea poco fiable o su disponibilidad sea baja –o lo que es lo mismo, que el enlace esté congestionado.

Entre las publicaciones de especificaciones para TCP, surgió otro protocolo en 1980, *User Datagram Protocol* (UDP). El RFC 768 o STD-6 define este protocolo basado en datagramas como complementario a TCP. Mientras que el anterior se encarga de dar servicio a aquellas aplicaciones que requieren una entrega fiable y ordenada, UDP está orientado a mensajes. Es decir, reduce enormemente los tiempos de envío, sin poder garantizar que todos los datos lleguen al receptor [20]. Se centra en ofrecer el servicio a aplicaciones que requieran baja latencia.

Estos protocolos, a pesar de ser los primeros, mantienen su relevancia a día de hoy, puesto que todas las comunicaciones se basan en uno o en otro –dependiendo si es necesario que sea orientado a la conexión o no. Han tenido que ir adaptándose a las necesidades de las redes actuales, aunque no siempre fuese posible. Motivo por el cual se empezaron a investigar nuevas formas de ofrecer las ventajas de TCP (fiable, seguro) y las de UDP (bajas latencias) en un único protocolo.

En el año 2000 se publicó un nuevo RFC (RFC 2960) el cual definía un nuevo protocolo de transporte, llamado *Stream Control Transmission Protocol* (SCTP). Como TCP, es un protocolo orientado a la conexión pero también ofrece la transferencia de datos orientada a mensajes, parecido a UDP. Asimismo, es un protocolo de transporte fiable, el cual ofrece una serie de características a sus usuarios, mostradas a continuación [21]:

- Transferencia sin errores, evitando duplicados.
- Fragmentación de datos (permitiendo ajustarse al tamaño máximo).
- Entrega secuenciada, con opción de entrega individual por orden de llegada de mensajes de usuario.
- Agrupación opcional de múltiples mensajes en un único paquete.
- Tolerancia a fallos a nivel de red y soporte *multihoming* o multiconexión (más de una red) en uno o ambos extremos.
- Capacidad de dividir sus datos en múltiples flujos o *streams* (*multi-streaming*). De esta forma, la pérdida de datos de un flujo solo afecta a dicho flujo, evitando inducir retardos en los otros –impide el bloqueo por *Head-of-line* (HOL), o también llamado *Head-of-line blocking*.

Además, el diseño de este protocolo es adecuado para evitar la congestión, así como capaz de resistir a diferentes ataques, tales como la suplantación de identidad (*masquerade)* o por sobrecarga del sistema de destino con múltiples peticiones (conocido como inundación o *flooding*). Aunque *a priori* este protocolo posee las ventajas de los primeros protocolos, e incluso proporciona nuevas, no es muy utilizado a nivel de usuario. Uno de los motivos se encuentra al enviar un fichero de datos pequeño, ya que se genera una sobrecarga en las cabeceras que prácticamente ocupa todo el paquete [22]. Otra de las razones de peso es la escasa aceptación, con pocas implementaciones (en Windows, sobre todo) y soporte prácticamente nulo [23]. Aún así, mantiene su relevancia en entornos de telecomunicaciones. Por ejemplo, es el protocolo utilizado por el *3rd Generation Partnership Project* (3GPP) en la pila de protocolos del segmento de acceso para la comunicación de plano de control. Además, sigue en constante mejora e investigación, sacando nuevas especificaciones –como el RFC 9260, publicado en junio de este año 2022 [24].

Unos años más tarde, en 2006, se propuso otro protocolo de transporte, llamado *Datagram Congestion Control Protocol* (DCCP) [25]. Se diseñó con el fin de dar servicio a las aplicaciones que necesiten bajas latencias, pero una entrega fiable y ordenada –TCP causa retrasos arbitrarios, algo perjudicial para dichas aplicaciones, y UDP necesita de una aplicación externa que se encargue del control de la congestión. DCCP es un protocolo de transporte orientado al mensaje que proporciona un control de congestión totalmente integrado, de tal forma que evita retrasos innecesarios incluso para flujos de datagramas no fiables. Otra de sus características es que permite la negociación tanto de sus opciones de enlace como en el algoritmo de control. Sin embargo, esta solución tuvo menos éxito que la anterior, cayendo rápidamente en desuso debido a que su implementación puede llegar a ser complicada, como se puede ver en [26].

Tras ver cómo los principales protocolos de transporte (sobre todo TCP) empezaban a quedarse atrás respecto a las tecnologías actuales, y que los nuevos protocolos no parecían funcionar con la misma relevancia, el número de soluciones que captaban la atención empezaba a estrecharse. Las alternativas que más interés generaron han sido dos: (1) la evolución directa de TCP, *Multipath TCP* (MPTCP); y (2) el protocolo diseñado por Google, QUIC.

En 2011 se publicó la especificación RFC 6181, en donde se definía una serie de extensiones para el protocolo TCP, de tal manera que proporcionara un entorno *multipath* –es decir, utilizar múltiples caminos en una única conexión (dos nodos finales), para que los datos lleguen con mayor rapidez. Este nuevo protocolo, llamado *Multipath TCP* (MPTCP) tiene tres objetivos claros: (1) mejorar la tasa binaria

o *throughput* –decrementando la latencia; (2) no dañar el comportamiento global de la red evitando utilizar más recursos de los necesarios; y (3) equilibrar la congestión de las colas (desviando el tráfico). En otras palabras, se quiere mejorar la robustez frente a errores y el rendimiento del sistema, respecto a lo que existía en TCP. Para ello, habilita cierta capacidad *multihoming* (multiconexión a nivel de red) y compatibilidad con técnicas de movilidad. Sin embargo, la utilización de múltiples rutas puede resultar compleja, incluso pudiendo ocasionar brechas de seguridad. Es por ello que se necesitan mecanismos de seguridad adicionales, razón tras la cual MPTCP tiene que ser extensible y flexible a la utilización de diferentes soluciones [27].

Por último, en 2021 se publicó, tras varios años en desarrollo, el protocolo QUIC en el RFC 9000 [16]. Desde un primer momento ha tenido buena acogida por parte de la comunidad investigadora y desarrolladora, promovido por la propia empresa Google –actualmente, se utiliza como protocolo de transporte por defecto del navegador Chrome [28]. Este protocolo será analizado en profundidad en la siguiente sección, puesto que su evaluación es un objetivo principal de este trabajo.

2.2.2. Protocolo QUIC

Quick UDP Internet Connections (QUIC) se define en el RFC 9000 como un protocolo de transporte, extremo a extremo (E2E) y orientado a la conexión [16]. Este protocolo, desarrollado por Google, se diseñó para mejorar la experiencia del usuario, por ello su objetivo se basa principalmente en mejorar tiempos (baja latencia), así como ser totalmente seguro y confiable. Para ello se basa en el protocolo UDP, evitando las tramas de conexión de TCP. De la misma manera, al transportar los paquetes QUIC en mensajes UDP, se facilita el despliegue en los sistemas y redes existentes. Aunque dentro de la pila de protocolos *Open Systems Interconnection* (OSI) QUIC se ubicaría en la capa de sesión, en este trabajo nos referiremos a QUIC como solución a nivel de transporte, ya que su funcionalidad y diseño se corresponden a esta capa. Como se ha dicho anteriormente, la especificación de QUIC sobre UDP se hace buscando compatibilidad, aceptación y facilidad en el despliegue.

Además, aunque se define en el propio RFC 9000 como un protocolo de transporte, también proporciona técnicas de seguridad, propias de capas superiores. En la Figura 2.3 se muestra la situación de QUIC en la pila de protocolos típicamente presentes en comunicaciones de Internet. Como se puede observar, su ubicación es más extensa que la de TCP, ya que, como se ha mencionado, presenta funcionalidades tanto de nivel de transporte (control de flujo, congestión, etc.) como de capas superiores.

Entre sus ventajas, podemos destacar dos principalmente. La primera de ellas es el *multistreaming* o la capacidad de tener diferentes *streams* o flujos de datos en una misma conexión. Esta característica de QUIC permite que cuando una pérdida afecta a un flujo concreto, el resto de ellos no se vean afectados. De este modo, se evita el bloqueo de la cabecera de línea (HOL *blocking*, en inglés), un fenómeno que limita enormemente el rendimiento del sistema. Esto es especialmente relevante en tráfico de web, donde sobre una misma conexión se pueden enviar datos referidos a objetos diferentes (p.e. partes de páginas web).

La siguiente gran ventaja es el hecho de que sus conexiones no estén ligadas a única ruta de red, es decir, tiene funcionalidad *multipath* de forma nativa. De esta manera, los paquetes QUIC pueden utilizar diferentes caminos hasta llegar a su destino, dependiendo de cuál sea el más óptimo en ese momento.

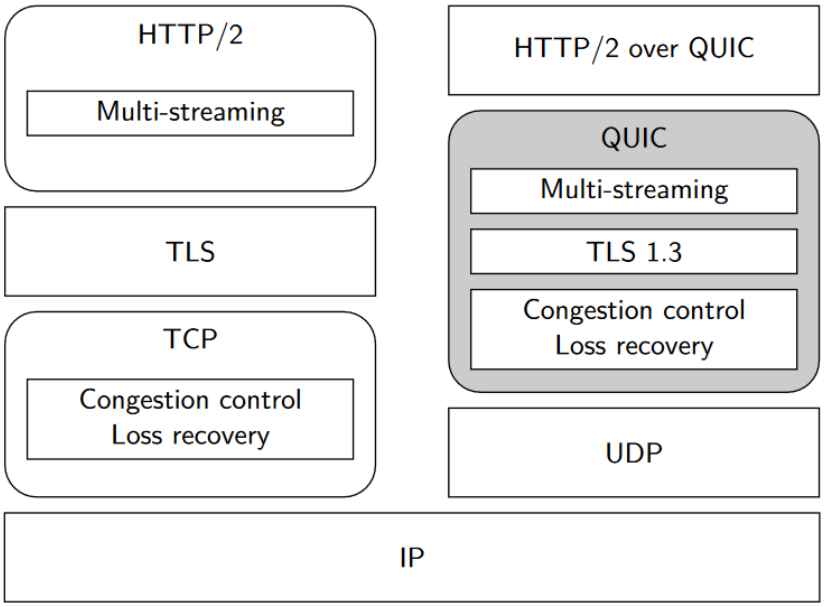


Figura 2.3: Arquitectura TCP vs QUIC [15].

El conjunto de estas características implica que, incluso en conexiones con cierta inestabilidad, se consiga reducir tanto la latencia como las pérdidas de datos. Esta es la principal razón que ha motivado a estudiar el protocolo QUIC en enlaces LMS (poco estables), analizando su funcionamiento en un entorno realista.

Una vez se conocen las cualidades más relevantes del protocolo para este trabajo, se detallará el encapsulado de los datos y el formato de un paquete QUIC. Este paso es crucial para poder implementar de manera correcta los nuevos algoritmos de *scheduling*, los cuales se explican en el Apartado 2.3.

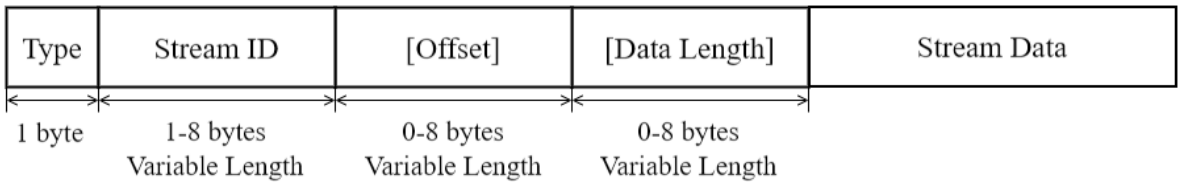


Figura 2.4: Formato del *Stream Frame*. Basado en [16].

Un paquete QUIC, al transportarse en mensajes UDP, se compone de la cabecera correspondiente UDP. Tras ella, existe otra cabecera, esta vez ya propia del protocolo QUIC. Seguido aparecen las tramas o *frames* de datos, las cuales pueden pertenecer a uno o varios *streams* diferentes. Este es un factor realmente importante en este trabajo, ya que conocer cómo está compuesto y cuáles son los posibles valores, resulta vital a la hora de la implementación de nuevos algoritmos de *scheduling* que gestionen el envío de datos de uno u otro *stream*. Cada trama (o llamado ya *Stream Frame*) tiene un formato propio, representado en la Figura 2.4. Se procede a explicar cada campo [16]:

- `Type`: tipo de *frame* a enviar. El conjunto de valores posibles va desde 0x08 hasta 0x0f (`0b00001XXX`), siendo los tres bits menos significativos los que determinan la naturaleza de los siguientes campos:

- Bit `OFF` (0x04): cuando está a 1, indica que existe el campo `Offset`. Cuando está a 0, significa que dicho campo está ausente –implica que estos son los primeros bytes del flujo actual.
- Bit `LEN` (0x02): cuando está a 1, indica que existe el campo `Data Length`. Cuando está a 0, significa que dicho campo está ausente, implicando que el campo de `Stream Data` se extiende hasta el final del paquete QUIC. Esto supone dos posibles casos: (1) solo hay información de este flujo a enviar –si es el primer *frame*; o (2) que no existen más tramas tras esta –si es la última.
- Bit `FIN` (0x01): marca que esta trama pertenece al final del *stream*. Por ende, el tamaño final del flujo es la suma entre los campos de su `Offset` y su `Data Length`.

- `Stream ID`: indica cuál es el identificador del flujo. Este identificador no solo señala el *stream* actual, sino que expone nueva información, como quién envía información (cliente o servidor) o el tipo de flujo. Su formato es de número entero de longitud variable.

  Los flujos pueden ser unidireccionales o bidireccionales, dependiendo de la dirección de los datos: si solo transporta datos del emisor al receptor (uni) o en ambas direcciones (bi). Para conocer qué papel tiene el extremo transmisor, este campo utiliza los dos bits menos significativos para indicar si se trata del cliente o el servidor, así como si es un flujo unidireccional o bidireccional. En la Tabla 2.1 se representan los cuatro posibles tipos.

Tabla 2.1: Tipos de `Stream ID` [16]-Tabla 1.

Bits	Tipo
0x00	Cliente como transmisor. Flujo bidireccional.
0x01	Servidor como transmisor. Flujo bidireccional.
0x02	Cliente como transmisor. Flujo unidireccional.
0x03	Servidor como transmisor. Flujo unidireccional.

- `Offset`: especifica el *offset* o desplazamiento de bytes en el *stream* para los datos en dicha trama. Si es la primera trama del flujo, este campo se omite, ya que sería 0. Este campo es necesario para conocer cuántos datos se han enviado de un *stream*. Su formato es de número entero de longitud variable.

- `Data Length`: especifica la longitud del campo de datos (`Stream Data`) de la trama actual. Si este *frame* es el único o el último del paquete, este campo se omite –se puede calcular con los demás campos de cabeceras. Su formato es de número entero de longitud variable.

- `Stream Data`: información (bytes) pertenecientes al *stream* a enviar.

Como se ha mencionado, varios de los anteriores campos no tienen una longitud fija, sino variable. Esta variabilidad depende justamente de su tamaño, puesto que se trata de campos donde su valor puede tomar un rango de posibilidades bastante amplio. Ya se ha comentado que se usan para definir campos

como el desplazamiento de bytes, en donde si se envían grandes ficheros su valor incrementa enormemente. En la Tabla 2.2 se presenta la codificación de este tipo de campos, en donde se muestra que su longitud puede variar desde 1 hasta 8 bytes, dependiendo su valor.

Tabla 2.2: Codificación de las cabeceras de longitud variable [16]-Tabla 4.

2 MSB	Longitud (bytes)	Bits disponibles	Rango
00	1	6	0-63
01	2	14	0-16383
10	4	30	0-1073741823
11	8	62	0-4611686018427387903

2.3. Algoritmos de *scheduling* implementados

Dentro de las comunicaciones en general existe un proceso el cual es crucial para la eficiencia de cualquier sistema. Es por ello que en este apartado se va a hablar de la planificación o, más conocido por su término anglosajón, del *scheduling*. Este componente es el responsable de gestionar cómo se comparten los diferentes recursos implicados en el envío de la información [29]. Existen múltiples algoritmos de *scheduling*, pero todos tienen un propósito claro, aparte de cumplir con los objetivos de diseño. Son técnicas que buscan maximizar el rendimiento mientras minimizan el tiempo de respuesta, es decir, conseguir el mejor resultado en el menor tiempo posible. Por ende, estos algoritmos son extremadamente eficientes (gracias a su sencillez), además de justos y predecibles.

En el caso concreto que nos ocupa, el *scheduling* se aplica a la selección de *stream* del cual enviar datos. Durante una conexión, la capacidad de envío de QUIC viene limitada por los algoritmos del control de flujo y de congestión. Dadas las altas capacidades de dispositivos, el control de flujo no suele ser un factor limitante, siendo el control de congestión la funcionalidad que dicta la capa real de transmisión. Ante una oportunidad de transmitir un paquete, el *scheduler* de QUIC selecciona datos de cada *stream* a enviar en el paquete. Estos flujos o *streams* se distribuyen, a su vez, en distintos *frames*. En esta sección se presentan los diferentes algoritmos de *scheduling* que se van a implementar, comenzando con el funcionamiento predeterminado de la implementación que se ha usado, el cual se basa en el algoritmo de *Round-Robin*. Tras ello, se hará una breve explicación sobre dos técnicas muy comunes como son *Fair Queuing* y *Weighted Fair Queuing*. Para finalizar con esta sección, se explicará de manera resumida en qué consiste el algoritmo propuesto en este trabajo. Esta técnica asegura estabilización en las colas de cada *stream*, una característica muy relevante para comunicaciones satelitales.

2.3.1. Round-Robin

El primer algoritmo que se va a explicar es el aplicado por defecto en la implementación de QUIC utilizada. Técnicamente, el algoritmo implementado no es un *Round-Robin*, pero al estar basado en él, se puede considerar que sí lo es. Se procede, entonces, a explicar cómo funciona dicha técnica.

El algoritmo RR se trata de uno de los más comunes cuando se trabajan con aplicaciones de mejor esfuerzo o *best-effort* –aquellas que no necesitan garantizar que sus datos lleguen al destino u ofrecer

determinados requisitos de calidad. La técnica del *Round-Robin* consiste en asignar turnos para el previo envío de manera circular. Es decir, cada proceso tiene el mismo tiempo de uso de los recursos. Dichos procesos, además, se organizan mediante colas *First In, First Out* (FIFO). De esta manera se asegura que los paquetes que llegan primero se envían antes, minimizando así los retardos [30].

Si trasladamos esta definición al ámbito de QUIC, esto significa que en cada paquete de datos envía únicamente información de un *stream* –dicho de otra manera, se envían diferentes *frames*, pero solamente de un único flujo. Si tras enviar este sigue teniendo datos pendientes de enviar, se posicionan al final de la cola, dejando paso primero a otros *streams*. Si por el contrario los datos a enviar no consiguen llenar el paquete QUIC, se completará dicho paquete con los datos del siguiente *stream*. De esta manera se realiza un reparto justo, en el cual se garantiza que se envían datos de todos los flujos implicados. También es una técnica equitativa, siempre y cuando todos *streams* posean la misma cantidad de información a enviar. En la Figura 2.5 se muestra un diagrama del funcionamiento de este algoritmo.

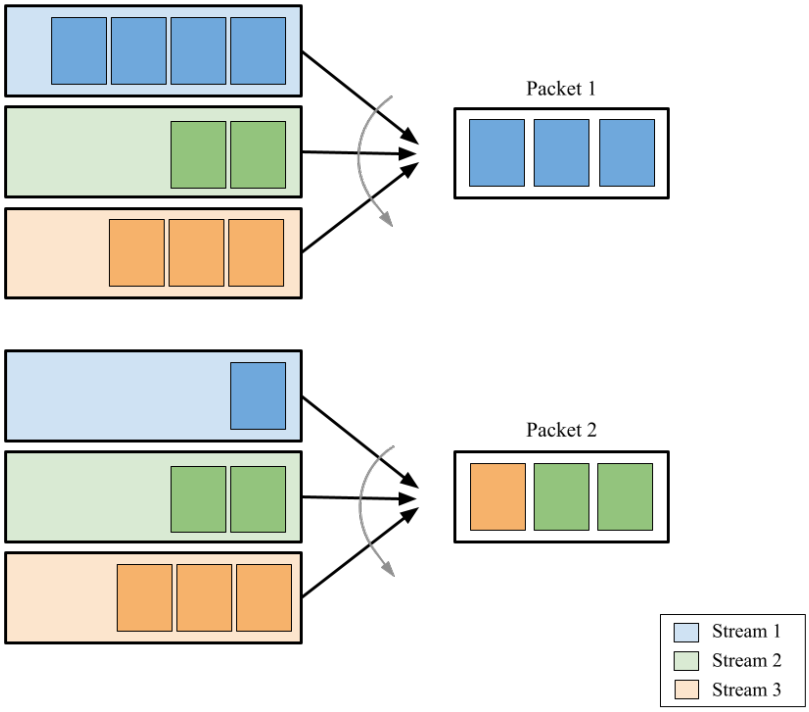


Figura 2.5: Diagrama del algoritmo RR.

En dicha figura, sin embargo, también se puede observar el primer problema de este algoritmo. Cuando el tamaño de la información difiere en exceso entre *streams*, un *buffer* puede crecer mucho mientras que otro se vacíe rápidamente. Este hecho deriva en problemas de eficiencia en el sistema. Además, esta técnica no permite dar prioridad a cierto tipo de información, como cuando se tienen paquetes de control –una funcionalidad muy útil para las comunicaciones satelitales. Es por estos motivos por los que se estudian otros algoritmos que permitan hacer dicha distinción.

2.3.2. Fair Queuing

El siguiente algoritmo a analizar es uno de los más utilizados a la hora de realizar un buen *scheduler*. La técnica del *Fair Queuing* es muy utilizada, sobre todo en los casos en los que existen grandes flujos heterogéneos en la red [31].

El algoritmo FQ destaca por su sencillez, ya que su objetivo es que todos los procesos posean las mismas capacidades y que, por tanto, envíen la misma cantidad de datos. Se trata de un algoritmo totalmente justo, tal como su nombre indica, en el que cada flujo recibe el mismo porcentaje de utilización de recursos [32]. Por lo tanto, la lógica a seguir sería la siguiente:

1. Calcular $\frac{C}{N}$
2. Mientras existan flujos en los que $r_i < \frac{C}{N}$:

$$C = C - \sum_{\forall i | r_i < \frac{C}{N}} r_i \quad ; \quad N = N - \sum_{\forall i | r_i < \frac{C}{N}} 1 \tag{2.1}$$

3. El resultado final es $r_i = \frac{C}{N}$

Por ejemplo, en el caso de existir cuatro flujos, la tasa binaria ofrecida a cada uno sería la cuarta parte de la total ($r_i = C/4$) –suponiendo que todos los flujos tienen capacidad suficiente de envío. Si alguno de los flujos no usa toda la capacidad, el remanente se reparte de forma equitativa entre el resto.

Se empezó a utilizar esta técnica en los *routers* para poder dotar de cierta capacidad de discriminación de flujos de datos en los momentos de sobrecarga. De esta manera, se aseguraba que todos los flujos que querían enviar información lo pudieran hacer, independientemente del tamaño del *buffer*. Además, su uso se hizo cada vez más popular al comprobar que era capaz de proporcionar una buena calidad de servicio en términos de latencia [33].

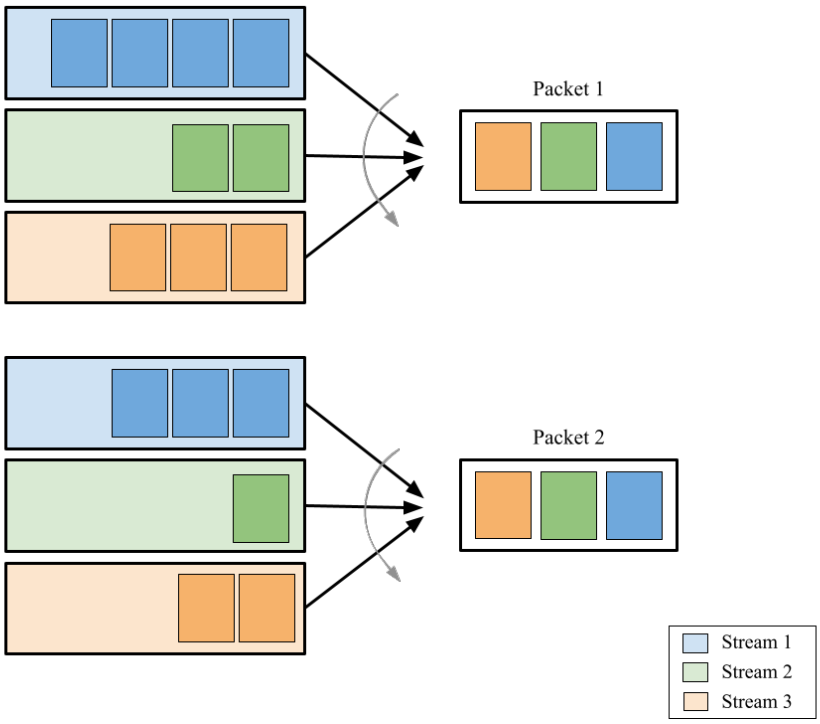


Figura 2.6: Diagrama del algoritmo FQ.

Para el protocolo QUIC el proceso es el mismo, tal y como se muestra en la Figura 2.6. En este caso, se observa como en ambos paquetes se envía la misma cantidad de *frames* provenientes de cada uno de

los *streams* disponibles. En el siguiente paquete ocurre exactamente lo mismo. Sin embargo, este envío implica que no quedan *frames* en el *buffer* del segundo flujo –se ha vaciado. Es por ello que la solución es repartir este recurso sobrante entre los otros dos, resultando en un tercer paquete que tuviese un *frame* y medio del primer y tercer flujo de datos, respectivamente. Este hecho es necesario para maximizar la eficiencia del sistema.

La desventaja principal del algoritmo FQ es que no tiene en cuenta la cantidad de datos existentes en los *buffers* de cada *stream*, por lo cual puede que uno de ellos crezca exponencialmente mientras que otros se vacíen rápidamente. En los casos donde los *streams* son poco homogéneos, implica grandes pérdidas en la eficiencia del sistema. Este inconveniente podría mitigarse dotando de prioridades a los flujos, pero al igual que el caso anterior, esta técnica no permite hacer uso de ellos. Es por este motivo por el que se quiso implementar el siguiente algoritmo a tratar.

2.3.3. Weighted Fair Queuing

La siguiente aproximación que se procede a explicar es la continuación directa del algoritmo *Fair Queuing*. Se trata de la técnica que permite dotar a los flujos de diferentes prioridades o pesos, siempre otorgando a los diferentes flujos la posibilidad de enviar datos. Conocido como *Weighted Fair Queuing*, es uno de los algoritmos más utilizados a la hora de distribuir los recursos de manera justa, teniendo en cuenta los flujos extremo a extremo –acercándose al principio E2E, ya comentado con anterioridad [31].

De nuevo y como ocurría en el algoritmo anterior, el propósito del WFQ reside en dotar a todos los flujos de la posibilidad de enviar sus *frames*. Vuelve a ser una técnica justa, en donde además se puede dotar a dichos *streams* de prioridades [32]. De esta manera, se mejora la eficiencia del sistema, siempre y cuando dichas prioridades se asignen correctamente. La lógica a seguir en este caso es muy similar a la anterior, con la única diferencia basada en tener en cuenta los pesos de cada flujo. Es decir, la tasa ofrecida se calcularía de la siguiente forma:

$$c_i = \frac{w_i \cdot C}{\sum_{\forall k} w_k} \tag{2.2}$$

En el caso en el que el *buffer* de uno de los *streams* no tuviese la cantidad necesaria como para rellenar su parte correspondiente, ocurriría lo mismo que en los casos anteriores. Es decir, sería necesario volver a repetir los cálculos para los siguientes flujos, teniendo en cuenta que la cantidad a repartir ahora es menor e ignorando el peso del flujo cuya tasa ya se ha asignado.

Si se añade esta funcionalidad en el protocolo QUIC, se observan varios cambios en la formación de los paquetes respecto a la técnica del apartado anterior. Estas variaciones se muestran en la Figura 2.7, en donde se puede ver cómo se gestiona la repartición de recursos. En este caso destaca uno de ellos, con una prioridad del 70%, siendo esta razón por la cual el último *stream* recibe más recursos que el resto –al igual que el primero tiene (ligeramente) más que el segundo. La misma lógica se sigue para transmitir el segundo paquete, en donde se observa que el tercer *stream* recibe tantos recursos como *frames* tiene. Por ende, sobra un recurso el cual se reparte entre los flujos restantes. El primer *stream* es el siguiente, puesto que su prioridad es mayor que la del segundo, enviando así el *frame* restante que le queda y el próximo. Por último, el segundo flujo vuelve a enviar lo que puede del *frame* que le quedó restante.

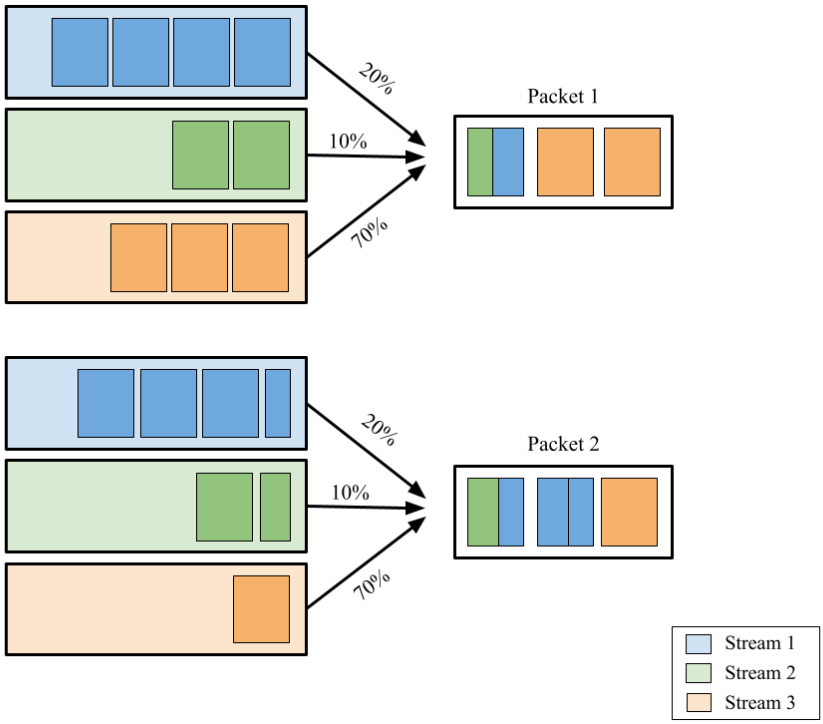


Figura 2.7: Diagrama del algoritmo WFQ. Se representan las prioridades de cada *stream* en porcentaje.

Sin embargo, este algoritmo no resulta del todo eficiente. En el caso propuesto en la anterior figura se observa que el primer flujo es el que más carga de datos posee, sin embargo, como su prioridad es baja, parece estancarse. Se vuelve, por tanto, a uno de los primeros problemas comentados. La eficiencia cae, ya que hay *buffers* que crecen exponencialmente mientras que otros se vacían con rapidez. Una solución a este inconveniente se basaría en la posibilidad de dar prioridades dinámicas. Siguiendo con esta línea de pensamiento se desarrolla el siguiente algoritmo.

2.3.4. Proposal Queuing

Debido a la inestabilidad existente en las comunicaciones entre satélites, en este trabajo se ha diseñado un algoritmo cuyo objetivo reside en la estabilización de las colas –basándose en la teoría de Lyapunov [34].

Esta técnica destaca por su sencillez, ya que se basa en los algoritmos de *max-weight*. Es decir, en cada instante de tiempo se observan las colas y, dependiendo de la capacidad de envío existente, se toma la decisión más óptima. La solución es única además de sencilla, ya que propone enviar todos los datos posibles de un mismo flujo, aquel que tenga una mayor ocupación en el *buffer*. No solo eso, sino que este algoritmo posee la funcionalidad de añadir penalizaciones en la toma de decisiones, dando así prioridades a los flujos y evitando los problemas comentados en los primeros algoritmos. Por ende, la solución vuelve a ser única: encontrar el *stream* con mayor coeficiente y enviarlo –derivando así en un *tradeoff* entre la ocupación de las colas en los diferentes flujos y la penalización de los mismos.

A continuación se muestra el problema resultante de aplicar la mencionada teoría de Lyapunov. La decisión en cada *slot* temporal ($\alpha(t)$) que se ha de tomar es la que minimice el resultado entre las penalizaciones ($p(t)$) con sus respectivos pesos (V) y la ocupación de las colas o capacidad de envío.

$$\min_{\alpha(t)} \quad V \cdot p(t) - \sum_{k=1}^{N} Q_k(t) \cdot b_k(t) \tag{2.3}$$

siendo $Q_k(t)$ y $b_k(t)$ el valor de la ocupación de la cola y las salidas de la cola del *stream* k en el *slot* t como consecuencia de la decisión tomada, respectivamente.

Como se ha mencionado, esta solución se corresponde con el algoritmo de *max-weight*, que se ha aplicado tradicionalmente a gestión de recursos radio. Asumiendo que el tráfico medio de entrada a las colas no excede la capacidad media del sistema, este algoritmo garantiza que las colas son *mean-rate-stable* [34], que se se define como:

$$\lim_{T\to\infty} \quad \frac{1}{T}\,\mathbb{E}\{Q_k(t)\} = 0 \tag{2.4}$$

En nuestro caso, la variable a analizar es la cola del *stream* en diferentes instantes temporales. Como se cumple la condición anterior en todos los casos, se puede deducir que el algoritmo propuesto es capaz de estabilizar las colas (de los diferentes flujos) a lo largo del tiempo.

En la Figura 2.8 se observa la implementación de este algoritmo en un entorno con el protocolo QUIC. En el primer paquete se puede ver que solo existen *frames* de un único *stream*, aquel que tiene mayor ocupación en su *buffer* –teniendo en cuenta que su penalización no es muy alta. En el segundo paquete se aplica la misma teoría, eligiendo el segundo flujo a enviar –a pesar que el tercero tenga más datos encolados, su penalización es más elevada. En este caso, sin embargo, el flujo a enviar no posee tanta información, por lo que quedan recursos disponibles. Es por ello que el tercer flujo es capaz de enviar información también.

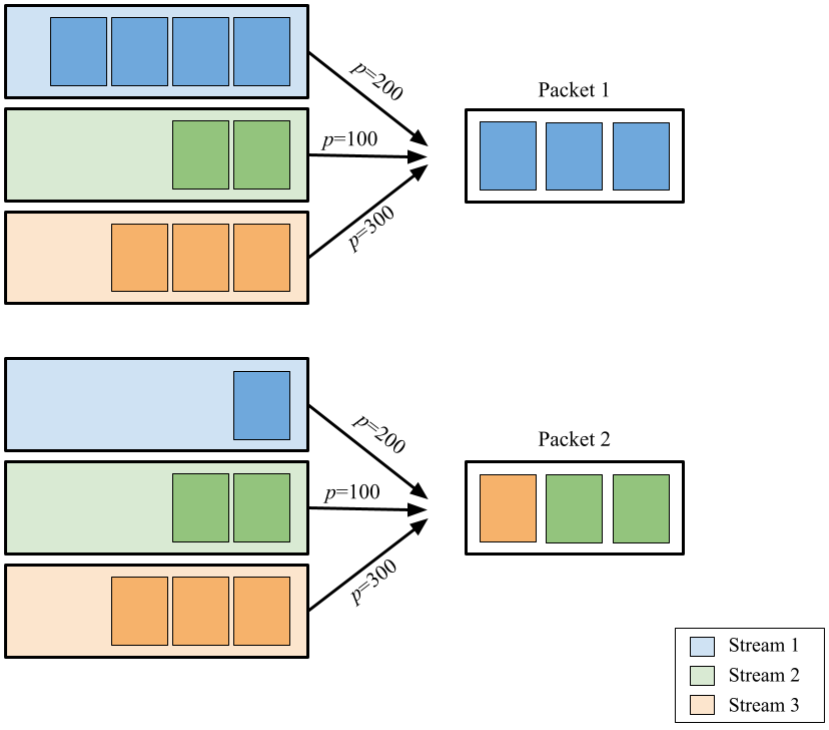


Figura 2.8: Diagrama del algoritmo *Proposal Queuing*. Se representan las penalizaciones de cada *stream*.

Si se compara este diagrama con el que se vio en la Figura 2.5 se puede observar que el resultado (paquetes) es el mismo. No obstante, con el paso del tiempo y en casos más complejos, el algoritmo propuesto es capaz de estabilizar las colas, de forma que las decisiones tomadas se adaptarán al estado de las mismas. Con este algoritmo se busca tener una solución adaptativa, de tal forma que tenga un comportamiento promedio similar al WFQ suponiendo que los pesos de este se fueran adaptando a la variación de tráfico.

Capítulo 3

Diseño, desarrollo e implementación

Una vez se conoce el contexto del trabajo, así como las motivaciones tras la realización del mismo, se procede a implementar todas las tecnologías anteriores en un único entorno. De esta manera, se pretende estudiar los resultados obtenidos para así entender las posibles ventajas que ofrece aunar dichas tecnologías.

La propia implementación consta de tres fases diferenciadas entre sí. La primera de ellas es el desarrollo de un entorno, de manera que todas las herramientas funcionen de tal forma que se emule un enlace LMS de una NTN –comunicación entre una estación terrena y un satélite. Para ello será necesario explicar cuáles son dichas herramientas, así como su conexión. La segunda fase se centra en entender dicho entorno, primero catalogando los diferentes modelos de canales en los que se realizarán las medidas, para luego definir el propio tráfico que se va a medir. Por este motivo, se prepararán diferentes escenarios de donde se obtendrán los ficheros de resultados a analizar. Por último, la última fase de implementación se centra en el *scheduler* y en cuáles son los cambios a realizar con el objetivo de mejorar el rendimiento del sistema anterior. Este punto del trabajo se basará en uno de los escenarios anteriores, analizando las diferencias entre los algoritmos que se desean implementar.

3.1. Entorno

La plataforma utilizada para la evaluación, tanto de los protocolos como de la eficiencia del sistema, se muestra en la Figura 3.1. Se utilizará este dibujo como refuerzo visual, de forma que se entienda mejor el entorno de trabajo. Esta es una plataforma compleja que combina la implementación de una parte real (protocolos de transporte) con una simulada (modelo de canal). Para ello resulta crucial el uso de dos herramientas: los contenedores Docker y el simulador de redes NS-3.

Docker[1] se define como una plataforma *software* que permite el despliegue de múltiples aplicaciones rápidamente. Su base radica en la utilización de "contenedores", una forma de aislamiento de procesos o virtualización ligera. Estos contenedores incluyen todos los recursos necesarios para que el usuario los utilice como servidores virtuales en la nube. De esta manera, Docker proporciona una serie de sencillos comandos para que el usuario pueda crear, personalizar, iniciar o detener dichos contenedores, ya que todo es configurable.

[1] `https://www.docker.com/`

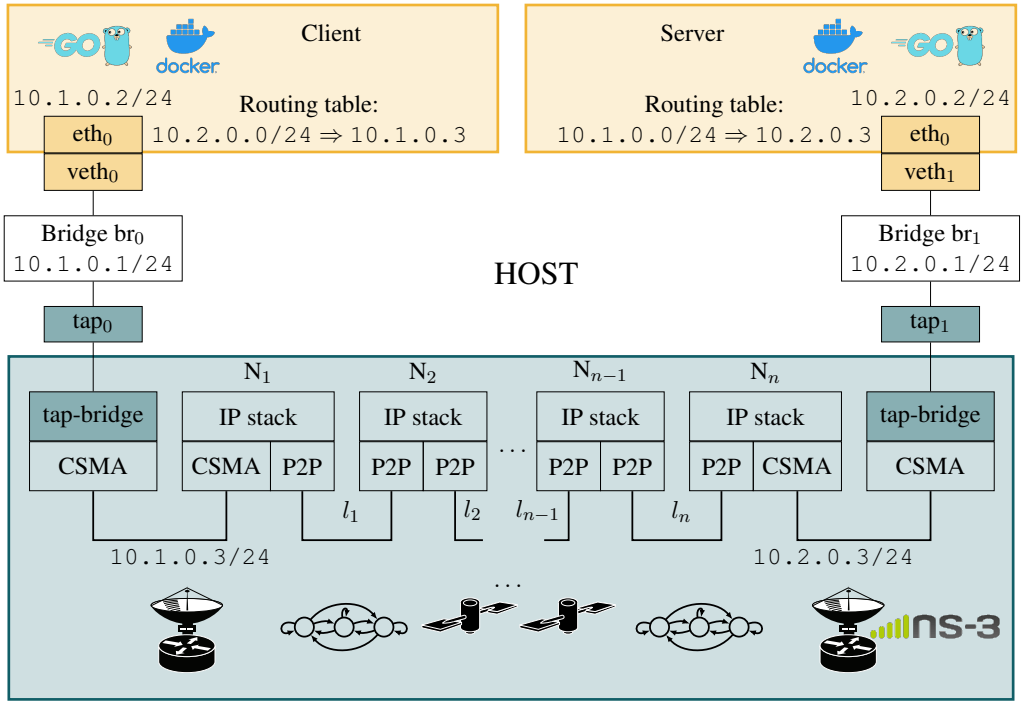


Figura 3.1: Diagrama del entorno en donde se hará el pertinente banco de pruebas. Integra las herramientas NS-3 y contenedores Docker.

En este caso se utilizan los contenedores Docker para empaquetar todas las aplicaciones que van a generar y consumir tráfico, tal y como se ve en la parte superior de la figura anterior. En este caso, cliente y servidor respectivamente. El tráfico será TCP o QUIC, dependiendo del caso de uso. Además, dichas redes Docker están totalmente aisladas la una de la otra, por lo que solamente son accesibles entre sí mediante la utilización de puentes virtuales (*bridges*) –generados por el propio motor de Docker. También se muestra la dirección *Internet Protocol* (IP) y puerto tanto del cliente como del servidor, así como su tabla de encaminamiento, en donde se puede ver la ruta que deben recorrer los datos.

La siguiente herramienta, el NS-3, se trata de un simulador de redes de eventos discretos [35]. Es un programa de *software* abierto y libre, el cual permite desarrollar un entorno de simulación en función de las necesidades del usuario. El uso de este simulador reside, sobre todo, en el campo de investigación y en el educativo.

En el contexto de este trabajo, se utiliza para simular el modelo del canal del enlace –parte inferior de la ya mencionada figura. Para ello, se explotan sus capacidades con el objetivo de realizar simulaciones en tiempo real, valiéndose del reloj del sistema en vez del reloj interno propio del NS-3. Los puentes virtuales creados por Docker son los que, mediante unos dispositivos virtuales llamados *tap devices*, se comunican con este entorno de simulación. Sin embargo, antes de poder llegar al primer nodo (N_1), debe atravesar un enlace *Carrier Sense Multiple Access* (CSMA). Dicho enlace se utiliza para conectar el *tap-brige*, correspondiente al entorno *dockerizado*, con el propio nodo simulado. Sus características, no obstante, son ideales: capacidad de transmisión infinita y retardo nulo –resultando así en un enlace totalmente transparente para el tráfico. De esta manera, se simula una estación terrena gracias a la unión del contenedor de la izquierda y el propio nodo N_1. La misma configuración se realiza al final de la cadena, conectando el servidor (contenedor de la derecha) con el último nodo (N_n).

Cabe destacar que se ha comprobado que esta cadena o configuración es correcta y funcional, ya que se ha utilizado en investigaciones previas como [36] o [37], publicaciones pertenecientes al mismo grupo de trabajo donde se desarrolla este proyecto. Además, gracias a la utilización de los contenedores y todos los elementos intermedios (*bridges* y *taps*), se consigue que el tráfico solo pueda ir por un camino, asegurando así que el intercambio de información entre aplicaciones pase por el simulador.

3.2. Canal del enlace

El siguiente apartado se centra en el ya comentado modelo de canal del enlace. Este enlace se sitúa en medio de las aplicaciones cliente-servidor, de tal forma que simula una comunicación entre satélites LEO. La topología subyacente contempla dos posibles enlaces distintos: LMS e *Inter-Satellite Link* (ISL). El primer tipo se refiere a aquel enlace que se establece entre una estación terrena y un satélite LEO, de ahí su nombre. Esta comunicación se da únicamente al principio y al final del intercambio, es decir, entre la estación terrena inicial con el primer satélite, conocido como *Uplink* (UL) o entre el último satélite con la estación terrena final, denominado *Downlink* (DL). El segundo tipo abarca las demás comunicaciones, en otras palabras, los enlaces entre dos satélites LEO consecutivos de la correspondiente constelación. Estos satélites se representan en forma de nodos en la Figura 3.1 –excepto el primero y el último, ya que como se ha mencionado, se corresponden a las estaciones terrenas. La unión entre dichos satélites (o nodos) se corresponden a enlaces punto a punto, los cuales tienen diferentes propiedades dependiendo del tipo de enlace (si son LMS o ISL).

El modelo aplicado al enlace LMS se basa en una cadena de Markov, mostrado en la Figura 3.2. Este modelo LMS (tomado del estudio en [38]) considera que los enlaces pueden experimentar diferentes condiciones que dificulten el intercambio de datos, así como el deterioro en la calidad del mismo. Teniendo en cuenta estas adversidades, se consideran tres posibles estados: (l) LoS, implicando un entorno ideal donde se pueden ver los satélites sin obstáculos; (m) *Mid Shadowing*, en donde las condiciones empeoran; y (d) *Deep Shadowing*, el peor caso posible, en donde la conexión está gravemente afectada. Aparte de estos tres estados, se representan en las transiciones entre ellos mediante unas probabilidades estadísticas regidas por una matriz de transición que se analizará más adelante.

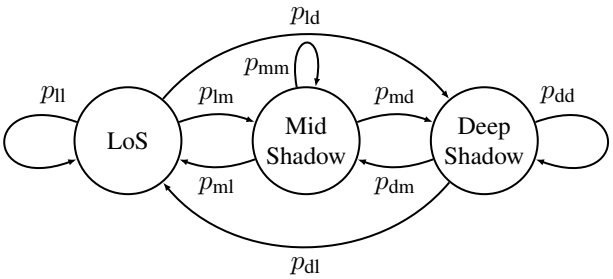


Figura 3.2: Modelo de cadena de Markov para los canales satelitales UL/DL.

Es importante mencionar que este modelo ya ha sido estudiado bajo un tráfico sintético, concretamente de *Poisson*, en trabajos anteriores del Grupo de Ingeniería Telemática (GIT); como en [39] y [40]. En ellos se analizó tanto los retardos como la probabilidad de pérdida de datos en diferentes situaciones, con cambios en las tasas binarias y en el tamaño del *buffer* del canal. Este trabajo, en contraposición a los anteriores, se centra en la evaluación del comportamiento tanto del protocolo TCP como de QUIC con tráfico real.

Volviendo a la utilización de la herramienta NS-3 para la simulación de estos enlaces, se procede a explicar cuáles son las diferencias entre los dos tipos (LMS e ISL), así como sus respectivas implementaciones en dicho programa.

El primer canal, destinado a la comunicación entre la estación base y un satélite LEO, se integra modificando la implementación de dispositivos punto a punto del propio programa NS-3. El cambio más relevante respecto a la implementación predeterminada se produce en la tasa de transmisión. Gracias a este valor, y en conjunto con la longitud del paquete, el simulador planifica las transmisiones de dichos paquetes a través de la interfaz. Por defecto, esta tasa se configura al principio de la simulación y permanece constante a lo largo de la misma. Sin embargo, se ha adoptado una versión modificada capaz de variar la tasa de transmisión durante la propia simulación. Para ello se genera una matriz de transición, la cual indica (en forma de probabilidades) la tasa binaria seleccionada hasta el siguiente cambio. De esta manera, se consigue implementar esa variabilidad en las tasas, recreando aquellos posibles estados dependiendo de las adversidades encontradas en el entorno realista.

El segundo canal está dedicado a la comunicación entre satélites. En este caso se modela siguiendo el funcionamiento por defecto del dispositivo punto a punto de la herramienta NS-3. En otras palabras, se utiliza una tasa de transmisión constante (configurada al principio de la simulación, pero inalterada a lo largo del mismo). Además, la implementación de NS-3 permite añadir un retraso determinista adicional que tuviera en cuenta el retardo de propagación entre satélites.

Una vez comprendido los modelos de canal que se van a analizar, y teniendo en mente que el objetivo de este Trabajo Fin de Máster se centra en estudiar los protocolos TCP y QUIC, se procede a mostrar la configuración elegida para dicho análisis. Para ello se evalúan dos bandas de frecuencias: la banda *Ka* (rango de frecuencias entre 26,5 `GHz` y 40 `GHz`) y la banda *S* (con un rango muy inferior, desde 2 hasta 4 `GHz`). Se eligen estas bandas ya que son bandas con rangos muy diferentes y, por tanto, tasas binarias que difieren enormemente. Asimismo, son bandas muy utilizadas en las comunicaciones satelitales, sobre todo la banda *Ka*. En la Tabla 3.1 se muestran las características precisas para las diferentes configuraciones a tratar. De esta manera, se consigue realizar un estudio completo del sistema.

Tabla 3.1: Configuración de los canales en las diferentes bandas.

Banda *S*	
Tasa LMS	$[4, 2, 0.8]$ `Mbps`
Matriz de transición LMS	$\mathcal{P} = \begin{pmatrix} 0 & 0.94076 & 0.059243 \\ 0.77084 & 0 & 0.22916 \\ 0.49418 & 0.50582 & 0 \end{pmatrix}$
Tiempo de permanencia LMS	$[0.5485, 0.4992, 0.3529]$ `s`
Duración del *slot* temporal	$\delta = 100$ `ms`
Banda *Ka*	
Tasa LMS	$[80, 40, 16]$ `Mbps`
Matriz de transición LMS	$\mathcal{P} = \begin{pmatrix} 0 & 0.93156 & 0.068437 \\ 0.34526 & 0 & 0.65474 \\ 0.070012 & 0.92999 & 0 \end{pmatrix}$
Tiempo de permanencia LMS	$[0.2530, 0.7299, 0.1666]$ `s`
Duración del *slot* temporal	$\delta = 100$ `ms`

El modelo LMS descrito en [38] define las matrices de probabilidad de transmisión que caracterizan a ambas bandas, tal y como se puede ver en la Tabla 3.1. Es importante destacar que estas transiciones de estado se producen únicamente en determinados intervalos periódicos. El modelo anterior también establece el tiempo de permanencia medio de cada estado, sin embargo, no proporciona la capacidad de transmisión en cada uno de ellos. Es por ello que en este trabajo se opta por utilizar las siguientes tasas máximas de 4 y 80 `Mbps` –para la banda *S* y banda *Ka*, respectivamente. Estas tasas corresponden al estado de LoS, mientras que los dos estados restantes se calculan como el 50% (*Mid Shadowing*) y 20% (*Deep Shadowing*) de la capacidad máxima anterior. Estos valores resultantes se muestran en la Tabla 3.1. La duración del *slot* temporal para calcular los tiempos medios en cada estado se define en todos los casos de 100 `ms`. Se recuerda que todos estos valores son variables, por lo que resulta sencillo hacer nuevas configuraciones en el caso que fuese necesario.

De esta manera se consigue simular los canales satelitales UL/DL, es decir, de subida o de bajada. La concatenación de los mismos implica un entorno *End to End* que abarcaría la comunicación entre cliente y servidor situados en tierra, pasando por un número arbitrario de satélites.

3.3. Tráfico de datos

Después de comprender cómo es el sistema en el que se va a trabajar, se puede entrar en detalle de qué es lo que hace realmente el cliente y el servidor. Estas aplicaciones están implementadas en el lenguaje de programación Go[2] –también conocido como "Golang". Este novedoso lenguaje fue diseñado por Google, de ahí que en múltiples ocasiones se le llame "Google Go", para cubrir las necesidades de la propia empresa en torno a los servicios *Fog/Cloud*. Es un lenguaje orientado a objetos, basado en C (rendimiento alto), el cual busca ser fácil de entender y utilizar; más parecido a Python en dicho aspecto. La elección de este lenguaje vino porque la implementación de QUIC utilizada en el trabajo también está implementada en Go. En concreto se trata de `QUIC-go`[3], versión 0.15.12. Por otro lado, la implementación de TCP es la propia del *kernel*, por lo que no impone limitaciones en cuanto al lenguaje de programación. Por razones prácticas, todas las aplicaciones, tanto usando QUIC como TCP, se han realizado en Go.

Las aplicaciones anteriores actúan de la siguiente manera. El servidor escucha hasta que recibe un petición de conexión y la acepta. Tras ello, el cliente comienza a enviar su fichero de datos en diferentes *streams* o flujos, dependiendo del caso. Mientras tanto, el servidor recibe cada flujo por separado y de manera ordenada. Para poder realizar el pertinente banco de pruebas existe un fichero de configuración en el cual se dicta cuántos *streams* se van a utilizar, así como el protocolo a usar, el tamaño del fichero a enviar y la tasa binaria de la transmisión.

En este fichero también se puede definir la distribución seleccionada para el tráfico a enviar. Es decir, el cliente puede utilizar dos distribuciones diferentes: fija o *fixed* y de *Poisson*. En el primer caso se podría considerar que no existe distribución *per se*, puesto que lo que supone es un valor de tasa binaria de envío constante. Este caso es útil para poder valorar los diferentes protocolos en un entorno ideal. Es por ello que se implementa también un tráfico con distribución de *Poisson*. En este caso se utiliza un tráfico sintético, mediante una distribución de probabilidad discreta mediante la cual se definen con independencia del tiempo transcurrido entre sí, en función de una tasa media configurada.

[2] `https://go.dev/`
[3] `https://github.com/lucas-clemente/quic-go`

Como pasó en la sección anterior, una vez comprendida la capa de aplicación, así como las distribuciones de tráfico implementadas, es necesario comentar la configuración elegida para este punto –tanto de la capa de aplicación como la capacidad máxima del *buffer* en el modelo del canal. Como refuerzo se muestra en la Tabla 3.2 los parámetros y valores escogidos. En primer lugar, en la tabla se indican los diferentes tamaños de *buffer* a utilizar en el enlace satelital, de forma que pueda haber desbordamiento y, por tanto, pérdidas de paquetes que haya que recuperar. Asimismo, se indican las tasas de generación de tráfico en cada banda y los tamaños de fichero que aseguran que los tiempos de envío sea semejantes independientemente de las tasas.

Tabla 3.2: Configuración de la capa de aplicación.

Capa de aplicación y *buffer*	
Tamaño del *buffer*	$\{7, 15, \infty\}$
Tasa (datos)	*S*: 2.48 `Mbps`; *Ka*: 45.32 `Mbps`
Tamaño fichero (datos)	*S*: 60 `MB`; *Ka*: 800 `MB`
Tamaño del paquete	1000 `Bytes`
QUIC #*streams*	$\{1, 2, 3\}$

Con estas configuraciones el cliente envía un fichero de datos de tamaño lo suficientemente grande como para poder visualizar el impacto de las variaciones en la tasa en el canal LMS; es decir, se asegura que se pase un número elevado de veces por cada estado del canal. En el caso de la banda *Ka* es un fichero de 800 `MB`, mucho mayor que el de la banda *S* –cuanto mayor es la tasa binaria, mayor debe ser el fichero. Este hecho se debe a que, como ya se ha mencionado, las transiciones de estado de la matriz anterior son periódicas e independientes a variaciones en las tasas binarias. Por ende, se necesita un tiempo de transferencia de datos elevado en el caso que las tasas binarias sean grandes. Continuando con la configuración el tamaño de paquete se mantiene igual en ambos casos. Volviendo a la tasa de envío en la aplicación, en ambas bandas, se utiliza una tasa cercana a la capacidad media del canal, calculada de la siguiente forma:

$$\overline{Rate}\ [\texttt{Mbps}] = \frac{t_l\ [\texttt{s}] \cdot rate_l\ [\texttt{Mbps}] + t_m\ [\texttt{s}] \cdot rate_m\ [\texttt{Mbps}] + t_d\ [\texttt{s}] \cdot rate_d\ [\texttt{Mbps}]}{t_{total}\ [\texttt{s}]} \tag{3.1}$$

$$\overline{Rate}_{bandS} = \frac{0.5485 \cdot 4 + 0.4992 \cdot 2 + 0.3529 \cdot 0.8}{1.4006} = 2.4809\ \texttt{Mbps}$$

$$\overline{Rate}_{bandKa} = \frac{0.2530 \cdot 80 + 0.7299 \cdot 40 + 0.1666 \cdot 16}{1.1495} = 45.3254\ \texttt{Mbps}$$

Para concluir con el análisis, se configura el protocolo QUIC con un número diferente de *streams* o flujos en los que repartirse el envío anterior. La finalidad es poder estudiar el beneficio de la función *multistreaming* que posee este protocolo, viendo sus ventajas o desventajas frente al protocolo TCP.

3.3.1. Ficheros de resultados

Para llevar a cabo la evaluación, resulta necesario comprobar que todo es enviado y recibido de la manera correcta. Es por ello que ambas aplicaciones generan unos archivos de registro (más conocidos como *logs*) en donde se almacenan los *timestamps* (registros temporales) así como el número de bytes

enviados/recibidos en dicho instante, dependiendo de si se trata del cliente o del servidor. Gracias a estos ficheros se puede obtener tanto la tasa binaria media del sistema como los retardos ocurridos en el transcurso del intercambio, utilizando dos simples fórmulas:

$$throughput = \frac{bitsReceived}{timestamp_{lastpkt} - timestamp_{firstpkt}} \quad [Mbps] \tag{3.2}$$

$$delay = \frac{\sum timestamp_{server} - \sum timestamp_{client}}{\#pkts} \quad [us] \tag{3.3}$$

Además, dentro del enlace simulado por el NS-3 se generan otros ficheros muy importantes a la hora de entender cómo funciona el control de congestión. En este sentido cabe indicar que tanto para QUIC como para TCP se usa el algoritmo CUBIC [41], que es la solución por defecto en la mayoría de implementaciones. El primer archivo generado desde NS-3 proporciona la ocupación del *buffer* a lo largo del tiempo. Asimismo, en otro fichero se almacena la evolución de la capacidad del canal a medida que transita entre los diferentes estados. Se recuerda que pueden ser tres estados, regidos por la matriz de transición LMS de cada banda (vistas en la Tabla 3.2): máxima capacidad (LoS), media (*Mid Shadowing*) o mínima (*Deep Shadowing*). Por último, el cliente también guarda los cambios de la ventana de congestión, de forma que se pueda comparar con los cambios en la capacidad del canal.

Gracias a todos estos resultados, se pueden comparar todas las casuísticas necesarias para analizar cómo es el rendimiento de los diferentes protocolos en entornos satelitales, tal y como se desarrollará en el Capítulo 4.

3.4. Modelo de servicio: Scheduler

Tal y como se ha ido mencionado a lo largo de esta memoria, no solo se desea estudiar cómo se comportan los protocolos de transporte en entornos satelitales, sino que también se propone mejorar el rendimiento de QUIC aprovechando su capacidad de gestionar varios flujos o *streams*. Es por ello que esta sección se centra en el ya comentado planificador o *scheduler*, el ente que se encarga del reparto de los datos en los diferentes paquetes. Saber cómo funciona este dispositivo es muy útil a la hora de entender el funcionamiento del propio protocolo, incluso llegar a mejorarlo –dependiendo del tipo de información a enviar o cómo sea el canal, controlar cómo se realiza dicha planificación puede ser determinante. Cabe indicar que, aunque la siguiente información es propia de la implementación *quic-go*, la lógica sigue la especificación, por lo que es de esperar una solución semejante en otras implementaciones.

En la Figura 3.3 se muestra, en forma de diagrama de flujo, el contexto del modelo del planificador. En este diagrama se puede ver la utilidad del *scheduler*, así como el recorrido que se debe seguir hasta llegar a él. Tal y como se observa, todo comienza cuando se desea enviar un paquete QUIC. Ya se conoce que este protocolo tiene como objetivo mejorar las latencias entre servidor y usuario, siendo en este caso extremadamente importante cómo se realiza el establecimiento de la conexión, o comúnmente conocido como *handshake*. La estructura tras ello es tal que permite el intercambio de datos de la aplicación lo antes posible, sin importar si dicho establecimiento está confirmado o no. En otras palabras, se evita la necesidad de respuesta por parte del servidor antes del envío, permitiendo así que el cliente transmita sin esperas. Esto implica la creación de una nueva opción basada en el tiempo de ida y vuelta o *Round-Trip*

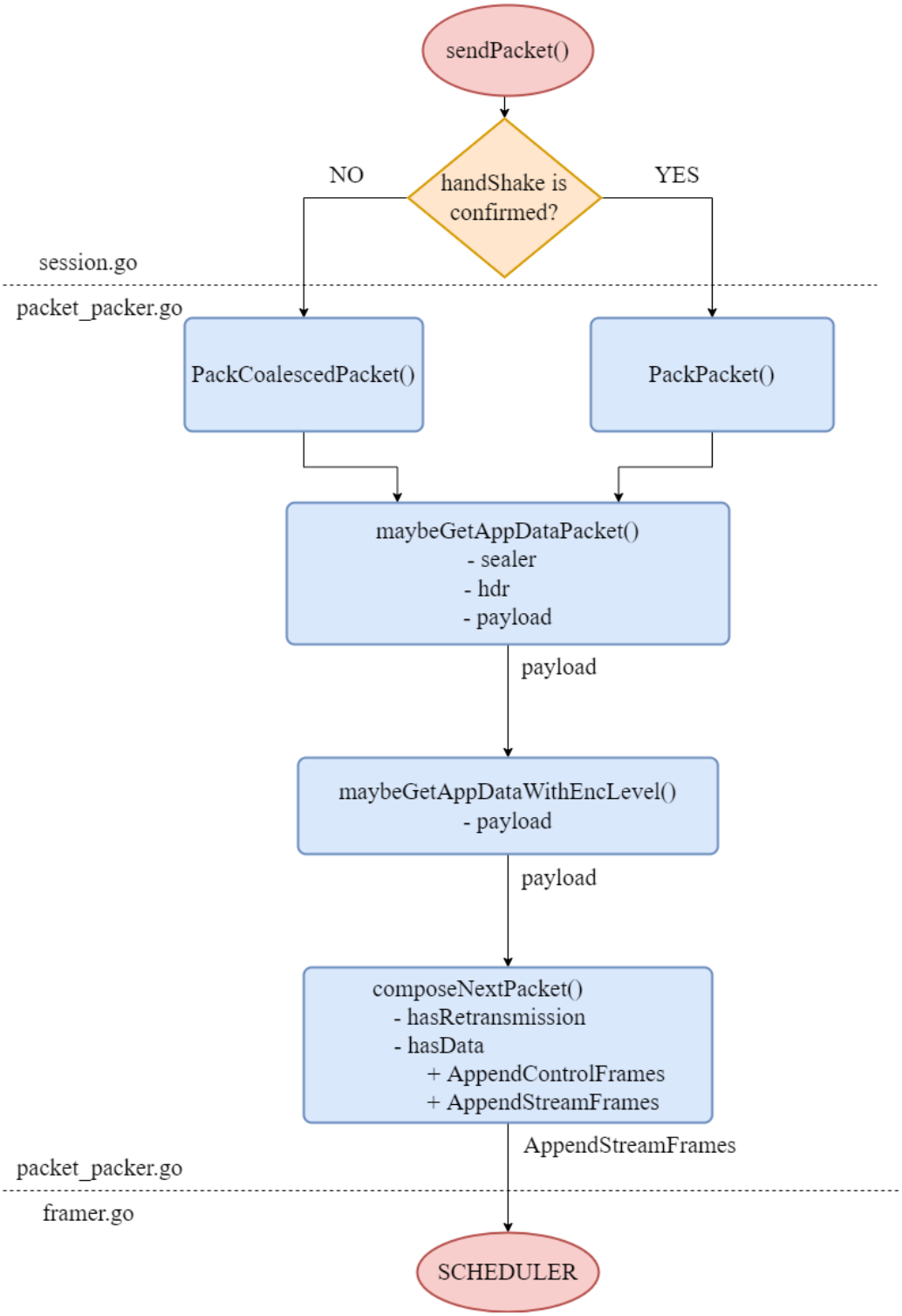


Figura 3.3: Diagrama de flujo: creación del paquete QUIC.

Time (RTT), llamada 0-RTT –para su utilización, se necesita una comunicación o configuración previa. Sin embargo, usarlo conlleva a una desventaja significativa en la seguridad, puesto que no posee protección ante ataques de repetición. En ambos casos, el paso siguiente se basa en una función la cual genera el paquete que se desea enviar. La diferencia entre las opciones anteriores radica en cómo se crea este paquete. En el caso en el que sí se haya confirmado el establecimiento de la conexión, solo será necesario empaquetar el contenido a enviar en el campo de datos de aplicación del paquete gracias a una función llamada `PackPacket()`. Sin embargo, en la opción 0-RTT no solo empaqueta datos de aplicación, sino que también puede generar paquetes de otro tipo (*Initial* o *Handshake*). En este caso, la función a llamar sería `PackCoalescedPacket()`. El objetivo es unir diferentes tipos de paquetes QUIC en un solo mensaje UDP, facilitando así el envío de paquetes total.

Pese a estas diferencias, como en los dos casos se necesita generar el paquete de datos de aplicación, ambos siguen el mismo camino hacia la siguiente función llamada `maybeGetAppDataPacket()`. Esta función es la encargada de generar tres objetos: el sellador o *sealer*, cabeceras o *hdr* y por último, el *payload*. Como el *scheduler* se enfoca en el apartado de los datos y los dos primeros son campos relacionados con la seguridad (en concreto, con el cifrado), el camino continúa en cómo se forma el campo de *payload*. Para ello, es necesaria otra función encargada de cifrar dicho apartado, llamada `maybeGetAppDataPacketWithEncLevel()`, la cual debe recurrir a la función `composeNextPacket()` para generar el paquete de datos (después se le aplicaría la seguridad ya mencionada). Esta función es realmente donde se genera el campo de *payload*, por lo que lo primero que debe hacer es revisar si existen datos de aplicación que reenviar (*retransmmissions*). Tras ello, en el caso que estos datos no sean suficientes, se comprueba el tamaño de los datos existentes en el *buffer* o cola, esperando a ser enviados. El paquete generado se compone de dos partes: la parte de control (`AppendControlFrames`) y la parte de datos de *stream* (`AppendStreamFrames`). El *scheduler* es quien se encarga de la planificación a la hora del envío de datos, por lo que el conjunto de *Stream Frames* se generarán por parte de este último componente.

Sin embargo, antes de comenzar con las explicaciones de las diferentes implementaciones, es necesario comentar un aspecto muy relevante encontrado en el desarrollo de aplicaciones sobre el protocolo QUIC. En concreto, dicha problemática surge en el envío de la información por parte del cliente hacia el servidor. El protocolo QUIC, aunque comúnmente se considera como un protocolo de transporte (y así se define en el RFC 9000), realmente es de sesión/aplicación –ya que el protocolo de transporte en el que se basa es UDP. Mientras que UDP posee tanto un *buffer* de recepción (lectura) como de envío (escritura), en la capa de aplicación ese *buffer* no existe. Es más, el *buffer* de escritura de UDP ni si quiera se tiene en cuenta en el envío de datos QUIC. La ausencia de este *buffer* se puede traducir en bloqueos en el transmisor, ya que la función de envío es bloqueante. De este modo, si en el momento en que nuestra aplicación quiere enviar hay datos pendientes de ser encapsulados, esta se queda esperando hasta que se libere espacio para hacerlo. Por ende, el cliente es incapaz de enviar la información a la misma tasa binaria que se ha establecido en su configuración, sino que debe hacerlo a una tasa menor.

En otras palabras, al desarrollar una aplicación sobre QUIC (como protocolo de capa de aplicación) es esta entidad quien debe responsabilizarse de la gestión del envío de datos. Por este motivo, se ha diseñado e implementado un *buffer* a dicho nivel, uno por cada *stream* de datos utilizado. De esta manera, el cliente puede realizar el envío a la tasa binaria anteriormente establecida. Es en dicho *buffer* donde se realiza

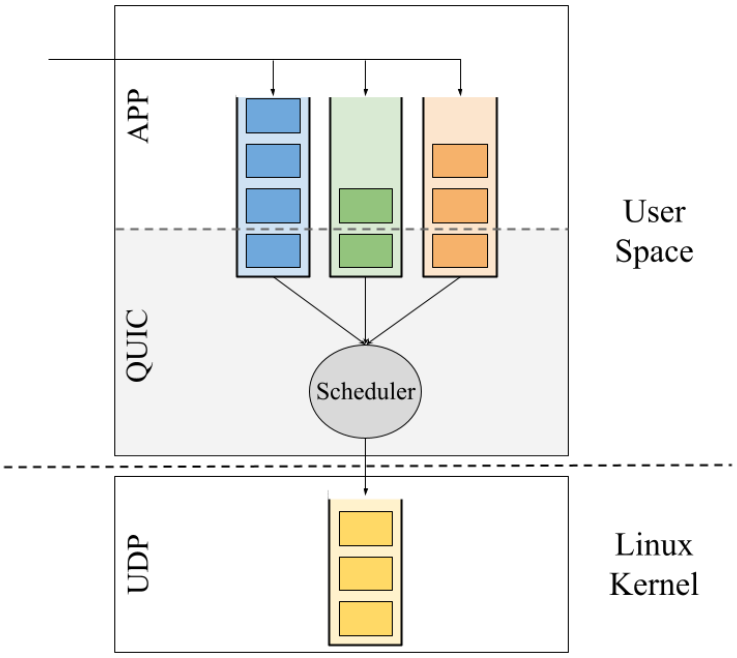


Figura 3.4: Arquitectura de los *buffers* necesarios para el envío de datos en la aplicación implementada.

el envío *per se*, por lo que si se producen bloqueos, no afectan al transmisor. Asimismo, el *buffer* sigue siendo totalmente accesible, por lo que el cliente puede seguir aumentándolo sin inconvenientes.

En este caso no solo ha sido necesario implementar dicho *buffer* para asegurar la tasa binaria de envío, sino que también es muy relevante para los algoritmos de *scheduling*. En concreto para el algoritmo propuesto, ya que se vale del estado de las diferentes colas (cada uno correspondiente a un flujo distinto) para realizar su planificación. Esto se puede ver en el cálculo mostrado en la Ecuación 2.3 para saber de cuál debe enviar. Al mismo tiempo, conocer la evolución del estado de los *buffers* en los diferentes algoritmos resulta interesante para la comparativa entre algoritmos.

En la Figura 3.4 se muestra la arquitectura resultante de la implementación anterior. En ella se observa cómo el emisor envía información por tres flujos diferentes (flechas). Estos datos se almacenan en sus respectivos *buffers* (azul, verde y naranja), implementados en *threads* independientes. Asimismo, estos *buffers* poseen un número diferente de paquetes, ya que su ocupación depende de la tasa binaria de cada flujo –la cual se supone distinta para cada uno. Esta primera parte ocurre a nivel de aplicación, en donde se da paso al entorno de QUIC. En este punto, se puede ver que el *scheduler* decide el envío de datos, extrayendo los datos más antiguos de cada *buffer*. Por último, los datos seleccionados por esta entidad son enviados al *buffer* de UDP.

Una vez se comprende el contexto que rodea al *scheduler*, se continúa con la implementación de los diferentes algoritmos. El primero de ellos es el ya impuesto por defecto (en la implementación de QUIC utilizada), un algoritmo de *scheduling* basado en el *Round-Robin*. Seguidamente, se expondrán las variaciones del mismo hasta llegar a los diferentes algoritmos de *scheduling* mencionados y desarrollados en el Apartado 2.3.

3.4.1. Round-Robin

El algoritmo *Round-Robin* es el que está detrás de la planificación en el protocolo QUIC (de forma predeterminada) a la hora de generar paquetes desde diferentes flujos de datos. Este algoritmo destaca por su sencillez, la cual se muestra en la Figura 3.5. Este diagrama describe los pasos a seguir, desde la selección del *stream* hasta el envío de datos. Al final del proceso, se devuelve una lista de *Stream Frames* (`frame[]`) con datos de diferentes *streams*.

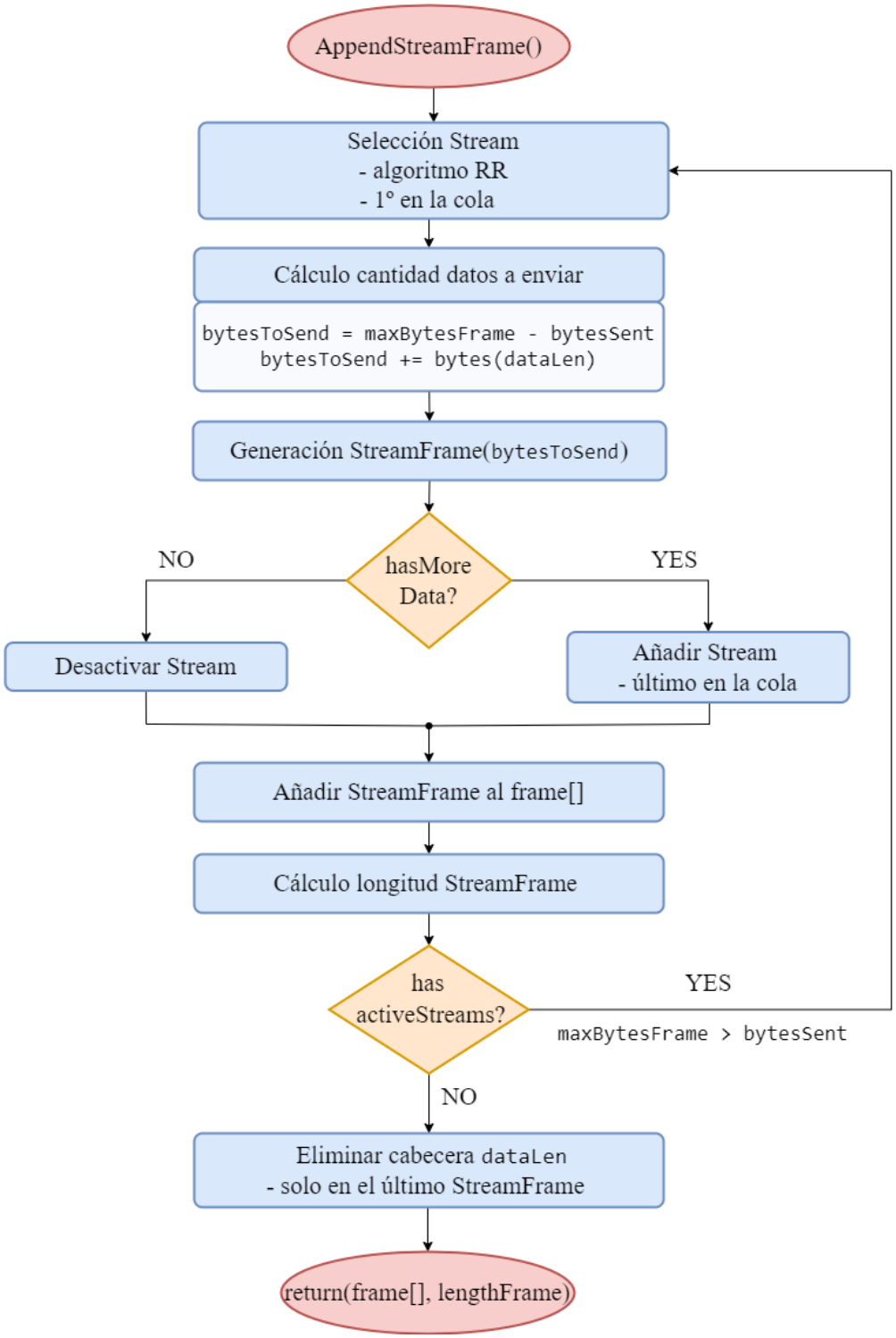


Figura 3.5: Diagrama de flujo: *Scheduler* usando el algoritmo RR.

El primer paso es seleccionar el primer (y quizá único) flujo del que se enviarán los datos. Para la selección de flujo, se almacenan sus identificadores en una cola. La elección depende exclusivamente de la propia cola de flujos disponibles en el momento actual, por lo que no es necesario comprobar prioridades, pesos o cantidad de datos en el *buffer* de cada *stream*. En otras palabras, se rige por el principio de colas FIFO, tal y comentó en la Sección 2.3.1. Tras ello, se debe calcular cuántos datos (bytes) tanto útiles (*payload*) como de cabeceras es posible que se envíen en esta primera trama. Para poder realizar este cálculo, es necesario comprobar el valor de dos variables: el tamaño máximo del campo de datos en la trama (bytes posibles a enviar máximos) y la cantidad de bytes ya enviados en tramas anteriores del mismo paquete QUIC. Como es lógico, para la creación del primer *Stream Frame* este valor será nulo. Es decir, para el primer *frame* que se genere en este paquete QUIC. Pero también se le debe añadir unos bytes adicionales pertenecientes a una cabecera llamada `Data Length` (o `dataLen`). Como se ha explicado en la Sección 2.2.2, esta cabecera se omite en el último *Stream Frame*, por lo que es importante añadirla para que todos los demás puedan implementarla. Una vez se conoce el tamaño máximo de la trama de datos, se genera el *Stream Frame* –gracias a la utilización de una función encargada de ello. Finalmente, si el flujo del que se han empaquetado datos aún tiene datos pendientes de envío se vuelve a añadir al final de la cola FIFO. Si por el contrario ya está vacío, se desactiva dicho flujo hasta que vuelva a tener datos.

Una vez se ha logrado generar el primer *Stream Frame*, primera trama con datos de un *stream* específico, se añade a la lista de *frames* (`frame[]`). Gracias a esta lista, se generará el paquete QUIC final. Además, se calcula la longitud (bytes enviados) de dicho *Stream Frame*. En este punto se debe comprobar si hay más flujos activos, es decir, pendientes de enviar datos. Si es así, se repite todo el proceso anterior, si no, se da por finalizado. Asimismo, existe otro caso en el cual se debe parar de generar *Stream Frames*. Este caso ocurre cuando la cantidad máxima del campo de datos en la trama se iguala a la cantidad de datos ya enviados. Para terminar y poder seguir con el proceso para enviar el paquete QUIC (devolviendo la lista de *frames* y su longitud total), se debe eliminar los bytes de la cabecera `dataLen` del último (o si es el único) *Stream Frame* –ya que siempre se añade en su creación, pero no siempre debe existir.

3.4.2. Fair Queuing

El primer algoritmo implementado para mejorar el rendimiento del protocolo QUIC es el *Fair Queuing*. El nombre que se le ha dado a esta función es `SchedulerFairQueuing()` y su proceso se puede ver en el diagrama de flujo mostrado en la Figura 3.6. Este caso difiere bastante del anterior, como se procede a explicar a continuación, aunque la estructura general se mantiene –las variables de entrada y salida son las mismas, por lo que solo es necesario variar la función correspondiente al algoritmo en el fichero `packet_packer.go` como se vio en la Figura 3.3.

Se recuerda que el algoritmo FQ pretende enviar información de todos los flujos disponibles, es decir, de aquellos flujos que tengan datos para enviar. Además, envía la misma cantidad de ellos (siempre que sea posible). Teniendo sus principios en cuenta, se comienza explicando el proceso. De nuevo, el primer paso consiste en seleccionar el primer *stream* del que se desea enviar los datos. Para ello es fundamental comprobar el tamaño de los datos a enviar (indicados por la variable `nextFrame`) de cada flujo. Esta trama `nextFrame` contiene la información que se va a transmitir tan pronto como sea posible. En concreto, estos datos proceden tanto de aquellos que están por transmitirse (ya sea porque en la trama

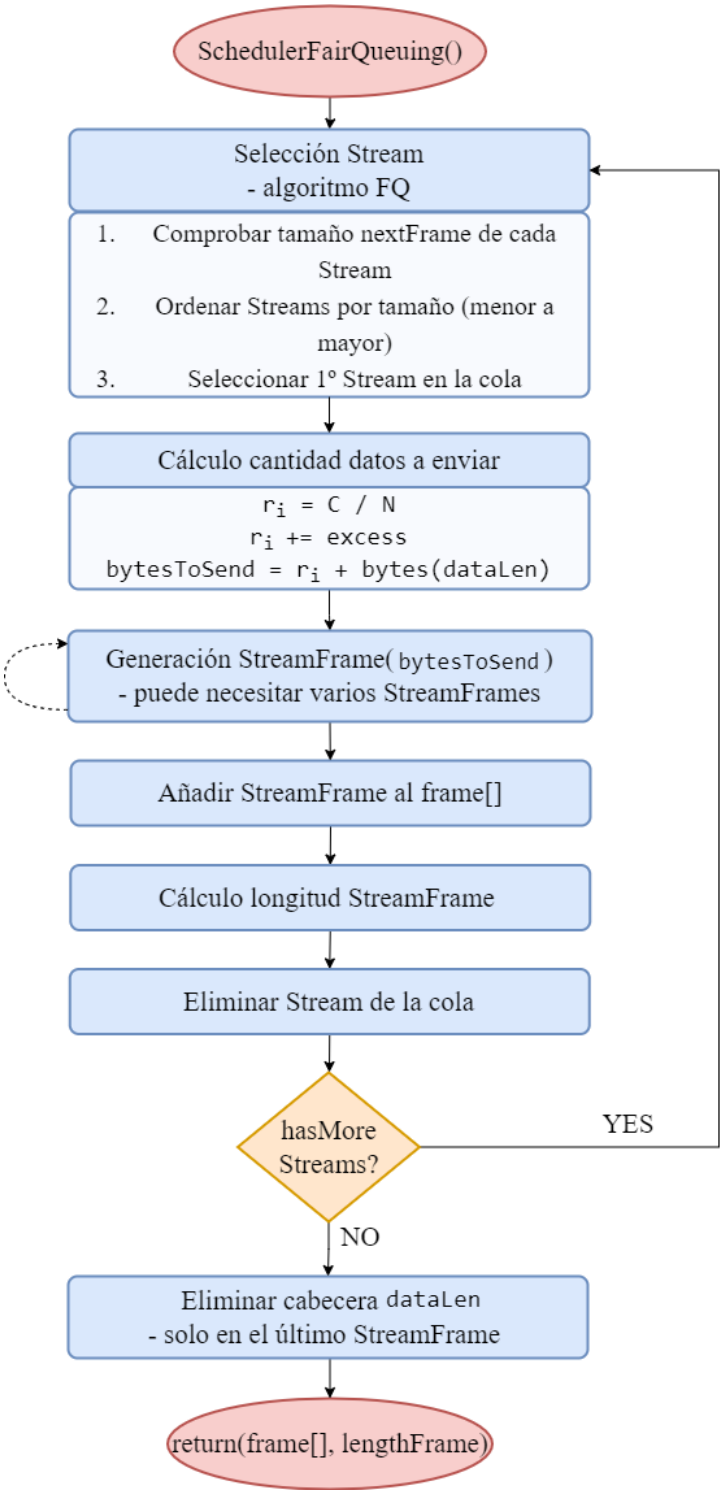


Figura 3.6: Diagrama de flujo: *Scheduler* usando el algoritmo FQ.

anterior no se tuvo oportunidad o porque su tamaño era demasiado pequeño) como aquellos que, por algún error en la transmisión, necesitan reenviarse. Para ello se crea una función que ofrezca esta información, llamada `TotalQueue`[4]. Tras obtener dicho valor para cada *stream*, es necesario ordenarlos en orden ascendente, ya que si uno de ellos tiene menos bytes para enviar que el resto, los recursos que no utilice se repartirán entre el resto.

[4]Definida en el fichero `send_stream.go`.

Tras ello, se necesita calcular cuántos datos puede enviar este flujo. Para ello es necesario seguir las fórmulas vistas en la Sección 2.3.2, es decir, dividir el tamaño máximo del campo de datos en la trama entre el número de *streams* activos. Asimismo, también se le deben añadir los bytes de exceso en el caso que alguno de los flujos previos no poseyera la cantidad de información que se le había asignado. De nuevo, el primer *stream* encontrará esta variable con valor nulo. Como ocurría con el algoritmo RR, hay que sumarle a dicho valor los bytes correspondientes de la cabecera `dataLen`.

Una vez obtenido el valor máximo del tamaño de la trama de datos, se genera el *Stream Frame* correspondiente. Los siguientes pasos son idénticos al algoritmo visto con anterioridad. Primero hay que añadir el *Stream Frame* generado y calcular el tamaño de los bytes que se han enviado en la totalidad del proceso. Adicionalmente, se elimina el *stream* de la cola, para que se puedan enviar los flujos restantes (repitiéndose el proceso anterior). En este caso no es necesario comprobar la cantidad de datos enviados respecto al tamaño máximo de la trama de datos, puesto que el cálculo anterior nunca puede resultar en un valor mayor que ese. Al terminar el envío de todos los *streams* pendientes de transmitir su información, se debe eliminar la cabecera de `dataLen`. Como las variables de retorno son las mismas, se devuelve la lista `frame[]` con todos los *Stream Frames* generados, así como la longitud total del mismo.

Sin embargo, aparece un problema relacionado con la naturaleza de este algoritmo. Como mínimo, se generan tantos *Stream Frames* como el número de flujos activos. Esto implica que se genera una sobrecarga de datos de cabecera, en contraposición al algoritmo RR que prioriza el envío de datos de un mismo *stream*. Esta sobrecarga se genera por el diseño del protocolo QUIC, no ideado para implementar un algoritmo de este tipo.

3.4.3. Weighted Fair Queuing

El siguiente algoritmo que se ha querido implementar es la continuación lógica del anterior, puesto que se trata del algoritmo *Weighted Fair Queuing*. La única variación respecto al anterior reside en cómo se debe calcular la cantidad de datos a enviar –aparte del nombre de la función, la cual se le ha dado `SchedulerWFQ()`. El resto de pasos, tanto previos como siguientes a este, se mantienen exactamente igual que la solución anterior, puesto que la lógica de ambos algoritmos es la misma. Para evitar repeticiones, esta sección se centrará únicamente en cómo se realiza el cálculo anterior. En la Figura 3.7 se muestra el recorrido lógico que hace el algoritmo WFQ para realizar de manera correcta el reparto de datos del *Stream Frame* pertinente.

La principal diferencia, como se ha explicado en la Sección 2.3.3, implica considerar ciertos pesos en los flujos a enviar. Si estos pesos se asignaran de manera equitativa, esta solución actuaría de la misma forma que el algoritmo FQ. Para que el *scheduler* sea capaz de obtener esta información se ha incluido en el propio archivo de configuración. Cabe destacar que, por este motivo, se supone que estos valores son constantes y no varían en el tiempo –una funcionalidad que se podría añadir en un futuro. Estos pesos varían el porcentaje de datos que puede enviar un *stream* respecto al total a enviar.

El cálculo, el cual se basa en la Ecuación 2.2, resulta sencillo, puesto que solo es necesario conocer dos variables. La primera de ellas es el tamaño disponible en el *Stream Frame* para el envío de información útil, una variable que depende únicamente del tamaño máximo de la trama actual y del número de bytes ya enviados. Aunque este valor no influye al comienzo del proceso, para el segundo *stream* de la cola tiene una gran relevancia, ya que si el flujo anterior envía menos datos de los que podía, el segundo

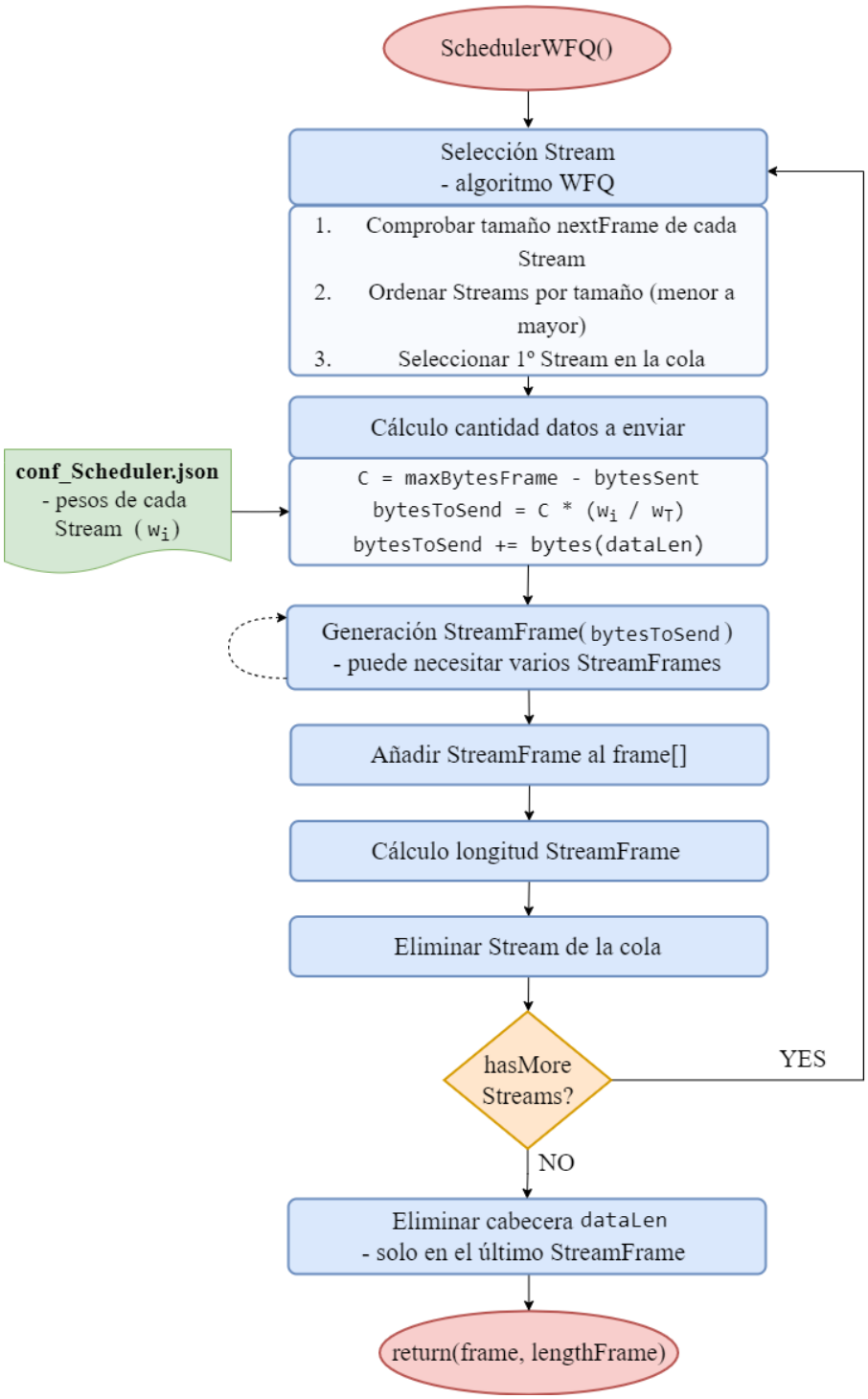


Figura 3.7: Diagrama de flujo: *Scheduler* usando el algoritmo WFQ.

puede aprovechar el espacio libre. La segunda variable, el peso, modula cuánta capacidad de la todavía disponible se puede usar por el *stream* del que se está generado la trama.

Una vez se tiene el valor de datos útiles para su posterior envío del primer flujo, se le debe añadir los datos de cabecera para la generación del *Stream Frame*, tal y como se explicó en el algoritmo anterior. Tras ello, se añade este elemento a la lista `frame[]` y se comienza el proceso desde el principio para el resto de flujos disponibles para su envío de datos.

3.4.4. Proposal Queuing

El último algoritmo implementado se ha llamado *Proposal Queuing* –función de nombre análogo– y se basa en la teoría de Lyapunov, descrita en la Sección 2.3.4. Esta opción recupera la estructura original y se basa en ella para realizar la generación de los diferentes *Stream Frames*. Es por ello que, como se ve en la Figura 3.8, su diagrama de flujo es similar al diagrama del algoritmo RR, tomando cierta distancia de los otros dos algoritmos. A continuación se explica paso a paso cómo funciona esta alternativa que busca la estabilización de las colas.

De manera semejante a los anteriores algoritmos, el primer paso es seleccionar el *stream* desde donde se va a enviar la información. Se recuerda que el algoritmo propuesto elige el flujo teniendo en cuenta dos factores: (1) el estado del *buffer* de cada flujo y (2) el valor de penalización de dicho flujo. La penalización se obtiene de la misma forma que el algoritmo anterior (WFQ), valiéndose de un fichero de configuración en donde se asignan estos valores (en vez de pesos) a los *streams* utilizados. Estos valores son invariantes en el tiempo, tal y como pasaba en el algoritmo previo. Para conocer el estado de los *buffers* de cada *stream*, se necesita conocer el tamaño tanto de la trama `nextFrame`, es decir, de los datos o bytes disponibles para su envío (función `TotalQueue`), como de los que se encuentran almacenados en el *buffer* de aplicación –se recuerda que este *buffer* es el que se ha tenido que implementar debido a su ausencia.

Tras obtener los dos factores clave para poder hacer la elección, se calcula la prioridad de cada flujo como se mostró en la Ecuación 2.3. Para facilitar cálculo, se determinan las penalizaciones totales de cada flujo como la resta entre la penalización propia del flujo y el tamaño (estado) del *buffer*. De esta forma, se disponen los flujos en orden ascendente de prioridad, de manera que el primer flujo a enviar es aquel que tiene una penalización inferior (mayor prioridad). La forma en que se clasifican los flujos es lo que realmente diferencia la implementación de este algoritmo al aplicado por defecto en la implementación de QUIC utilizada, ya que los siguientes pasos son similares.

La cantidad de datos que puede enviar un *stream* depende de la capacidad máxima de la trama de datos del paquete QUIC y los bytes ya enviados. Si es el primer flujo, el segundo valor será nulo, por lo que intentará enviar el mayor número de bytes de datos posibles. Además, y como ya ocurría en los demás algoritmos, se le debe añadir los bytes correspondientes a la cabecera `dataLen` –dependientes de cuántos bytes de datos útiles se envíen, tal y como se vio en la Tabla 2.2. Tras obtener el valor máximo de datos que se pueden enviar (útiles y de cabeceras), se procede a generar el *Stream Frame* correspondiente al flujo seleccionado.

Una vez generado, se desactiva el flujo activo y se añade la trama recién creada la lista de *frames* (`frame[]`). Antes de continuar, se debe calcular la longitud (bytes) que ocupa dicha trama, ya que así también se descubre cuántos bytes de datos han sido enviados. En este momento se debe comprobar si existen más flujos disponibles, así como si hay capacidad en la trama de datos para su envío. Si existe espacio para enviar otra trama, el proceso anterior se debe repetir. En este caso no es necesario volver a realizar el proceso entero de selección del *stream*, sino que se escoge el flujo siguiente en la cola. Una vez terminado todo el proceso, se debe eliminar la cabecera `dataLen` (como en los casos anteriores) únicamente del último *Stream Frame* generado. Se concluye devolviendo tanto la lista de tramas generadas como su longitud total.

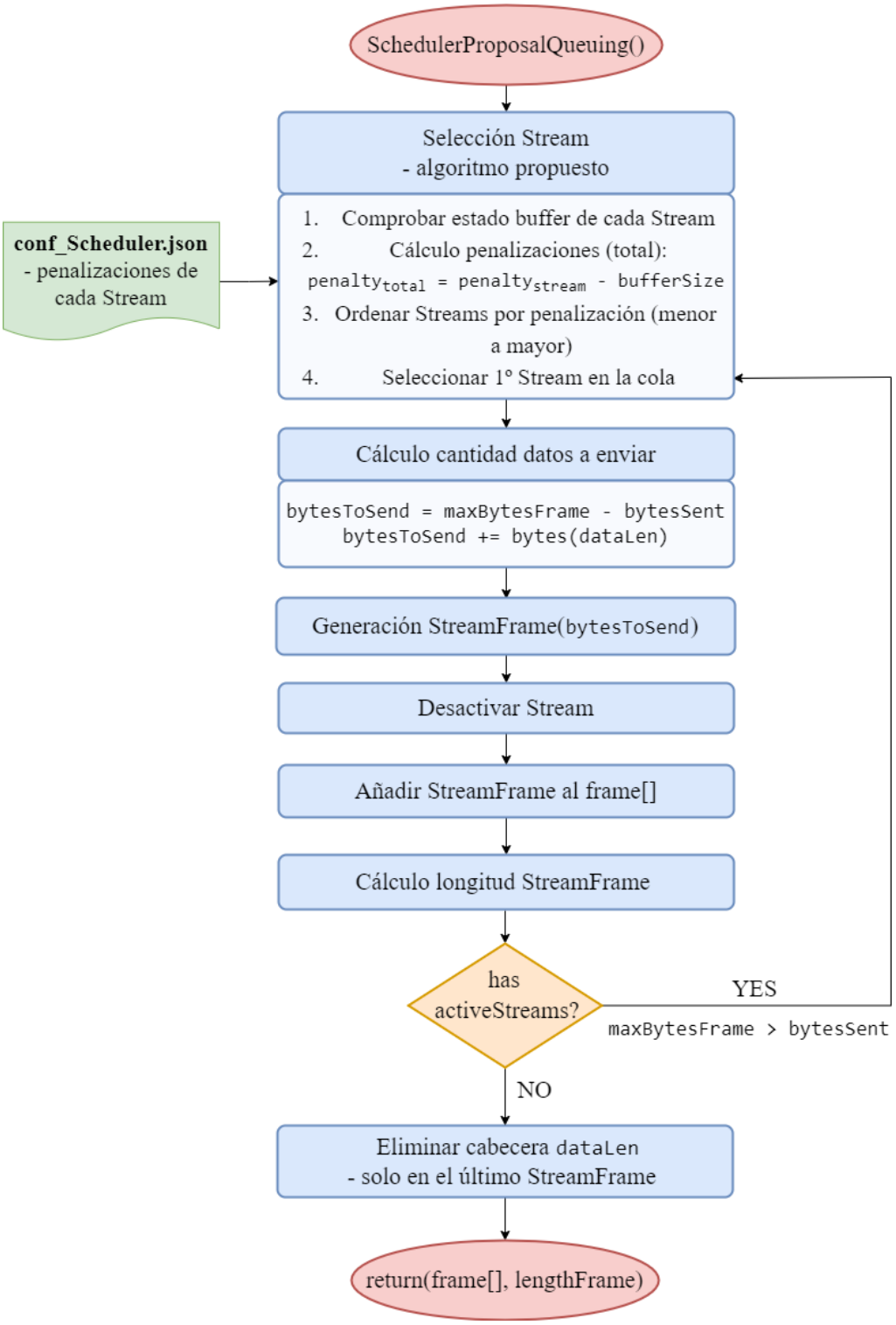


Figura 3.8: Diagrama de flujo: *Scheduler* usando el algoritmo *Proposal Queuing*.

Este algoritmo, al ser más parecido al aplicado por defecto en la implementación de QUIC utilizada (basado en el algoritmo RR), su implementación es más sencilla que el algoritmo FQ o WFQ. Asimismo, al distanciarse de estas dos implementaciones (en concreto, al WFQ), ya no necesita conocer *a priori* cómo envía la información para ofrecer buenos resultados –es decir, conocer cuál es la distribución del tráfico. En cambio, se vale simplemente de estudiar cuál es el estado de los *buffers* para hacer su elección.

Capítulo 4

Comparación de rendimiento y resultados

Este capítulo se centra en el análisis de los diferentes datos y resultados obtenidos con el entorno y técnicas descritas en el Capítulo 3. Se puede dividir en tres secciones, aunque las dos primeras están estrechamente relacionadas.

La primera parte de esta capítulo se centra en el estudio del comportamiento de los diferentes protocolos de transporte, tanto QUIC como TCP, en las comunicaciones satelitales. Para ello, el análisis utiliza el modelo LMS, es decir, aquel enlace que simula una conexión entre la estación terrena y el primer satélite LEO. Seguidamente, se entra en detalle sobre el comportamiento de dichos protocolos, analizando y comparando sus respectivos rendimientos –centrándose en los *delays* o retrasos en diferentes escenarios. Para terminar, se evaluarán los diferentes algoritmos de planificación implementados. De esta forma, se busca comprobar cuál de ellos es el que mejor resultados consigue, teniendo en cuenta que se trabaja en un entorno con grandes inestabilidades.

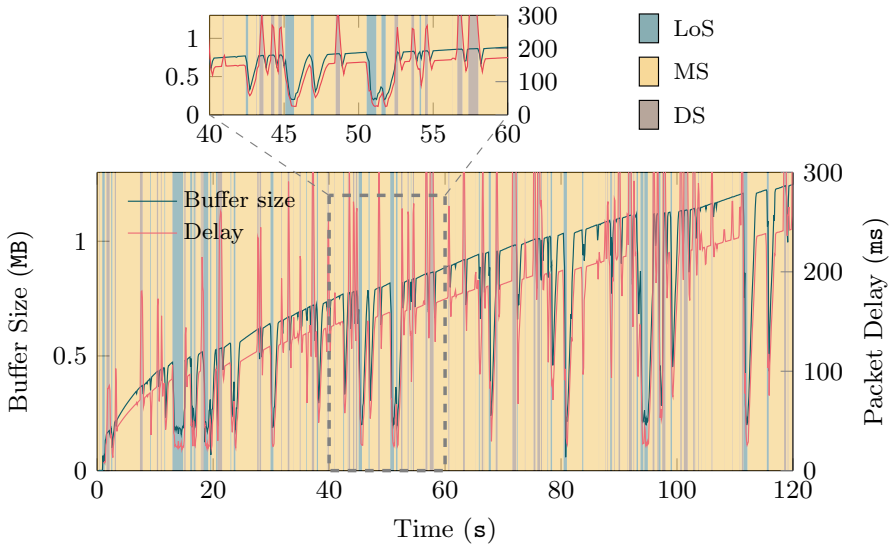
4.1. Comportamiento y comparativa de los protocolos de transporte QUIC y TCP

Como ya se ha comentado, en esta sección se pretende estudiar el comportamiento de los diferentes protocolos de transporte QUIC y TCP en entornos inestables, concretamente, en comunicaciones satelitales. Asimismo, se realiza una comparativa para comprobar si es correcto afirmar que el protocolo más reciente, QUIC, opera mejor en este tipo de entornos. Para ello, se presentan dos escenarios. El primero consiste en utilizar un único enlace LMS, simulando la comunicación entre una estación terrena y el primer satélite LEO de la cadena. El segundo resulta una ampliación del primero, puesto que se simula una comunicación LEO completa. Es decir, se concatenan dos enlaces LMS obteniendo así una enlace E2E. Cabe destacar que, como este apartado no se centra en la distribución del tráfico enviado, se utiliza tráfico con tasa constante en los dos escenarios.

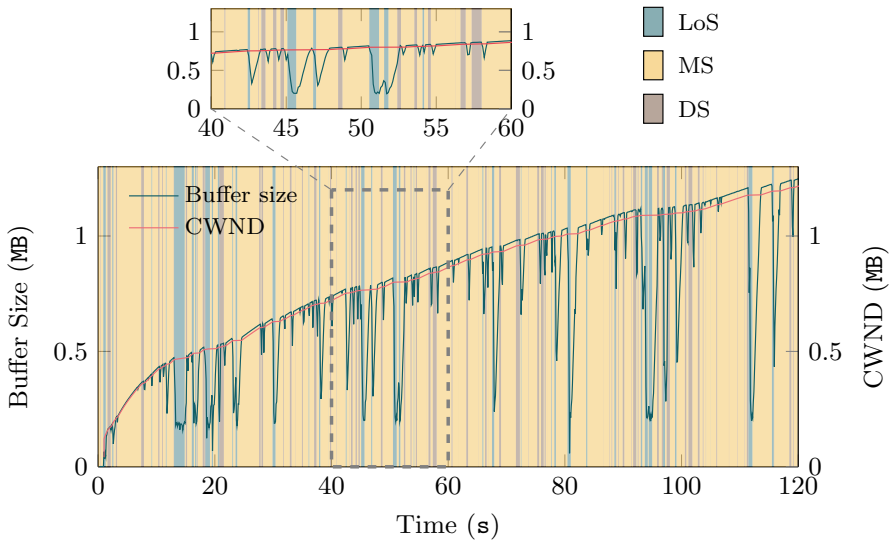
4.1.1. Enlace LMS

Se comienza analizando la evolución instantánea de tres aspectos claves de una comunicación: (1) la ocupación del *buffer* en la entrada al enlace LMS; (2) la ventana de congestión o simplemente CWDN; y (3) el retardo. Todo ello se estudia bajo un único canal LMS, aunque variando la capacidad máxima del *buffer* del mismo. Con este análisis se puede observar de forma clara la interacción de estos parámetros y validar que el entorno en su conjunto tiene un comportamiento razonable.

En el caso de la Figura 4.1, no existe ninguna restricción, lo cual implica un *buffer* infinito. Sin embargo, en la Figura 4.2 se configura el canal con una gran limitación, puesto que solo puede albergar 7 paquetes en espera al mismo tiempo. El objetivo es poder ver cómo afecta estos canales a los protocolos bajo estudio.



(a) *Buffer* Vs. Retardo (*delay*)



(b) *Buffer* Vs. CWND

Figura 4.1: Ejemplo de la evolución del tamaño del *buffer*, ventana de congestión y del retardo experimentado en un canal LMS con *buffer* infinito.

Se aclara que, pese a que en ambas figuras se representan resultados conseguidos tras utilizar el protocolo QUIC, las conclusiones que se obtengan se pueden extrapolar al protocolo TCP. Este hecho se basa en la utilización de un algoritmo de congestión, concretamente, CUBIC. Esta solución es la más extendida y utilizada en la mayoría de implementaciones, tanto para el protocolo QUIC como TCP.

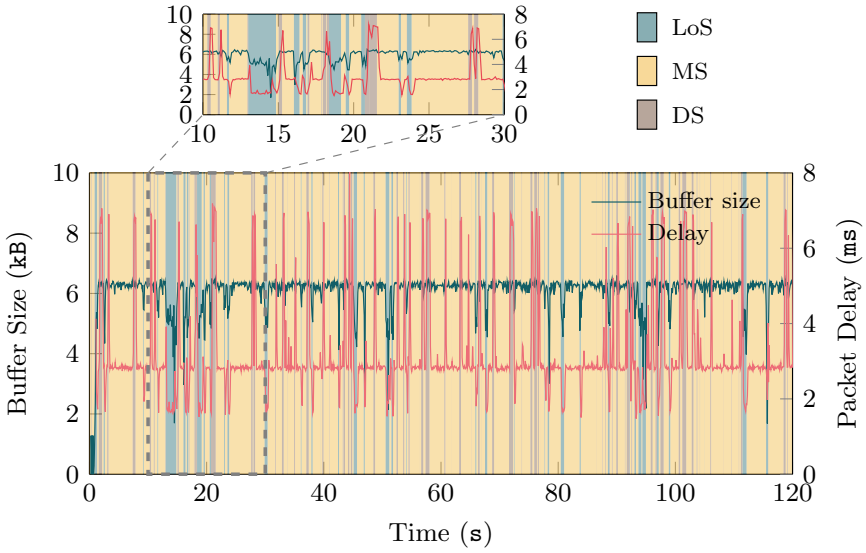
Lo primero es explicar la composición de las figuras ya mencionadas. Como se puede observar, el fondo de las figuras muestra en qué estado se encuentra el canal en cada instante temporal del experimento. Asimismo, y como se explicó en el capítulo anterior, los diferentes estados corresponden a distintas situaciones en función la línea de visión de los satélites. El color que predomina es el amarillo, indicando que el estado dominante en el tiempo es el *Mid Shadowing* o MS –seguido por el mejor caso, el LoS y finalmente, el contrario, el *Deep Shadowing* o DS. Ambas figuras, adicionalmente, se dividen en dos subapartados: en la parte superior se muestra tanto la evolución del tamaño del *buffer* como del retraso de los paquetes, mientras que en la parte inferior se puede ver (de nuevo) la evolución del *buffer* esta vez contra la variación de la ventana de congestión.

Una vez comprendido qué es lo que se muestra, se analizan los datos obtenidos. Se comienza por la Figura 4.1, en donde se realiza un experimento con un canal LMS con *buffer* infinito, el cual nunca se va a desbordar. Este hecho es el que produce que, tanto en la Figura 4.1a como 4.1b, se vea cómo la ocupación del *buffer* aumenta constantemente –eje ordenadas de la izquierda. No lo hace de manera lineal, sin embargo, puesto que tiene una clara relación a la situación del canal. Cuando se encuentra en el estado LoS, su ocupación disminuye abruptamente, vaciándose. En el caso contrario (estado DS), la capacidad aumenta ligeramente. Se pueden ver con mayor claridad ambas situaciones en el *zoom* realizado (parte superior).

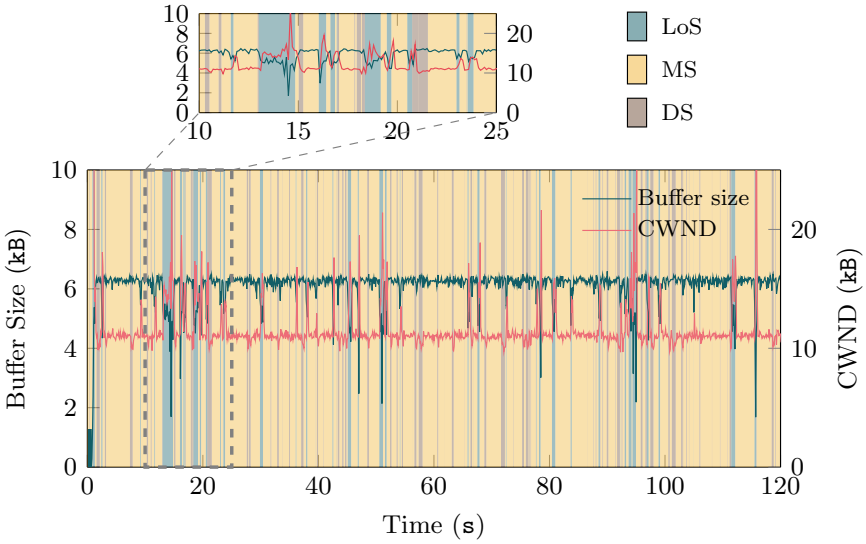
Para estudiar la evolución del retardo a lo largo del tiempo se analiza la Figura 4.1a –eje ordenadas de la derecha. En líneas generales se puede observar que, como el tamaño del *buffer* es infinito, el *delay* de los paquetes aumenta con el paso del tiempo debido a que se almacenan en el dispositivo de entrada al enlace LMS. Sin embargo, sufre unas variaciones más abruptas. Centrándose en el *zoom*, se puede ver que estas subidas y bajadas se corresponden, de nuevo, a los cambios producidos en el estado del canal. De la misma forma que el caso anterior, el retardo aumenta rápidamente en la situación más perjudicial (DS), se mantiene en el estado MD y prácticamente se elimina en el LoS.

El último elemento que queda por evaluar se corresponde con la ventana de congestión, mostrada en la Figura 4.1b con una línea sólida de color rojo. Al contrario de lo ya visto, esta evolución no parece reaccionar a los cambios en el estado del canal, sino que se limita a crecer de forma constante y prácticamente lineal. Este comportamiento es el que provoca el aumento continuo (pero lento) del tamaño del *buffer* y los retardos, puesto que demuestra la incapacidad de la CWND de adaptarse a las variaciones del canal. Este comportamiento es el esperado, ya que el *buffer* infinito hace que el transmisor no detecte el cambio de capacidad (se percibiría como congestión), y continúe enviando. El efecto producido, llamado *buffer bloating*, es bien conocido y se da típicamente al usar conexiones inalámbricas.

La Figura 4.2 se centra en un caso en el que el *buffer* sí esté limitado, en concreto, con únicamente 7 paquetes posibles como máximo. Poniendo números, y como dichos paquetes son de 1000 `Bytes`, la capacidad máxima es de 7000 `Bytes`. En este ejemplo, la evolución del tamaño del *buffer* (línea azul en ambas subfiguras) no parece aumentar paulatinamente, tal y como ocurría en la figura previa. En cambio, se mantiene prácticamente estable a lo largo del tiempo, por debajo ligeramente de su capacidad



(a) *Buffer* Vs. Retardo (*delay*)



(b) *Buffer* Vs. CWND

Figura 4.2: Ejemplo de la evolución del tamaño del *buffer*, ventana de congestión y del retardo experimentado en un canal LMS con un *buffer* de capacidad máxima igual a 7 paquetes.

máxima. Aunque, tal como pasaba en el caso anterior, se puede ver en el *zoom* superior como se repite el comportamiento al encontrarse en el estado LoS, vaciándose el *buffer*. En los otros dos estados aumenta o se mantiene, como era de esperar.

En la Figura 4.2a se muestra la evolución del retardo (en rojo), con un comportamiento muy similar al anterior parámetro. De nuevo, el *delay* parece ser bastante estable con un valor medio entorno a los 4 `ms`. Este efecto se observa incluso a pesar de las abruptas subidas o bajadas, dependiendo si se encuentra en el estado DS o LoS, respectivamente. Para finalizar, se analiza la evolución de la ventana de congestión en la Figura 4.2b –eje ordenadas de la derecha. Como era de esperar viendo la tendencia de los otros parámetros, la ventana de congestión también se mantiene estable. Asimismo, se observan variaciones (picos), en este caso más marcadas, que coinciden con el paso al estado con mejores características, aprovechando

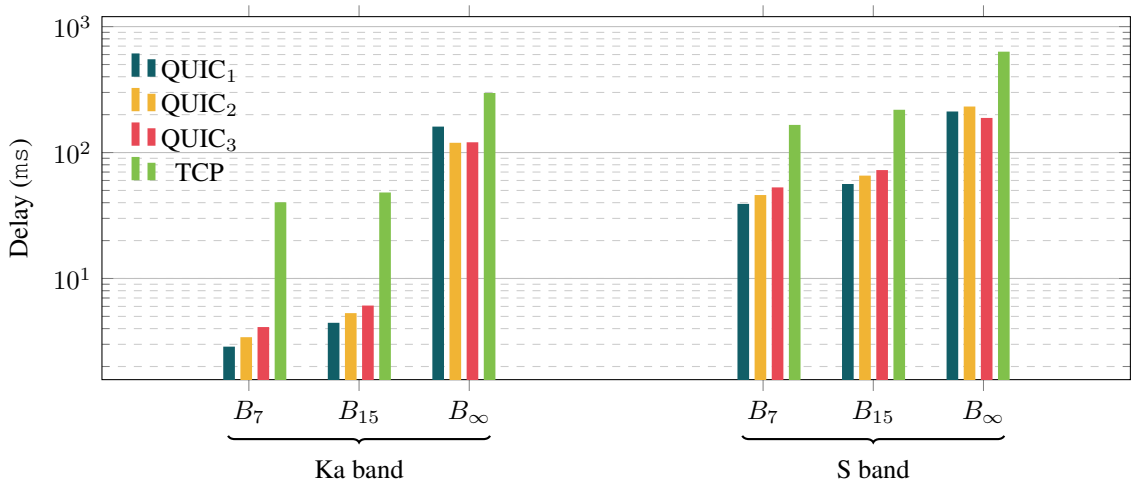


Figura 4.3: Retardo experimentado por el tráfico TCP y QUIC en un único enlace LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha).

capacidades más elevadas. Este comportamiento, totalmente contrario al visto en la figura con *buffer* infinito, implica una clara mejora en la capacidad de adaptación del sistema, estabilizándose. Al reducir el tamaño del *buffer* este se desborda, produciéndose pérdidas. A su vez, estas pérdidas se detectan como congestión, lo que reduce la ventana y así el uso del *buffer*.

Tras los resultados obtenidos se puede concluir que los canales LMS, así como la configuración del *buffer* de los dispositivos implicados, pueden tener un impacto bastante relevante en el rendimiento de los protocolos de transporte. A su vez, se ha podido observar que el entorno de emulación proporciona comportamientos razonables y que los parámetros de los diferentes componentes, tanto del cliente/servidor en los contenedores Docker como de los dispositivos de NS-3, interaccionan correctamente. Dicho esto, se puede continuar la investigación con la comparación entre los dos protocolos seleccionados, QUIC y TCP.

El parámetro más relevante para la comparativa de ambos protocolos de transporte es el retardo o *delay* que sufren los diferentes paquetes por su paso por el enlace LMS. Para ello se realizan 30 ejecuciones totalmente independientes entre sí, enviando un fichero de 800 `MB` para la banda *Ka* y de 60 `MB` para la banda *S*. Las medidas se han realizado en distintos escenarios, desde diferentes limitaciones en el tamaño del *buffer* del canal (7, 15 o infinitos paquetes), hasta distinto número de flujos o *streams* utilizados (1, 2 o 3) –todo ello se representa con los subíndices respectivos.

En la Figura 4.3 se muestran los valores obtenidos en `ms`. Se puede observar que la tendencia en ambas bandas es similar, siendo lógico que los retardos en la segunda banda sean más elevados, puesto que las tasas binarias manejadas eran muy inferiores. El comportamiento parece indicar que a mayor tamaño de *buffer*, mayor es el *delay* medido. Asimismo, al aumentar el número de *streams* implementados, parece aumentar el retardo. Por ende, se puede suponer que la función *multistreaming* no es ventajosa en entornos con gran inestabilidad. Sin embargo, en el caso de *buffer* infinito, esta tendencia cambia por completo. En dicho caso (para ambas bandas), un aumento en los flujos utilizados mejora los retardos, disminuyéndolos. Pese a ello, en todos los escenarios el retardo máximo siempre se consigue por la utilización del protocolo TCP. Este comportamiento parece suavizarse cuanto mayor es el *buffer*, puesto que las diferencias entre protocolos se hacen menos notables. El rendimiento de TCP parece ser peor que el conseguido en QUIC,

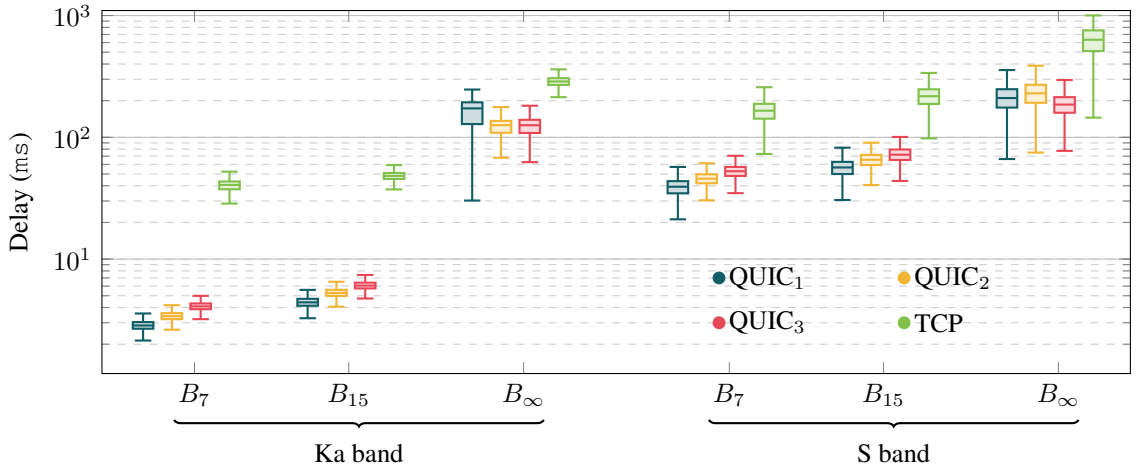


Figura 4.4: *Boxplot* o diagrama de caja del retardo experimentado por el tráfico TCP y QUIC en un único enlace LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha).

sobre todo al fijarse en los *buffers* más pequeños (7 y 15 paquetes) de la banda *Ka* y comparándolo a los resultados de la Figura 4.2a.

La Figura 4.4 muestra la distribución de los retardos anteriormente analizados. Gracias a esta representación, se puede estudiar el *jitter* (fluctuaciones del retardo) introducido por los dos protocolos de transporte al interactuar con los canales LMS. El diagrama de caja es una representación completa que muestra: la mediana o percentil 50 con una línea horizontal; los percentiles 75 y 25 como límites superior e inferior de la caja, respectivamente; y, por último, los percentiles 95 y 5 con los *whiskers* (líneas que sobresalen de la caja) superior e inferior, respectivamente.

Lo que muestra dicha figura, por tanto, es la variabilidad del retardo. El mejor comportamiento se muestra para las configuraciones de *buffer* finito, independientemente de la banda de funcionamiento. Sin embargo, hay comportamientos que merecen ser comentados en profundidad. Algo muy llamativo es lo que ocurre en el caso de QUIC con un único flujo, *buffer* infinito y banda *Ka*. Este caso es el más claro, pero también aparece a lo largo de las configuraciones de *buffer* finito de la otra banda –las configuraciones con dos y tres flujos parecen más estables en términos de *jitter*. Este efecto de mayor dispersión se repite para el protocolo TCP, en concreto en la banda *S* y, sobre todo, para *buffer* infinito.

La conclusión de esta primera comparativa entre protocolos es clara. *A priori*, el protocolo QUIC muestra un mejor rendimiento (basándose en *delays*) que TCP. La diferencia se puede deber a las mejoras en la detección de congestión implementadas en QUIC [42], tanto en la numeración, cálculo de tiempos y gestión de reconocimientos. El efecto concreto de cada una de las mejoras supone una línea futura de análisis derivado de este trabajo.

También se observa un mejor comportamiento de QUIC en las distribuciones, ya que el protocolo TCP muestra variaciones ligeramente más elevadas. Sin embargo, no queda claro cuál es el papel de la propiedad *multistreaming* de QUIC. Se debería estudiar más afondo y buscar un compromiso entre número de *streams* y el retardo subyacente (y su variabilidad), ya que aunque cuantos más flujos se utilicen, mayor es el *delay* (pero mayor es la estabilidad de los mismos).

4.1.2. Comunicación LEO extremo a extremo

Se continúa el análisis midiendo una comunicación LEO extremo a extremo o E2E. Para lograrlo, el escenario se compone de dos canales LMS y uno adicional que simula las conexiones ISL. Al igual que en el caso anterior, los resultados que se proceden a analizar se obtienen de 30 ejecuciones independientes, enviando un fichero suficientemente grande en cada banda –800 (banda *Ka*) y 60 `MB` (banda *S*).

En la Figura 4.5 se muestra la distribución del retardo obtenido de los resultados anteriores para diferentes tamaños de *buffer*, en donde se puede observar su valor medio. Una de las primeras conclusiones que se puede ver es que el protocolo QUIC introduce retardos menores que TCP, en especial en los casos de limitación de *buffer* (7 y 15 paquetes). No obstante, TCP parece ser más estable que QUIC, un comportamiento apreciable en los escenarios con *buffer* infinitos de ambas bandas. Es más, en dichos casos, los valores del retardo de ambos protocolos de transporte son muy parejos; exceptuando el caso de un único flujo de la banda *S*. En esta excepción se observa que QUIC produce los mejores resultados, tanto de valor medio como de variabilidad escasa, en comparación con las otras configuraciones de QUIC (dos y tres *streams*) y TCP.

Siguiendo con este análisis, la Figura 4.6 muestra la desviación estándar relativa o *Relative Standard Deviation* (RSD) –valor absoluto del coeficiente de variación. Estos valores son muy útiles para poder comparar de forma equitativa la distribución de las configuraciones con diferentes rangos de retardo. Esta comparación justa de dispersiones (del retardo) reside en cómo se calcula la métrica, basándose en la relación entre la desviación estándar y la media de los *delays* de cada configuración.

Una de las primeras apreciaciones a realizar es que la RSD, de todos los escenarios en los que se usa el protocolo QUIC, es mucho mayor que la obtenida con TCP. Este hecho se intensifica cuando la longitud del *buffer* no tiene restricciones (es decir, *buffer* infinito), tanto en la banda *Ka* como en la banda *S*. Este comportamiento indica una variabilidad mayor, tal y como se podía observar en la figura anterior. Asimismo, se observa una tendencia clara al aumentar el número de flujos en el sistema, disminuyendo su RSD. Sin embargo, en la figura anterior se vio una excepción al utilizar un único flujo QUIC en la configuración de banda *S* y *buffer* infinito, en donde se apreciaba un retardo menor que los demás casos (2

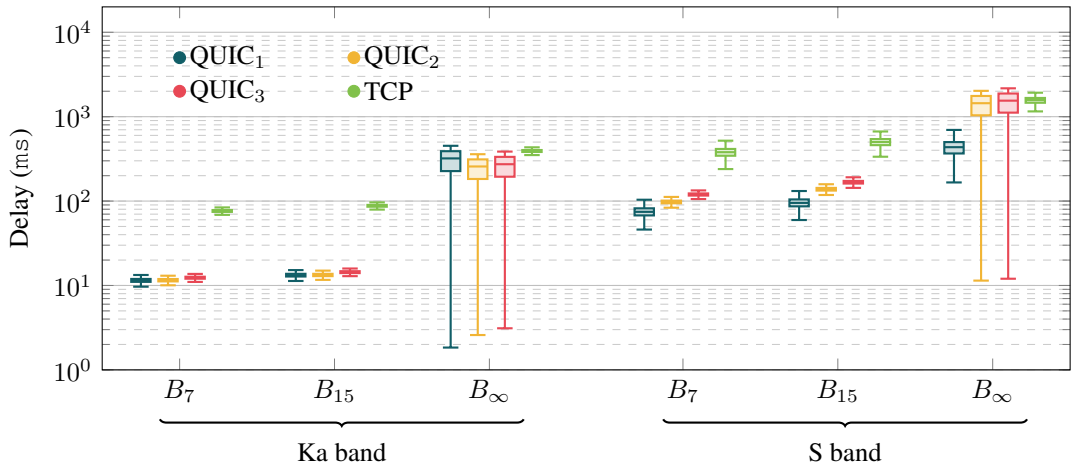


Figura 4.5: *Boxplot* o diagrama de caja del retardo experimentado por el tráfico TCP y QUIC en una conexión E2E que comprende 2 enlaces LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha).

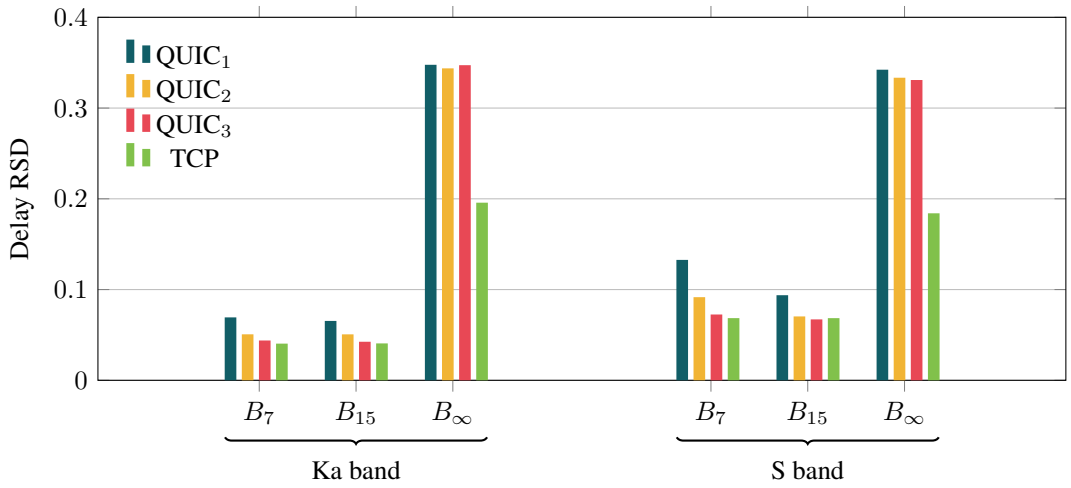


Figura 4.6: Desviación estándar relativa (RSD) del retardo experimentado por el tráfico TCP y QUIC en una conexión E2E que comprende 2 enlaces LMS –al operar tanto la banda *Ka* (izquierda) como en la banda *S* (derecha).

y 3 flujos con QUIC y para TCP). En cambio, en esta figura se observa que la RSD mayor se produce justamente en dicha configuración, aunque por una variación ligeramente superior.

Estos resultados demuestran, una vez más, que el protocolo QUIC ofrece un rendimiento mayor que el conseguido con TCP en cuanto a retardo medio. Sin embargo, aunque los valores medios son menores, su variabilidad parece mayor, lo que podría afectar al rendimiento de algunos servicios. Asimismo, se ha comprobado que en conexiones E2E, al contrario que en los escenarios de un único canal LMS, el peor comportamiento de QUIC se produce con un único flujo. Esto se debe a que sus valores son ligeramente inferiores, pero mucho más variantes.

Se continúa ahora analizando el impacto de la tasa de tráfico (en la aplicación) sobre el *delay*. Para ello se ha seleccionado la configuración en banda *Ka* con un *buffer* finito de 15 paquetes, comparando los retardos resultantes con QUIC (un único *stream*) y TCP. El motivo tras la elección de este escenario se basa en los resultados anteriores, puesto que es el que ofrecía rendimientos más similares.

En la Figura 4.7 se muestra la distribución del retardo, en ambos protocolos de transporte, aumentando la tasa binaria a nivel de aplicación –desde 10 hasta 45 `Mbps`, con saltos de 5 `Mbps`. Cabe destacar que en las configuraciones anteriores (tanto canal LMS como E2E) la tasa utilizada era la máxima (45 `Mbps`, muy cercana a la capacidad media del canal). Lo primero que se puede observar es cómo la variabilidad disminuye al aumentar tasa de datos, al mismo tiempo que las diferencias entre ambos protocolos se vuelven más relevantes. En tasas bajas, QUIC y TCP ofrecen resultados y rendimientos muy similares. Sin embargo, mientras QUIC se mantiene en valores constantes (e incluso con más estabilidad), TCP muestra valores cada vez mayores del retardo (aunque más estables).

Para complementar los resultados anteriores y ampliar el estudio, se muestra en la Figura 4.8 la RSD del retardo –para las diferentes tasas de tráfico. Como se podía observar en la figura anterior, la tendencia se mantiene igual. Es decir, al aumentar la tasa de tráfico en la aplicación, se disminuye la RSD. Sin embargo, ocurre algo curioso respecto a la comparación de los dos protocolos de transporte. En las tasas más bajas (de 10 a 30 `Mbps`), se observa que el protocolo TCP muestra valores superiores a los de QUIC (un único flujo). En cambio, al aumentar dicha tasa, las tornas se cambian y es TCP quien obtiene el *jitter* ligeramente menor que el observado en QUIC.

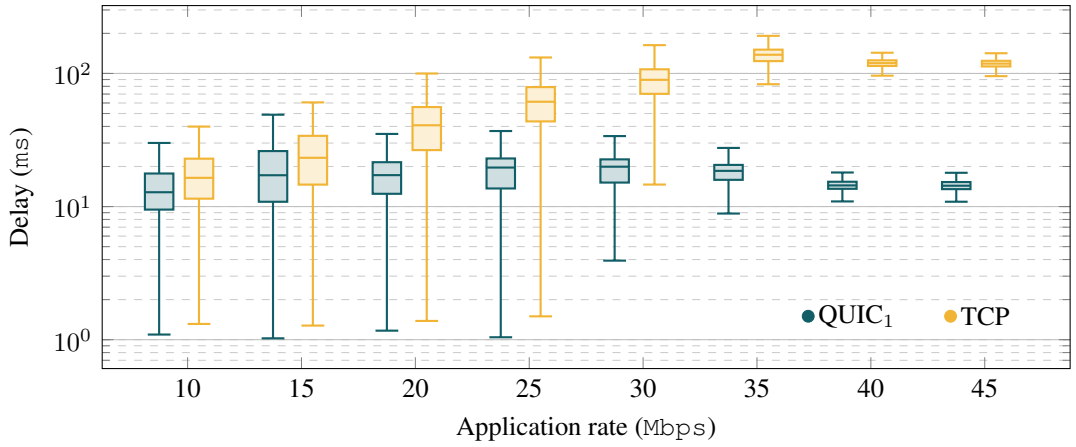


Figura 4.7: *Boxplot* o diagrama de caja del retardo experimentado por el tráfico TCP y QUIC en la banda *Ka* en una conexión E2E para diferentes tasas de tráfico. La longitud del *buffer* se fija en 15 paquetes.

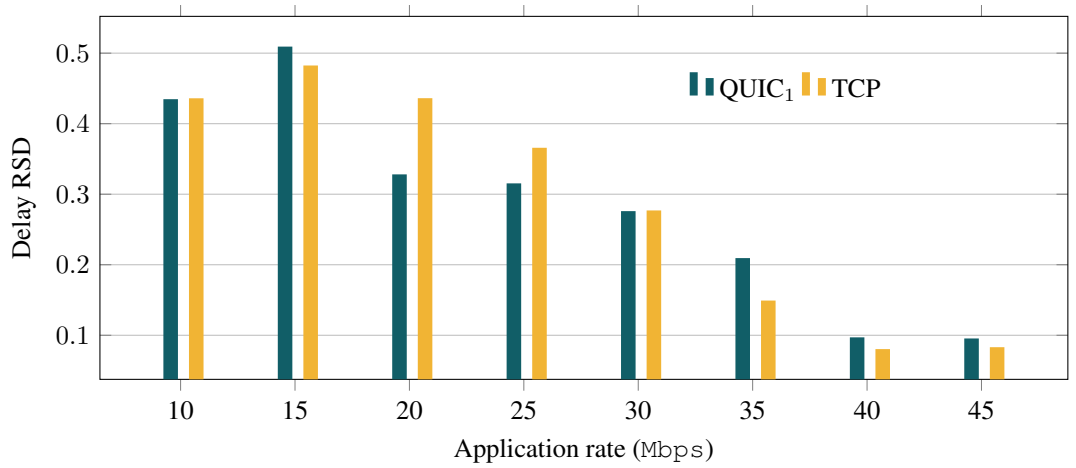


Figura 4.8: Desviación estándar relativa (RSD) del retardo experimentado por el tráfico TCP y QUIC en la banda *Ka* en una conexión E2E para diferentes tasas de tráfico. La longitud del *buffer* se fija en 15 paquetes.

Por ende, se termina la comparativa de protocolos concluyendo que en general QUIC ofrece un mejor rendimiento (tanto del valor medio como de la distribución del retardo) que TCP, sobre todo con tasas cercanas a la capacidad media del canal.

4.2. Análisis del rendimiento del Scheduler con diferentes estrategias

Una vez se afirma que el protocolo QUIC ofrece mejoras en el rendimiento del sistema en comunicaciones satelitales, se procede a analizar cómo funciona el planificador que aparece por defecto en la implementación de QUIC utilizada. Para ello, se estudiará el rendimiento de la propiedad *multistream* al aplicar diferentes algoritmos de *scheduling*.

Para todos los resultados que se analizarán en este apartado, el algoritmo WFQ posee un reparto de pesos que beneficia al primer flujo. En concreto, los pesos seleccionados en el fichero de configuración son de 2 y 1. Asimismo, se ha tomado la decisión de no penalizar a ningún *stream* para el caso del algoritmo propuesto. Es decir, que las penalizaciones existentes en el fichero de configuración son nulas (0).

4.2.1. Comportamiento de los algoritmos en un enlace controlado

El primer punto es comprender cómo funcionan los cuatro algoritmos que se han implementado –basado en el RR, FQ, WFQ y el propuesto. Para ello, se procede a explicar la configuración elegida tanto en la capa de aplicación como el enlace utilizado, la cual se puede observar en la Tabla 4.1.

Tabla 4.1: Configuración de la capa de aplicación, *buffer* y del canal de control.

Capa de aplicación y *buffer*	
Tamaño del *buffer*	∞
Tasa (datos)	*Stream*$_1$: 27 `Mbps`; *Stream*$_2$: 15 `Mbps`
Tamaño fichero (datos)	*Stream*$_1$: 202.5 `MB`; *Stream*$_2$: 112.5 `MB`
Tamaño del paquete	1000 `Bytes`
QUIC #*streams*	2
Canal de control	
Tasa media del canal	45.33 `Mbps`
Tasa de control (estados)	$[80, 40, 16]$ `Mbps`
Duración del *slot* temporal	$\delta = 50$ `ms`

En la capa de aplicación se encuentran dos flujos totalmente diferenciados, puesto que uno tiene casi el doble de tasa que el otro. Asimismo, el tamaño del fichero que envía cada uno es diferente e inversamente proporcional a la tasa, ya que se desea comprobar cómo se realiza la planificación del envío de información de diferentes puntos, por lo que ambos *streams* deben enviar datos durante la misma cantidad de tiempo. El tamaño del paquete se mantiene igual que en los escenarios vistos con anterioridad. Además, el tamaño del *buffer* permanece sin restricciones para poder analizar mejor las diferencias entre los algoritmos sin la influencia del control de congestión. Por último, es importante mencionar que el tráfico sintético utilizado sigue la distribución de *Poisson*.

Cabe destacar que, a pesar que en este apartado se procede a calificar a los diferentes flujos con el subíndice 1 o 2, en realidad sus `Stream ID` son 0 y 4, respectivamente. Esto se debe al tipo de *stream* que se envía, tal y como se pudo ver en la Tabla 2.1, ya que en este caso el transmisor de la información es el cliente y el flujo es bidireccional –por ende, el `Stream ID` de cada uno sería `0100` ó `0x004` (4) y `1000` ó `0x008` (8).

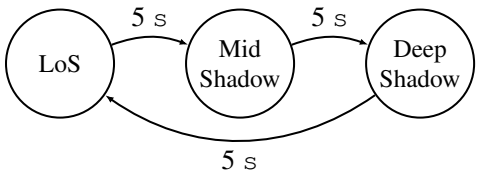
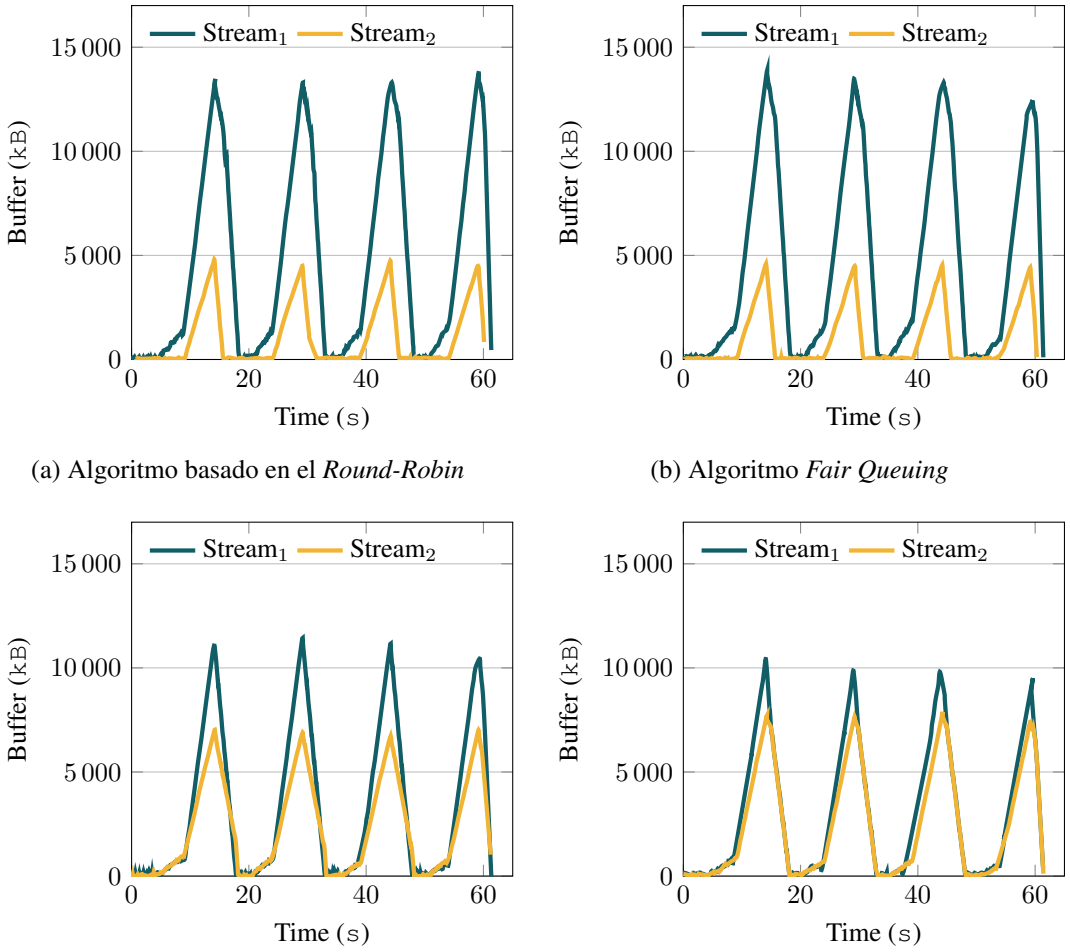


Figura 4.9: Modelo de un enlace controlado, basado en la cadena de Markov de LMS.

Para poder validar el comportamiento de los algoritmos, se utiliza un entorno controlado en el cual se simplifica el modelo del canal previamente utilizado. Se mantienen los tres estados característicos del modelo del canal LMS, sin embargo, el tiempo de permanencia en cada uno de ellos se fija a un valor constante. En este caso, se ha elegido utilizar la banda *Ka* y que la tasa del canal varíe cada cinco segundos. Primero en el canal más ventajoso (80 `Mbps`), después el levemente perjudicado (40 `Mbps`) y, por último, el que peor capacidad ofrece (16 `Mbps`). Como se puede ver en la Figura 4.9, la cadena es similar a la



(a) Algoritmo basado en el *Round-Robin*

(b) Algoritmo *Fair Queuing*

(c) Algoritmo *Weighted Fair Queuing*

(d) Algoritmo *Proposal Queuing*

Figura 4.10: Evolución del *buffer* de la cola de los diferentes *streams* a lo largo del tiempo, utilizando diferentes algoritmos de *scheduling* y un enlace de control.

vista en la Figura 3.2. Sin embargo, en este caso el enlace es totalmente previsible (cambios producidos con probabilidad 1), lo cual ayudará a poder estudiar los escenarios a tratar.

Una vez conocido el entorno, se procede a analizar de qué manera evoluciona el *buffer* de las colas a lo largo del tiempo –en concreto, durante un minuto. Para ello se muestra en la Figura 4.10 el valor que toma la cantidad de datos o *bytes* que existen en el *buffer* de cada uno de los flujos (en azul y amarillo). Se recuerda que este *buffer* está compuesto tanto por el *buffer* generado a nivel de aplicación como los que están disponibles para su envío (función `TotalQueue`).

Lo primero que se puede observar es que la tendencia, independientemente del *stream* y del algoritmo implementado, es la misma en todos los casos. Se puede ver con claridad los cambios en el estado del canal, ya que a los cinco segundos el tamaño del *buffer* crece ligeramente. En el siguiente cambio de estado (encontrándose en el más perjudicado), se puede ver como ese crecimiento aumenta bruscamente, generando un pico. En los diez segundos siguientes los valores caen en picado, sobre todo en el primer rango de tiempo.

Sin embargo, lo importante en dicha figura son las diferencias encontradas entre los algoritmos utilizados. En la Figura 4.10a (algoritmo RR) se observa una clara disparidad entre el valor máximo

(y medio) que consigue el *buffer* de cada flujo a lo largo del tiempo. El primer flujo posee un valor máximo entorno a 14 *MB*, mientras que el segundo no llega ni a la mitad de ese valor (casi 5 *MB*). Este comportamiento es muy similar o se acentúa ligeramente en el algoritmo FQ. Este hecho ocurre ya que, como se ha mencionado, el primer flujo envía información con una tasa binaria prácticamente el doble que el segundo flujo. Como en ambos algoritmos el reparto se produce tratando a ambos flujos del mismo modo, el primer *stream* se resiente, incrementando abruptamente su *buffer*. De la misma manera, esto beneficia al segundo flujo, puesto que al utilizar una tasa muy inferior, puede vaciar su cola rápidamente. En todo caso, cabe indicar que no hay una pérdida de recursos, ya que si un flujo vacía su *buffer* antes de usar todos los recursos, estos están a disposición del otro *stream*.

En cambio, en la Figura 4.10c, en donde se ha utilizado el algoritmo WFQ, la distancia de los valores entre los dos flujos se empieza a recortar. El valor máximo del *buffer* del primer *stream* se reduce hasta 11 *MB*, mientras que el valor máximo del segundo incrementa hasta unos 7 *MB*. Este cambio se debe a una planificación y reparto más óptimo, puesto que los recursos se reparten conociendo de antemano la tasa media de cada flujo: 66.66% para el primer flujo y 33.33% para segundo. Al observar la evolución del *buffer* para el último algoritmo, el propuesto, se puede ver que este comportamiento se acentúa aún más. Los valores se superponen entre sí en una gran parte del tiempo de ejecución, con una diferencia de unos 2 *MB* en los valores máximos (picos). Esto se debe a la naturaleza del propio algoritmo, el cual toma una decisión basándose en el valor del *buffer* de cada *stream*.

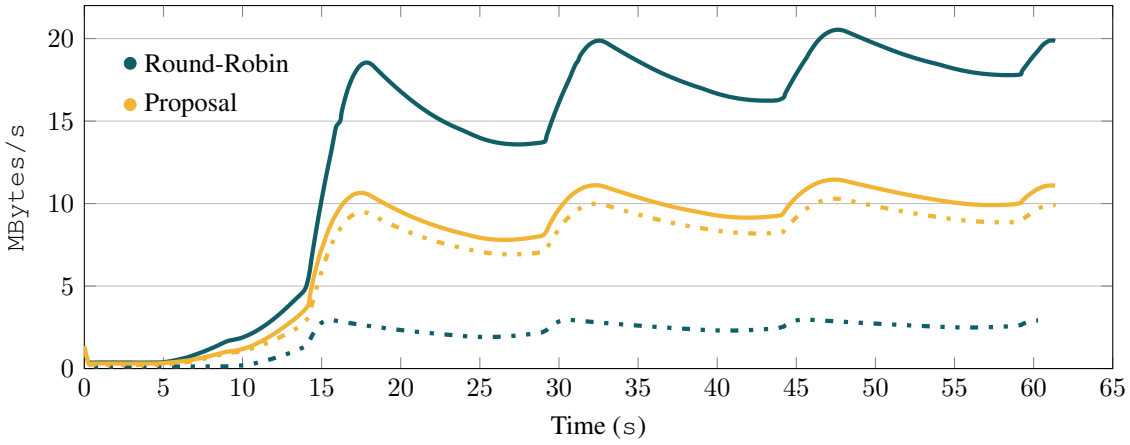
El uso de la ocupación del *buffer* como métrica permite tomar decisiones de manera oportunista, a diferencia del comportamiento promedio configurado el en WFQ. Ante tráfico variable, como lo es el tráfico de *Poisson* de las aplicaciones, la solución propuesta adapta las decisiones a la generación de tráfico en cada envío.

Para concluir esta sección, se procede a comprobar la métrica de estabilidad de colas propia del algoritmo propuesto. Para ello, se comparan únicamente entre dos algoritmos, el basado en el RR y el propuesto, ya que son los más representativos. En concreto, se pretende comprobar una métrica de estabilidad más robusta que la teórica (*mean rate stability*, representada en la Ecuación 2.4), puesto que es más restrictiva. Conocida como *strong stability*, esta métrica se define de la siguiente forma:

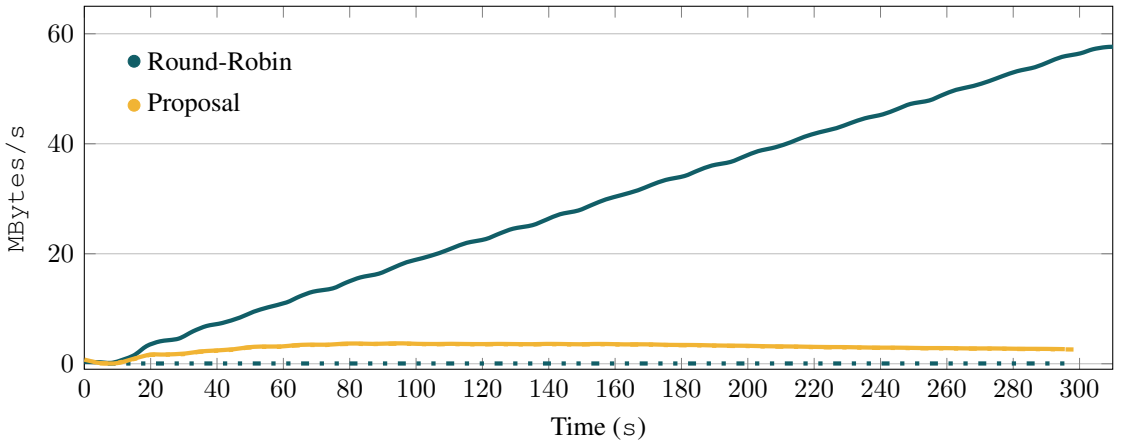
$$\lim_{T\to\infty} \quad \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{Q_k(t)\} < \infty \tag{4.1}$$

Como se puede observar, esta métrica tendría unidades de bits por segundo. En la Figura 4.11 se muestra la evolución de los *buffer* tanto en valor absoluto, como usando la métrica de *strong stability*. Estos resultados se han obtenido con una tasa de aplicación cercana a la saturación del canal.

Se comienza analizando el escenario que utiliza la configuración previa, cuyos resultados se muestran en la Figura 4.11a. Es decir, en este caso se usa una tasa conjunta (ambos *streams*) de 42 *Mbps* –siendo la capacidad del primer flujo de 27 *Mbps*, mientras que la del segundo es de 15 *Mbps*. En la Tabla 4.1 se establece el tamaño del fichero a enviar para conseguirlo. En esta figura se puede observar perfectamente las diferencias entre los algoritmos, así como la transición entre los diferentes estados del canal. La solución basada en el RR muestra evolución de valores totalmente distintas para cada flujo, mientras que el algoritmo propuesto proporciona valores muy parejos. Este comportamiento se debe a la lógica tras las diferentes soluciones, el cual se va a analizar en detalle a continuación. Asimismo, en todos los flujos



(a) Evolución del *buffer* de cada flujo.

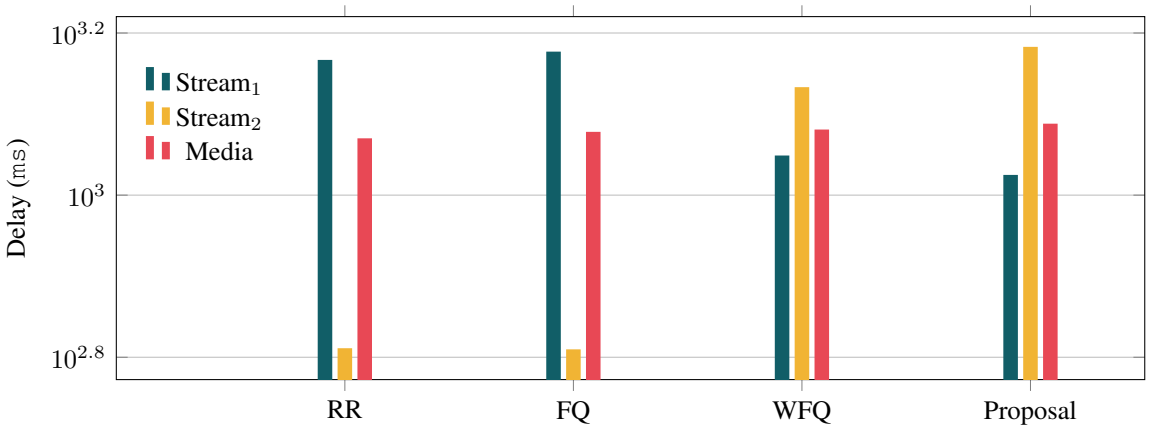


(b) Métrica *Strong stability* de los *buffer* de cada flujo.

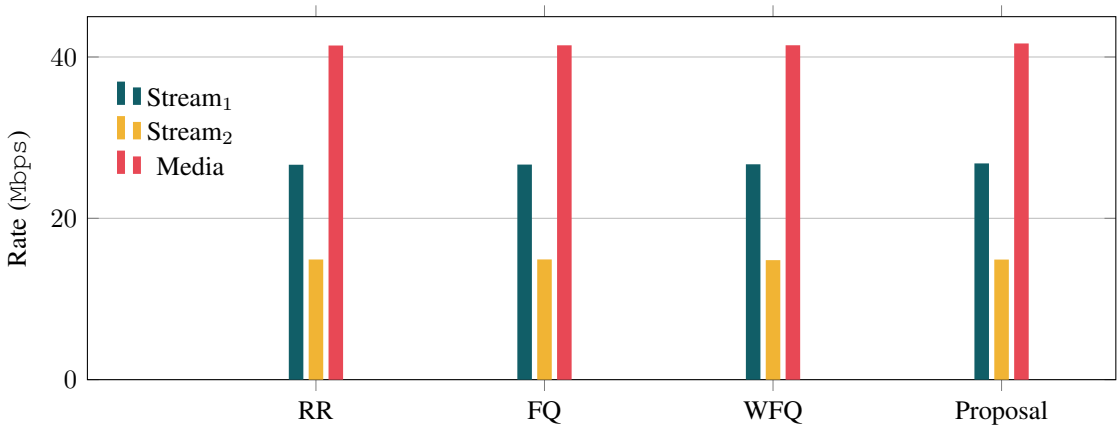
Figura 4.11: Evolución de los *buffer* al utilizar los algoritmos de *scheduling Round-Robin* y *Proposal*, en un canal controlado. Se representan los valores de los dos *streams* implementados, el primero de ellos con una línea sólida y el segundo con guiones punteados.

se puede ver la tendencia ondulada que se pudo ver en la figura anterior, causada por los cambios en el estado del canal.

La Figura 4.11b se centra en demostrar la estabilidad de colas. Para ello, se ha utilizado una configuración similar, pero no idéntica a la mostrada en la tabla anterior. En concreto, las variables que han sufrido una modificación han sido a nivel de aplicación, por un lado la tasa binaria de cada flujo y por otro, el tamaño del fichero a transmitir. La consecuencia de ello es poder realizar un envío de información (por parte del cliente) en un periodo de tiempo mayor. En este caso, se ha optado por simular durante cinco minutos –ficheros de 562.5 *MB* y 281.25 *MB*, respectivamente. Además, se ha disminuido las tasas, con la finalidad de alejarse de la capacidad de saturación del canal. Dando valores, el primer *stream* envía a 15 *Mbps* y el segundo a la mitad, 7.5 *Mbps*. La tasa total media es de 22.5 *Mbps*, el 50% de la tasa de saturación del canal. Lo que se puede ver a simple vista es cómo la velocidad del primer *stream* utilizando una planificación basada en el RR tiende a infinito (ya que el segundo flujo permanece a tasas extremadamente bajas). En cambio, la tendencia de la velocidad en los dos *streams* planificados por el algoritmo propuesto se mantiene completamente estable. Tanto es así que incluso los valores de ambos flujos se superponen.



(a) Retardo (*delay*)



(b) *Throughput*

Figura 4.12: Retardo y *throughput* experimentado por los diferentes algoritmos de *scheduling* implementados, en un canal controlado.

Por ende, se confirma que el algoritmo propuesto no solo cumple la métrica de estabilidad teórica, sino que en la práctica consigue una estabilidad de colas aún más exigente. Sin embargo, el algoritmo implementado por defecto en la implementación de QUIC utilizada no posee esta característica.

Para terminar de estudiar el comportamiento de los diferentes algoritmos de planificación, se muestra en la Figura 4.12 el retardo (Figura 4.12a) y el *throughput* (Figura 4.12b) experimentado a lo largo de las diferentes configuraciones –tras 30 ejecuciones independientes. Las diferencias más significativas se encuentran en la figura superior, sobre todo en los valores obtenidos al medir cada flujo por separado. Mientras que el retardo medio se mantiene similar en los cuatro algoritmos estudiados, existen grandes diferencias entre los *streams* en función del algoritmo adoptado.

Se comienza analizando el algoritmo basado en el RR, en donde se observa una clara diferencia entre los valores de los dos flujos. El primero de ellos tiene un retardo muy elevado, prácticamente el doble que el obtenido con el segundo *stream*. Esto se debe a que utiliza una tasa binaria mucho mayor, prácticamente el doble del siguiente *stream*. Es decir, tal y como se veía en la Figura 4.10, su *buffer* se incrementa rápidamente, mientras que el *buffer* del segundo flujo se vacía con mayor velocidad. Esta tendencia se repite al utilizar el algoritmo FQ, puesto que su lógica es muy similar a la del algoritmo previo. Sin embargo, al utilizar pesos (caso del algoritmo WFQ), el comportamiento empieza a variar debido que se

prioriza el envío del primer *stream* frente al segundo. Los valores del retardo entre flujos se asemejan más, incrementando el *delay* del segundo y decrementando el primero. La diferencia entre ambos es mucho menor, de unas pocas centenas de milisegundo. Por último, el algoritmo *Proposal Queuing* muestra un comportamiento parecido al algoritmo WFQ, aumentando ligeramente la distancia entre los valores del retardo de ambos flujos –al intentar equilibrar el tamaño del *buffer*, se aumenta el retardo del que utiliza la menor tasa.

Sin embargo, como se puede ver en la figura inferior, las tasas binarias tanto de los *streams* de forma independiente como la media en el sistema permanecen invariantes por el cambio de algoritmo. Es decir, las tasas binarias no se ven afectadas a pesar de las diferencias encontradas en el retardo.

La mejora del rendimiento en la planificación de los flujos al utilizar algoritmos que toman una decisión en función del estado del canal o del *buffer* es patente. La estabilidad de los diferentes *streams* ayuda a paliar el problema de la inestabilidad característica en canales variables, sin penalizar al retardo medio o al *throughput*.

Una vez se comprende el comportamiento de los diferentes algoritmos en un canal controlado, se procede a analizar el rendimiento de dichas implementaciones en un enlace más realista de comunicaciones satelitales.

4.2.2. Rendimiento de los algoritmos en un enlace LMS

En esta sección se quieren mostrar los resultados obtenidos al usar los diferentes algoritmos de planificación en un entorno inestable, como lo son los enlaces entre satélites LEO. Es por ello que se utiliza el canal LMS, previamente analizado en el apartado anterior. Se recuerda que la configuración de este canal se encuentra especificada en la parte inferior de la Tabla 3.1 (banda *Ka*). Estos valores indican que la capacidad media del canal es de 45.32 *Mbps*, tal y como se pudo comprobar al realizar el cálculo de la Ecuación 3.1.

Además, aparte de estudiar el rendimiento de los diferentes algoritmos analizando el retardo producido por cada solución, se desea evaluar la repercusión que puede tener el cambio de la tasa de bit de los diferentes *streams*.

Repercusión del cambio de la tasa de bit

Para comenzar el estudio, se procede a evaluar el impacto de la tasa binaria sobre el rendimiento de los *schedulers*. Se generan tres escenarios dentro de la configuración de la capa de aplicación, tal y como se muestra en la Tabla 4.2 –se mantiene tanto el tamaño del *buffer* como el del paquete de datos, así como el uso de dos *streams* para el envío de datos. Asimismo, se mantiene la distribución del tráfico (*Poisson*).

En el primer escenario se seleccionan las mismas tasas que se utilizaron para comprobar la métrica de estabilidad en la Figura 4.11a. Es decir, una dupla de tasas que suponen la mitad de la capacidad máxima o de saturación del canal LMS –cuyo valor se sitúa en unos 45.32 *Mbps*. Igualmente, se mantiene que uno de los flujos transmita a más velocidad que el otro, puesto que se desea comprobar cómo realiza la planificación el *scheduler* seleccionado. El segundo escenario mantiene la composición seleccionada al comienzo de este apartado, cercana al límite de la tasa del canal. En este caso, el conjunto de las tasas binarias de los flujos es de 42 *Mbps*. Finalmente, el tercer escenario se corresponde con una capacidad

Tabla 4.2: Configuración de la capa de aplicación y *buffer*.

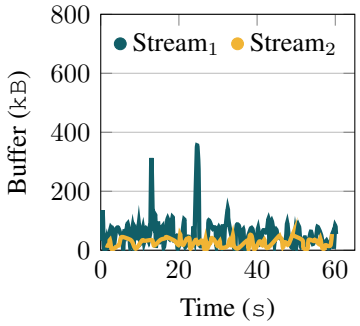
Capa de aplicación y *buffer*	
Tamaño del *buffer*	∞
Tamaño del paquete	1000 `Bytes`
QUIC #*streams*	2
	Escenario 1
Tasa (datos)	$Stream_1$: 15 `Mbps`; $Stream_2$: 7.5 `Mbps`
Tamaño fichero (datos)	$Stream_1$: 112.5 `MB`; $Stream_2$: 56.25 `MB`
	Escenario 2
Tasa (datos)	$Stream_1$: 27 `Mbps`; $Stream_2$: 15 `Mbps`
Tamaño fichero (datos)	$Stream_1$: 202.5 `MB`; $Stream_2$: 112.5 `MB`
	Escenario 3
Tasa (datos)	$Stream_1$: 30 `Mbps`; $Stream_2$: 15 `Mbps`
Tamaño fichero (datos)	$Stream_1$: 225 `MB`; $Stream_2$: 112.5 `MB`

prácticamente igual a la máxima, ya que las tasas de los dos flujos suman 45 *Mbps*. En este caso, se puede decir que nos encontramos en una situación de saturación.

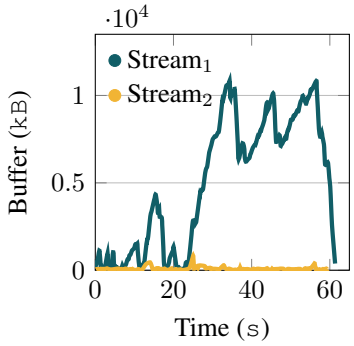
Se procede, por ende, a analizar la evolución de los *buffers* de las colas de cada flujo a lo largo de un periodo de tiempo. Igual que en el canal controlado, se emulan transmisiones de datos de un minuto. Se representa en la Figura 4.13 dicha evolución, tanto para los diferentes algoritmos implementados (filas) como para los distintos escenarios o tasas binarias (columnas).

El comportamiento general de los diferentes algoritmos implementados es coherente con lo observado en la sección anterior, prácticamente invariable al cambio en la tasa binaria de envío en los *streams*. En la mayoría de las soluciones, la ocupación del *buffer* del primer *stream* se mantiene superior a la existente en el segundo, ya que la relación entre las tasas individuales se ha mantenido. La excepción a este comportamiento se encuentra en el algoritmo propuesto, el cual iguala los valores de los dos flujos –aumentando la ocupación del *buffer* del segundo y reduciendo el primero. Es más, la superposición de valores es más evidente con el enlace LMS que con el canal de control (Figura 4.10). Como se puede observar, este comportamiento es más acusado al incrementar la tasa media de envío, acercándose a la de saturación del canal.

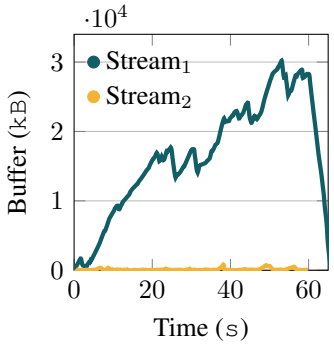
Se continúa la evaluación centrándose en las diferencias representadas en los valores obtenidos entre los distintos escenarios, es decir, los cambios entre las tasas binarias de cada flujo en cada algoritmo. En ellas, en concreto en el eje de ordenadas, se puede ver cómo aumenta el tamaño del *buffer*. Como era de esperar, cuanto menor es la tasa de envío, menor es la ocupación en las colas. Tanto es así que en el primer escenario se muestra como el canal es capaz de mantener ambos *buffers* vacíos la mayor parte del tiempo. Sin embargo, se produce una variación en la tendencia del comportamiento del algoritmo WFQ en el caso de máxima tasa binaria posible. Curiosamente, en esta configuración, los valores de los flujos toman una gran distancia entre sí, mientras que en la capacidad previa (cercana a la de saturación) se puede apreciar que dicha brecha es mucho menor. Este comportamiento se debe tanto al nivel de saturación, como a la variabilidad del escenario –tanto del canal como del tráfico. El algoritmo WFQ realiza un reparto medio proporcional al tráfico. Sin embargo, no aprovecha de forma óptima situaciones en las que la relación de



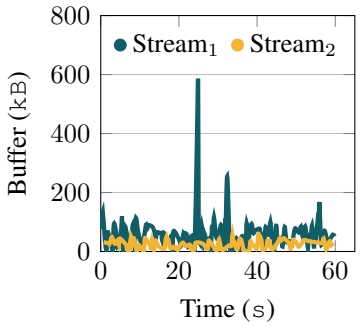
(a) Algoritmo RR.
Tasa binaria media de 22.5 *Mbps*.



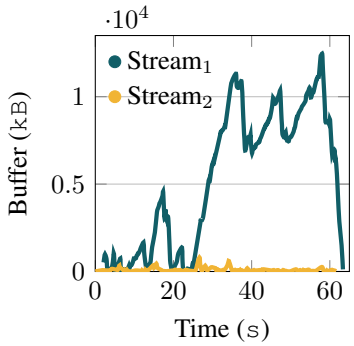
(b) Algoritmo RR.
Tasa binaria media de 42 *Mbps*.



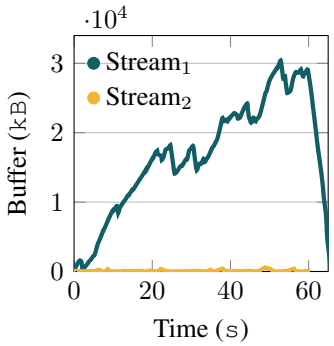
(c) Algoritmo RR.
Tasa binaria media de 45 *Mbps*.



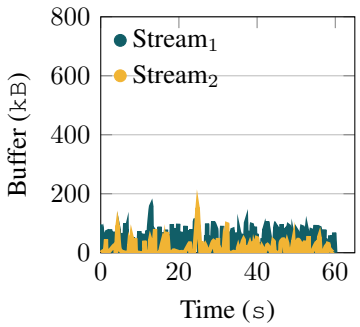
(d) Algoritmo FQ.
Tasa binaria media de 22.5 *Mbps*.



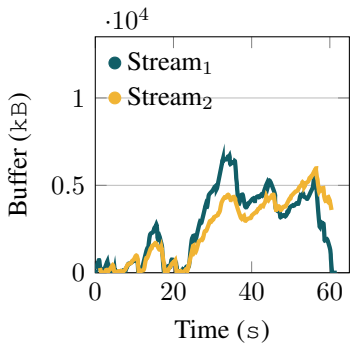
(e) Algoritmo FQ.
Tasa binaria media de 42 *Mbps*.



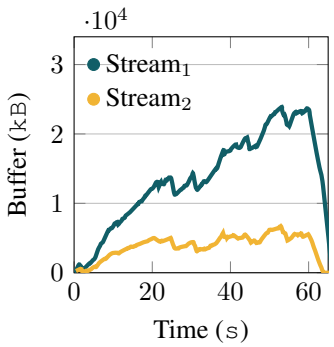
(f) Algoritmo FQ.
Tasa binaria media de 45 *Mbps*.



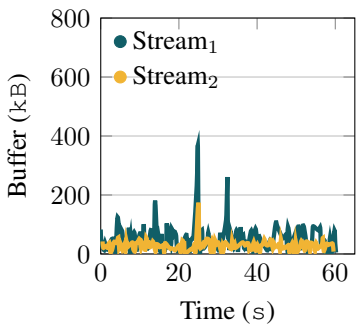
(g) Algoritmo WFQ.
Tasa binaria media de 22.5 *Mbps*.



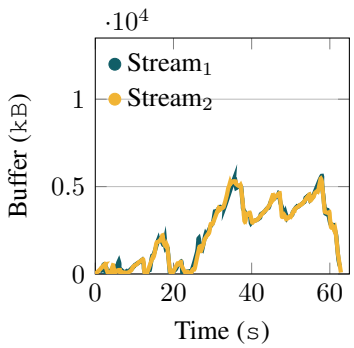
(h) Algoritmo WFQ.
Tasa binaria media de 42 *Mbps*.



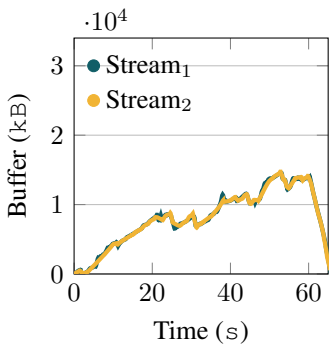
(i) Algoritmo WFQ.
Tasa binaria media de 45 *Mbps*.



(j) Algoritmo *Proposal Queuing*.
Tasa binaria media de 22.5 *Mbps*.



(k) Algoritmo *Proposal Queuing*.
Tasa binaria media de 42 *Mbps*.



(l) Algoritmo *Proposal Queuing*.
Tasa binaria media de 45 *Mbps*.

Figura 4.13: Evolución del *buffer* de la cola de los diferentes *streams* a lo largo del tiempo, utilizando tanto diferentes algoritmos de *scheduling* como tasas binarias de envío, en un enlace LMS.

tráfico no se corresponde con la media. Este comportamiento sub-óptimo se corrige a lo largo del tiempo (en media) cuando el canal no está saturado, pero en situaciones cercanas a la saturación esta corrección no se da.

Es importante aclarar que, para cada uno de los escenarios, se ha elegido un tamaño de fichero el cual permitiera realizar el envío de datos (por parte del cliente) en un minuto. Sin embargo, se puede observar en la figura que, cuanto menor es la tasa media de los flujos, menor es el tiempo en el cual se vacían por completo los *buffers*. En el primer escenario, la simulación acaba al minuto; en el segundo, sin embargo, aumenta ligeramente unos segundos. El tercer escenario es el que más tiempo necesita para desocupar los *buffers* empleados en la transmisión.

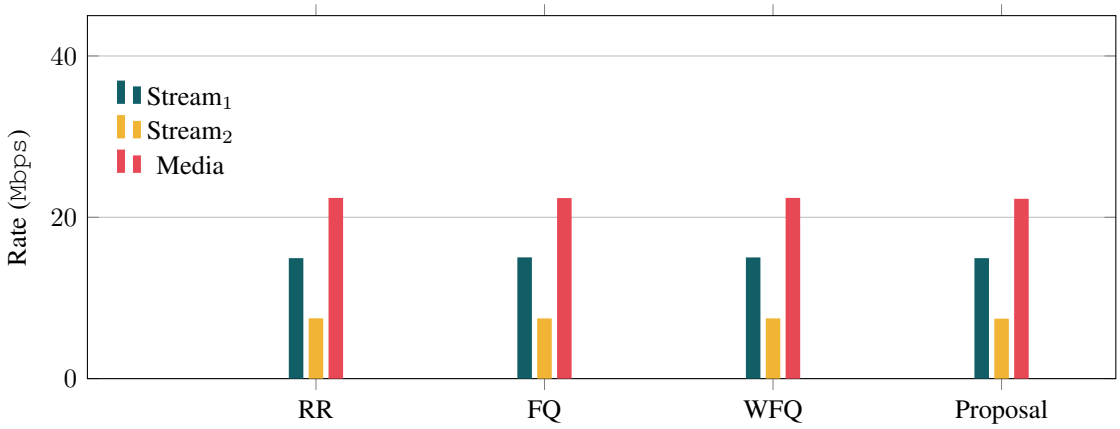
Una vez estudiado cómo afectan las diferentes tasas binarias en la ocupación de las colas, se analiza qué repercusión pueden tener dichos cambios en el *throughput* del sistema. Para ello se representa en la Figura 4.14 la tasa de recepción calculada al terminar la comunicación, de cada *stream* individual como la media del sistema.

La conclusión clara que se puede obtener tras estudiar las tres gráficas es que la variación en el algoritmo de planificación no afecta a la tasa binaria, ni individual ni media, del sistema. Asimismo, y como era de esperar, los valores del *rate* de cada flujo se corresponden con la tasa seleccionada para ellos. Por ende, se podría afirmar que prácticamente no se pierde capacidad en el envío, sobre todo en los dos primeros escenarios. Sin embargo, en el último escenario se observa que el *throughput* del primer flujo se penaliza en mayor medida –pierde casi un 10%, unos 3 *Mbps*. Este comportamiento se repite al evaluar la tasa media en los tres escenarios. Mientras que en los dos primeros el valor medio del *throughput* se asemeja a la tasa binaria de envío, en el tercero sufre una penalización de casi 5 *Mbps*. La explicación tras este hecho es simple, puesto que la configuración de las tasas seleccionadas en dicho escenario se diseñó con la idea de saturar el canal. Por ende, los resultados en este escenario son negativos.

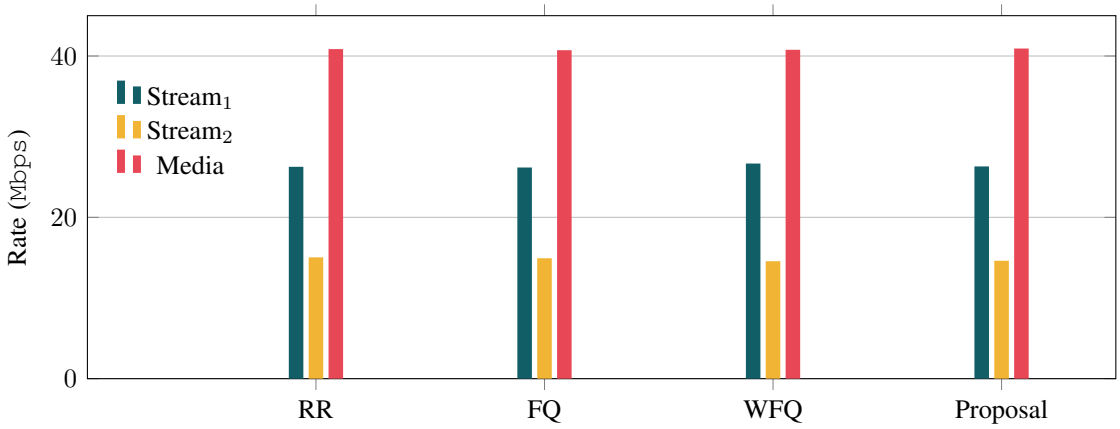
Se ha comprobado que la tasa seleccionada en la capa de aplicación tiene relevancia en el sistema. Cuando el canal se satura, los resultados obtenidos se ven afectados –ya sea por no lograr un buen rendimiento (limitando las tasas) o por la repercusión de este hecho en el comportamiento de los algoritmos (sobre todo, en el WFQ). Por otro lado, aunque el primer escenario no penaliza al rendimiento de la transmisión, al ocupar el canal únicamente al 50% no ofrece un entorno óptimo para analizar las distintas soluciones empleadas en el *scheduler*. Por otro lado, el tercer escenario se encuentra muy cercano a la saturación, por lo que el rendimiento también se ve mermado. Es por ello que se continúa el análisis con el segundo escenario.

Análisis del retardo obtenido en cada solución

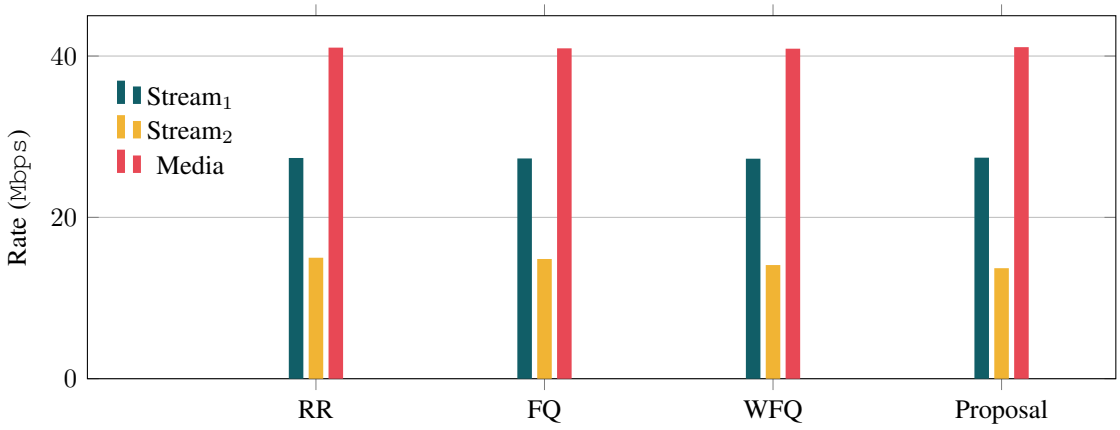
Para terminar de caracterizar los algoritmos de *scheduling* implementados, se estudia el retardo (tanto medio como individual) que sufren los diferentes *streams*. Tal y como ya se ha comentado, en esta sección se pretende hacer la evaluación con una configuración ya conocida. Para la capa de aplicación se implementa el segundo escenario, en el cual la tasa binaria media se encuentra cercana a la capacidad de saturación del canal. Asimismo, el canal consiste en un único enlace LMS, cuyas características se han descrito al comienzo de este capítulo.



(a) Escenario 1 – Tasa binaria media de envío igual a 22.5 *Mbps*

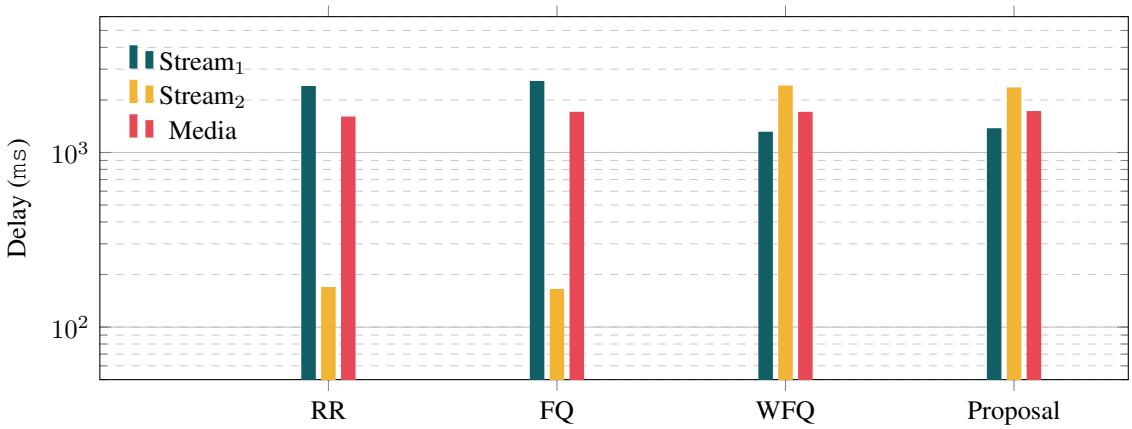


(b) Escenario 2 – Tasa binaria media de envío igual a 42 *Mbps*

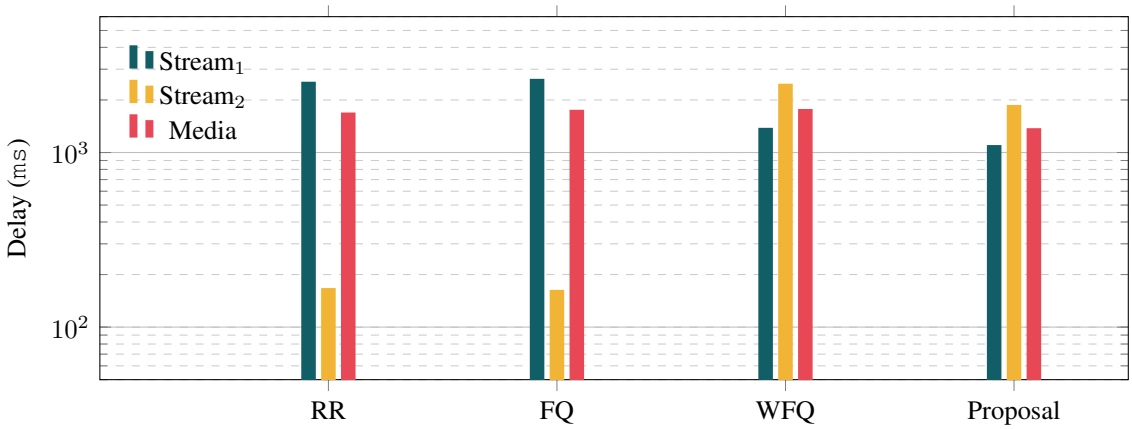


(c) Escenario 3 – Tasa binaria media de envío igual a 45 *Mbps*

Figura 4.14: *Throughput* experimentado tanto por diferentes algoritmos de *scheduling* como tasas binarias de envío, en un canal LMS.



(a) Tráfico con distribución constante o *fixed*



(b) Tráfico con distribución de *Poisson*

Figura 4.15: Retardo experimentado por los diferentes algoritmos de *scheduling* implementados, en un único enlace LMS.

En contraposición a las configuraciones previas, en este caso se pretende variar la distribución del tráfico generado. Para ello se evalúan los algoritmos trabajando tanto con tráfico de tasa constante (o *fixed*) y como con distribución de *Poisson*.

Para empezar, se analizan los valores del retardo experimentado en los cuatro algoritmos tanto para los dos flujos de manera independiente como de forma conjunta, calculando el retardo medio. En la Figura 4.15, se pueden ver los resultados obtenidos, los cuales se han generado tras realizar 30 ejecuciones independientes entre sí, con ambas distribuciones de tráfico –en la parte superior el caso más simple con tráfico constante, y en la inferior con tráfico de *Poisson*.

Al fijarse detenidamente en dicha figura, se puede apreciar como el retardo que se consigue tanto al utilizar un tráfico constante como con una distribución de *Poisson* es muy similar –para todos los algoritmos y flujos. La única diferencia notoria se encuentra en el caso del algoritmo propuesto, ya que para el segundo tráfico (Figura 4.15b) se obtienen retardos ligeramente menores.

Analizando las cuatro soluciones implementadas se puede observar el mismo comportamiento que se vio con el enlace de control. Mientras que el *delay* medio de la transmisión permanece prácticamente constante en todos los algoritmos, las diferencias se centran en los valores del retardo de los *streams* de manera independiente. Como era de esperar, para el algoritmo RR, el retardo experimentado por el primer

flujo es mayor que el del segundo. Este hecho se debe, de nuevo, a la ocupación del *buffer*. O en otras palabras, por la diferencia entre las tasas binarias elegidas. Por estas razones, el algoritmo FQ también tiene el mismo comportamiento. Como el planificador mantiene en espera a un mayor número de paquetes para el primer *stream*, su retardo incrementa. Sin embargo, tanto en el WFQ como en el *Proposal*, se consigue ajustar el envío de los *streams* de tal forma que los retardos de estos se igualan –siendo ahora el segundo ligeramente más elevado que el primero. Este hecho ocurre por mantener los paquetes del segundo flujo a la espera de ser enviados, mientras que transmite los del primero.

Por último, se muestra en la Figura 4.16 la distribución del retardo, para ambas distribuciones de tráfico. Gracias a estas gráficas se estudia el *jitter* (o variabilidad del retardo) producido por cada algoritmo de planificación. Asimismo, se representa el valor medio mediante marcadores circulares, el cual se acaba de analizar en la figura anterior.

Como se acaba de explicar para la gráfica previa, en este caso también se observa la diferencia de tasas entre los *streams*, sobre todo en las dos primeras soluciones. El *jitter* experimentado por el segundo flujo es menor, puesto que tiene más oportunidad de envío. Gracias a ello, sus datos se transmiten prácticamente sin demoras. Sin embargo, para los siguientes algoritmos la distribución de retardo en ambos flujos se iguala. Este comportamiento es consecuencia de que estos *schedulers* tienden a elegir con más probabilidad el primer flujo, induciendo mayor retardo y variación en el segundo.

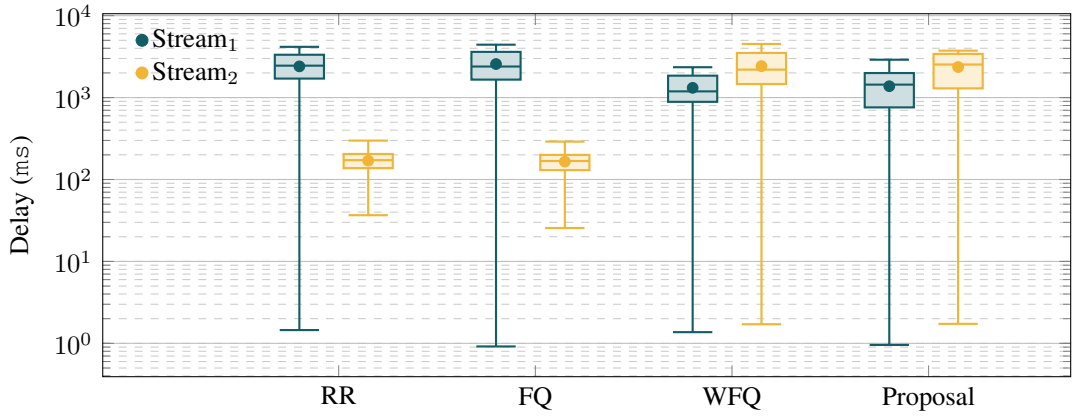
Al comparar ambas distribuciones de tráfico, *Fixed* y *Poisson*, se puede ver como el *jitter* parece aumentar ligeramente para todas las soluciones y flujos implementados con el segundo tráfico. La razón tras este comportamiento se encuentra en la aleatoriedad de la distribución del tráfico de *Poisson*. Pese a ello, no parece haber otras diferencias significativas.

Aunque por la distribución mostrada en el retardo (ambos flujos) para los algoritmos WFQ y *Proposal* parece indicar que el retardo medio del canal incrementa respecto a las otras soluciones, cabe destacar que no es así. Como se pudo ver en la Figura 4.15, el valor medio del retardo se mantiene muy similar para las cuatro soluciones. Este hecho se debe a la cantidad de información que se envía en cada flujo, ya que el retardo medio se calcula como la media de los retardos de todos los paquetes transmitidos.

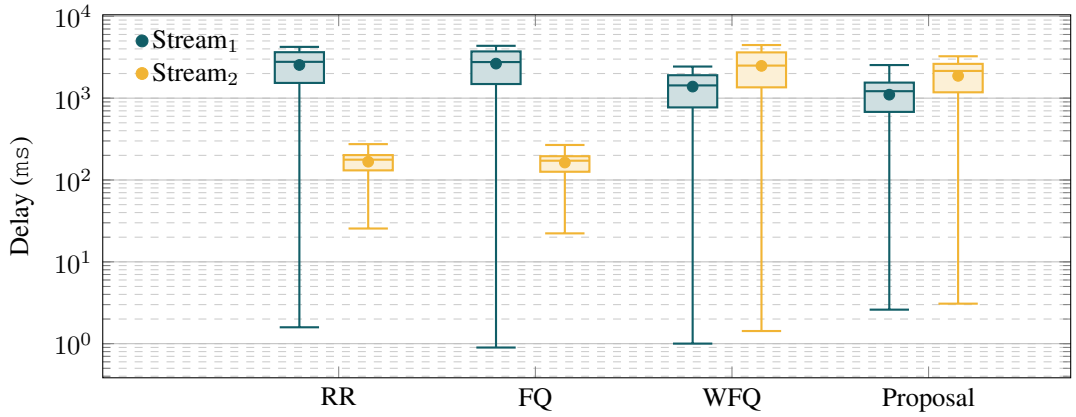
Para finalizar este apartado, se procede a recopilar las conclusiones que se han ido obteniendo a lo largo del mismo.

Una de las primeras conclusiones tiene que ver con el rendimiento del algoritmo de planificación aplicado por defecto por la implementación de QUIC. Este ha demostrado, junto con el algoritmo FQ, ser una solución funcional pero no óptima. Aunque consigue mantener una tasa binaria de transmisión correcta y el retardo medio no se ve gravemente afectado, no está diseñado para adaptarse a situaciones de alta variabilidad tanto de tasa como de capacidad (p. e. canales satelitales).

El siguiente algoritmo, la solución WFQ, proporciona un buen equilibrio entre el tamaño del *buffer* de los flujos utilizados. Sin embargo, tiene un gran inconveniente –sin contar la sobrecarga de bytes de cabecera. Este algoritmo necesita conocer *a priori* cuál es la distribución del envío de la información entre los *streams*, puesto que los pesos asignados a cada flujo van acorde a las tasas binarias (y tamaño de fichero). Es por ello que el algoritmo *Proposal Queuing* es el que ofrece realmente el mejor resultado, ya que sin penalizar al retardo o *throughput* medio del sistema, estabiliza las colas de los *streams*. Para ello solo necesita conocer cuál es el estado de dichos *buffers*, sin necesitar conocer cómo es el tráfico.



(a) Tráfico con distribución constante o *fixed*



(b) Tráfico con distribución de *Poisson*

Figura 4.16: *Boxplot* o diagrama de caja del retardo experimentado por los diferentes algoritmos de *scheduling* implementados, en un único enlace LMS. Se incluye el valor medio de cada realización con un marcador circular.

Capítulo 5

Conclusiones

El trabajo realizado se enmarca en un entorno complejo, el cual busca ofrecer al usuario dos propiedades esenciales hoy en día: elevadas tasas binarias y bajas latencias. A esto hay que añadir otro elemento fundamental, como es la cobertura. Las tecnologías de nueva generación, como 5G o *Beyond 5G*, son capaces de ofrecer las características deseadas, pero su escasa cobertura hace a esta solución demasiado complicada de implementar en algunos sectores. Este es el motivo principal por el que existe una creciente relevancia en el uso de las NTN para los próximos despliegues celulares. Sin embargo, aún existe cierta incertidumbre sobre el rendimiento que puedan proporcionar a los servicios. En concreto, se necesita conocer en detalle cuál es el comportamiento de los protocolos de transporte existentes sobre ellas, teniendo en cuenta la inestabilidad que las caracteriza. Es por ello que en este proyecto se ha investigado en este campo.

La metodología propuesta combina implementaciones reales junto con técnicas de virtualización y simulación. Para ello se han tenido que aunar diferentes tecnologías y herramientas, entre las que se destaca el simulador de redes NS-3. Este programa ha permitido la integración de un modelo realista de un canal LMS, basado en dos canales de frecuencias, la banda *Ka* y la banda *S*.

La primera parte del proyecto consistía en analizar cómo era el comportamiento de dos protocolos de transporte, TCP y QUIC, en distintos escenarios típicos de las comunicaciones LEO, obteniéndose varias conclusiones. La primera de ellas evidencia la mejora de QUIC respecto a los retardos obtenidos, menores que los observados con TCP. Sin embargo, en determinados escenarios, ello conllevaba un aumento de la variabilidad o *jitter*. Asimismo, se ha estudiado también la relevancia de la funcionalidad de *multistream* propia de QUIC. No obstante, los resultados obtenidos no eran del todo positivos, pues no siempre se produce una mejora relevante en el sistema.

Pese a lo anterior, se puede concluir que el comportamiento de los protocolos de transporte (tanto tradicionales como más novedosos) sobre las NTN (o, en este caso, sobre las redes LEO) no resulta óptimo. Los retardos siguen siendo algo elevados para las aplicaciones de hoy en día, en especial debido al efecto del algoritmo de control de congestión. Sin embargo, esta limitación es común a todos los protocolos. Es por ello que se ha querido analizar más a fondo las posibilidades que ofrece la gestión de los múltiples *streams* en QUIC. Tras comprobar que su rendimiento es mayor que el de TCP, se busca mejorarlo aún más, implementando para ello diferentes algoritmos de *scheduling –Fair Queuing*, *Weighted Fair Queuing* y uno que se propone en el marco del trabajo, *Proposal Queueing*.

Así, en la segunda parte del Trabajo Fin de Máster se ha examinado en profundidad la relevancia de la característica *multistream*, propia del protocolo QUIC. Para ello se ha evaluado el planificador o *scheduler*, aplicando diferentes algoritmos de planificación y comparándolos con la solución por defecto (en la implementación de QUIC utilizada), basada en el algoritmo RR. Este estudio ha esclarecido que el algoritmo por defecto no resulta el más óptimo para entornos con alta variabilidad, como los satelitales, puesto que no es capaz de estabilizar los retardos. Asimismo, se ha comprobado que tampoco estabiliza el tamaño del *buffer* de transmisión. Este comportamiento se repite al analizar la siguiente solución, basada en el algoritmo FQ.

En cambio, se ha comprobado que existen dos soluciones que proporcionan un comportamiento adecuado, basadas en los algoritmos WFQ y *Proposal Queueing*. En ambos casos parece que se consigue equilibrar los retardos que afectan a los datos enviados por los diferentes *streams*, sin penalizar al *delay* o *throughput* medio de la transmisión. Sin embargo, el algoritmo WFQ necesita conocer *a priori* la distribución del tráfico, un claro inconveniente. Es por ello que el algoritmo más apropiado, en el contexto de las comunicaciones LEO, es el propuesto en este trabajo, ya que tiene la capacidad de estabilizar las colas simplemente analizando el estado de los *buffers* de la aplicación.

Asimismo, los algoritmos justos (FQ y WFQ) no están realmente diseñados para utilizarse como planificadores en el protocolo QUIC. Uno de los motivos es la sobrecarga de información de cabeceras que añaden, ya que siempre envían información de todos los flujos empleados en la transmisión. Además, otro motivo importante reside en la retransmisión de *Stream Frames* por errores. Si hay una pérdida de un paquete, esta afecta a todos los flujos, acentuando el efecto *Head-of-line* (HOL) *blocking*. No obstante, al estudiar dichos algoritmos en un entorno realista de comunicaciones satelitales, este inconveniente no es demasiado importante –son entornos con grandes inestabilidades, pero baja probabilidad de errores y, normalmente, los *buffer* están sobredimensionados.

5.1. Trabajo futuro

A pesar de que este trabajo ha conseguido cumplir los objetivos propuestos al principio del mismo, a lo largo de la investigación llevada a cabo han aparecido nuevas líneas de trabajo. A continuación se enumeran las más relevantes y en las que se podría realizar un futuro estudio:

- Extender el estudio y la funcionalidad de *schedulers* basados en teoría de Lyapunov en varios sentidos:
    - Analizar el efecto de las penalizaciones o preferencias sobre los flujos junto con la estabilidad.
    - En este trabajo la ocupación de los *buffer* se ha utilizado como métrica de forma directa. Sin embargo la teoría permitiría definir funciones de esta ocupación, lo que permitiría establecer diferentes políticas.
    - Asimismo, existen trabajos en los que se demuestra que se puede garantizar la estabilidad del sistema usando el retardo acumulado del tráfico en el *buffer* de aplicación como métrica. Este enfoque, junto con el anterior, podría tener importantes beneficios a nivel de retardo.

- Explotar la característica de *multipath* que posee el protocolo QUIC. Esto permitiría agregar capacidad de transmisión de varios enlaces de forma simultánea. La interacción de esta funcionalidad con el *multi-stream* y el control de congestión podría dar lugar a diferentes esquemas de gestión de tráfico.
- Para el caso del *scheduler* de FQ y WFQ, realizar la distribución de paquetes en el tiempo. De esta manera se evita que en un paquete QUIC se envíe información de diferentes flujos, lo que puede acarrear problemas en un entorno con errores.
- Estudio de diferentes técnicas de control de congestión sobre el rendimiento del sistema. En concreto, los algoritmos de control de congestión basados en retardo podrían mitigar el efecto *buffer bloating*.
- Análisis sobre el efecto de las mejoras en la detección de congestión implementadas en el RFC 9002 de QUIC –numeración, cálculo de tiempos y gestión de reconocimientos.

Bibliografía

[1] T. Dillon, C. Wu y E. Chang. "Cloud Computing: Issues and Challenges". En: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. 2010, págs. 27-33. DOI: 10.1109/AINA.2010.187.

[2] L. Sánchez. "Cloud Computing". En: *Protocolos y Servicios para Redes de Nueva Generación* (2021). URL: https://www.tlmat.unican.es/index.php?l=es&p=teaching&s=postgraduate&ss=m_psrng& (visitado 18-07-2022).

[3] O. Rius. *Practical Applciations of Cloud, Edge and Fog Computing*. 2020. URL: https://www.e-zigurat.com/innovation-school/blog/cloud-edge-fog-computing-practical-applications/ (visitado 19-07-2022).

[4] F. Bonomi, R. Milito, J. Zhu y S. Addepalli. "Fog computing and its role in the internet of things". En: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. ACM Press, 2012, pág. 13. DOI: 10.1145/2342509.2342513.

[5] A. Gupta y R. K. Jha. "A Survey of 5G Network: Architecture and Emerging Technologies". En: *IEEE Access* 3 (2015), págs. 1206-1232. DOI: 10.1109/ACCESS.2015.2461602.

[6] "5G & Beyond". En: *NIST – National Institute of Standards and Technology* (2021). URL: https://www.nist.gov/programs-projects/5g-beyond (visitado 20-07-2022).

[7] I. Sarrigiannis, L. M. Contreras, K. Ramantas, A. Antonopoulos y C. Verikoukis. "Fog-Enabled Scalable C-V2X Architecture for Distributed 5G and Beyond Applications". En: *IEEE Network* 34.5 (2020), págs. 120-126. DOI: 10.1109/MNET.111.2000476.

[8] H. Inc. *Digital 2021 - Social Media Marketing & Management Dashboard*. URL: https://www.hootsuite.com/es/recursos/tendencias-digitales-2021 (visitado 21-07-2022).

[9] M. Hosseinian, J. P. Choi, S.-H. Chang y J. Lee. "Review of 5G NTN Standards Development and Technical Challenges for Satellite Integration With the 5G Network". En: *IEEE Aerospace and Electronic Systems Magazine* 36.8 (2021), págs. 22-31. DOI: 10.1109/MAES.2021.3072690.

[10] T. Darwish, G. K. Kurt, H. Yanikomeroglu, M. Bellemare y G. Lamontagne. "LEO Satellites in 5G and Beyond Networks: A Review From a Standardization Perspective". En: *IEEE Access* 10 (2022), págs. 35040-35060. DOI: 10.1109/ACCESS.2022.3162243.

[11] Z. Tang, H. Zhou, T. Ma, K. Yu y X. S. Shen. "Leveraging LEO Assisted Cloud-Edge Collaboration for Energy Efficient Computation Offloading". En: *2021 IEEE Global Communications Conference (GLOBECOM)*. 2021, págs. 1-6. DOI: 10.1109/GLOBECOM46510.2021.9685309.

[12] J. A. Moraño Fernández. "Análisis y ejemplos de órbitas circulares". En: *Departamento de Matemática Aplicada – Universitat Politècnica de València* (2019). URL: https://riunet.upv.es/handle/10251/132655 (visitado 23-07-2022).

[13] A. Bishen. *5G And Non-Terrestrial Networks*. 2022. URL: https://www.5gamericas.org/5g-and-non-terrestrial-networks/ (visitado 23-07-2022).

[14] N. Pachler, I. del Portillo, E. F. Crawley y B. G. Cameron. "An Updated Comparison of Four Low Earth Orbit Satellite Constellation Systems to Provide Global Broadband". En: *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2021, págs. 1-7. DOI: 10.1109/ICCWorkshops50388.2021.9473799.

[15] L. Diez. "Tema 4. Análisis de técnicas y protocolos de transporte: TCP". En: *Diseño y Operación de Redes Telemáticas* (2020). URL: https://www.tlmat.unican.es/siteadmin/submaterials/3358.pdf (visitado 23-07-2022).

[16] J. Iyengar y M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. 2021. DOI: 10.17487/RFC9000. URL: https://www.rfc-editor.org/info/rfc9000 (visitado 23-07-2022).

[17] J. Dizdarević y A. Jukan. "Experimental Benchmarking of HTTP/QUIC Protocol in IoT Cloud/Edge Continuum". En: *ICC 2021 - IEEE International Conference on Communications*. 2021, págs. 1-6. DOI: 10.1109/ICC42927.2021.9500675.

[18] P. Qian, N. Wang y R. Tafazolli. "Achieving Robust Mobile Web Content Delivery Performance Based on Multiple Coordinated QUIC Connections". En: *IEEE Access* 6 (2018), págs. 11313-11328. DOI: 10.1109/ACCESS.2018.2804222.

[19] V. Cerf, Y. Dalal y C. Sunshine. *Specification of Internet Transmission Control Program*. RFC 675. 1974. DOI: 10.17487/RFC0675. URL: https://www.rfc-editor.org/info/rfc675 (visitado 25-07-2022).

[20] J. Postel. *User Datagram Protocol*. RFC 768. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768 (visitado 25-07-2022).

[21] R. R. Stewart, C. Sharp, M. Kalla, D. V. Paxson, T. Taylor, K. Morneault, D. H. Schwarzbauer, Q. Xie, I. Rytina y L. Zhang. *Stream Control Transmission Protocol*. RFC 2960. 2000. DOI: 10.17487/RFC2960. URL: https://www.rfc-editor.org/info/rfc2960 (visitado 25-07-2022).

[22] T. Dreibholz, E. P. Rathgeb, I. Rüngeler, R. Seggelmann, M. Tüxen y R. R. Stewart. "Stream control transmission protocol: Past, current, and future standardization activities". En: *IEEE Communications Magazine* 49.4 (2011), págs. 82-88. DOI: 10.1109/MCOM.2011.5741151.

[23] S. Hogg. *What About Stream Control Transmission Protocol (SCTP)?* en. 2012. URL: https://www.networkworld.com/article/2222277/what-about-stream-control-transmission-protocol--sctp--.html (visitado 25-07-2022).

[24] R. R. Stewart, M. Tüxen y K. Nielsen. *Stream Control Transmission Protocol*. RFC 9260. 2022. DOI: 10.17487/RFC9260. URL: https://www.rfc-editor.org/info/rfc9260 (visitado 25-07-2022).

[25] S. Floyd, M. J. Handley y E. Kohler. *Datagram Congestion Control Protocol (DCCP)*. RFC 4340. 2006. DOI: 10.17487/RFC4340. URL: https://www.rfc-editor.org/info/rfc4340 (visitado 25-07-2022).

[26] Y.-N. Lien e Y.-C. Ding. "Can DCCP Replace UDP in Changing Network Conditions?" En: *2011 IEEE International Conference on Advanced Information Networking and Applications*. ISSN: 2332-5658. 2011, págs. 716-723. DOI: 10.1109/AINA.2011.101.

[27] M. Bagnulo. *Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6181. 2011. DOI: 10.17487/RFC6181. URL: https://www.rfc-editor.org/info/rfc6181 (visitado 25-07-2022).

[28] Google. *QUIC, a multiplexed transport over UDP*. URL: https://www.chromium.org/quic/ (visitado 29-07-2022).

[29] D. C. Marinescu. "Chapter 9 - Cloud Resource Management and Scheduling". En: *Cloud Computing (Second Edition)*. 2018, págs. 321-363. DOI: https://doi.org/10.1016/B978-0-12-812810-7.00012-1.

[30] P. Cabalar. "TEMA III. PROCESOS". En: *Sistemas Operativos* (2022). URL: https://www.dc.fi.udc.es/~so-grado/SO-Procesos-planif.pdf (visitado 04-07-2022).

[31] R. Agüero. "Tema 3. Algoritmos y Protocolos de Nivel de Red". En: *Diseño y Operación de Redes Telemáticas* (2015). URL: https://ocw.unican.es/course/view.php?id=22 (visitado 04-07-2022).

[32] A. E. García. "Concepto de Integración de Servicios". En: *Arquitecturas de Red para la Integración de Servicios* (2021). URL: https://www.tlmat.unican.es/index.php?l=es&p=teaching&s=postgraduate&ss=m_aris& (visitado 04-07-2022).

[33] G. Varghese. "Chapter 18 - Conclusions". En: *Network Algorithmics*. 2005, págs. 417-432. DOI: https://doi.org/10.1016/B978-012088477-3/50022-9.

[34] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan y Claypool Publishers, 2010.

[35] G. F. Riley y T. R. Henderson. "The ns-3 Network Simulator". en. En: *Modeling and Tools for Network Simulation*. Ed. por K. Wehrle, M. Güneş y J. Gross. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, págs. 15-34. DOI: 10.1007/978-3-642-12331-3_2. URL: http://link.springer.com/10.1007/978-3-642-12331-3_2 (visitado 01-08-2022).

[36] M. Zverev, P. Garrido, F. Fernández, J. Bilbao, Ö. Alay, S. Ferlin, A. Brunstrom y R. Agüero. "Robust QUIC: Integrating Practical Coding in a Low Latency Transport Protocol". En: *IEEE Access* 9 (2021), págs. 138225-138244. DOI: 10.1109/ACCESS.2021.3118112.

[37] P. Garrido, I. Sanchez, S. Ferlin, R. Aguero y O. Alay. "rQUIC: Integrating FEC with QUIC for Robust Wireless Communications". En: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, págs. 1-7. DOI: 10.1109/GLOBECOM38437.2019.9013401.

[38] F. Fontan, M. Vazquez-Castro, C. Cabado, J. Garcia y E. Kubista. “Statistical modeling of the LMS channel”. En: *IEEE Transactions on Vehicular Technology* 50.6 (2001), págs. 1549-1567. DOI: 10.1109/25.966585.

[39] N. J. H. Marcano, L. Diez, R. A. Calvo y R. H. Jacobsen. “Finite Buffer Queuing Delay Performance in the Low Earth Orbit Land Mobile Satellite Channel”. En: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. 2022, págs. 132-137. DOI: 10.1109/WCNC51071.2022.9771859.

[40] N. J. H. Marcano, L. Diez, R. A. Calvo y R. H. Jacobsen. “On the Queuing Delay of Time-Varying Channels in Low Earth Orbit Satellite Constellations”. En: *IEEE Access* 9 (2021), págs. 87378-87390. DOI: 10.1109/ACCESS.2021.3089005.

[41] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert y R. Scheffenegger. *CUBIC for Fast Long-Distance Networks*. RFC 8312. 2018. DOI: 10.17487/RFC8312. URL: https://www.rfc-editor.org/info/rfc8312 (visitado 04-08-2022).

[42] J. Iyengar e I. Swett. *QUIC Loss Detection and Congestion Control*. RFC 9002. 2021. DOI: 10.17487/RFC9002. URL: https://www.rfc-editor.org/info/rfc9002 (visitado 19-09-2022).