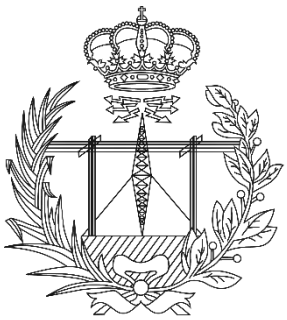
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Trabajo Fin de Grado

Evaluación de las prestaciones de un entorno 5G para la Internet táctil
(Performance evaluation of a 5G framework for Tactile Internet)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Miguel Barón Herrá

Septiembre - 2022

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

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Miguel Barón Herrá
Director del TFG: Luis Muñoz Gutiérrez, José Ramón Juárez
Título: "Evaluación de las prestaciones de un entorno 5G para la Internet táctil"
Title: "Performance evaluation of a 5G framework for Tactile Internet"

Presentado a examen el día: 5 de septiembre de 2022

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Muñoz Gutiérrez, Luis
Secretario (Apellidos, Nombre): Crespo Fidalgo, José Luis
Vocal (Apellidos, Nombre): García Arranz, Marta

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

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

En primer lugar, me gustaría agradecer a mi director de Trabajo de Fin de Grado, el Catedrático Luis Muñoz Gutiérrez, por la confianza depositada en mí y su inestimable disposición. Asimismo, a la entidad donde he podido desarrollar el proyecto, Ikerlan, S. Coop. y, principalmente, a mi tutor, José Ramón Juárez, por todas las facilidades que me han puesto a la hora de realizar un proyecto de tanta actualidad y proyección de futuro.

Querría destacar la altísima calidad humana de los demás miembros de las distintas áreas de la empresa, así como del Grupo de Ingeniería Telemática del Departamento de Ingeniería de Comunicaciones de la Universidad de Cantabria, especialmente, el equipo de IoT y Plataformas Digitales y los miembros del Laboratorio de Redes y Servicios, por hacerme mucho más amenos mis días, tanto en Mondragón como en Santander.

Por último, me gustaría agradecer a mi familia y amigos, por su constante apoyo e infinita paciencia, con quienes me siento muy afortunado de poder contar. Sin las fuerzas y seguridad que me han dado, no hubiera podido llegar hasta aquí.

Resumen Ejecutivo

Las retransmisiones en directo en línea, los videojuegos con múltiples jugadores, los procesos de automatización industrial, las intervenciones quirúrgicas con realidad aumentada, las comunicaciones V2X para conducción autónoma, o la automatización de terminales portuarias son algunos de los ejemplos de los servicios y aplicaciones que las tecnologías 5G están llamadas a cubrir, ofreciendo latencias ultra bajas, tasas de transferencia elevadas y una disponibilidad del servicio óptima.

Con estas condiciones de contorno, en este trabajo se realiza un análisis del rendimiento de la infraestructura pública 5G y se evalúa su comportamiento en una aplicación con requerimientos de baja latencia, propia del paradigma de la Internet táctil. Se desarrolla, para ello, una implementación del protocolo de comunicaciones MQTT, estableciendo una conexión entre un *broker* local y un *broker* en la parte extrema de la red, a través de conexión celular en la que se ponen en evidencia diferentes parámetros relacionados con la calidad de servicio, para los diferentes estándares, a saber, 3G, 4G y 5G.

Executive Abstract

Video live streaming, online multiplayer video games, industrial automation processes, augmented reality assisted surgery, V2X communications for autonomous driving or terminal automation are some of the examples of the services and applications that 5G technologies are called to cover, offering ultra-low latencies, high throughputs, and optimal service availability.

With these boundary conditions, in this project a performance analysis of the 5G public infrastructure and evaluation of its behaviour in an application with low latency requirements, typical of the Tactile Internet paradigm, is carried out. For this purpose, an implementation of the MQTT communication protocol, is developed, establishing a connection between a local broker and a broker at the Edge Network, through a cellular connection in which parameters related to quality of service are revealed, for the different standards, i.e., 3G, 4G y 5G.

Índice general

Índice de figuras

Índice de tablas

Acrónimos

3GPP Third Generation Partnership Project

5GS Fifth Generation System

ACK Acknowledgement

AMQP Advanced Message Queuing Protocol

AN Access Network

APN Access Point Name

AR Augmented Reality

ARFCN Absolute Radio-Frequency Channel Number

AT Attention

BS Base Station

BTB Board-to-Board

BTS Base Transceiver Station

CN Core Network

CoAP Constrained Application Protocol

CP Cyclic Prefix

DL Downlink

DSS Dynamic Spectrum Sharing

E-UTRAN Evolved UTRAN

Ec/Io Energy per chip to Interference power ratio

EDGE Enhanced Data Rates for GSM Evolution

eMBB enhanced Mobile Broadband

EN-DC E-UTRAN New Radio Dual Connectivity

eNB Evolved Node B

EPC Evolved Packet Core

EVB Evaluation Board

FDD Frequency Division Duplex

gNB Next generation Node B

GPRS General Packet Radio Service

GSM Global System for Mobile communications

HMI Human-Machine Interaction

HOL Head-of-line

HSDPA High Speed Downlink Packet Access

HSUPA High Speed Uplink Packet Access

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ICI Intercarrier interference

IETF Internet Engineering Task Force

IIoT Industrial IoT

IoT Internet of Things

IP Internet Protocol

ISI Intersymbol interference

JSON JavaScript Object Notation

LAC Location Area Code

LAN Local Area Network

LPWA Low-Power Wide-Area

LTE Long Term Evolution

M2M Machine-to-Machine

MBMS Multimedia Broadcast Multicast Service

MCC Mobile Country Code

MMS Multimedia Messaging Service

mMTC massive Machine Type Communications

MNC Mobile Network Code

MQTT Message Queue Telemetry Transport

MSM Mobile Station Modem

NAS Network Access Service

NB Node B

NB-IoT Narrowband IoT

ng-eNB Next Generation eNB

NR New Radio

NSA Non-Stand Alone

OFDM Orthogonal Frequency-Division Multiplexing

PC Personal Computer

PCI Physical Cell ID

PDB Packet Delay Budget

PPP Point-to-Point Protocol

QAM Quadrature Amplitude Modulation

QMI Qualcomm MSM Interface

QoS Quality of Service

RAC Routing Area Code

RAN Radio Access Network

REST Representational State Transfer

RSCP Received Signal Code Power

RSRP Reference Signal Received Power

RSRQ Reference Signal Received Quality

RSSI Received Signal Strength Indicator

RTT Round-Trip Time

SA Stand Alone

SCS Subcarrier Spacing

SIM Subscriber Identity Module

SINR Signal to Interference and Noise Ratio

SMS Short Message Service

SSL Secure Sockets Layer

TAC Tracking Area Code

TCP Transmission Control Protocol

TDD Time Division Duplex

TI Tactile Internet

TIC Tecnologías de la Información y de la Comunicación

TLS Transport Layer Security

UDP User Datagram Protocol

UE User Equipment

UI User Identifier

UIT Unión Internacional de Telecomunicaciones

UL Uplink

UMTS Universal Mobile Telecommunications System

URLLC Ultra Reliable and Low Latency Communications

USB Universal Serial Bus

UTRAN UMTS Terrestrial RAN

V2X Vehicle to Everything

VR Virtual Reality

WAGF Wireless Access Gateway Function

WCDMA Wideband Code Division Multiple Access

Capítulo 1

Introducción

En este primer capítulo, se presentan los distintos conceptos y tecnologías que se abordan en el trabajo. Asimismo, se exponen los objetivos fijados y la estructura adoptada para la redacción de la presente memoria, sintetizando los aspectos principales de cada capítulo constitutivo.

Las Tecnologías de la Información y la Comunicación (TIC) prosiguen con su vertiginosa evolución en lo que parece un proceso sin fin. Ahora, con la llegada de la quinta generación de tecnologías de comunicaciones móviles (5G), se espera un salto aún mayor, especialmente en términos de velocidad de transmisión, latencia y consumos energéticos, con respecto a las tecnologías predecesoras.

Estas mejoras quieren suponer una revolución dentro del mundo del Internet de las cosas (IoT, *Internet of Things*), puesto que permiten obtener la respuesta a diferentes sucesos o acciones realizadas, con un nivel de inmediatez muy elevado y en dispositivos que no requieren utilizar demasiados recursos. La gran cantidad de conexiones simultáneas que el 5G permite gestionar también será un aspecto muy ventajoso en aplicaciones IoT.

Otro terreno que podrá verse muy beneficiado es el del paradigma de la Internet táctil (TI, *Tactile Internet*): la Unión Internacional de Telecomunicaciones (UIT) presenta la combinación de latencias extremadamente bajas junto con la alta disponibilidad, confiabilidad y seguridad, como los elementos definitorios de la TI [1].

Dicho paradigma conlleva la materialización de la interacción ser humano-máquina (HMI, *Human-Machine Interaction*) a un siguiente estado, dado que habilita la interacción háptica con medios físicos o contenidos audiovisuales, para tecnologías inmersivas con realidad virtual (VR, *virtual reality*) o realidad aumentada (AR, *augmented reality*).

1.1. Objetivos y estructura de la memoria

Dentro de este marco de la Internet táctil, se plantea, como principal objetivo, realizar el desarrollo y despliegue de una aplicación de baja latencia en un entorno 5G, que permita mostrar y validar el comportamiento de la tecnología de una manera práctica, en base a parámetros relacionados con la experiencia del usuario.

Se propone utilizar un sensor con tecnología infrarroja, que capte las manos del usuario, y poder visualizar los movimientos en remoto. Para ello deberá desarrollarse una aplicación que permita recoger los datos proporcionados por el sensor y los envíe a un servidor en el *Edge*, segmento extremo de la red, de modo que cualquier otro cliente pueda acceder a ellos.

Se utilizará un módem de comunicaciones móviles para aportar conectividad inalámbrica al sistema. Deberá poder ser controlado remotamente, así como accesible desde cualquier equipo, de manera que puedan obtenerse datos útiles de la red y del propio módem, que permitan monitorizar y caracterizar las conexiones de una forma más avanzada.

Se persigue, además, analizar el rendimiento de la infraestructura pública 5G, en base a medidas de latencia y caudal útil, así como otros parámetros de calidad que puedan obtenerse en diferentes escenarios reales. Los resultados deberán ayudar a comparar las prestaciones de la tecnología 5G con las de las tecnologías celulares de tercera y cuarta generación.

Para todo ello, se hará uso del protocolo *Message Queue Telemetry Transport* (MQTT), cuya presencia en escenarios de comunicaciones IoT está muy extendida.

Con objeto de cubrir los objetivos anteriormente mencionados, el presente documento se organiza en base a cinco capítulos. Tras este primero, dedicado a la introducción y objetivos del proyecto, en el segundo se analiza el estado del arte de las principales tecnologías y protocolos de comunicaciones celulares para aplicaciones IoT. En el Capítulo 3 se presenta el sistema propuesto, detallando el diseño y las implementaciones llevadas a cabo. Seguidamente, en el Capítulo 4, se analizan los resultados obtenidos. Finalmente, en el Capítulo 5, se detallan las conclusiones y se exponen futuras líneas de investigación que derivan del trabajo realizado.

Capítulo 2

Revisión del estado del arte

En este capítulo se realiza una revisión del estado del arte de las tecnologías de mercado más relevantes dentro del marco de este trabajo, como son las redes y módulos para comunicaciones celulares, así como los protocolos más habituales en aplicaciones IoT.

2.1. Redes celulares

En este apartado se presenta la evolución que han seguido las tecnologías de redes celulares, desde las denominadas de primera generación, hasta las presentes, de quinta generación.

2.1.1. Evolución

En los últimos cincuenta años la telefonía móvil ha sido, indudablemente, una de las tecnologías que más cambios ha experimentado. Las redes celulares han jugado un papel de suma importancia en este proceso donde los avances son cada vez mayores y se producen cada vez más rápido.

Partiendo de comunicaciones que solo permitían voz analógica, se han venido desarrollando tecnologías que posibilitan interconectar miles de millones de dispositivos, con latencias ultra bajas y con velocidades de transmisión de hasta 10 Gbps.

Las redes de comunicaciones de primera generación (1G) fueron desplegadas, por primera vez, en 1979 en Japón. No fue hasta principios de los años ochenta que comenzaron a popularizarse en Europa y Estados unidos. Se trataban de comunicaciones analógicas, de calidad limitada y que no contaban con ningún tipo de seguridad.

Con la llegada de las comunicaciones móviles de segunda generación (2G), a principios de los años noventa, se digitalizan las comunicaciones, llegándose a soportar tasas relativamente bajas, en comparación con las más actuales, pero que en el momento supusieron un gran avance tecnológico. La introducción del servicio de mensajería *Short Message Service* (SMS) o la itinerancia transnacional fueron algunas de las aportaciones del *Global System for Mobile communications* (GSM): el estándar europeo clave en las comunicaciones móviles 2G.

Entre los años 2000 y 2003 se estandarizaron las tecnologías *General Packet Radio Service* (GPRS) y *Enhanced Data Rates for GSM Evolution* (EDGE), que conllevaron una importante mejora en las velocidades de transmisión. A caballo entre las redes de segunda y tercera generaciones, son popularmente conocidas como redes 2.5 G.

Con el estándar *Universal Mobile Telecommunications System* (UMTS) se consolidan las comunicaciones móviles de tercera generación (3G). Surgieron de un intento de crear un estándar a nivel mundial, con tasas binarias mucho más elevadas y con mayor seguridad. Servicios como las videollamadas, la navegación por Internet, el correo electrónico o el acceso a vídeos bajo demanda y otros contenidos multimedia llegaron de la mano de esta tecnología.

En 2010 fueron desplegados los primeros servicios pertenecientes a la cuarta generación de tecnologías de telefonía móvil (4G), las cuales se acogían al estándar *Long Term Evolution* (LTE). Este provee de velocidades superiores a las tecnologías precedentes, con alta calidad y gran capacidad de usuarios. Se minimizan los costes de los servicios de voz y datos, multimedia y voz sobre IP.

Las características fundamentales de cada tecnología de comunicaciones móviles se muestran en la Tabla 2.1.

Tabla 2.1: Características fundamentales de las tecnologías de comunicaciones celulares. Elaboración propia. Basado en [2].

	1G	*2G*	*2.5G*
Año	1979 (Japón), 1981	1991	2000 – 2003
Estándares	AMPS	GSM	GPRS EDGE
Tecnología	Analógica	Digital	Digital
Velocidad	-	13 kbps	GPRS → 115 kbps EDGE → 384 kbps
Multiplexación / Acceso	FDMA	TDMA	GMSK (GPRS) 8-PSK (EDGE)
Conmutación	Conmutación de circuitos	Conmutación de circuitos	Conmutación de paquetes
Frecuencia	800 – 900 MHz	850 – 1900 MHz	850 – 1900 MHz
Ancho de banda	25 kHz 832 canales	Radiocanales de 200 kHz divididos en 8 canales de 25 kHz	Radiocanales de 200 kHz divididos en 8 canales de 25 kHz
	3G	*4G*	*5G*
Año	2000	2010	2015
Estándares	UMTS (GSM) CDMA 2000 (IS-95) TD – SCDMA	LTE – TDD LTE – FDD WiMAX	IP LAN W AN PAN & WWWW
Tecnología	Digital	Digital, IP	Digital
Velocidad	384 kbps – 2 Mbps	100 Mbps (Movimiento) 1 Gbps (Reposo)	1 – 10 Gbps
Multiplexación / Acceso	WCDMA HSPA HSPA+ CDMA2000 1x	OFDM MC – CDMA CDMA LAS – Red – LMDS	CDMA BDMA
Conmutación	Conmutación de circuitos Conmutación de paquetes	Conmutación de paquetes	Conmutación de paquetes
Frecuencia	800 MHz – 2,5 GHz	700, 800, 900, 1800, 2600 MHz (EU)	3 – 300 GHz
Ancho de banda	5 – 20 MHz	5 – 20 MHz (opcional – 40 MHz)	Hasta 2 GHz

3GPP-Third Generation Partnership Project

De manera que se pudieran realizar estandarizaciones globales para tecnologías móviles, en 1998 se origina el proyecto de asociación de tercera generación (3GPP, *Third Generation Partnership Project*), constituido por las siete organizaciones de normalización de la industria de las telecomunicaciones de América, Asia y Europa de mayor relevancia (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA y TTC).

Hasta la fecha, se han aprobado un total de diecinueve versiones o *releases*, como se refleja en la Tabla 2.2. A la fecha de la realización de este Trabajo, se están ultimando los detalles de la *Release* 17, y comenzando con el desarrollo de la *Release* 18.

Tabla 2.2: Detalles de las diferentes entregas de 3GPP. Elaboración propia. Basado en [3].

Release	*Fecha*	*Detalles*
Phase 1	1992	Características básicas de **GSM**.
Phase 2	1995	Características de GSM, codificación EFR.
Release 96	1997	Mejoras de GSM, 14.4 kbps para datos de usuario.
Release 97	1998	Características adicionales de GSM, GPRS.
Release 98	1999	Características adicionales de GSM, GPRS para PCS 1900, AMR, EDGE.
Release 99	2000	**UMTS**, acceso radio WCDMA.
Release 4	2001	Mejoras de UMTS, red de núcleo all-IP.
Release 5	2002	IMS y HSDPA.
Release 6	2004	HSUPA, MBMS, mejoras de IMS (PoC, GAN), operación con WLAN.
Release 7	2007	Mejoras en QoS y latencia, VoIP, HSPA+, EDGE Evolution, NFC.
Release 8	2008	**LTE**, SAE, OFDMA, FDE, MIMO, Dual-Cell HSDPA.
Release 9	2009	Mejoras de SAE, interoperabilidad LTE/UMTS, WiMAX, Dual-Cell HSUPA, Dual-Cell HSDPA con MIMO, LTE HeNB, eMBMS.
Release 10	2011	**LTE Advanced**, Multi-Cell HSDPA.
Release 11	2012	HetNet, CoMP, IDC, IPXS.
Release 12	2015	Mejoras en micro celdas, agregación de portadora, SECAM.
Release 13	2016	**LTE Advanced Pro**, LTE-Unlincensed, integración LTE con Wi-Fi.
Release 14	2017	Eficiencia energética, LCS, Mission Critical Data sobre LTE, Mission Critical Video sobre LTE, FMSS, MBSP, mejoras para servicios de TV sobre eMBMS, MIoT, CBS.
Release 15	2018	**5G NR**, V2X, IMS, FRMCS.
Release 16	2020	Mejoras de 5G, NR-U, Satellite Access, IIoT/TSN, IAB, posicionamiento.
Release 17	2022	Mejora de rendimiento de sistema 5G, características de eMBB, URLLC, mMTC.
Release 18	2023	5G Advanced.

A continuación, se realizará una exposición del estado del arte de las redes de quinta generación de comunicaciones móviles (5G), basándose en las últimas versiones del estándar publicadas.

2.1.2. Redes 5G

La llegada de las redes 5G trae consigo una plétora de opciones tecnológicas que las empresas podrán aprovechar para desarrollar proyectos que anteriormente era inimaginable llevar a la práctica. El mundo del Internet de las cosas (IoT, *Internet of Things*) será, indudablemente, una de las ramas que pueda verse más beneficiada, gracias a las altas velocidades y capacidades de red y a las mínimas latencias que prometen podrán alcanzarse con las tecnologías 5G. Estos aspectos serán cruciales en procesos propios de las Industria 4.0, caracterizada por la interconexión, la transparencia de la información, las decisiones descentralizadas y la asistencia técnica [4], que requieren datos en tiempo real, con un grado de inmediatez muy elevado.

Se esperan conexiones con tasas de datos de hasta 10 Gbps (de diez a cien veces mayor velocidad que las generaciones anteriores), latencias de 1 ms, una banda ancha mil veces más rápida por unidad de área y la capacidad de tener hasta cien veces más dispositivos conectados por unidad de área (en comparación con LTE). También, una cobertura del 100 % y una

disponibilidad del 99.999 %, así como la reducción del 90 % en el consumo de energía de la red y el aumento de la duración de la batería a 10 años, en los dispositivos IoT de baja potencia [5].

Versión 15

La *Release* 15 [6], de marzo 2018, se centra en definir una primera fase de las comunicaciones 5G. Se especifican en ella las siguientes categorías: *enhanced Mobile Broadband* (eMBB), *Ultra-Reliable and Low Latency Communications* (URLLC) y *massive Machine Type Communications* (mMTC).

- **Enhanced Mobile Broadband (eMBB)**

Orientado a una mejora de la banda ancha móvil con altas tasas de transferencia de datos, con picos de hasta 10 Gbps. Desarrollado para escenarios de despliegue y cobertura, en diferentes áreas de servicio, donde se requieren anchos de banda elevados, con velocidades de datos sostenidas, independientemente de la localización y movilidad del usuario. El flujo de vídeo de alta definición, las realidades virtual y aumentada. el procesamiento de ingentes cantidades de datos en tiempo real, los eventos a gran escala o las comunicaciones en zonas con una alta densidad de población son algunos ejemplos de escenarios para comunicaciones eMBB.

- **Ultra-Reliable and Low Latency Communications (URLLC)**

Para comunicaciones críticas y con requerimientos de muy baja latencia, de hasta 1 ms, disponibilidad del servicio muy elevada y con tasas de pérdidas extremadamente bajas. Está destinada para un amplio abanico de escenarios, como la Industria 4.0, aplicaciones médicas, comunicaciones militares, la seguridad en el trasporte o comunicaciones *Vehicle to Everything* (V2X).

- **Massive Machine Type Communications (mMTC)**

Para entornos en los que gran cantidad de dispositivos generan volúmenes de datos reducidos y en las que se permiten mayores retardos.

Se presenta la nueva interfaz de radio, *New Radio* (NR), como la principal característica del 5G. Ofrece la flexibilidad necesaria para dar soporte a los tipos de comunicaciones descritos anteriormente.

Se definen las arquitecturas *Non-Stand Alone* (NSA) y *Stand Alone* (SA), siendo la primera de ellas aquella en la que la red de acceso (AN, *Access Network*) 5G utiliza una red central (CN, *Core Network*) 4G (EPC, *Evolved Packet Core*); y la segunda, en la que una AN 5G se conecta con una 5G CN (5GC). Será tan solo en esta última arquitectura donde estará soportado el conjunto completo de servicios 5G de esta primera fase, mientras que en una arquitectura NSA se soportarán solamente los servicios 4G, con las capacidades que ofrece 5G NR (bajas latencias, etc.).

Se distinguen dos rangos de frecuencia de operación: bandas por debajo de los 7.125 GHz (FR1, *Frequency Range 1*) y bandas, denominadas milimétricas (*mmWave*), entre los 24.25 GHz y los 52.6 GHz (FR2, *Frequency Range 2*). Esta última es una zona que, pese a tener bastante atenuación, de la posibilidad de ofrecer a los operadores un ancho de banda mucho mayor, lo que permite capacidades y velocidades de descarga más altas, incluso en situaciones donde hay una gran concentración de usuarios [7].

A diferencia de en LTE, en NR se utiliza multiplexación por división de frecuencias ortogonales (OFDM, *Orthogonal Frequency-Division Multiplexing*) con prefijo cíclico (CP, *Cyclic Prefix*)[1] en el enlace ascendente (UL, *uplink*) y descendente (DL, *downlink*). Se disminuye la banda de protección de la forma de onda, dejando de estar limitada al 90 % de ancho de banda. Se consigue, por tanto, una mayor eficiencia espectral.

Se utiliza una estructura de tramas flexible con diferentes espaciados de subportadoras (SCS, *Subcarrier Spacings*[2]), lo que se conoce como múltiples numerologías. Mientras que en LTE tan solo podía tomar el valor de 15 kHz (7.5 kHz para *Multimedia Broadcast Multicast Services* [MBMS]), en NR cabe la posibilidad de encontrar distancias de 15, 30, 60, 120 o 240 kHz. Esto puede ser muy ventajoso, puesto que un mayor espaciamiento de subportadoras provoca un aumento del ancho de banda y una disminución en los tiempos de transmisión de un símbolo OFDM. Se conseguiría, así, una reducción de las latencias en la capa física [8].

Como se observa en la Figura 2.1, las tramas, de 10 ms, están divididas en 10 subtramas de 1 ms con un número de ranuras variable, dependiendo del SCS, cada uno con 14 símbolos OFDM precedidos por un prefijo cíclico.

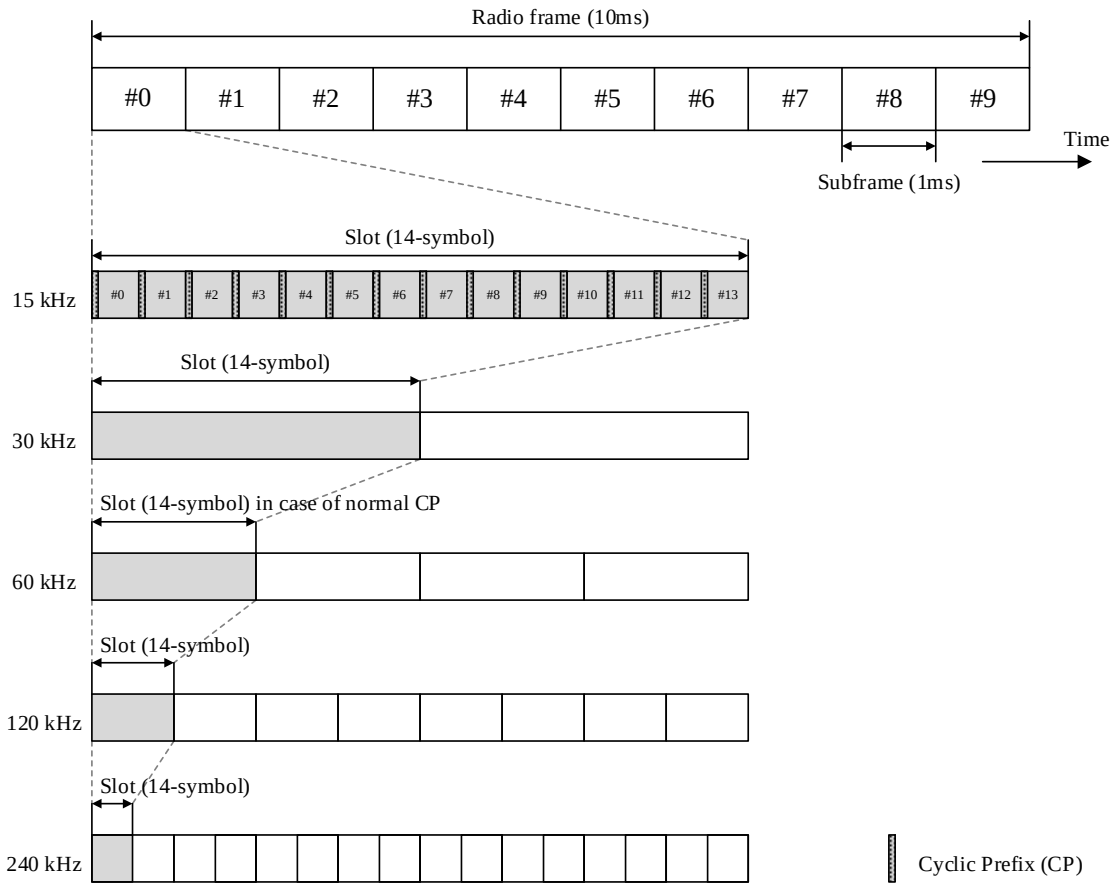


Figura 2.1: Trama OFDM. Elaboración propia. Basado en [6].

[1] *Cyclic Prefix*: tiempo de guarda que se agrega al inicio de cada símbolo OFDM para evitar la interferencia entre símbolos (ISI, *intersymbol interference*) y entre portadoras (ICI, *intercarrier interference*) producida por la llegada con retraso de señales reflejadas o *multipath*. Debe ser mayor que el *delay spread* para evitar que el *multipath* se convierta en interferencia.

[2] *Subcarrier Spacings*: distancia entre los centros de dos portadoras consecutivas.

Tabla 2.3: Múltiples numerologías en NR. Elaboración propia. Basado en [6].

Cyclic Prefix	*SCS (kHz)*	*Slot/Frame*	*Slots/Subframe*	*Symbols/Slot*	*Frequency*
Normal	15	10	1	14	FR1
Normal	30	20	2	14	FR1
Normal	60	40	4	14	FR1, FR2
Extended	60	40	4	12	FR1, FR2
Normal	120	80	8	14	FR2
Normal	240	160	16	14	FR2

Tal y como refleja la Tabla 2.3, a medida que el SCS aumenta, la longitud de las ranuras disminuye, en la misma proporción, con lo que se incrementa el número de aquellos por subtrama. Para celdas con una gran dispersión de retardo de propagación o *delay spread*[3], existe la posibilidad, para un SCS de 60 kHz, de utilizar un prefijo extendido, de cuatro veces mayor longitud, de modo que se ofrezca una mayor protección contra interferencias, a cambio de una pérdida de capacidad.

Normalmente, las transmisiones se llevan a cabo en una sola ranura, pero en ciertas ocasiones, tales como en servicios URLLC, cuando se requieren muy bajas latencias, pueden realizarse sobre, tan solo, una fracción de ranura, habiéndose establecido un mínimo de dos símbolos por transmisión.

Versión 16

Con la *Release* 16 [9], de septiembre 2019, se producen mejoras en la interfaz radio 5G (NR), con el incremento de la tasa binaria neta, gracias, entre otros aspectos, a la introducción de configuraciones de agregación de portadoras y de 256 QAM[4]. Además, se abre la posibilidad del acceso al espectro sin licencia entre los 5 y 6 GHz (NR-U, *New Radio-Unlicensed*) y se realiza un perfeccionamiento de la movilidad y de ahorro de energía del equipo del usuario.

Se incrementa la versatilidad y fiabilidad del *5G System* (5GS), de tal forma que se adecúe para una gran variedad de sectores ("verticales") (sanidad, transporte, industrias con fábricas automatizadas, etc.), con mejoras en las comunicaciones URLLC, como el soporte de transmisión redundante para comunicaciones de alta fiabilidad, el monitoreo de Calidad de Servicio (QoS, *Quality of Service*), la división del *Packet Delay Budget*[5] (PDB) y las mejoras del mecanismo de continuidad de la sesión.

También, se implementan la segmentación de la red en subredes virtuales, con la arquitectura *Network Slicing*, y la computación en el borde de la red, con el paradigma de computación *Edge Computing*. Se desarrolla el *cellular IoT* y se producen mejoras en el soporte para redes privadas (NPN, *Non-Public Network*, o SNPN, *Standalone Non-Public Network*), y en los servicios de posicionamiento, de alta precisión. Además, se desarrollan servicios con funcionalidades similares a las redes de área local (LAN, *Local Area Network*), pero con las capacidades propias de las comunicaciones 5G. Se mejora, también, el uso del 5G como una red

[3] *Delay spread*: diferencia de tiempo entre la llegada al receptor de la última señal reflejada y la primera.
[4] *QAM*: *Quadrature Amplitude Modulation*, modulación de amplitud de cuadratura.
[5] *Packet Delay Budget*: presupuesto de retraso de paquetes, define el límite superior del retraso sufrido por un paquete entre el equipo del usuario (UE, *User Equipment*) y la función de cumplimiento de políticas y cargos (PCEF, *Policy and Charging Enforcement Function*).

de comunicaciones subyacente, de tal forma que pueda ser utilizada de manera transparente por aplicaciones externas a la red, principalmente sobre *Northbound APIs*[6].

Por otro lado, se perfecciona la coexistencia del 5GS con sistemas que no son 3GPP, admitiendo estas redes como confiables. Para ello, se incluye un nuevo nodo de red: *Wireline Access Gateway Function* (W-AGF). Se realizan optimizaciones de red, como el uso de un *User Identifier* (UI) para la autenticación de la identidad de los usuarios; y se producen mejoras en el área del entretenimiento (retransmisiones en vivo, difusión de los medios de comunicación, etc.).

Con todo esto, se puede ver que en esta última *release* desarrollada por 3GPP destacan las mejoras del 5GS en términos de capacidad, cobertura, potencia, movilidad, fiabilidad, facilidad de implementación y latencia, entre otras.

Versión 17

Con la *Release* 17, con vistas para su normalización a mediados de 2022, se pretende mejorar el rendimiento del 5GS, así como admitir nuevos casos de uso y verticales, y proporcionar conectividad ubicua en diferentes condiciones y escenarios de implementación [10].

En la Figura 2.2, se muestran ciertas mejoras de las tecnologías 5G, descritas en las distintas versiones del estándar, haciéndose énfasis en aquellas definidas en la última *release*.

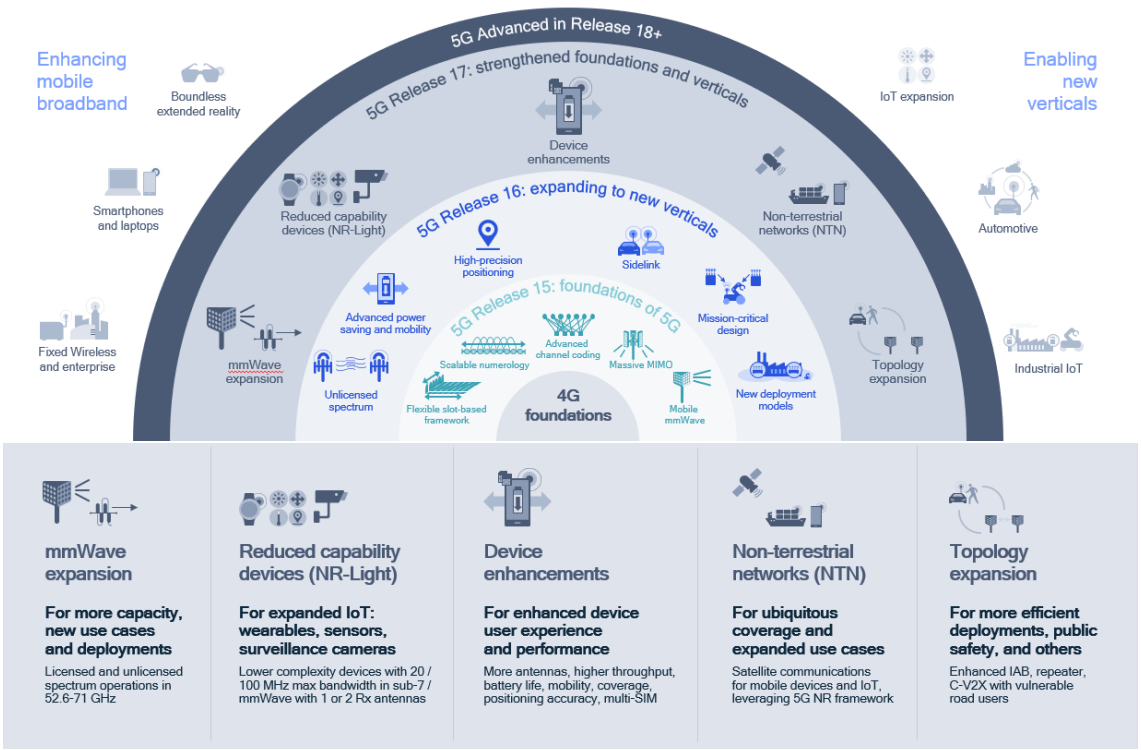


Figura 2.2: Mejoras definidas en la *Release* 17 de 3GPP [11].

[6] *Northbound API*: interfaz de programación de aplicaciones con capacidad de comunicarse con un componente de nivel superior.

2.2. Módulos para comunicaciones celulares

A continuación, se presenta un análisis del mercado [12] de los distintos módulos y dispositivos que se encuentran disponibles, a la fecha de la redacción de este documento, así como diferentes formas de acceder a la configuración de estos.

2.2.1. Mercado de módulos y/o dispositivos para redes celulares

Tras un 2020 en el que el mercado de módulos IoT celulares apenas creció, respecto al año anterior; en 2021, la demanda de módulos celulares aumentó notablemente, llegándose a registrar alrededor de 391 millones de unidades de envíos de módulos IoT celulares.

Se estima que en 2022 el mercado continúe al alza, siguiendo la misma dinámica que este pasado año. No obstante, la escasez en el suministro de semiconductores, que ha marcado estos dos últimos años y que se prevé que persista hasta 2023, probablemente esté planteando un exceso de existencias y ventas que podría verse reflejado en un futuro próximo, con una ralentización del crecimiento del mercado hacia 2023. El desarrollo del mercado entre los años 2019 y 2026 se puede observar en la Figura 2.3.

Existen numerosas aplicaciones del ámbito de la industria basada en IoT (IIoT, *Industrial IoT*) que no requieren una elevada tasa de datos, pero sí una amplia cobertura geográfica. Es por eso por lo que entre el 2020 y el 2021, alrededor de un 80 % de los envíos de módulos son 2G/3G o LTE/LPWA (*Low-Power Wide-Area*) de categoría baja.

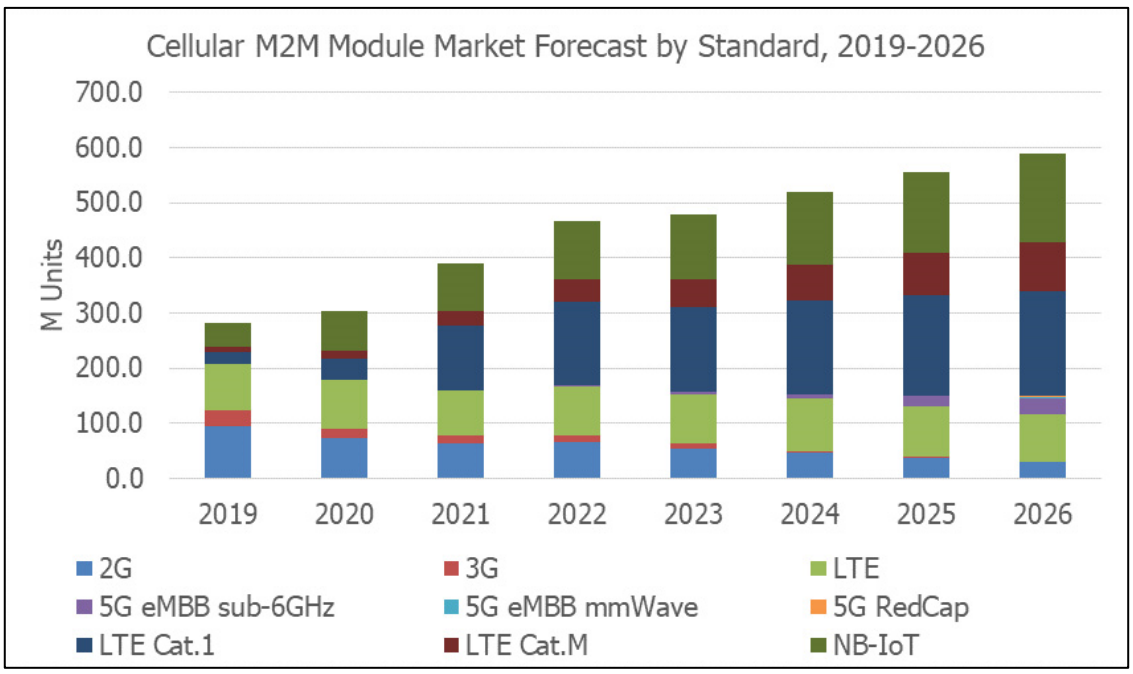


Figura 2.3: Evolución del mercado de módulos [12].

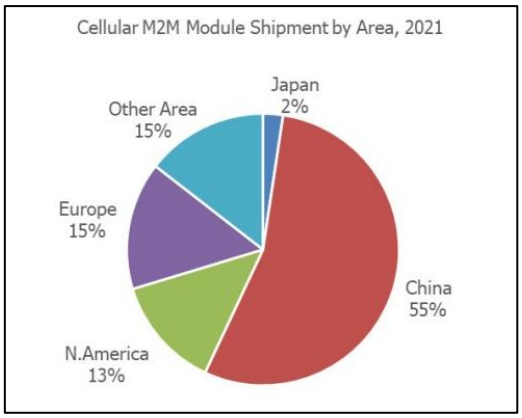


Figura 2.4: Mercado de módulos M2M por área [12].

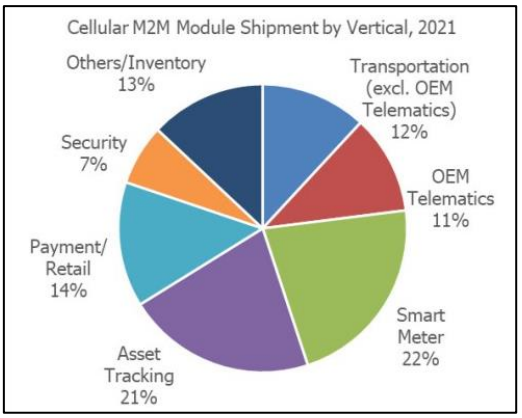


Figura 2.5: Mercado de módulos M2M por vertical [12].

Cabe destacar el considerable aumento del mercado de LTE Cat.1. Aunque moderado fuera de China, su crecimiento en el país del Asia Oriental ha sido tan elevado que ha llegado a superar a la banda estrecha de Internet de las cosas (NB-IoT, *Narrowband IoT*), logrando convertirse en el estándar celular de mayor alcance. Los operadores de la Unión Europea han mostrado interés en los módems LTE Cat.1 de bajo coste, de cara a la migración 2G/3G.

América del Norte y Japón han sido las zonas en que se ha registrado un mayor crecimiento del mercado de LTE Cat.M. En Europa y Asía Pacífico también ha aumentado la demanda, tanto de LTE Cat.M como de NB-IoT, para cubrir la red de múltiples operadores en un solo SKU[7].

Centrándose en áreas geográficas, se puede ver en la Figura 2.4 que China es, con un amplio margen, el mayor mercado de módulos celulares *Machine to Machine* (M2M), representando más del 50 % de la cuota de mercado. Europa y Norte América se quedan lejos, sin llegar a un 30 % entre los dos.

En cuanto a las aplicaciones, se estima que las verticales con mayor peso son los contadores inteligentes y el seguimiento de activos, seguidas por el pago en comercios, el transporte y los dispositivos telemáticos integrados en los vehículos de fabricantes de equipos originales (OEM), como se detalla en la Figura 2.5.

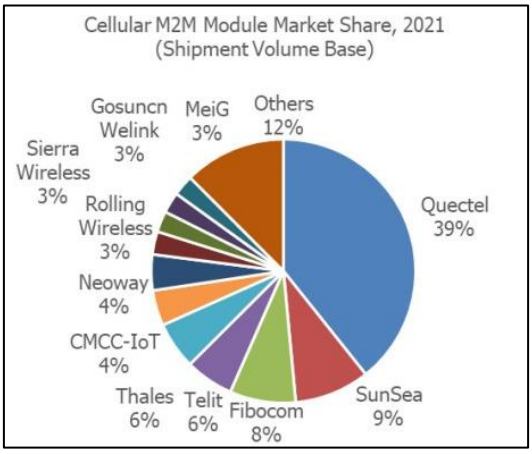


Figura 2.6: Cuota de mercado de módulos M2M [12].

[7] *SKU*: *Stock Keeping Unit*.

Tabla 2.4: Proveedores de módulos celulares. Elaboración propia. Basado en [12].

Estándar	*Proveedores de módulos celulares*
LTE	Quectel, SunSea, Fibocom, Rolling Wireless, MeiG, LG Innotek, Thales
LTE Cat.1 (Fuera de China)	Telit, Thales, Quectel, Sierra Wireless, ublox, WNC, Abit
LTE Cat.1 (Dentro de China)	Quectel, Fibocom, SunSea, Neoway, MeiG, Gosuncn Welink, MobileTek, Lierda
LTE Cat.M	Telit, Sierra Wireless, u-blox, Thales, WNC, Murata, Taiyo Yuden
NB-IoT	Quectel, CMCC IoT, SunSea, Gouncn Welink, Neoway, Lierda

Gracias a la fuerte demanda, la mayoría de los proveedores de módulos celulares ha incrementado sus ventas durante 2021. Tal y como refleja la Figura 2.6, Quectel representa casi el 40 % de la participación de mercado. Tras la compañía china se encuentran otras como SunSea, Fibocom, Telit, Thales o CMCC-IoT. Se espera que Fibocom, Rolling Wireless, MeiG y Telit, además de Quectel, aumenten considerablemente los envíos de módulos en los próximos años. En la Tabla 2.4 se muestran los principales proveedores de módulos, según el estándar de comunicación celular.

En la Figura 2.7 se pueden ver los principales proveedores de módulos celulares en el último trimestre de 2021, en diferentes regiones geográficas. Con Quectel liderando el marcado en buena parte del mundo, cabe destacar la fuerte presencia de Telit y Thales, este último en Europa de manera más notoria.

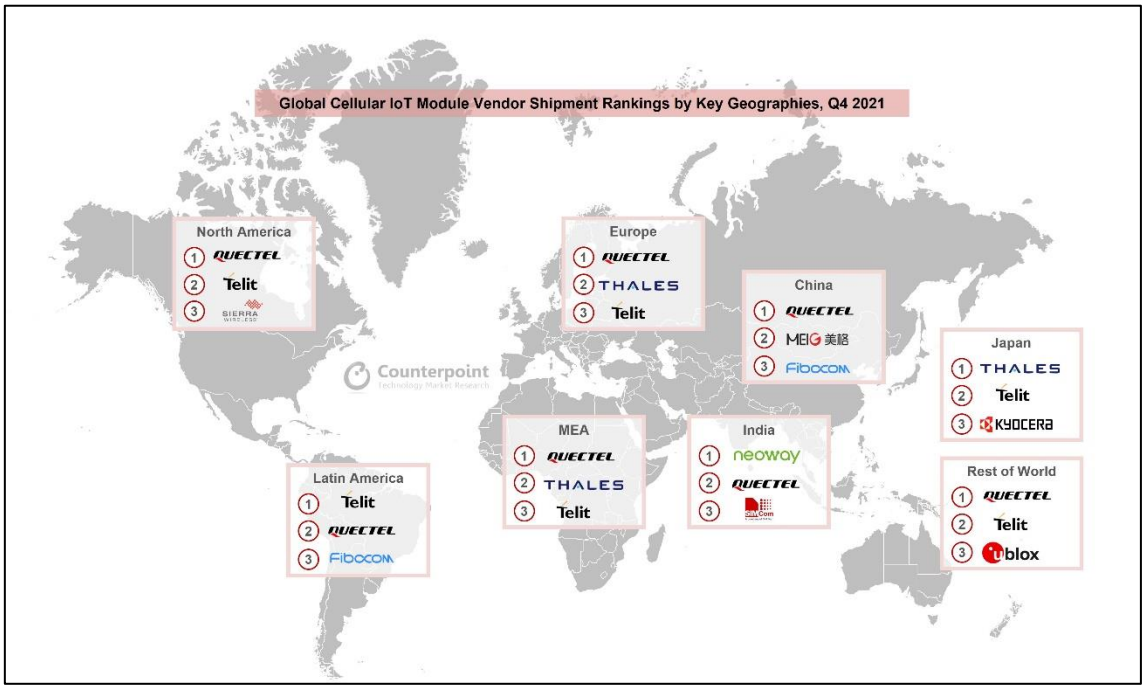


Figura 2.7: Principales proveedores de módulos celulares por región geográfica en el último trimestre de 2021 [12].

Tabla 2.5: Proveedores de chipsets. Elaboración propia. Basado en [12].

Estándar	*Proveedores de chipsets*
GSM	MediaTek, UNISOC, Intel
3G	Qualcomm, Intel
LTE	Qualcomm, Intel
5G	Qualcomm, HiSilicon, UNISOC
LTE Cat.1 (Fuera de China)	Qualcomm, Intel, Sony, Sequans
LTE Cat.1 (Dentro de China)	UNISOC, ASR (2022-: Eigencomm, Xinyi, M-Link)
LTE Cat.M	Qualcomm, Sony, Nordic, Sequans, u-blox
NB-IoT	HiSilicon, UNISOC, MediaTek, Eigencomm, Xinyi, M-Link

Los proveedores de circuitos integrados de módems celulares más relevantes difieren según el estándar celular, tal y como refleja la Tabla 2.5. Destacan, como se observa en la Figura 2.8, Qualcomm, UNISOC, ASR Microelectronics, HiSilicon, MediaTek e Intel, entre los principales proveedores de chipsets para módulos IoT celular. Estos dos últimos proporcionan fundamentalmente módems 2G/3G, por lo que su cuota de mercado se va viendo reducida con el paso de los años.

HiSilicon se ha impuesto en el mercado de NB-IoT, en detrimento de MediaTek, aunque con la sanción que el Gobierno de los Estados Unidos impuso a Huawei en 2019, no podrá producir nuevos chips basados en nodos de procesos avanzados. Compañías, como UNISOC y ASR Microelectronics, han ganado mucha participación de mercado, gracias a la expansión de LTE Cat.1 en China.

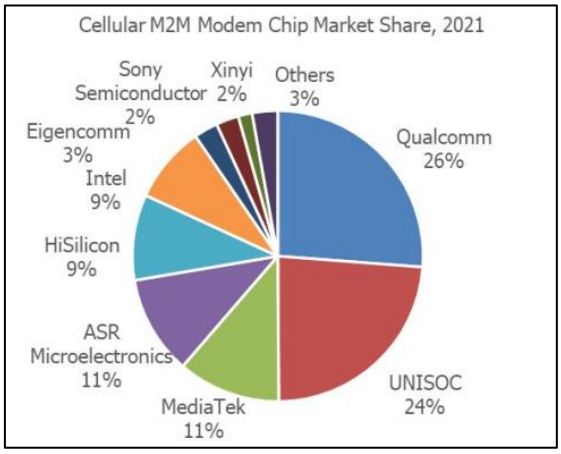


Figura 2.8: Cuota de mercado de chips de módulos M2M [12].

2.2.2. Configuración de módulos comerciales

Seguidamente, se describen las posibles opciones de configuración que contemplan la mayoría de los módulos que se encuentran en el mercado.

A través de comandos AT

El conjunto de comandos Hayes [13], comúnmente conocidos como comandos AT, es un lenguaje de comandos desarrollado por Dennis C. Hayes en 1981, que sentó las bases para la comunicación con módulos celulares o de radiofrecuencia.

Se trata de cadenas de caracteres ASCII que deben ir precedidos en cada línea de comando por el prefijo "AT" o "at" (*Attention*). La sintaxis de los comandos AT se refleja en la Tabla 2.6.

Utilizados principalmente para configurar y depurar los módems, a través del puerto serie del ordenador al que están conectados, así como para habilitar y deshabilitar la conexión de la red, se desarrollaron de manera que la compatibilidad entre diferentes módulos estuviera garantizada, facilitando en gran medida el control de estos.

Proporcionan acceso a información acerca del dispositivo y de la tarjeta SIM[8], así como a servicios de mensajería (SMS, MMS), fax y de voz y datos. Permiten enviar y recibir múltiples parámetros de la red y del dispositivo, como obtener datos del modelo y fabricante del módulo o el estado actual de la red, así como, por ejemplo, cambiar la tecnología de red que se está utilizando.

Los módems de banda ancha móvil han sido controlados durante mucho tiempo a través de extensiones del protocolo AT específicas para celulares, definidas normalmente por las organizaciones encargadas de implementar estándares de tecnología de banda ancha móvil (ETSI TS 127 007 [14]) o por los propios proveedores de equipos [15]. Estos módems, al haber heredado la mayor parte de la lógica de los módems de conexiones por línea conmutada, deben establecer una sesión PPP[9] para que actúe como interfaz entre el ordenador y el módem.

A medida que los requisitos del rendimiento de las conexiones fueron en aumento y surgieron nuevas tecnologías para la interfaz entre el dispositivo y el ordenador, los fabricantes de módems comenzaron a desarrollar e implementar sus propios protocolos para controlar los dispositivos de banda ancha móvil.

Por consiguiente, el uso de comandos AT para comunicaciones con módems, a pesar de que siga estando disponible en la mayoría de los dispositivos, se ha restringido mayoritariamente para aquellos sistemas que no hacen uso de los nuevos protocolos.

Tabla 2.6: Sintaxis de los comandos AT. Elaboración propia. Basado en [13].

		Comando
Sintaxis Básica		`AT<COMMAND><SUFFIX><DATA>`
Sintaxis Extendida	*Test*	`AT+<COMMAND NAME>=?`
	Read	`AT+<COMMAND NAME>?`
	Write	`AT+<COMMAND NAME>=<…>`
	Execute	`AT+<COMMAND NAME>`

[8] *SIM*: *Subscriber Identity Module*, módulo de identificación de abonado.
[9] *PPP*: *Point-to-Point Protocol*, protocolo punto a punto.

A través de drivers para Linux

Uno de estos nuevos protocolos implementados para controlar los módems de banda ancha móvil es el protocolo *Qualcomm MSM Interface* (QMI), desarrollado por Qualcomm. Define una clasificación en diferentes grupos o servicios, cada uno con sus distintas indicaciones, peticiones y respuestas, para interactuar con el módem. Se distinguen los siguientes servicios:

- Device Management Service (DMS)
- Network Access Service (NAS)
- Wireless Data Service (WDS)
- Wireless Messaging Service (WMS)
- Position Determination Service (PDS)
- Phonebook Management Service (PBM)
- User Identity Module Service (UIM)
- Open Mobile Alliance Service (OMA)
- Wireless Data Administrative Service (WDA)

QMI está disponible en los sistemas operativos basados en el *kernel* de Linux a través de dos controladores de dispositivos o *drivers*[10]: *GobiNet* y *qmi_wwan*.

Se realiza, a continuación, una comparativa de ambos controladores, puesto que aportan un enfoque del protocolo QMI totalmente diferente, como se refleja en la Tabla 2.7.

- **qmi_wwan**

Qmi_wwan se mantiene dentro del *kernel* de Linux ascendente (*in-tree*), por lo que obtiene todas las actualizaciones que descargue el *kernel*, de manera automática. Además, es válido para múltiples dispositivos, siempre y cuando sean compatibles con QMI. Deja al espacio de usuario la gestión completa del protocolo, incluyendo las asignaciones y liberaciones de clientes, así como el servicio de control interno (CTL), lo que dota al *driver* de una muy alta simplicidad.

- **GobiNet**

Por el contrario, *GobiNet* es *out-of-tree* y, al tener un repositorio de código fuente privado, las versiones del controlador no están alineadas con las actualizaciones del *kernel*. Es de mayor complejidad: implementa la mayor parte de la lógica del protocolo a nivel de *kernel*, incluyendo el CTL. Tan solo podrán hacer uso de este *driver* algunos dispositivos desarrollados por Qualcomm.

Tabla 2.7: Comparativa *GobiNet* y *qmi_wwan*. Elaboración propia. Basado en [15].

Característica	*GobiNet*	*qmi_wwan*
Linux kernel	Out-of-tree	In-tree
Complejidad	Muy complejo	Simple
Gestión de CTL	Kernel space	User space
Dispositivos	Solo para algunos dispositivos propietarios de Qualcomm	Para dispositivos compatibles con QMI

[10] *Driver*: componente software que permite al sistema operativo interactuar con dispositivos conectados al PC

2.3. Protocolos para comunicaciones IoT

A continuación, se realiza un breve estudio del estado del arte de distintos protocolos de interés en comunicaciones relacionadas con el Internet de las cosas.

2.3.1. Estado del arte

Se comentan brevemente los aspectos más relevantes de tres de los protocolos IoT más habituales: MQTT [16], AMQP [17] y CoAP [18].

- **MQTT**

Message Queue Telemetry Transport [16] (MQTT) es un protocolo de comunicaciones muy ligero, de gran conveniencia para aplicaciones IoT donde se utilicen sensores de baja potencia y ancho de banda mínimo, con una pequeña huella de código, gracias a ciertas ventajas que ofrece, como su fácil integración y buen rendimiento.

Es un protocolo de red IoT y M2M de tipo *Message Queue*. El *Router* genera una cola de mensajes única para cada cliente y discrimina los mensajes con el identificador de cliente (*ClientId*). Los mensajes recibidos, por tanto, se mantienen en el *Router* hasta que son entregados al cliente, incluso cuando este esté desconectado.

Como refleja la Figura 2.9, está basado en el patrón *Publisher/Subscriber* (*Pub/Sub*). Los clientes publican mensajes (*PUBLISH*) a cerca de un tema o *topic*, al que deberán suscribirse (SUBSCRIBE) aquellos que deseen recibirlos. El *broker* será el servidor encargado de aceptar los mensajes publicados y de entregarlos a los clientes suscritos.

Se describen los siguientes tres niveles de QoS para clientes MQTT:

- QoS 0: *at most once delivery*. El mensaje se envía una sola vez, sin respuesta por parte del receptor. El mensaje llega una vez a destino, o no llega.
- QoS 1: *at least once delivery*. El mensaje se envía hasta que el receptor da constancia de que lo ha recibido correctamente, con un PUBACK. El mensaje llega al menos una vez a destino.
- QoS 2: *exactly once delivery*. El mensaje se envía y el receptor da constancia de que lo ha recibido correctamente, evitando la duplicación y asegurando la recepción del mensaje, con un handshake de cuatro pasos. El mensaje llega una vez a destino.

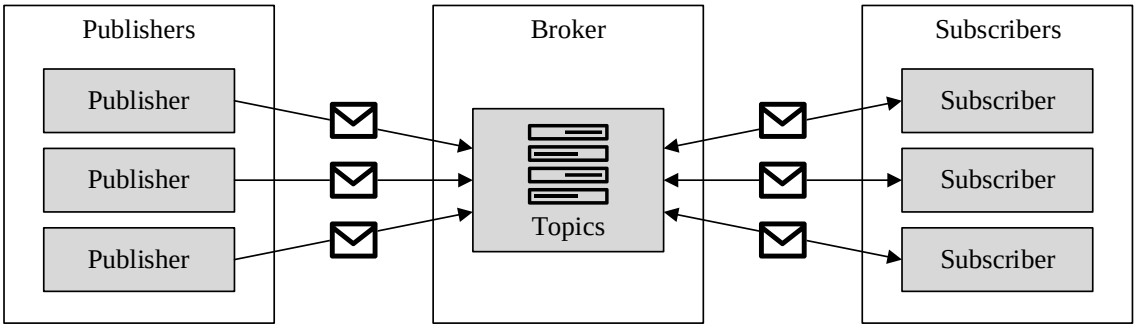


Figura 2.9: Modelo de comunicación de MQTT. Elaboración propia.

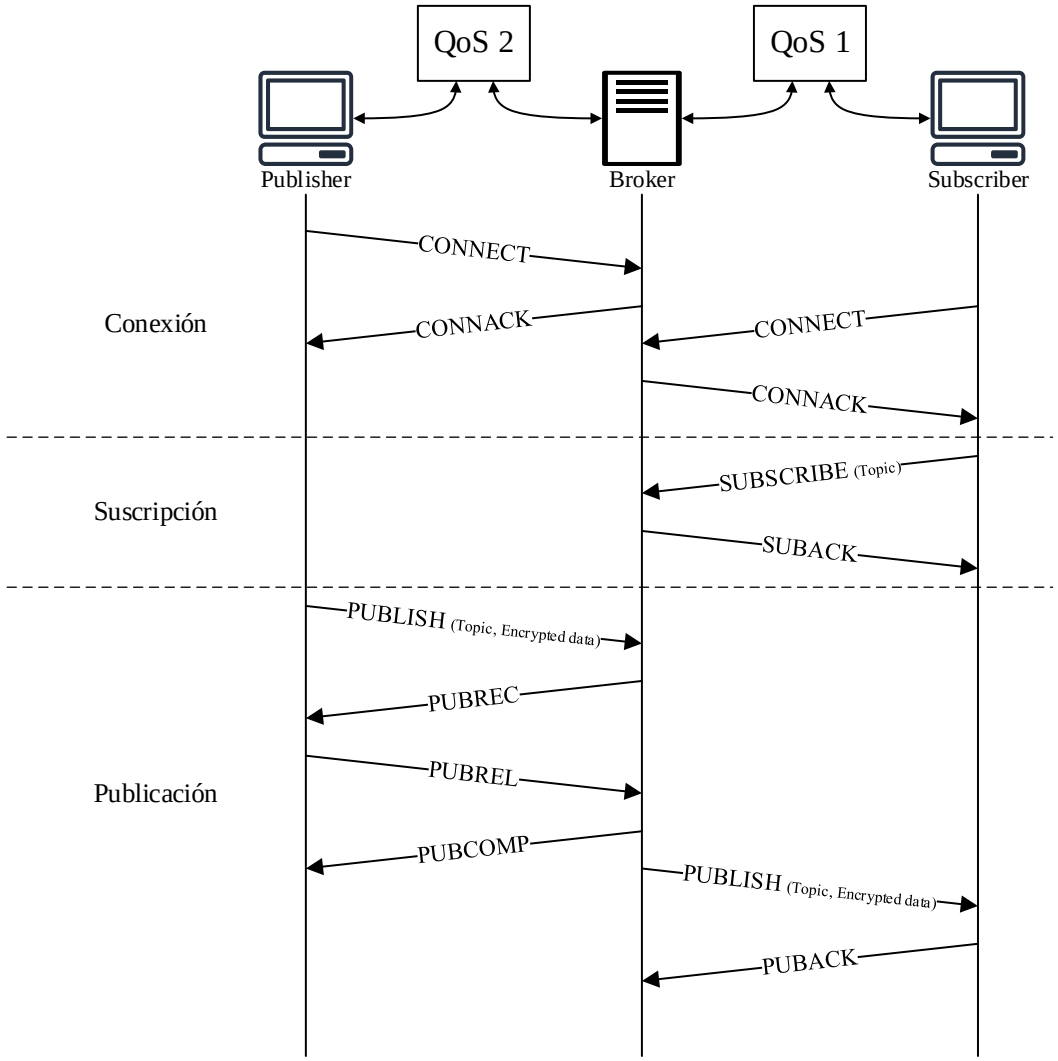


Figura 2.10: Sesión MQTT con QoS 1 y QoS 2. Elaboración propia.

En la Figura 2.10, el intercambio de tramas de dos conexiones Publisher-Broker y Broker-Subscriber, con QoS 2 y QoS 1, respectivamente.

Como se observa en la Figura 2.11, normalmente se utiliza *Transmission Control Protocol* [19] (TCP) como capa de transporte, junto a *Internet Protocol* [20] (IP) como capa de red, permitiéndose la posibilidad de proporcionar seguridad en la capa de transporte con *Transport Layer Security* [21] (TLS). Se utiliza por defecto el puerto 1883, salvo en el caso de MQTT TLS, donde está reservado el 8883.

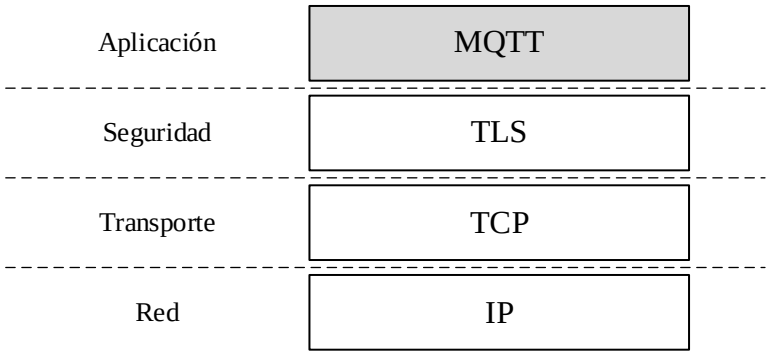


Figura 2.11: Arquitectura de MQTT. Elaboración propia.

- **AMQP**

El protocolo de capa de aplicación *Advanced Message Queue Protocol* [17] (AMQP) surge de la necesidad por obtener un protocolo en que la entrega de mensajes esté, en todo caso, garantizada. Destaca, por tanto, por su fiabilidad, pero también por su alta seguridad e interoperabilidad. Estas cualidades hacen de AMQP un protocolo, a pesar de haber sido diseñado originalmente para aplicaciones empresariales, muy adecuado para entornos IoT que requieran una capacidad de procesado elevada.

Se dispone de un sistema de colas que permite mantener una conversación asíncrona entre transmisor y receptor, de modo que los mensajes que se envíen puedan ser recuperados según la disponibilidad del destinatario.

La arquitectura de AMQP, como refleja la Figura 2.12, está compuesta por un productor (*producer*) que publica en un *broker* el mensaje que recibirá un consumidor (*consumer*). Dentro del *broker*, se distinguen varias entidades: el encargado de aceptar los mensajes provenientes del productor es el intercambiador (*exchange*), que los enruta a las diferentes colas (*queues*); el *binding* es el que relaciona al *exchange* con las *queues*, puesto que es responsable de determinar las reglas que se deben seguir a la hora de enrutar los mensajes.

Se distinguen cuatro tipos de *exchange* distintos, como se muestra en la Tabla 2.8, diferenciados por el algoritmo utilizado para determinar la cola a la que enviar los mensajes.

Generalmente se basa en el modelo TCP/IP, usándose el puerto 5672 o, cuando se aplique seguridad con TLS, el 5671.

Tabla 2.8: Tipos de *exchanges* en AMQP. Elaboración propia. Basado en [22].

Tipo	*Descripción*
Direct exchange	Envío de mensaje con clave de enrutamiento (*routing key*) a cola con misma clave de vinculación (*binding key*).
Fanout exchange	Envío de mensaje a todas las colas.
Topic exchange	Envío de mensajes a colas con mismo patrón de claves
Headers exchange	Envío de mensajes a colas según cabecera del mensaje

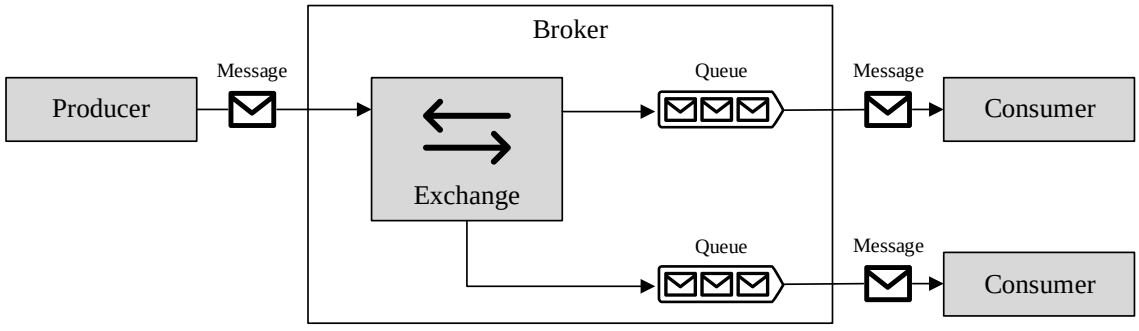


Figura 2.12: Modelo de comunicación de AMQP. Elaboración propia. Basado en [23].

- **CoAP**

Otro protocolo habitual en el ámbito del Internet de las cosas es el *Constrained Application Protocol* [18] (CoAP). Está basado en la arquitectura de desarrollo web REST[11], diseñado para realizar un mapeo directo con HTTP[12] [24] de la manera más sencilla posible. La Figura 2.13 muestra un ejemplo de una comunicación CoAP.

Es muy ligero y funciona particularmente bien en entornos restringidos, donde la red y los dispositivos están altamente congestionados, primando la velocidad por encima de la confiabilidad, al utilizar el protocolo de datagramas de usuario (UDP, *User Data Protocol* [25]) como capa de transporte. El puerto configurado por defecto es el 5683.

Para proteger la transmisión de información confidencial, CoAP utiliza la seguridad de la capa de transporte de datagramas (DTLS, *Datagram Transport Layer Security* [26]) como protocolo de seguridad para la comunicación y autenticación de los dispositivos, en el puerto 5684.

Como muestra la Figura 2.14, CoAP está dividido en dos subcapas: mensajes y peticiones/respuestas. La primera de ellas interactúa con UDP y soporta transmisiones de mensajes asíncronos, fiables y no fiables. Por otra parte, la segunda subcapa es la encargada de la gestión de las peticiones y respuestas.

Las diferentes peticiones que se pueden realizar (GET, POST, PUT, DELETE) pueden ser transportadas en un mensaje confirmado (CON) o, en su defecto, en un mensaje no confirmado (NON).

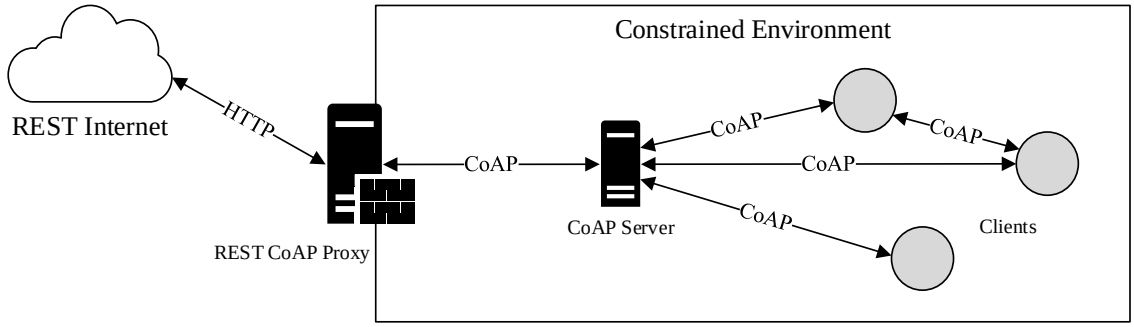


Figura 2.13: Modelo de comunicación de CoAP. Elaboración propia. Basado en [27].

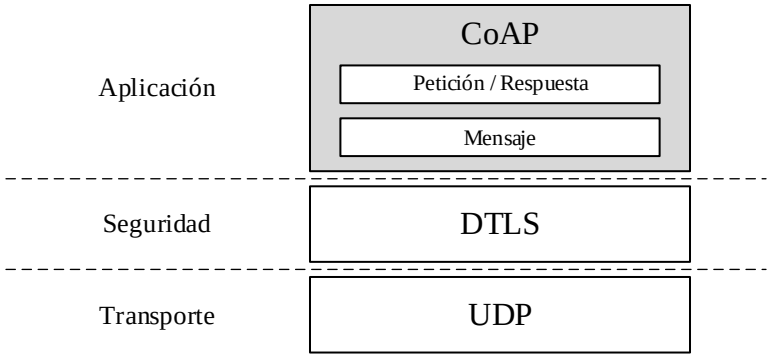


Figura 2.14: Arquitectura de CoAP. Elaboración propia. Basado en [28].

[11] *REST*: *Representational State Transfer*, transferencia de estado representacional. Estilo de arquitectura software que permite interconectar varios sistemas basados en HTTP.
[12] *HTTP*: *Hypertext Transfer Protocol*, protocolo de transferencia de hipertexto.

En el primer caso, si la información requerida está disponible inmediatamente, la respuesta puede ser transportada, bien en el mensaje de *acknowledgementk* (ACK) (*piggybacking*), o bien en otro mensaje confirmado. Si en la tramitación del mensaje CON ha ocurrido algún error, se enviará un mensaje de *reset* (RST). Si se trata, por el contrario, de una solicitud enviada en un mensaje no confirmado, la respuesta es enviada en otro mensaje NON.

Como se muestra en la Figura 2.15, se utiliza un *Token* para asociar las peticiones con sus respectivas respuestas y, así, permitir la asincronía en las transmisiones.

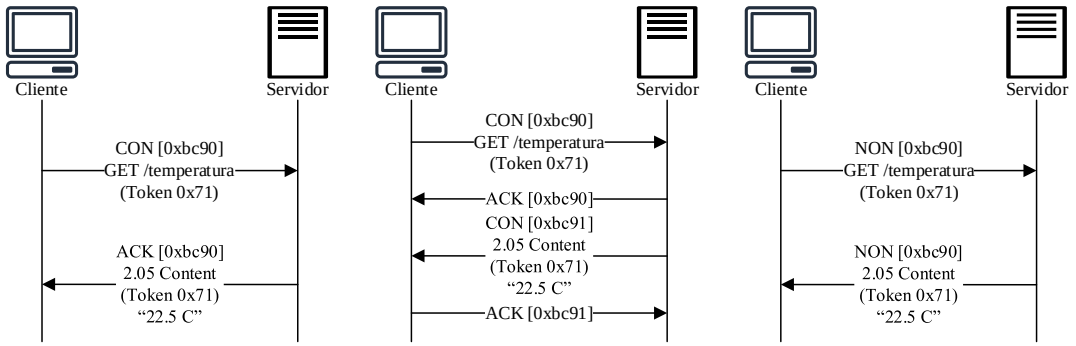


Figura 2.15: Transmisiones en CoAP. Elaboración propia.

2.3.2. Tabla resumen comparativa

Seguidamente, se muestra en la Tabla 2.9, una comparación entre los protocolos de comunicaciones IoT tratados en el anterior punto.

Tabla 2.9: Comparativa de protocolos IoT. Elaboración propia. Basado en [27][29].

Características	*MQTT*	*AMQP*	*CoAP*
Protocolo base	TCP	TCP	UDP
Seguridad	TLS/SSL	TLS/SSL	DTLS
Modelo	Publisher-Subscriber	Producer-Consumer	Request-Response
RESTful	No	No	Sí
Consumo energía	Medio	Medio	Bajo
Confiabilidad	Media-alta	Alta	Baja
Complejidad	Baja	Alta	Media
Cabeceras	2B	8B	4B
Mensajes	Asíncronos	Asíncronos	Síncronos y asíncronos

2.3.3. Mecanismos de transporte novedosos en protocolos IoT

Como se ha indicado anteriormente, el uso de TCP como protocolo de transporte suele ser habitual en las comunicaciones IoT. No obstante, cuenta con una serie de desventajas que hacen que quizás pueda no ser considerada como la mejor opción, en vistas a un futuro próximo. No estar garantizados los mínimos retardos requeridos para algunas aplicaciones es una de ellas. Es por eso por lo que se han propuesto varias modificaciones de TCP y han surgido otras alternativas que cumplen de mejor manera con estos requerimientos en la capa de transporte, como es el caso del protocolo QUIC.

QUIC[30], previamente acrónimo de *Quick UDP Internet Connections*, es un protocolo de transporte experimental, desarrollado por *Google Inc.* y estandarizado por la IETF[13], con el objetivo de reducir las latencias y los tiempos de establecimiento de la conexión, así como de mejorar la seguridad en las comunicaciones, con el cifrado de extremo a extremo de cabecera y *payload* [31].

QUIC mejora la latencia del establecimiento de la conexión (*handshake*), con una reducción notable de los tiempos de ida y vuelta (RTT, *Round-Trip Time*), con respecto a TCP, lográndose conexiones seguras con TLS 1.3[32] en 1 RTT, o 0 RTT si los puntos finales ya habían realizado un establecimiento de la conexión con anterioridad, como se muestra en la Figura 2.16. Se utilizan datagramas UDP para facilitar la implementación en sistemas y redes existentes [31].

Otra de las grandes aportaciones de QUIC es la multiplexación de *streams* en la capa de transporte, técnica utilizada para evitar el *head-of-line blocking*[14] (HOL *blocking*). Sobre una única conexión, se definen flujos de datos o *streams* multiplexados, identificados por un *stream ID*, de tal forma que no sea necesario abrir varias conexiones desde el mismo cliente.

Otra causa por la que QUIC logra reducir las latencias son los algoritmos de detección de pérdidas que implementa, que incluyen "retransmisiones tempranas" y sondas de pérdida de cola [33]. Así como algoritmos de control de la congestión. Son descritos en [34].

En la Figura 2.17 se muestra el modelo de arquitectura de QUIC.

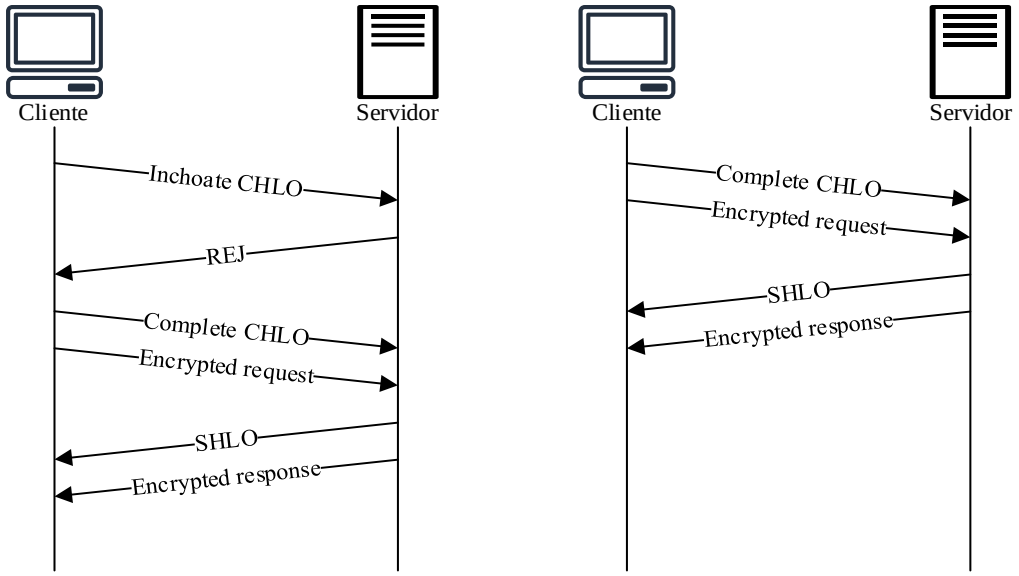


Figura 2.16: Establecimiento de la conexión con QUIC con 1-RTT y 0-RTT. Elaboración propia. Basado en [35].

[13] *IETF*: *Internet Engineering Task Force*, grupo de trabajo de ingeniería de Internet.
[14] *Head-of-line blocking*: limitación del rendimiento de un sistema de colas por la detención de una línea de paquetes debido a la pérdida de un primer paquete.

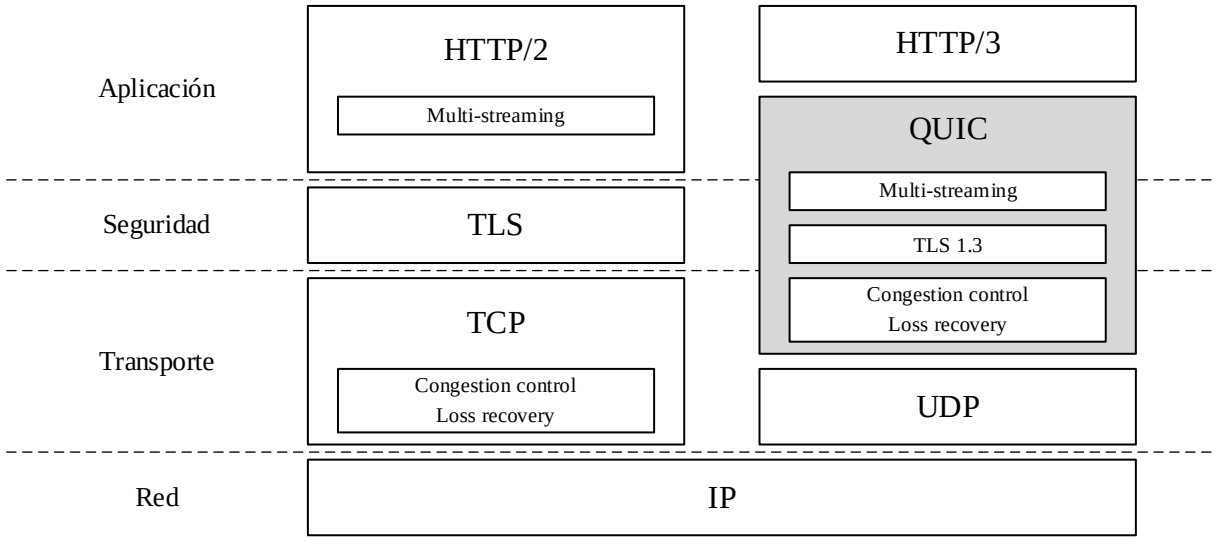


Figura 2.17: Arquitectura de QUIC. Elaboración propia. Basado en [31].

Capítulo 3

Diseño e implementación

En este tercer capítulo, se realiza una exposición del equipo empleado para el desarrollo de los módulos que conforman el núcleo de este trabajo. Asimismo, se incluyen una detallada explicación de estos y los pasos a seguir para su correcta puesta en funcionamiento.

3.1. Arquitectura del sistema

En este apartado se presenta la arquitectura del sistema a implementar, que será tratado, en adelante, con mayor profundidad.

Para cumplir con los objetivos propuestos, se requiere un dispositivo que capte los movimientos de las manos del usuario con la máxima precisión y con el mínimo retardo, de forma que se puedan obtener las posiciones exactas al instante. Estos datos deberán ser enviados a un servidor, a través de un módulo que proporcione conectividad inalámbrica y, así, poder controlar remotamente un brazo robótico que esté enlazado a dicho servidor.

De esta manera, tal y como refleja la Figura 3.1, el sistema se creará en base a los siguientes elementos, fundamentales para la comunicación:

- Un dispositivo sensor, con el que detectar las manos del usuario.
- Un módulo de comunicaciones celulares, para proveer de conectividad inalámbrica al sistema.
- Un dispositivo actuador, capaz de traducir los datos en movimiento.

Se ha optado por la utilización MQTT [16] como protocolo de comunicaciones IoT para todas las conexiones establecidas entre los diferentes módulos del sistema. La elección de este protocolo de aplicación viene dada por la baja latencia asociada, así como por su simplicidad, eficiencia y facilidad de implementación, con las que se consigue mejorar el desempeño del sistema. Para evitar mayores retardos, no se hace uso del protocolo de seguridad en la capa de transporte, TLS.

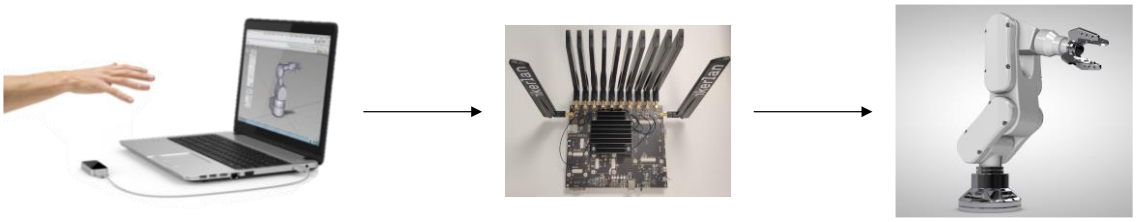


Figura 3.1: Arquitectura del sistema. Elaboración propia.

A continuación, se detallan los recursos de que se han podido disponer a la hora de la realización de este Trabajo, para llevar a cabo el sistema propuesto.

3.1.1. Sensor - Leap Motion Controller

Se trata de un módulo desarrollado por *Ultraleap* [36], que cuenta con dos cámaras y varios leds, que permiten captar, con un muy alto grado de detalle, el movimiento de las manos y los dedos de los usuarios, posibilitando la interacción de estos con contenidos físicos y digitales. El software del controlador puede discernir hasta veintisiete elementos distintos de la mano y rastrearlos incluso estando ocultos por otras partes de la mano.

Los leds iluminan las manos con pulsos de luz infrarroja no ionizante, imperceptibles para el ojo humano. Son captados por las dos cámaras de infrarrojo cercano de que consta el módulo, de 640x240 píxeles, que operan en el rango espectral de 850 nm ± 25, a una frecuencia de 120 Hz, y que tienen la capacidad de capturar imágenes a 2000 fotogramas por segundo.

Cuenta con un campo de visión de 150º y un rango efectivo que se extiende desde 25 a 600 milímetros por encima del dispositivo. Su funcionamiento se optimizará en tanto en cuanto tenga una visión más clara y un mayor contraste del objeto a detectar. Puede obtenerse más información acerca del sensor en [37].

El dispositivo, a medida que rastrea, proporciona actualizaciones en forma de un conjunto de datos o *frames*, que contienen las posiciones medidas y más información acerca de las manos detectadas en un determinado instante de tiempo. Estos *frames* son enviados vía USB[15] al PC[16], de manera que sean gestionados como se considere oportuno.

3.1.2. Módem de comunicaciones inalámbricas

Seguidamente, se describen las características y parámetros principales del módulo que se utiliza para dotar de conectividad inalámbrica al sistema. Las pruebas se realizan sobre el kit de desarrollo 5G EVB Kit [38], una placa de evaluación (EVB, *Evaluation Board*) de *Quectel*, que se muestra en la Figura 3.2.

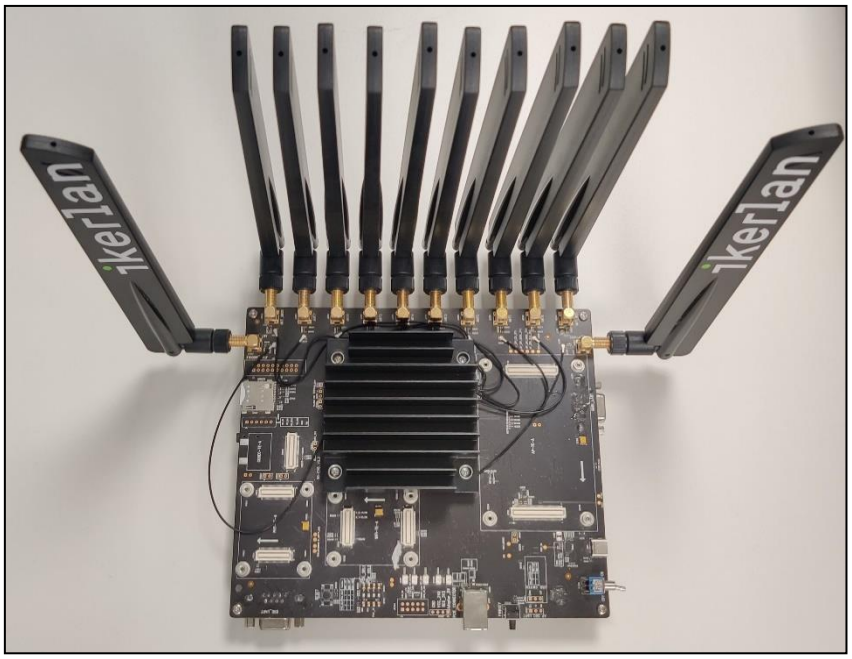


Figura 3.2: Quectel 5G EVB Kit utilizado. Elaboración propia.

Descripción y especificaciones hardware del módulo

Se dispone de un módulo de *Quectel*, modelo RG500Q-EA, capaz de trabajar en las bandas de 3G (UMTS), 4G (LTE) y 5G (NSA y SA). Las bandas de frecuencia en las que el módulo puede operar y las velocidades de transferencia máximas se muestran en las Tablas 3.1 y 3.2, respectivamente. Puede encontrarse más información al respecto en [39].

[15] *USB*: *Universal Serial Bus*.
[16] *PC*: *Personal Computer*.

Tabla 3.1: Bandas de frecuencia de operación del módulo *Quectel RG500Q-EA*. Elaboración propia. Basado en [39].

Modo	*Bandas de frecuencia*
5G NR	n1/n3/n5/n7/n8/n20/n28/n38*/n40*/n41/n77/n78/n79
LTE-FDD[17]	B1/B3/B5/B7/B8/B18/B19/B20/B26/B28/B32
LTE-TDD[18]	B34/B38/B39/B40/B41/B42/B43*
WCDMA	B1/B3/B5/B6/B8/B19

Tabla 3.2: Tasas de transferencia del módulo *Quectel RG500Q-EA*. Elaboración propia. Basado en [39].

Modo	*Transmisión (DL)*	*Transmisión (UL)*
5G SA Sub-6	2.1 Gbps	900 Mbps
5G NSA Sub-6	2.5 Gbps	650 Mbps
LTE	1.0 Gbps	200 Mbps
UMTS	42 Mbps	5.76 Mbps

3.1.3. Infraestructura de red - Edge Ikerlan

El despliegue del sistema se realiza dentro de la red de Ikerlan. En el momento del desarrollo del proyecto, se estaba integrando una solución privada de conectividad móvil, que permitirá disponer de una MPN[19] donde poder experimentar con aplicaciones en arquitecturas 5G MEC[20].

Mientras tanto, para poder acceder desde el dispositivo 5G a un servidor remoto, simulando una arquitectura Edge en 5G, se ha habilitado un servidor, accesible desde Internet y, al mismo tiempo, desde la red de Ikerlan, que contiene un *broker* MQTT y que es el responsable de recoger los datos, enviados por el módem, del posicionamiento de las manos.

Dicho servidor dispone, también, de un servicio de *Grafana*, conectado localmente a este *broker*, para la recogida de la información proveniente del módem y de los resultados de la latencia de la conexión, como se detallará en adelante.

3.1.4. Emulación del robot

En lugar de utilizar un brazo robótico, cuya puesta en marcha escaparía del marco de este Trabajo, se ha preferido aprovechar una interfaz gráfica que proporciona *Ultraleap*, con la que se consigue emular el funcionamiento del robot, mostrando, con gran precisión, un modelo esquemático de las manos del usuario capturadas por el sensor.

De esta manera, se puede demostrar de una manera práctica, el comportamiento de las distintas tecnologías celulares utilizadas en la transmisión de los datos.

El módulo utilizado, *Hand-Visualizer*, es una modificación realizada en Ikerlan del facilitado por *Ultraleap*, *css-visualizer*, para que se adapte de mejor manera a las conveniencias del sistema.

[17] *FDD*: *Frequency Division Duplex*, duplexación por división de frecuencia.
[18] *TDD*: *Time Division Duplex*, duplexación por división de tiempo.
[19] *MPN*: *Mobile Private Network*.
[20] *MEC*: *Multi-access Edge Computing*, arquitectura de red que proporciona las funciones de la computación en la nube en el borde la red, haciendo uso de tecnologías móviles. Aporta movilidad y una mayor robustez, reduciendo, además, las latencias y la carga del servidor de respaldo de la red.

3.2. Descripción funcional del sistema

A continuación, se presentan las funcionalidades principales del sistema propuesto.

Partiendo de un servicio o *daemon* que ponga en marcha el sensor y mande datos a un *broker* local, debe establecerse una conexión MQTT con el *Edge*, de manera que puedan visualizarse en remoto los gestos de las manos captadas por *Leap Motion Controller*. Esta conexión entre el nodo local y el nodo frontera se realiza a través de un módem de comunicaciones celulares.

Además de la emulación de los movimientos de la mano robótica, se lleva a cabo una monitorización del *ping*, de modo que se pueda llevar un control de las latencias generadas en cada una de las conexiones del módem (UMTS, LTE, 5G). Para poder simular un entorno hostil, con pérdidas y errores de conexión, o establecer otros parámetros de control de tráfico (gestión de anchos de banda, eliminar ciertos tráficos, etc.) en la conexión, se ofrece la posibilidad de realizar una configuración remota de la red. También se permite determinar la tecnología celular deseada, así como monitorizar ciertos datos proporcionados por el módem, de tal forma que se pueda acceder en remoto al estado de la conexión y del propio módem.

En la Figura 3.3, se presenta un esquema donde se distinguen los distintos módulos del sistema, que se detallan a continuación.

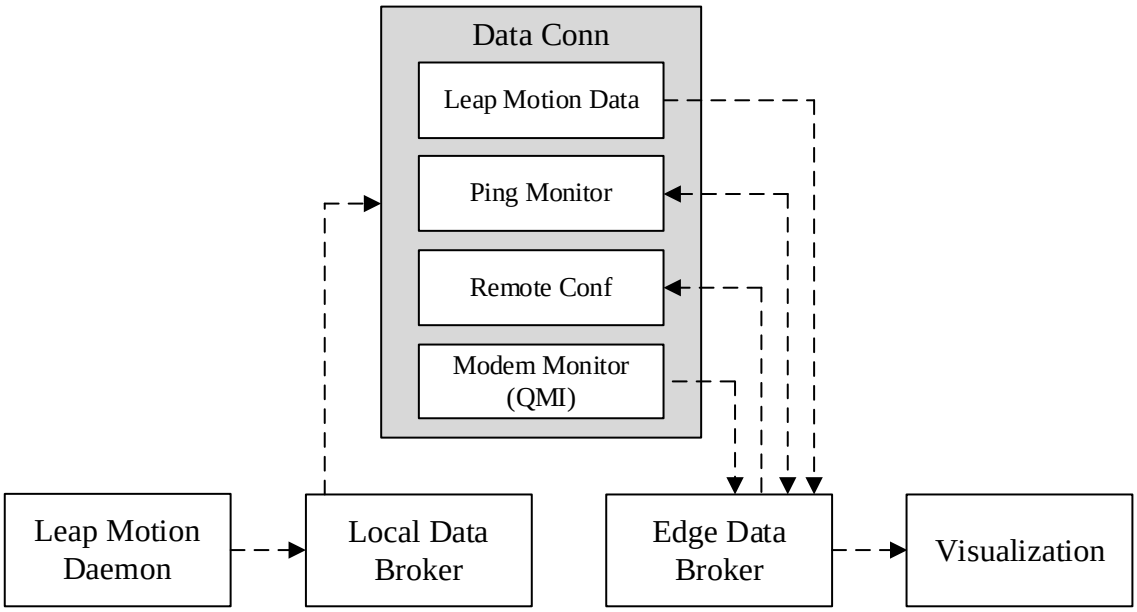


Figura 3.3: Diagrama de bloques del sistema. Elaboración propia.

3.2.1. Leap Motion Daemon

Se utiliza un servicio de Linux, ya creado, para comenzar a tomar muestras de los datos con el controlador *Leap Motion Controller*. Con el *daemon*, además de inicializar el sensor, se establece una conexión MQTT en la que un cliente recibe los *frames* provenientes del sensor. Los publica en un *topic* configurable, por defecto, */leap_motion/raw_data*, de un *broker*, con la dirección IP que se le indique, por defecto, la del *localhost*, 127.0.0.1.

3.2.2. Local Data Broker

En el propio equipo donde está conectado el dispositivo, se inicializa un *broker* MQTT *Mosquitto*[40] en el que se publican los datos de las posiciones de las manos que proporciona el Leap Motion Daemon, en el *topic /leap_motion/raw_data*. De este modo cualquier cliente dentro de la misma red, o de la propia máquina, puede conectarse al servidor, suscribirse al *topic* y obtener dichos datos de una manera sencilla, con tan solo conocer la dirección IP de la máquina.

Se consigue, así, que no haya necesidad de que el sensor esté en el mismo ordenador en que se ejecutan los demás módulos con los que cuenta el sistema, aportando un mayor grado de flexibilidad.

3.2.3. Edge Data Broker

Se hace uso de un *broker* MQTT EMQX [41] que se encuentra en el *Edge*, y que está gestionado por Ikerlan. En él se publica toda aquella información que deba ser accesible desde cualquier equipo, de manera remota.

Todo *topic* de este *broker* en que se realice una publicación, debe ser de la forma */leap_motion/{id}/{subtopic}*. De este modo, gracias al identificador (id), se logra distinguir de qué cliente es cada mensaje MQTT, para facilitar la obtención de una mejor comparativa entre ellos. Con el *subtopic* se consigue diferenciar qué tipo de información ha sido recibida, para su posterior procesado.

3.2.4. Visualización

Para poder observar los resultados obtenidos en el *broker* remoto, se presentan dos interfaces de visualización: *Hand-Visualizer* y *Grafana*.

- **Hand-Visualizer**

Para la visualización de las manos se usa una aplicación Web, programada a partir del módulo facilitado por *Ultraleap*, *css-visualizer*, como se mencionó en el Apartado 3.1.4. Ha sido modificado de manera que los datos de entrada se puedan obtener por medio de una conexión MQTT. Para ello, se crea un cliente que tras conectarse al *broker* en el *Edge*, se suscribe al *topic /leap_motion/{id}/data*. De esta forma pueden obtenerse y mostrarse gráficamente, en tiempo real, todos los datos de los movimientos de las manos que esté captando el sensor.

Otro cambio respecto al original es que cada cliente conectado al *broker* se establece en una capa diferente, para que se puedan contrastar más fácilmente las diferencias en las latencias de las distintas tecnologías, así como las que se producen cuando se simulan propiedades de redes en escenarios hostiles, con peor calidad de red, mayores pérdidas, etc.

En la Figura 3.4, un ejemplo de visualización de las manos de un usuario desde la interfaz *Hand-Visualizer*.

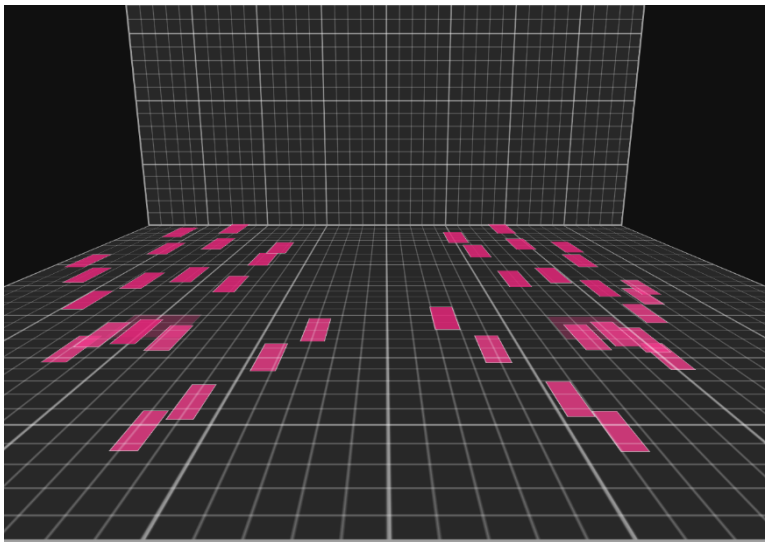


Figura 3.4: Visualización de las manos vista con la interfaz de *Hand-Visualizer*. Elaboración propia.

- **Grafana**

Por otra parte, para obtener de una manera más visual los *pings* resultados, así como información acerca del módem y la conexión, se utiliza *Grafana*. Se trata de una aplicación web que ofrece la posibilidad de graficar y analizar datos métricos en tiempo real.

Una de sus características más interesantes es la opción de habilitar un *plugin* de MQTT[21], de forma que las gráficas puedan ser generadas con datos obtenidos por un cliente suscrito a cierto *topic*. Así, para obtener, por ejemplo, el *ping*, tan solo debe establecerse una conexión entre el cliente y el *broker* en el *Edge* y realizarse una suscripción al *topic /leap_motion/+/ping*.

Si se quiere obtener información acerca de varios clientes, debe realizarse una suscripción a cada *topic* individualmente, para que puedan formarse una gráfica distinta para cada uno. Es decir, si los clientes que han publicado mensajes son aquellos con los identificadores 0 y 1, el cliente de *Grafana*, para obtener los RTT de las transmisiones, debe suscribirse a los *topics /leap_motion/0/ping* y */leap_motion/1/ping*.

Puede verse la interfaz de la herramienta, con una muestra de datos obtenidos con el *plugin* de MQTT, en la Figura 3.5.

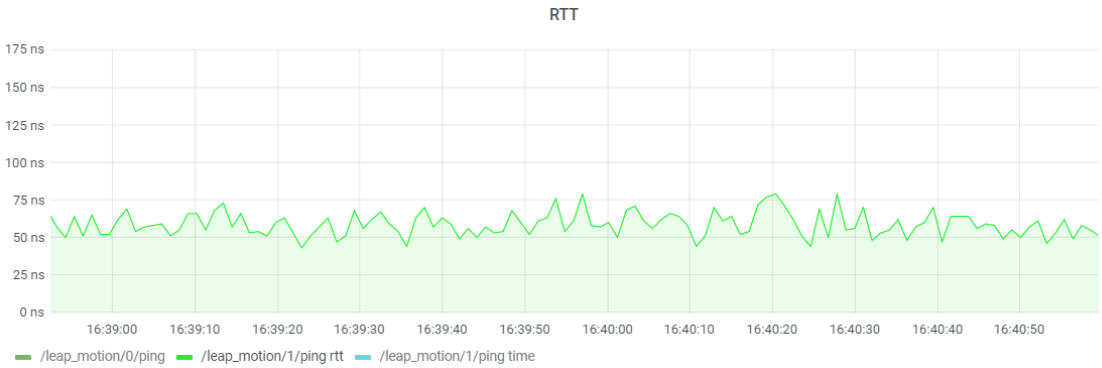


Figura 3.5. Muestra de datos obtenidos en tiempo real con una conexión MQTT, en un intervalo de dos minutos, vista con la herramienta *Grafana*. Elaboración propia.

[21] https://github.com/grafana/mqtt-datasource (accessed Jun. 29, 2022).

3.2.5. Conector de datos

Para interconectar los dos *brokers* mencionados anteriormente, se implementa un módulo denominado *Data Conn*. Permite, además de reenviar los datos de posicionamiento de las manos del *Local Data Broker* al *Edge Data Broker*, realizar una monitorización del *ping* y del módem, así como configurar, en tiempo real, la calidad de la red y ciertos aspectos del módem. El código del módulo se puede encontrar en el repositorio [42], así como en el Anexo Código al final de este documento.

- **Reenvío de datos de posicionamiento**

Los datos del posicionamiento de las manos, como se mencionó en el Apartado 3.1.1., se envían del sensor al ordenador vía USB. Como se refleja en la Figura 3.6, tras ser procesados gracias al software proporcionado por *Ultraleap*, se lanza un cliente MQTT, con el servicio *Leap Motion Daemon*, que publica los datos en el topic */leap_motion/raw_data* en el *broker* local.

A continuación, se crean otros dos clientes, dentro del módulo *Data Conn*, que actúa de puente entre los dos servidores. El primero de ellos se conecta al *broker* local y se suscribe al *topic /leap_motion/raw_data*. En cuanto se recibe un mensaje, el segundo cliente lo publica en el *topic /leap_motion/{id}/data* del *broker* en el borde de la red, al que se ha conectado previamente.

A este último topic se suscribe el cliente MQTT creado en el módulo *Hand-Visualizer*, que se ha conectado al *broker* en el *Edge*. Recibe, de esta forma, los datos para la representación de las manos.

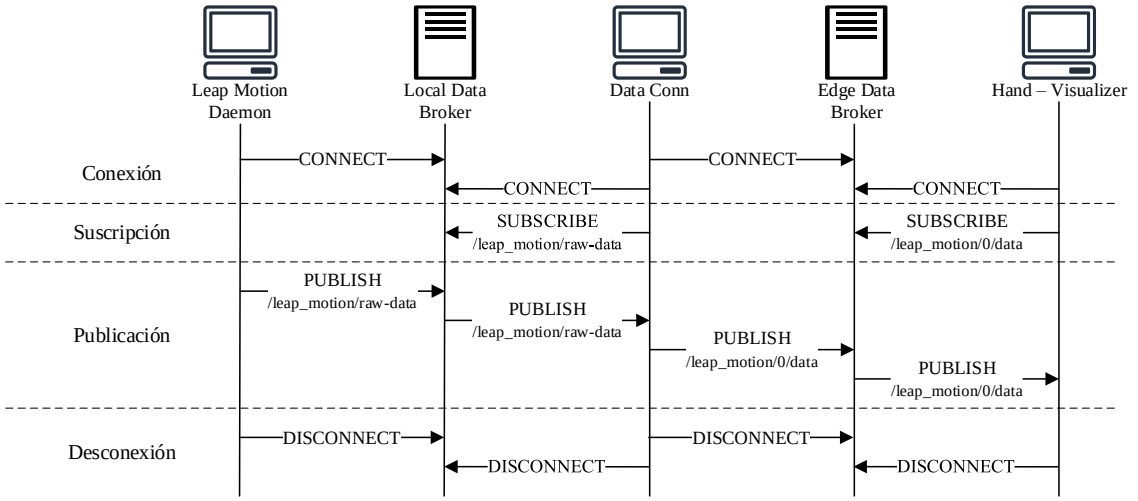


Figura 3.6: Fases de una sesión MQTT para el tráfico de datos. Elaboración propia.

- **Monitorización del ping**

El cliente del módulo *Data Conn* conectado al *broker* en el *Edge* publica, cada segundo, un mensaje en el *topic /leap_motion/{id}/ping* y, además, se suscribe a él. Con esto se puede conocer el momento en que se publicó y recibió el mensaje, permitiendo calcular los tiempos de ida y vuelta de la transmisión. El RTT es, de hecho, el contenido de los mensajes que son publicados periódicamente.

Así, cualquier otro cliente MQTT conectado al servidor remoto puede suscribirse a dicho *topic* y obtener las latencias generadas. Para la implementación realizada, se lanza un cliente gracias al *plugin* MQTT del software *Grafana* mencionado en el Apartado 3.2.4. Se conecta al *broker* en el *Edge* y se suscribe al *topic /leap_motion/{id}/ping*, para representar gráficamente y a tiempo real, los RTT de la comunicación.

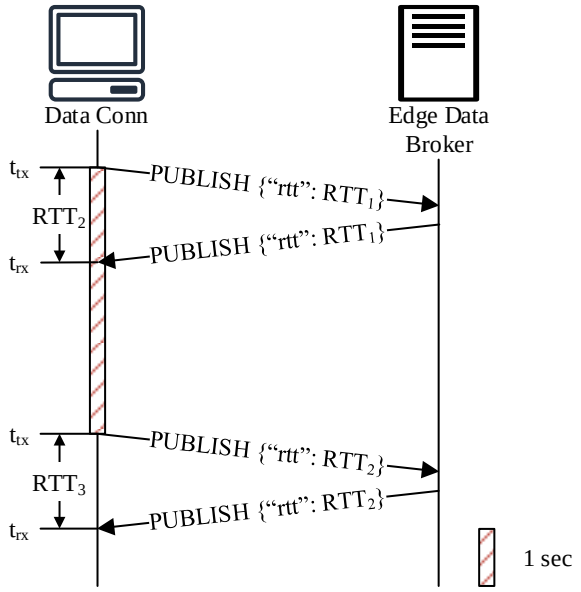


Figura 3.7: Cálculo de los tiempos de ida y vuelta. Elaboración propia.

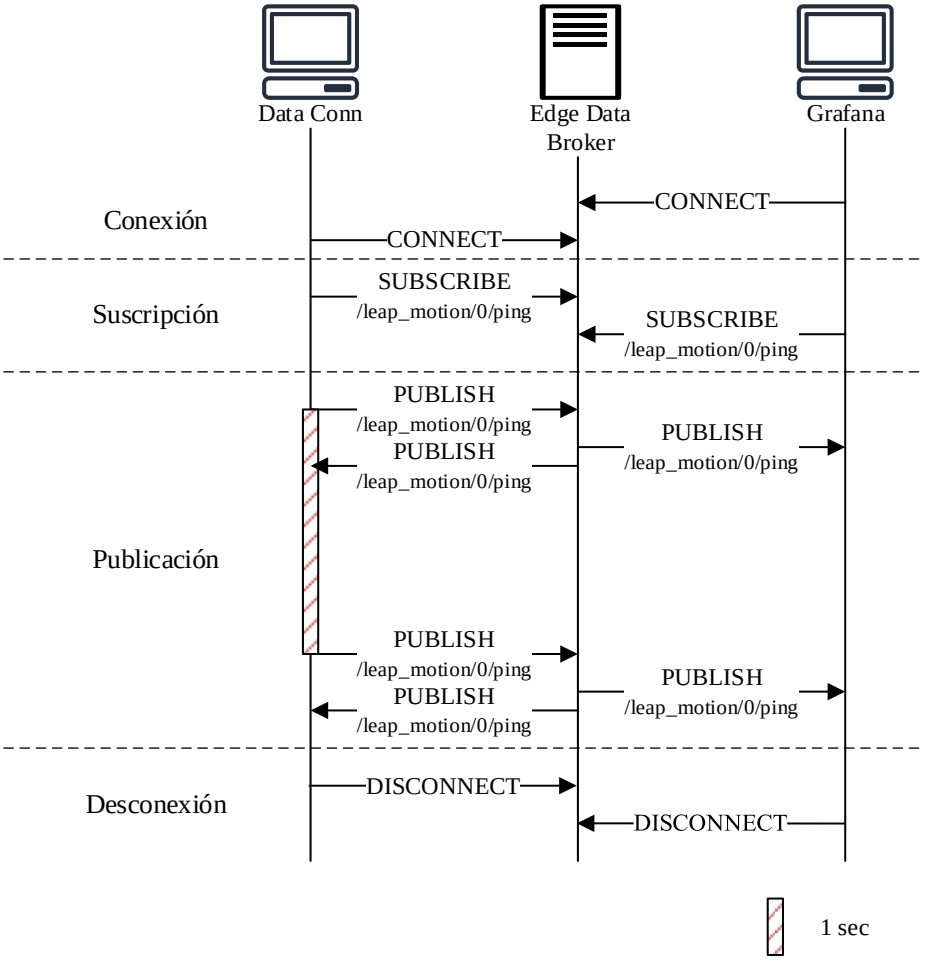


Figura 3.8: Fases de una sesión MQTT para el tráfico de ping. Elaboración propia.

En la Figura 3.7, se muestra el cronograma para el cálculo del *Round Trip Time* (RTT). Resulta de la diferencia, en milisegundos, entre el instante de tiempo en que el cliente ha recibido el mensaje y el que envió el mensaje, como se recoge en la Ecuación 3.1.

$$RTT_x = t_{rx} - t_{tx}\ [ms] \tag{3.1}$$

Podría pensarse, por tanto, que permite calcular el tiempo exacto que tarda en llegar el mensaje de un servidor a otro, dividiendo el RTT entre dos. No obstante, al estar incluidos en este resultado los tiempos de procesado, no puede relacionarse con esta latencia de una manera unívoca. Además, puede darse el caso de que la latencia sea asimétrica entre los dos *brokers*.

El método utilizado en el sistema puede entenderse, sin embargo, como un cálculo de la latencia real a nivel de aplicación, puesto que permite ver cómo se comporta realmente la aplicación que maneja un actuador conectado en el otro extremo.

En la Figura 3.8 se muestran las fases de una sesión MQTT para el tráfico del *ping* de la transmisión.

- **Configuración remota**

Para poder simular el comportamiento de la conexión en un entorno hostil, donde se puedan generar pérdidas de paquetes y fallos de comunicación, se permite configurar la red remotamente.

Como refleja la Figura 3.9, un cliente MQTT puede, tras haberse conectado al *broker* remoto, publicar en el *topic /leap_motion/{id}/conf* un mensaje con formato JSON[22], como se muestra en el Anexo A. De esta forma, el usuario tiene la capacidad de realizar un control de tráfico, modificando ciertos parámetros, como la tasa de ancho de banda o los tiempos de latencia o, por ejemplo, estableciendo un porcentaje de pérdidas.

Además, si se trata de una conexión a través del módem, en este mismo *topic* se puede publicar otro mensaje con formato JSON (Anexo A) en el que se especifique la tecnología móvil a utilizar (UMTS, LTE, NR5G).

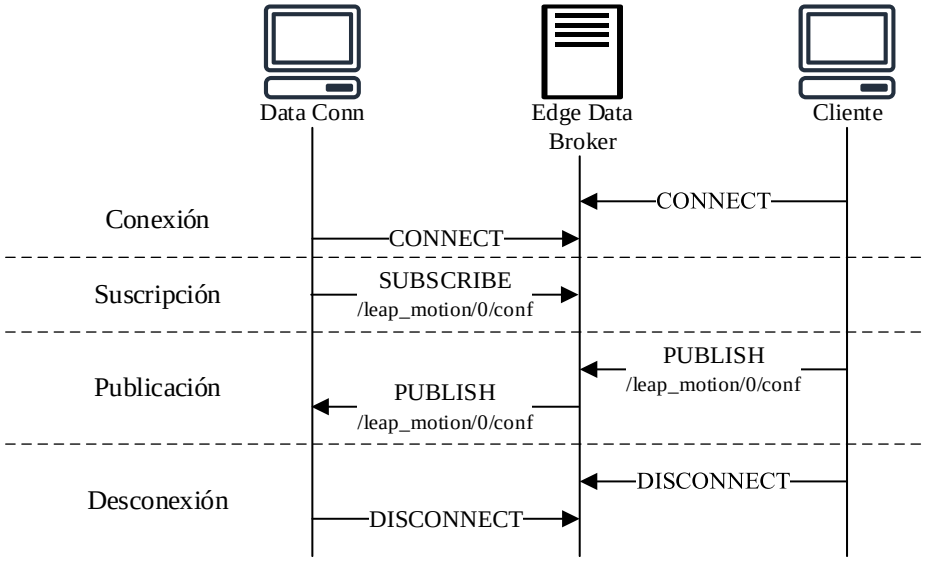


Figura 3.9: Fases de una sesión MQTT para el tráfico de configuración remota. Elaboración propia.

[22] *JSON*: *JavaScript Object Notation*.

El cliente del módulo *Data Conn* conectado al *broker* en el Edge se suscribe al *topic*. Con los mensajes obtenidos, desde el módulo puede configurarse la conexión, siguiéndose las indicaciones del usuario.

- **Monitorización del módem**

Existe también la posibilidad de llevar un seguimiento remoto del estado de la red y del módem. Cada 10 segundos se realizan consultas al módem, para obtener cierta información, como el modelo de dispositivo, la calidad de la red o la tecnología que se está utilizando. De este modo, el usuario puede llevar, con mayor exactitud, un registro de las latencias obtenidas, ya que se conoce más información acerca del entorno en que se están produciendo.

Las respuestas son analizadas y convertidas al formato de texto JSON (Anexo A) dentro del módulo *Data Conn*. Son publicadas por el cliente conectado al *broker* remoto en el *topic /leap_motion/{id}/status*. El cliente lanzado con *Grafana* puede, tras haber establecido una conexión con el servidor, suscribirse al *topic* y obtener los datos, para mostrarlos gráficamente. Las diferentes fases de la conexión pueden observarse en la Figura 3.10.

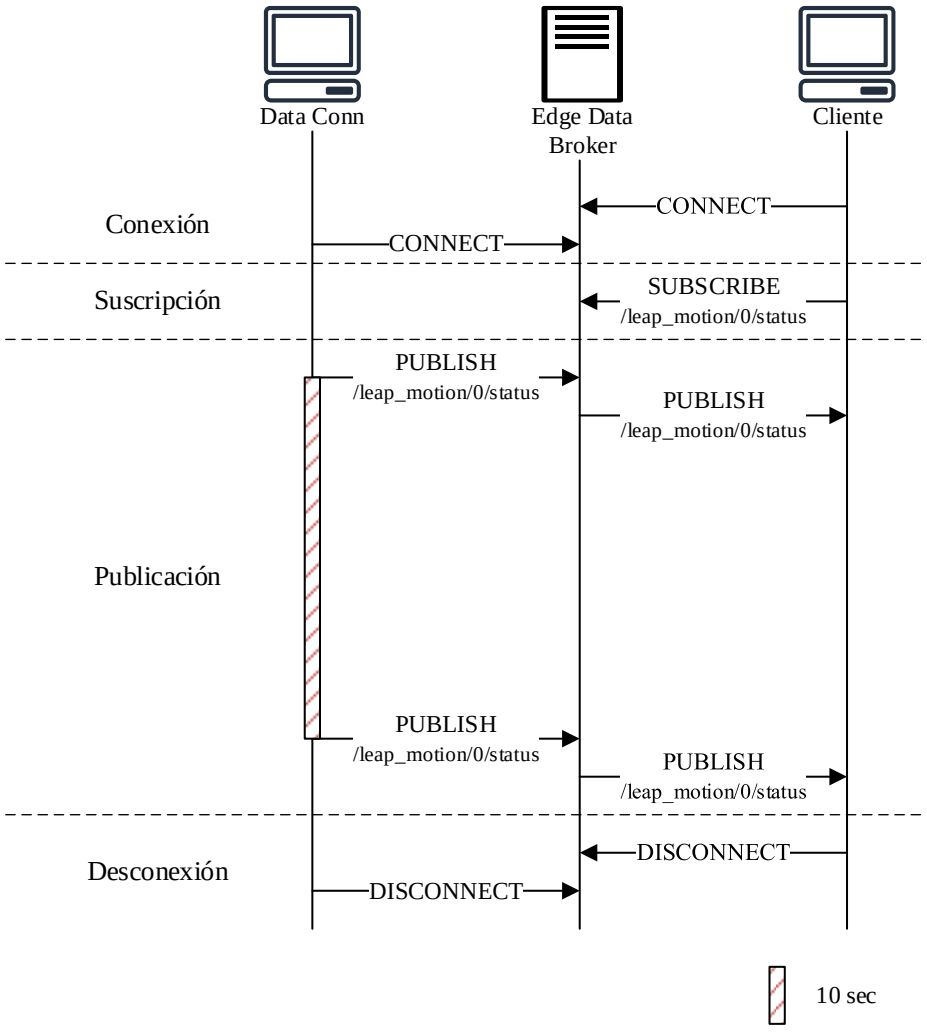


Figura 3.10: Fases de una sesión MQTT para el tráfico de estado de la conexión. Elaboración propia.

3.3. Integración y puesta en marcha

Seguidamente, se realiza una exposición del módulo implementado, así como de los aspectos necesarios para la puesta en operación del sistema completo, el cual se ha desarrollado sobre una máquina virtual con la distribución Linux Ubuntu.

3.3.1. Implementación del módulo *Data Conn*

Se exponen, a continuación, ciertas utilidades necesarias para implementar las funcionalidades propias del módulo *Data Conn* [42] desarrollado en este proyecto.

Lenguaje de programación - GO

El módulo *Data Conn* ha sido realizado en el lenguaje de programación GO. Desarrollado por Google, ha sido elegido por su conveniencia para futuras implementaciones de MQTT sobre el protocolo QUIC.

Para instalar el entorno de GO, se deben descargar los binarios de GO y extraer el archivo. A continuación, añadir la ruta de GO a la variable PATH del sistema y actualizar dicha variable en el terminal activo.

```
$ wget https://redirector.gvt1.com/edgedl/go/go1.9.2.linux-amd64.tar.gz
$ sudo tar -C /usr/local -xzf go1.9.2.linux-amd64.tar.gz
$ echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.profile
$ source ~/.profile
```

Para los clientes MQTT, se instala la librería de GO *Eclipse Paho-MQTT 3.1/3.1.1*[23]. Permite que las aplicaciones se conecten a un *broker* MQTT, para publicar mensajes y suscribirse a *topics* y recibir mensajes publicados.

```
$ go get github.com/eclipse/paho.mqtt.golang
```

Linux Traffic Control

Se usa la utilidad de Linux *Traffic Control* (tc) para llevar un control del tráfico de la red. Para la configuración de la red con el módulo *Data Conn*, en concreto, se utiliza el comando *tcset*[24], que permite agregar una regla de control de tráfico a un interfaz de red desde el terminal.

Para poder utilizar *tcset*, debe instalarse *tcconfig* con el administrador de paquetes software escritos en Python3, *pip3*.

```
$ sudo pip3 install tcconfig
$ curl -sSL https://raw.githubusercontent.com/thombashi/tcconfig/master/script
s/installer.sh | sudo bash
```

Los comandos se ejecutan directamente sobre el terminal Linux, a partir de los mensajes con formato JSON (Anexo A) publicados por un cliente en el *broker* remoto, como se indicaba en el Apartado 3.2.3.

[23] https://github.com/eclipse/paho.mqtt.golang (accessed Jun. 17, 2022).
[24] https://tcconfig.readthedocs.io/en/latest/pages/usage/tcset/index.html (accessed Jun. 17, 2022).

Permite, por ejemplo, añadir retardos de ida y vuelta o modificar las tasas de pérdida de paquetes, de anchos de banda o de duplicación de paquetes de ida y vuelta. Para ello, es necesario indicar el interfaz de red por donde se transmita el flujo de datos con estas modificaciones. A continuación, se muestra un comando *tcset* con el que se añade un retardo de 100 ms y se establece un porcentaje de pérdidas del 0.2 % al puerto Ethernet:

```
$ tcset eth0 --delay 100 --loss 0.2
```

Interfaz de línea de comandos de QMI

El módulo de comunicaciones inalámbricas utilizado en el sistema, RG500Q-EA, soporta conexiones por la interfaz QMI. Es por ello por lo que, para la monitorización del módem, así como para modificar la tecnología de red de la conexión, dentro de *Data Conn* se utiliza la interfaz de línea de comandos de QMI, *qmicli*[25], que permite comunicarse con el módem desde el terminal Linux.

Se instalan las utilidades *libqmi-utils*, para poder hace uso de QMI desde la línea de comandos.

```
$ sudo apt-get install libqmi-utils
```

Para establecer las comunicaciones con el módem, debe indicarse la ruta del dispositivo y abrirse el *proxy* QMI[26], junto a cada petición. Se muestran, a continuación, peticiones realizadas a través de *qmicli* para obtener el modelo del módem y seleccionar la tecnología 5GNR.

```
$ qmicli --device=/dev/cdc-wdm0 --device-open-proxy --dms-get-model
$ qmicli --device=/dev/cdc-wdm0 --device-open-proxy --nas-set-system-selection
-preference=5GNR
```

3.3.2. Entorno MQTT

Se presenta el *broker* local que se utiliza en el sistema, así como clientes MQTT de interés para realizar pruebas de conexiones desde el escritorio o el terminal de Linux.

Eclipse Mosquitto™

Para el servidor local, se hace uso de uno de los *brokers* de código abierto más habituales en implementaciones de MQTT para comunicaciones IoT: *Eclipse Mosquitto*[40]. Goza de gran popularidad, entre otros aspectos, por su ligereza y por ser apto para múltiples plataformas y para una amplia variedad de escenarios.

Se muestra a continuación el proceso de instalación y puesta en marcha del *broker* en Ubuntu.

[25] *qmicli*: *QMI command-line interface.*
[26] *Proxy QMI*: *proxy* que configura un socket abstracto que escucha en una dirección predefinida y se encarga de sincronizar el acceso a un conjunto de puertos QMI compartidos.

1. Instalación de *Mosquitto*.

```
$ sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa
$ sudo apt-get update
$ sudo apt-get install python-software-properties
$ sudo apt-get install mosquitto
```

2. Inicialización de Mosquitto.

```
$ sudo mosquitto
```

3. Configuración del broker.

Al ejecutar *Mosquitto*, se utiliza por defecto la configuración del fichero *mosquitto.conf*. Si se desea aplicar una configuración personalizada, es posible bien modificar dicho fichero, o bien crear un nuevo archivo dentro del directorio *conf.d*. En ese caso, debe ser indicada la ruta del directorio del archivo correspondiente a la configuración que se desea utilizar.

```
$ sudo mosquitto -c /etc/mosquitto/mosquitto.conf
$ sudo mosquitto -c /etc/mosquitto/conf.d/conf_file.conf
```

Se puede hacer uso del servicio de *Mosquitto* para acceder al estado del servidor (activo, inactivo, fallido), así como para activar y detener su ejecución en segundo plano.

```
$ sudo service mosquitto status
$ sudo service mosquitto start
$ sudo service mosquitto stop
```

Para poder realizar pruebas y simulaciones de conexiones MQTT, se instalan los clientes MQTT de línea de comandos de *Mosquitto*, *mosquitto_sub* y *mosquitto_pub*.

```
$ sudo apt-get install mosquitto-clients
```

Cliente MQTT X

Para obtener todos los mensajes deseados de una transmisión realizada con MQTT de una manera más estética y visual, se puede utilizar el cliente *MQTT X* [43]. Se trata de un cliente de escritorio multiplataforma de código abierto, desarrollado por *EMQ*, que muestra en forma de conversación todo el intercambio de mensajes de los *topics* a los que se haya suscrito, de una conexión MQTT. Es, en definitiva, un cliente MQTT más de la comunicación, por lo que puede actuar también como *Publisher*.

```
$ sudo snap install mqttx
```

Conexión básica de clientes MQTT con un broker Mosquitto

Se detallan, a continuación, las fases de una conexión básica entre clientes de línea de comandos y un cliente *MQTT X*, con un *broker Mosquitto*.

1. En un terminal Linux, se inicializa el broker en local, en el puerto 1883, sin utilización de seguridad con TLS. Para ello, no es necesario modificar el fichero de configuración.

```
$ sudo mosquitto

mosquitto version 2.0.14 starting
Using default config.
```

```
Starting in local only mode. Connections will only be possible from clients
running on this machine.
Create a configuration file which defines a listener to allow remote access.
For more details see https://mosquitto.org/documentation/authentication-
methods/
Opening ipv4 listen socket on port 1883
Opening ipv6 listen socket on port 1883
mosquitto version 2.0.14 running
```

2. Conexión del cliente MQTT desde la herramienta MQTT X al broker en local, con ID del cliente Client_MQTTX. Como se puede ver en la Figura 3.11, no se utiliza TLS, ni usuario y contraseña.

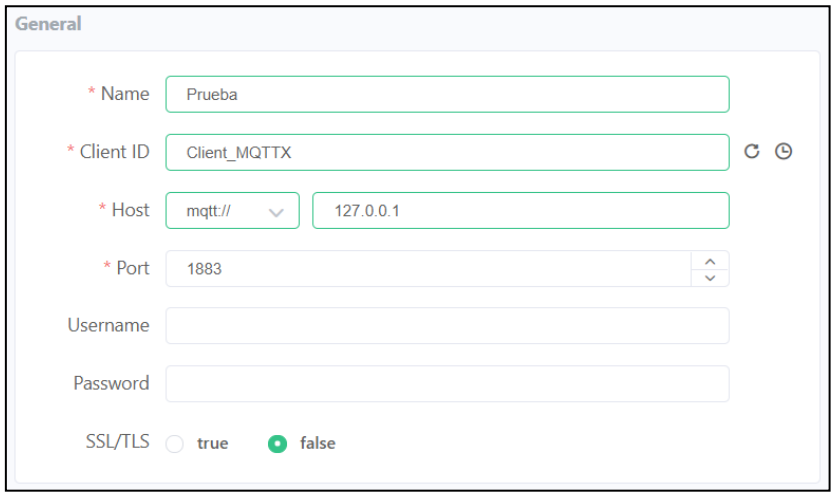


Figura 3.11: Ventana de creación de conexión en MQTT X. Elaboración propia.

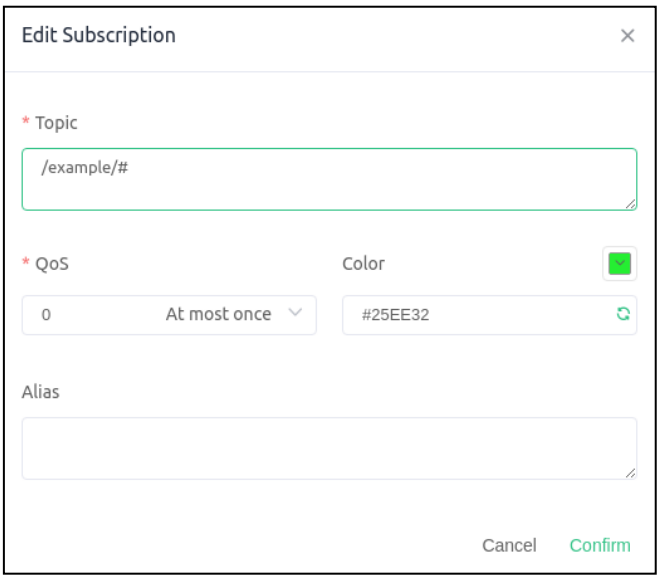


Figura 3.12: Ventana de suscripción en MQTT X. Elaboración propia.

3. Suscripción del cliente Client_MQTTX al topic /example/#. Se suscribe a todos los subtopics dentro de /example y se establece la calidad de servicio QoS 0. Se muestra en la Figura 3.12.

4. Conexión del cliente MQTT con ID subscriber_1 al broker Mosquitto y suscripción al topic /example/1.

```
$ mosquitto_sub -i subscriber_1 -h 127.0.0.1 -t /example/1
```

5. Conexión del cliente MQTT con ID publisher_1 al broker Mosquitto, publicación en el topic /example/1 del mensaje “Hello world!” y desconexión.

```
$ mosquitto_pub -i publisher_1 -h 127.0.0.1 -t /example/1 -m “Hello world!”
```

6. Conexión del cliente MQTT con ID publisher_2 al broker de Mosquitto, publicación en el topic /example/2 del mensaje “Hello world!” y desconexión.

```
$ mosquitto_pub -i publisher_2 -h 127.0.0.1 -t /example/2 -m “Hello world!”
```

7. Desconexión de los clientes subscriber_1 y client_MQTTX y del broker.

En la terminal donde se inició el *broker Mosquitto* se muestran todas las conexiones y desconexiones de los clientes que han participado a lo largo de toda la transmisión.

```
New connection from 127.0.0.1:52290 on port 1883.
New client connected from 127.0.0.1:52290 as Client_MQTTX (p2, c1, k60).
New connection from 127.0.0.1:52292 on port 1883.
New client connected from 127.0.0.1:52292 as subscriber_1 (p2, c1, k60).
New connection from 127.0.0.1:52294 on port 1883.
New client connected from 127.0.0.1:52294 as publisher_1 (p2, c1, k60).
Client publisher_1 disconnected.
New connection from 127.0.0.1:52296 on port 1883.
New client connected from 127.0.0.1:52296 as publisher_2 (p2, c1, k60).
Client publisher_2 disconnected.
Client subscriber_1 disconnected.
Client Client_MQTTX disconnected.
```

El suscriptor de línea de comandos tan solo se suscribió al *topic /example/1*, por lo que solo recibió el mensaje publicado por el primer cliente *publisher_1*.

```
$ mosquitto_sub -i subscriber_1 -h 127.0.0.1 -t /example/1
Hello world!
```

No es el caso del cliente de MQTT X. Al haberse suscrito a todos los *subtopics* de */example*, pudo obtener la conversación completa, como se refleja en la Figura 3.13.

Figura 3.13: Conversación vista desde MQTT X. Elaboración propia.

3.3.3. Conectividad inalámbrica

Se detallan, a continuación, los pasos que se han seguido para poner en marcha el módem de *Quectel*, así como otros aspectos necesarios para dotar de conectividad inalámbrica al sistema.

Puesta en marcha

Para la puesta en marcha el dispositivo, se procede de la siguiente manera:

1. Conectar el módulo RG500Q-EA a la EVB en los conectores BTB27 J0101 y J0102.
2. Insertar la tarjeta MicroSIM en el conector de tarjetas SIM J1401
3. Conectar la EVB por el puerto J0303 a una alimentación de 5 V.
4. Encender el interruptor de alimentación S0301 (ON).
5. Presionar el pulsador S0202 (PWRKEY) durante al menos 500 ms.
6. Conectar la EVB al PC a través del conector USB de Tipo C J1101.

Los planos del kit de desarrollo se encuentran en el Anexo B. Para más información, consultar la guía de usuario del kit de desarrollo [44].

Instalación de driver qmi_wwan

Para poder utilizar el módem, es necesario instalar un controlador de dispositivos. Se ha optado por utilizar el *driver qmi_wwan* ya integrado en el sistema operativo de Ubuntu, puesto que el que proporcionaba *Quectel* causaba ciertos problemas con el *kernel* 5.4.0-120 de Ubuntu 18.04 en que el sistema es desplegado.

Pruebas con comandos AT en Windows

Una vez instalado el controlador, con el fin de realizar comprobaciones del correcto funcionamiento del módem, se puede hacer uso de los comandos AT. Para ello, es preciso contar con una interfaz que permita comunicaciones con dispositivos que estén conectados al puerto serie, como son los emuladores de terminal *TeraTerm*, *Hyper Terminal* o *Putty*. No obstante, para sistemas operativos Windows, *Quectel* provee de una herramienta propia para ello, *QCOM*, en su versión 1.6.

A continuación, se enumeran los pasos a seguir para establecer una comunicación con la herramienta *QCOM*.

1. Localizar el puerto serie dedicado para comunicaciones con comandos AT, desde el administrador de dispositivos de Windows, como se muestra en la Figura 3.14.
2. Ejecutar el archivo QCOM_V1.6.exe.
3. Seleccionar el puerto serie y la tasa de baudios.
4. Opcionalmente, elegir los bits de parada, la paridad, el tamaño de byte y el control de flujo.
5. Abrir el puerto serie.
6. Enviar un archivo o una cadena de comandos AT.
7. Cerrar el puerto serie.

En la Figura 3.15, se ejemplifica una configuración de la herramienta QCOM.

[27] *BTB*: *Board-to-board*.

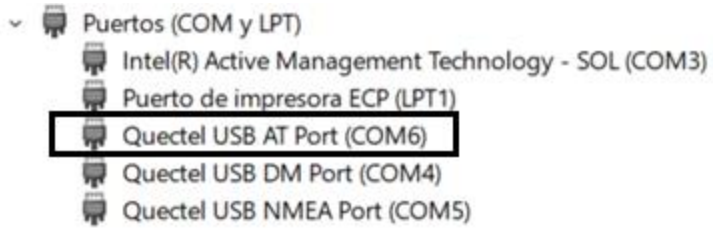


Figura 3.14: Vista de puertos en el administrador de dispositivos. Elaboración propia.

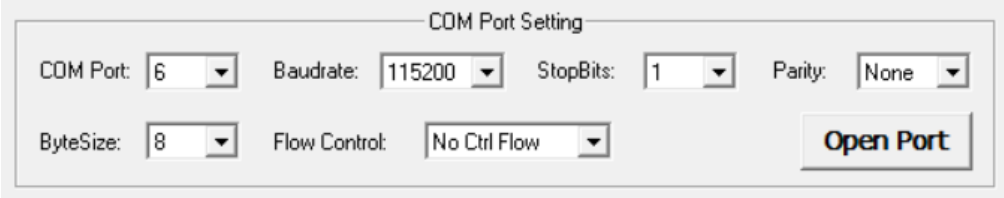


Figura 3.15: Configuración de la herramienta QCOM. Elaboración propia.

Pruebas con comandos AT en Linux

En Linux, en cambio, el fabricante no ofrece ninguna herramienta propietaria. En su lugar, se ha optado por utilizar *Minicom*, de uso muy común en sistemas operativos Unix.

Se instala la herramienta desde el terminal de Linux.

```
$ apt-get install minicom
```

Minicom debe ser configurada correctamente para que pueda realizarse la comunicación entre el ordenador y el módem. Para ello, se entra en el modo configuración y se siguen los siguientes pasos:

1. Seleccionar la configuración del puerto serie (*Serial port setup*)
2. *Serial Device*: cambiar el puerto serie a utilizar. Normalmente, */dev/ttyUSB2*
3. *Bps/Par/Bits*: cambiar la tasa de baudios por una adecuada.

```
$ sudo minicom -s

    +-----------------------------------------------------------------------+
    | A -    Serial Device      : /dev/ttyUSB2                              |
    | B - Lockfile Location     : /var/lock                                 |
    | C -   Callin Program      :                                           |
    | D -  Callout Program      :                                           |
    | E -    Bps/Par/Bits       : 115200 8N1                                |
    | F - Hardware Flow Control : No                                        |
    | G - Software Flow Control : No                                        |
    |                                                                       |
    |    Change which setting?                                              |
    +-----------------------------------------------------------------------+
```

Una vez instalado y configurado, se envía el comando *AT* como comprobación de que el ordenador y el módulo pueden comunicarse. Si se recibe un *OK* como código de resultado, la conexión ha sido establecida correctamente.

A continuación, se utiliza el comando *AT+CPIN* para comprobar si es necesario ingresar alguna contraseña ((U)SIM PIN, (U)SIM PUK, PH-SIM PIN, etc.) y, de ser así, introducirla:

```
AT+CPIN?
+CPIN: SIM PIN

OK
AT+CPIN=1234
OK

+CPIN: READY
+QUSIM: 1
+QIND: SMS DONE
+QIND: PB DONE
```

Pueden realizarse una gran variedad de consultas acerca del dispositivo, como el fabricante, el modelo, la versión del firmware o el IMEI[28]:

```
AT+GMI
Quectel

OK
AT+GMM
RG500Q-EA

OK
AT+GMR
RG500QEAAAR10A02M4G

OK
AT+GSN
86689704010****

OK
```

MódemManager

Otra forma de acceder a la información y más opciones del módem es a través del software estándar de Linux, *MódemManager*. Ofrece la posibilidad de comunicarse a través de canales de control de dispositivos, como los comandos AT o QMI. Para establecer la comunicación desde el terminal Linux, se hace uso de la interfaz de línea de comandos *mmcli*[29].

```
$ sudo apt install modemmanager

$ sudo systemctl start ModemManager.service

$ mmcli --list-modems
/org/freedesktop/ModemManager1/Modem/0 [Quectel] RG500QEA_VH

$ mmcli -m 0 --simple-connect="pin=1234, apn=movistar.es"
successfully connected the modem

$ mmcli --modem=0
  --------------------------------
  General  |            dbus path: /org/freedesktop/ModemManager1/Modem/0
           |            device id: fdedcc9aff9e676c6c19df0b23088f18b7a90092
  --------------------------------
  Hardware |         manufacturer: Quectel
           |                model: RG500QEA_VH
           |             revision: RG500QEAAAR10A02M4G
           |         h/w revision: 20000
           |            supported: gsm-umts, lte
           |              current: gsm-umts, lte
           |         equipment id: ***************
  --------------------------------
 (...)
```

[28] *IMEI*: *International Mobile Equipment Identity*, identidad internacional del equipo móvil.
[29] *mmcli*: *ModemManager command-line interface*.

Si se desea establecer u obtener otros parámetros, puede activarse el modo *debug* y utilizar *mmcli* para comunicarse con el módem por comandos AT. Así, por ejemplo, puede seleccionarse la tecnología de red (AUTO, WCDMA, LTE o NR5G) o consultarse datos de la celda de servicio, o información del operador, como la tecnología de acceso actual.

```
$ sudo systemctl stop ModemManager.service

$ sudo /usr/sbin/ModemManager –debug

$ mmcli -m 0 --command="+QNWPREFCFG=\"mode_pref\",AUTO"
response: ''

$ mmcli -m 0 --command="+QENG=\"servingcell\""
response: '+QENG: "servingcell","NOCONN"
+QENG: "LTE","FDD",214,07,30D5C1F,111,6400,20,3,3,5083,-106,-13,-75,11,255,-
32768,17
+QENG:"NR5G-NSA",214,07,65535,-32768,-32768,-32768'

$ mmcli -m 0 --command="+COPS?"
response: '+COPS: 0,0,"Movistar",13'

$ sudo systemctl start ModemManager.service
```

NetworkManager

Para gestionar las conexiones del ordenador y poder establecer la conexión a través del módem, se utiliza el software de código abierto de Linux *NetworkManager*. Se emplea *nmcli*[30], la herramienta de línea de comandos del software, desde el terminal de Linux.

```
$ sudo apt-get install network-manager
$ sudo systemctl start NetworkManager

$ nmcli device
DEVICE    TYPE      STATE         CONNECTION
enp0s3    ethernet  connected     Wired connection 1
docker0   bridge    connected     docker0
cdc-wdm0  gsm       disconnected  --
lo        loopback  unmanaged     --

$ nmcli connection show
NAME                    UUID                                  TYPE      DEVICE
Wired connection 1      4690a2fa-96af-373a-8902-b0bc48701ecd  ethernet  enp0s3
docker0                 7368e78c-18c1-41af-94f2-255a8cf2392b  bridge    --
docker0 Movistar (Tel…  4c8a4746-b0f9-4723-9cb2-c6885fc15950  gsm       --

$ nmcli connection up 4c8a4746-b0f9-4723-9cb2-c6885fc15950
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/0)

$ nmcli device
DEVICE    TYPE      STATE      CONNECTION
enp0s3    ethernet  connected  Wired connection 1
docker0   bridge    connected  docker0
cdc-wdm0  gsm       connected  Movistar (Telefónica) Movistar (USB modems)
lo        loopback  unmanaged  --
```

Como comprobación del correcto funcionamiento de la conexión, se puede enviar un ping, por ejemplo, a la dirección IP del servidor DNS de *Google*, por el interfaz del módem.

```
$ ping -I wwp0s12u2i4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) from 176.***.***.*** wwp0s12u2i4: 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=49.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=47.8 ms
```

[30] *nmcli*: *NetworkManager command-line interface*.

```
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=45.8 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 45.825/47.593/49.115/1.377 ms
```

3.3.4. Puesta en marcha del sistema

Seguidamente, se detallan los pasos a seguir para poner en funcionamiento todo el sistema y conseguir los resultados esperados.

Desde un terminal de Linux, se inicializa el *broker Mosquitto* que actúa como servidor local. Para que se mantenga activo en segundo plano, se hace uso del servicio *Mosquitto*.

```
$ sudo service mosquitto start
```

Mediante un cable USB, se conecta el sensor *Leap Motion Controller* al ordenador. A continuación, se lanza un *daemon* de Linux ejecutando el archivo *run_leap_daemon.sh*.

```
$ cd ~/leap-motion-docker
$ ./run_leap_daemon.sh
```

Para establecer una transmisión a través del módem de tecnologías móviles, tras la puesta en marcha que se detalla en el punto 3.3.3., el usuario debe introducir, con *mmcli*, el APN[31] correspondiente al proveedor de telefonía móvil y la clave PIN de la tarjeta SIM insertada.

También, ha de asegurarse de que se está utilizando el interfaz de red adecuado y, de no ser así, activarlo. Puede comprobarse con la herramienta *nmcli*.

```
$ mmcli -m 0 --simple-connect="pin=1234, apn=movistar.es"
$ nmcli connection show
$ nmcli connection up Movistar
```

Para que los datos lleguen al *broker* en el *Edge*, se ejecuta, con privilegios de seguridad de usuario *root*, el archivo *Data_Conn.go*, indicándose todas las opciones que se deseen.

Debe indicarse la dirección pública del *broker* remoto e introducirse la opción *--modem*, de modo que se puedan obtener todos los datos de la monitorización, así como habilitar la posibilidad de configuración remota de la tecnología de red. Pueden especificarse direcciones, puertos, *topics* u otros modos, que se encuentren dentro de las opciones del módulo, o mantenerlos con sus valores por defecto.

```
$ sudo go run Data_Conn.go --modem --dst_host=193.***.***.***
```

Visualización

Para obtener la visualización de las manos, debe ejecutarse el archivo HTML[32] del módulo *Hand-Visualizer*, indicando la dirección y el puerto del servidor en la URL en la barra de búsqueda. Para obtener las gráficas con *Grafana*, se accede al servidor remoto desde el buscador, en el puerto correspondiente.

```
file:///C:/Users/mbaron/Desktop/css-visualizer-mqtt/src/css-visualizer-
mqtt.html?wshosts=172.***.***.***:9001

http://172.***.***.***:3000/d/O0NcgFy7k/demo-leap-motion-5g
```

[31] *APN*: *Access Point Name*, nombre del punto de acceso.
[32] HTML: Hypertext Markup Language, lenguaje de marcado de hipertexto.

Capítulo 4

Resultados

Se presentan, en este capítulo, los resultados obtenidos y se realiza un análisis de estos, en base a las campañas de medida llevadas a cabo con diferentes tecnologías de acceso celular y en distintos escenarios.

En este proyecto, todas las pruebas de conectividad 5G han sido realizadas sobre la arquitectura 5G NSA, al estar haciéndose uso de la infraestructura pública del operador Movistar en España, donde las redes 5G SA aún no han sido desplegadas.

Seguidamente, se presenta una breve base teórica para complementar y mejorar la comprensión de los resultados obtenidos.

4.1. Marco teórico

Arquitecturas y redes de acceso radio

Las redes de acceso radio (RAN, *Radio Access Network*) son aquellas que comprenden la conexión de un equipo de usuario (UE, *User Equipment*) y la red principal, a través de estaciones base que ofrecen cobertura de radioenlace. En la Tabla 4.1 se muestra cómo se denomina a la estación base para cada tecnología celular.

En el marco de este Trabajo, se distinguen las siguientes arquitecturas RAN [6]:

- UTRAN, *UMTS Terrestrial Radio Access Network*. Permite a los UE acceder al núcleo de red de UMTS.
- E-UTRAN, *Evolved UMTS Terrestrial Radio Access Network*. Evolución de UTRAN, para dar acceso a redes LTE.
- EN-DC, *E-UTRAN-NR Dual Connectivity*. Configuración para 5G NSA, con un nodo maestro 4G eNB y un nodo secundario 5G en-gNB (*E-UTRAN-NR gNB*), conectados a un EPC.

Existen otras configuraciones, como E-UTRAN conectado a 5GCN, que proporciona a nodos ng-eNB (*Next Generation Enhanced NB*) conectividad con una red central 5G, a través de radioenlaces LTE; o NR conectado a 5GCN, para conexiones NR SA, entre un nodo gNB y la red 5GC.

El comando AT de lectura *AT+COPS?*, permite conocer algunos aspectos acerca del operador y la tecnología de acceso seleccionada. Como se refleja en la Tabla 4.2, cada tecnología se corresponde con un código numérico.

Tabla 4.1: Nomenclatura de las estaciones base para cada tecnología celular. Elaboración propia.

Tecnología	*Estación base*	*Siglas*
1G	Base Station	BS
2G	Base Transceiver Station	BTS
3G	Node B	NB
4G	Evolved Node B	eNB
5G	Next generation Node B	gNB

Tabla 4.2: Códigos relativos a las tecnologías de acceso permitidas por el módulo. Elaboración propia. Basado en [45].

Código	*Tecnología de acceso*
2	UTRAN
4	UTRAN W/HSDPA[33]
5	UTRAN W/HSUPA[34]
6	UTRAN W/HSDPA & HSUPA
7	E-UTRAN
10	E-UTRAN connected to a 5GCN
11	NR connected to 5GCN
12	NG-RAN
13	E-UTRAN-NR dual connectivity

Asignación de bandas de frecuencia

En el Cuadro Nacional de Atribución de Frecuencias [46] (CNAF) puede consultarse el uso de las distintas bandas de frecuencia en que se divide el espectro radioeléctrico en España, para los diferentes servicios de radiocomunicaciones.

Para la telefonía móvil, están, por el momento, reservadas las bandas de 700, 800, 900, 1500, 1800, 1900, 2100, 2600 y 3500 MHz, según se recoge en la Tabla 4.3, con la tecnología para la que se utilizan.

En concreto, para las comunicaciones con tecnología 5G, se define la banda n78, a 3500 MHz, como banda principal de alta velocidad; y la banda n28, a 700 MHz, como segunda banda, para reforzar la cobertura. Además, las bandas n3 y n1, a 1800 y 2100 MHZ, respectivamente, son utilizadas para 5G DSS[35]. La banda n258, a 26 GHz, se subastará en España próximamente, con vistas a ser dedicada para 5G mmWave.

Tabla 4.3: Clasificación de bandas de frecuencia para tecnologías celulares. Elaboración propia. Basado en [47].

Frecuencia	*Bandas*	*Tecnología*
700 MHz	28	5G
800 MHz	20	4G
900 MHz	8	2G/3G/4G
1500 MHz	32	4G SDL
1800 MHz	3	2G/4G/5G DSS
2100 MHz	1 y 39	3G/4G/5G DSS
2600 MHz	7 y 38	4G
3500 MHz	78	5G

[33] *HSDPA*: *High Speed Downlink Packet Access*.
[34] *HSUPA*: *High Speed Uplink Packet Access*.
[35] *DSS*: *Dinamic Spectrum Sharing*. Técnica utilizada para la coexistencia de LTE y NR en la misma frecuencia, para facilitar la migración hacia 5G.

4.2. Escenarios de despliegue

La totalidad de las pruebas de conectividad del proyecto se han realizado en las instalaciones de Ikerlan, S. Coop., en la localidad de Mondragón, Guipúzcoa, donde aún no se dispone de red en servicio 5G NR 3500. Se encuentra operativa, no obstante, la tecnología 5G-DSS en la banda n1, a 2100 MHz.

Para analizar con mayor profundidad el comportamiento de las distintas tecnologías celulares (UMTS, LTE, 5G), se han obtenido resultados desde dos escenarios de despliegue diferentes, de modo que pueda analizarse si se producen alteraciones.

Así, se examinan las latencias y las tasas de transferencia efectivas, además de diversos parámetros de calidad de la señal, en una primera ubicación (α), con una visión relativamente directa a la estación base, y otra segunda (β), que a priori tendrá un peor desempeño, al estar localizada en un lugar más recóndito dentro de la propia empresa. Se muestra un mapa con los escenarios en la Figura 4.1.

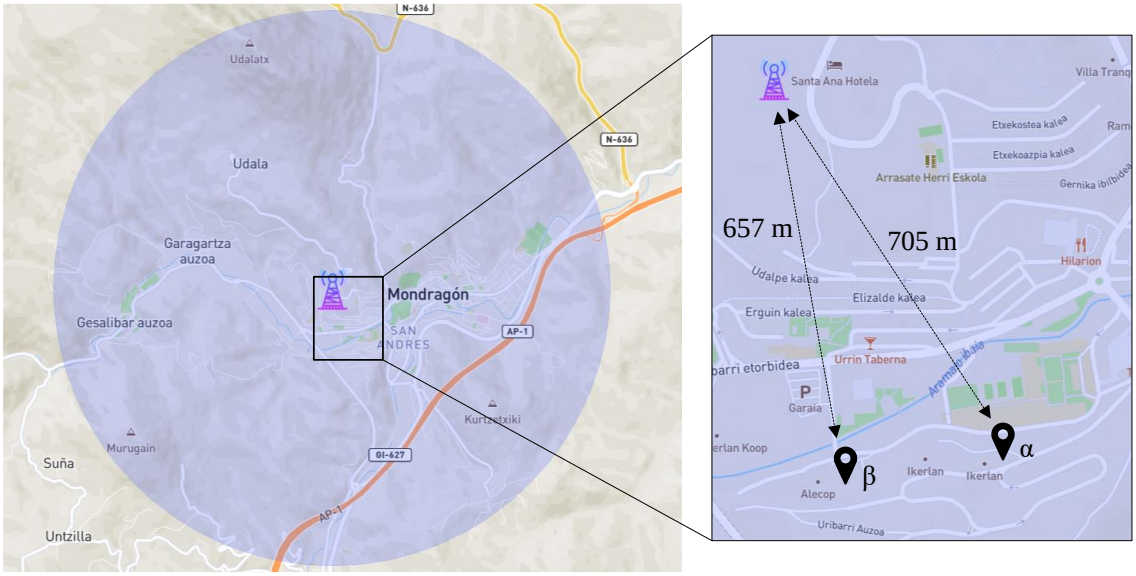


Figura 4.1. Localización de los escenarios α y β, ambos en la localidad de Mondragón, Guipúzcoa. Elaboración propia. Basado en [48].

4.3. Caracterización de las conexiones

Aprovechando la posibilidad que ofrece la interfaz de línea de comandos de la aplicación ModemManager, vista en el Apartado 3.3.3., de mantener una conversación con el módem vía comandos AT, se realizan una serie de peticiones y consultas, desde el puerto serie, para caracterizar de una mejor manera las conexiones establecidas.

Como se presentó en el Apartado 3.3.1., también existe la opción de realizar consultas a través del servicio de acceso a la red (NAS, *Network Access Service*) de la interfaz de línea de comandos de QMI, con el que se obtienen datos similares.

Para efectuar el análisis de las diferentes tecnologías móviles, se deben configurar las preferencias de búsqueda de red, seleccionando la tecnología preferente y fijando una banda de frecuencias adecuada, antes de realizar las consultas pertinentes. Se utiliza para ello el comando *AT+QNWPREFCFG*.

Tabla 4.4: Parámetros para la caracterización de las conexiones. Elaboración propia.

Parámetro	*Tecnología*	*Descripción*
MCC	Común	Código móvil de país (*Mobile Country Code*).
MNC	Común	Código de red móvil (*Mobile Network Code*).
CellID	Común	Identificador de la celda (*Cell ID*).
PCI	LTE, 5G	Identificador de celda física (*Physical Cell ID*).
LAC	UMTS	Código de área de localización (*Location Area Code*).
RAC	UMTS	Código de área de enrutamiento (*Routing Area Code*).
TAC	LTE	Código de área de rastreo (*Tracking Area Code*).
ARFCN	Común	Número de canal absoluto de radiofrecuencia (*Absolute Radio-Frequency Channel Number*). Dependiendo de la tecnología de acceso, puede tratarse de NRARFCN (NR), EARFCN (E-UTRAN) o UARFCN (UTRAN).

Se expone, a continuación, una caracterización de las diferentes conexiones establecidas, en las posiciones α y β, con datos del operador, de la celda de servicio y de la tecnología de acceso seleccionada, obtenidos a través de comandos AT (*AT+QENG="servingcell", AT+COPS, AT+QNWINFO*) y QMI (*--nas-get-system-info*, *--nas-get-cell-location-info*, *--nas-get-system-selection-preference*). Los parámetros que se han analizado se presentan en la Tabla 4.4.

- **Conexiones con tecnología UMTS**

La dupla MCC/MNC obtenida en ambas conexiones es 214/07, correspondiente al operador Movistar en España. Con los distintos parámetros obtenidos se demuestra que, en las dos conexiones, el UE se ha enlazado con la misma celda, cuyo CellID es 27566813. La ubicación del área de la celda viene dada por el código de enrutamiento 25 (RAC), dentro del área con código de localización 2061 (LAC).

Observando la tecnología de acceso, se puede verificar que se está utilizando la red de acceso UTRAN, con tecnología de acceso múltiple por división de código de banda ancha (WCDMA, *Wideband Code Division Multiple Access*). Se está operando en la frecuencia de 900 MHz, correspondiente a la banda 8.

Esto se demuestra también fijándose en el UARFCN. En este caso, se está utilizando el canal de radiofrecuencia 3034. Corresponde a la banda E-GSM, para tecnología UMTS FDD, en la banda 8, a una frecuencia de 901.8 MHz para el enlace ascendente y de 946.8 MHz para el descendente. Tiene un ancho de banda de 5 MHz.

- **Conexiones con tecnología LTE**

Al igual que en las conexiones con UMTS, se puede comprobar que el operador encargado de proveer de conectividad inalámbrica de cuarta generación al sistema es Movistar. Los parámetros referentes a la localización de las celdas evidencian que las conexiones α y β se encuentran en la misma área, con código de rastreo 20611 (TAC) y PCI 111. El identificador de la celda es el 51207199.

Se puede comprobar que la tecnología de acceso utilizada es E-UTRA, con LTE FDD en la banda 20 a 800 MHz, con anchos de banda de subida y bajada de 10 MHz. El canal EARFCN del que se está haciendo uso es el 6400, correspondiente a la banda 20, e indica que las frecuencias en los enlaces ascendente y descendente son de 857 MHz y 816 MHz, respectivamente.

- **Conexiones con tecnología 5G NSA**

De la misma manera que con las tecnologías celulares analizadas anteriormente, se puede comprobar que la conexión se realiza a través del operador Movistar. Al estar utilizándose una arquitectura NSA, se distinguen dos celdas diferentes: una celda LTE y otra celda NR5G-NSA. Puede comprobarse que se está utilizando, como era de esperar, la tecnología de acceso EN-DC.

El identificador de la celda LTE a la que se está enlazado en ambas ubicaciones difiere del de aquella a la que se conectó el UE en la conexión LTE, tratándose del CellID 51207199. No obstante, la celda está ubicada en la misma localización., puesto que los valores de los parámetros TAC y PCI son idénticos a los obtenidos anteriormente.

Se utiliza, al igual que en LTE, el canal EARFCN 6400, en la banda 20, con 857 MHz en el enlace ascendente y 816 en el descendente y anchos de banda de subida y bajada de 10 MHz.

La celda NR5G-NSA, por otra parte, con identificador PCI 284, utiliza el canal NRARFCN 433450, correspondiente a la banda n1, denominada IMT-2000, trabajando a una frecuencia de 1977.25 MHz en el enlace ascendente y de 2167.25 MHz en el descendente. Se trata de una banda utilizada para 5G-DSS, donde se comparte el espectro entre LTE y 5G.

4.4. Calidad de la señal

Para determinar la calidad de la señal de las diferentes tecnologías analizadas, en los distintos escenarios contemplados, se tienen en cuenta los siguientes parámetros:

- RSCP, *Received Signal Code Power*. Potencia recibida en un código.
- Ec/Io, *Energy per chip to Interference power ratio*. Relación de energía de chip por nivel de interferencia.
- RSSI, *Received Signal Strength Indicator*. Indicador de fuerza de la señal recibida. Depende de la distancia, cobertura, interferencias y congestión de la red.
- RSRP, *Reference Signal Received Power*. Potencia recibida de la señal de referencia. Promedio de todas las señales que llegan a la celda. Ligada a la velocidad de la banda.
- RSRQ, *Reference Signal Received Quality*. Calidad de la señal de referencia recibida.
- SINR, *Signal to Interference and Noise Ratio*. Relación de la señal a ruido más interferencia.

Son obtenidos gracias a ciertos comandos que proporcionan información acerca de la señal, como *--nas-get-signal-info* o el comando *AT+QENG="servingcell"*, ya mencionado anteriormente. Para valorar cualitativamente las conexiones, se han tenido como referencia las recomendaciones de fuerza de señal móvil, válidos para módulos *Quectel*, tratadas en [49].

En la Tabla 4.5.a, se observa cómo la potencia de la señal obtenida con UMTS en el escenario β es, en general, bastante pobre; a diferencia de la hallada en α, que, sin llegar a tener un rendimiento excelente, ofrece unos resultados notablemente mejores.

Con respecto a la tecnología de cuarta generación, mostrándose en la Tabla 4.5.b, destaca la excelente intensidad de la señal recibida en α, obteniendo una señal idónea, con un nivel de potencia óptimo, que debería suponer alcanzar unas tasas de transferencia muy estables. En β, sin embargo, las potencias obtenidas son de valores más bajos. En ambas ubicaciones, no son excelentes los niveles de calidad y la relación con el ruido es bastante mala.

Tabla 4.5: Parámetros de calidad de la señal con las diferentes tecnologías analizadas. Medidas tomadas en una ubicación, α, con buena visión a la estación base; y en otra, β, que ofrece peores prestaciones. Elaboración propia.

a) Parámetros de calidad en UMTS.

Parámetro	*α*	*β*
RSCP	-74 dBm	-97 dBm
RSSI	-76 dBm	-97 dBm
Ec/Io	-7 dBm	-11 dBm

b) Parámetros de calidad en LTE.

Parámetro	*α*	*β*
RSRP	-87 dBm	-110 dBm
RSRQ	-14 dB	-12 dB
RSSI	-55 dBm	-77 dBm
SINR	4,4 dB	4,2 dB

c) Parámetros de calidad en 5G-NSA.

1) Celda LTE

Parámetro	*α*	*β*
RSRP	-87 dBm	-111 dBm
RSRQ	-15 dB	-12 dB
RSSI	-55 dBm	-78 dBm
SINR	5,8 dB	2 dB

2) Celda NR5G-NSA

Parámetro	*α*	*β*
RSRP	-95 dBm	-121 dBm
RSRQ	-11 dB	-12 dB
SINR	13 dB	6,5 dB

La Tabla 4.5.c está dividida, a su vez, en dos Tablas, 4.5.c.1 y 4.5.c.2, para las celdas LTE y NR5G-NSA, respectivamente, que dan servicio al UE. Los parámetros de calidad de la señal LTE son muy semejantes a los obtenidos para la conexión utilizando únicamente la tecnología de cuarta generación. Estos resultados son coherentes con lo analizado anteriormente, puesto que se verificó que en ambas conexiones se utilizaba la misma celda LTE.

En la celda NR5G-NSA, destaca la diferencia entre los valores obtenidos en las ubicaciones α y β. Pese a que en ninguna de ellas se obtengan resultados óptimos, es palpable la mejoría que ofrece la conexión en α respecto a β. En esta segunda localización, la potencia de la señal es prácticamente nula, afectando en la inestabilidad de la señal. La relación de la señal con las interferencias en α es notable, a diferencia de la resultada en el segundo escenario. Además, en ambos escenarios se obtiene una señal con bastante calidad.

4.5. Medidas del rendimiento de las conexiones

Tal y como se mencionó en el Apartado 3.2.5., estableciendo una conexión con el *broker* en el *Edge* y suscribiéndose al *topic /leap_motion/+/ping*, se pueden obtener los tiempos de ida y vuelta asociados a la transmisión MQTT, generados con el módulo implementado, *Data Conn*. Se utiliza, para ello, un cliente Mosquitto de línea de comandos.

```
$ sudo mosquitto_sub -t /leap_motion/+/ping -h 193.***.***.*** -v
```

Se realizan también medidas de las tasas de transmisión con las diferentes tecnologías, mediante el comando *iperf3*, sobre el servidor de Ikerlan. Mide los datos transmitidos por varias conexiones TCP, con un número de clientes variable, en un tiempo concreto, y calcula el caudal de información útil con el cociente entre el mensaje enviado y el tiempo invertido en cada transmisión. Se realizan pruebas en los sentidos UL y DL, para obtener las tasas generadas en la subida y descarga de datos.

```
$ iperf3 -c 193.***.***.*** -p 1883 -f m -P <#CLIENTES> -t <TIEMPO>
$ iperf3 -c 193.***.***.*** -p 1883 -R -f m -P <#CLIENTES> -t <TIEMPO>
```

Seguidamente, se realiza un análisis de estas latencias y caudal útil asociados a cada tecnología celular (UMTS, LTE, 5G-NSA), en los escenarios referidos previamente, α y β. Los resultados han sido graficados con la plataforma de programación y cálculo numérico, *MATLAB*.

4.5.1. Escenario α

Las Figuras 4.2 y 4.3 reflejan el comportamiento de cada tecnología en términos de latencia. En la primera de ellas, se observan todas las muestras obtenidas, con su latencia asociada (Figuras 4.2.a, 4.2.c, 4.2.e), así como las funciones de densidad de probabilidad de dichas latencias (Figuras 4.2.b, 4.2.d, 4.2.f). Con el diagrama estadístico de la Figura 4.3, se muestra la dispersión de las latencias de una manera más precisa.

Se observa que, para la conexión UMTS, los retardos tienen una varianza muy elevada, llegándose a superar, en algunos casos, los 400 ms. La media de las latencias no es, por tanto, muy representativa para caracterizar esta conexión. En cambio, las latencias asociadas a las tecnologías de cuarta y quinta generaciones tienen una dispersión mucho menor, demostrando un comportamiento más estable, rondando los 58 ms, en ambos casos.

Cabe destacar que el comportamiento semejante que muestran estas dos últimas tecnologías celulares es debido a que podrían estar compartiendo el espectro, al estar utilizando la arquitectura 5G-DSS.

Pese a obtenerse algún valor atípico, por encima de los 100 ms, los resultados para LTE y 5G-NSA reflejan un rendimiento con una alta calidad, en comparación con el de la tecnología de tercera generación.

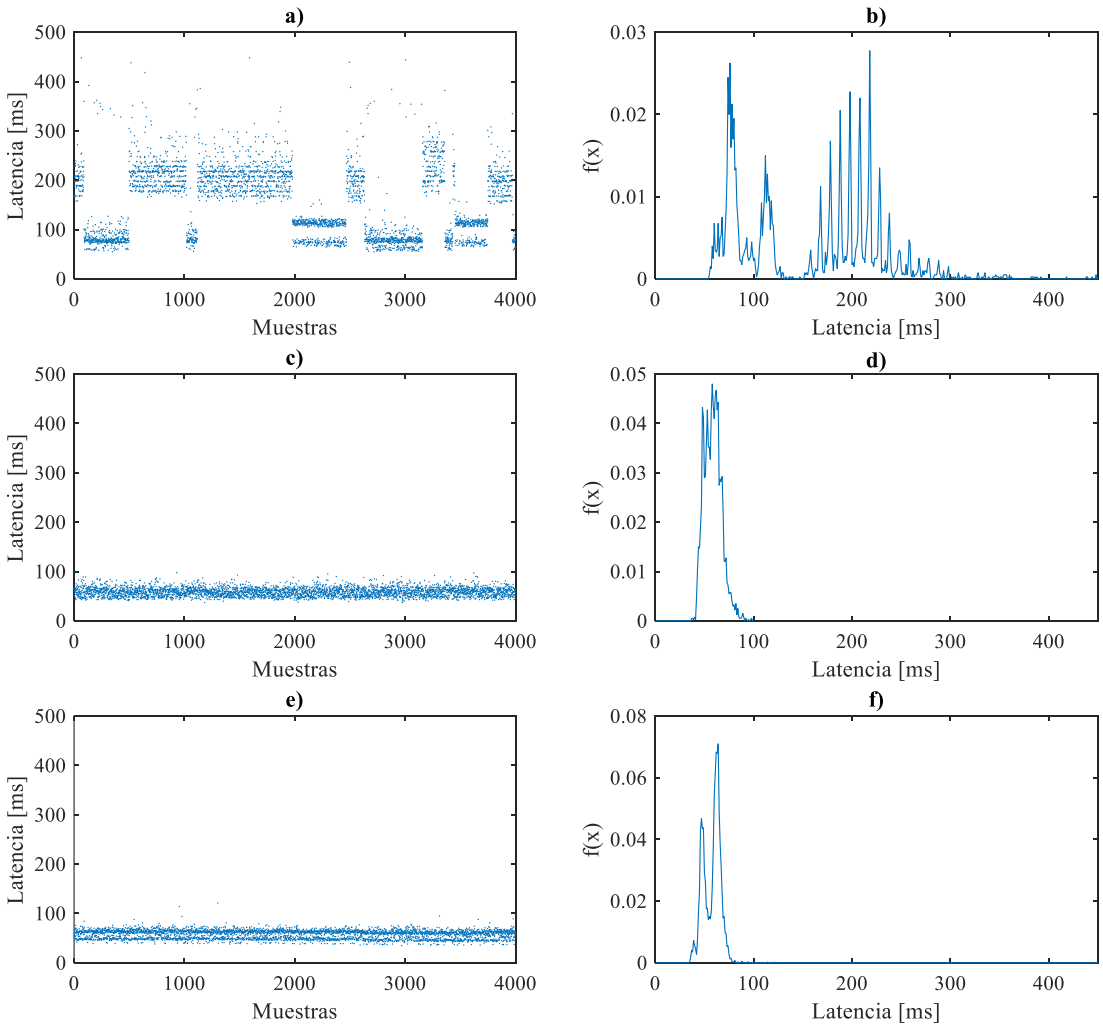


Figura 4.2. Latencias y funciones de densidad de probabilidad asociadas a las tecnologías celulares analizadas en el escenario α: UMTS (a, b), LTE (c, d) y 5G-NSA (e, f). Aparecen representadas 4000 muestras para cada tecnología, tomadas cada segundo, con el tiempo de ida y vuelta obtenido en ese instante de tiempo, en ms. Elaboración propia.

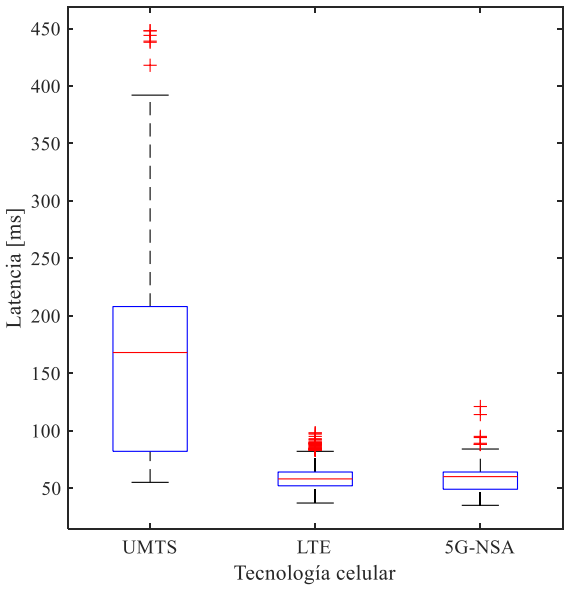


Figura 4.3. Dispersión de latencia de las tecnologías celulares analizadas en el escenario α: UMTS, LTE y 5G-NSA.

Analizando el caudal asociado a las tres tecnologías, visto en la Figura 4.4, se observa la mejora intergeneracional. En 5G-NSA, pese a tener una dispersión mucho mayor, el caudal llega a alcanzar tasas de bajada hasta cinco veces mayores que con LTE en el enlace descendente (Figura 4.4.a). En el ascendente, destaca la mejora en el rendimiento asociado a LTE, acercándose a los valores alcanzados por la tecnología de quinta generación. Las tasas obtenidas tanto con UMTS, como con 5G-NSA, son muy simulares a las del enlace anterior, disminuyéndose ligeramente la dispersión de ambas.

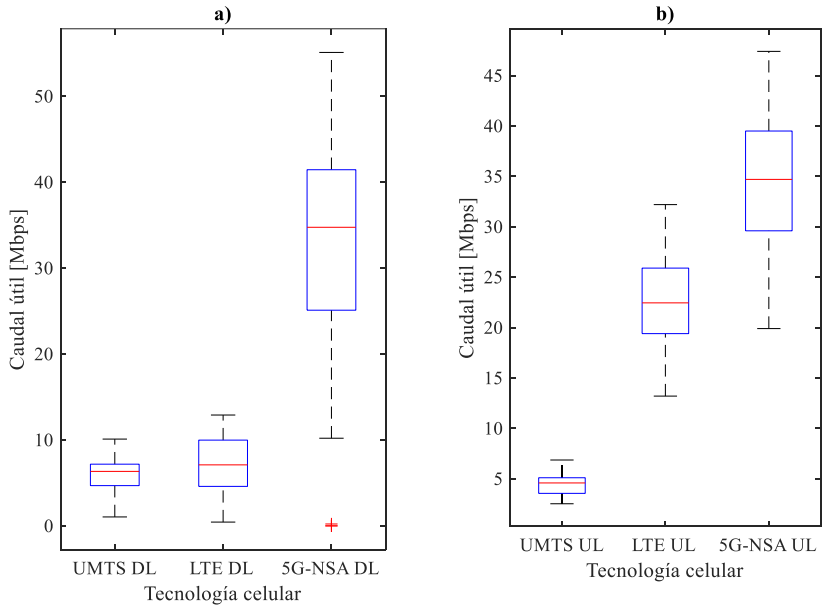


Figura 4.4. Dispersión del caudal útil en sentidos *downlink* (a) y *uplink* (b) de las tecnologías celulares analizadas en el escenario α: UMTS, LTE y 5G-NSA.

4.5.2. Escenario β

Vistos los parámetros de calidad analizados en el Apartado 4.4., en las medidas tomadas en el escenario β se espera un rendimiento más bajo que en la primera ubicación, especialmente en la tecnología de quinta generación, donde la señal resultaba muy inestable.

Esto es ratificado con el comportamiento de las tecnologías celulares, en términos de latencia, en este segundo escenario, mostrado en las Figuras 4.5 y 4.6. Se muestran tres picos de latencias que podrían estar relacionadas con la existencia de paquetes reenviados, por la baja cobertura: las latencias más bajas, semejantes a las obtenidas con LTE, serían las que se obtienen en ciertos momentos con mejor cobertura; el siguiente pico de latencias podría deberse a una retransmisión, con una dispersión elevada; y la tercera zona, con latencias más altas, correspondería a una segunda retransmisión, con una dispersión aún mayor. La distancia entre los picos se puede deber a una implementación de 5G no optimizada. Un mensaje de *ping* ocupa tan solo un paquete, y su pérdida se detecta con un tiempo de espera. En una fase inicial del despliegue 5G este tiempo de espera, como muchos otros parámetros, podría necesitar ajustes.

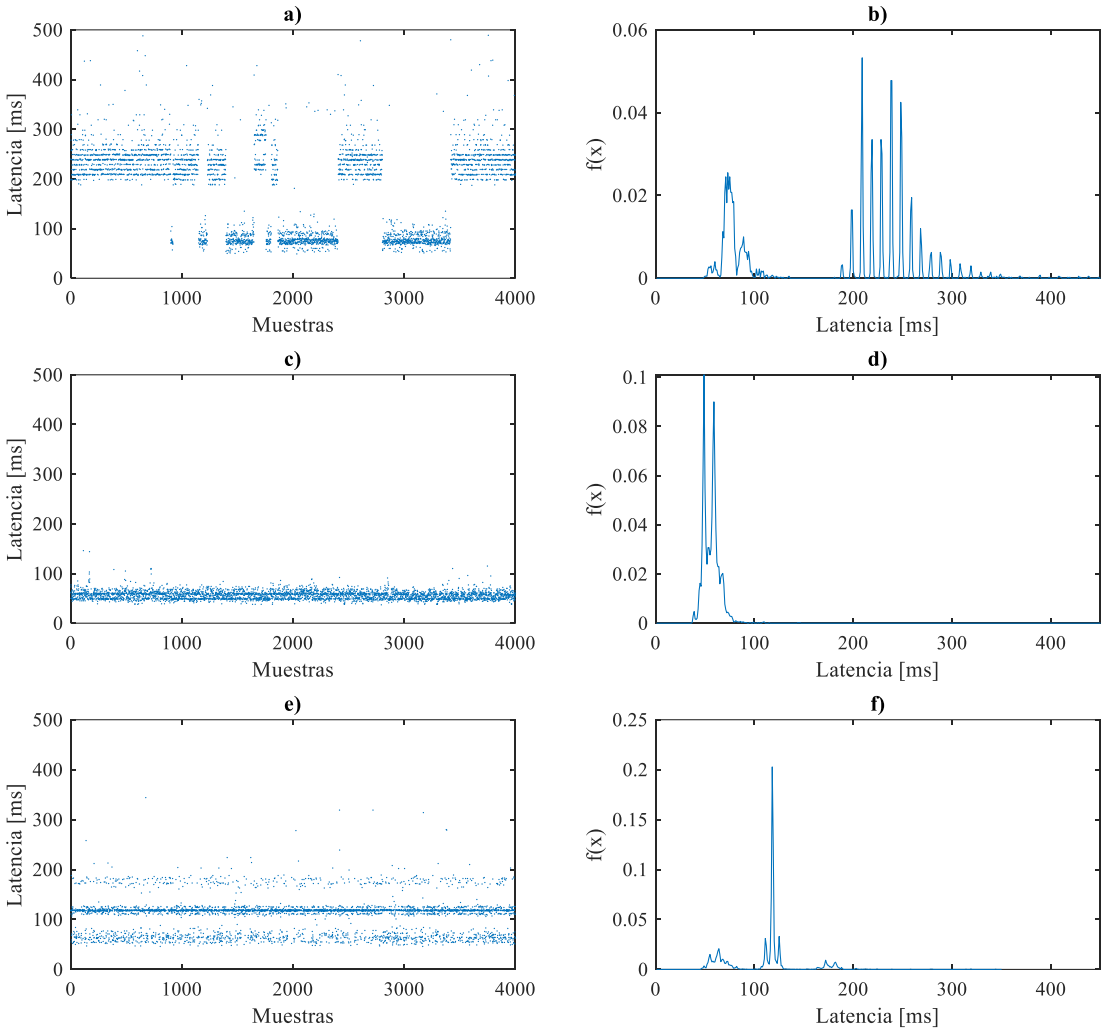


Figura 4.5. Latencias y funciones de densidad de probabilidad asociadas a las tecnologías celulares analizadas en el escenario β: UMTS (a, b), LTE (c, d) y 5G-NSA (e, f). Aparecen representadas 4000 muestras para cada tecnología, tomadas cada segundo, con el tiempo de ida y vuelta obtenido en ese instante de tiempo, en ms. Elaboración propia.

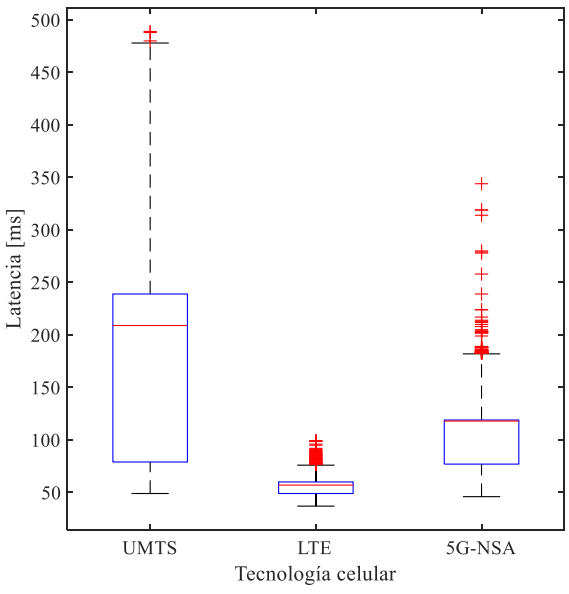


Figura 4.6. Dispersión de latencia de las tecnologías celulares analizadas en el escenario β: UMTS, LTE y 5G-NSA.

La Figura 4.7.a muestra, sin embargo, cómo la tecnología celular con mayor caudal en el enlace de bajada sigue siendo 5G-NSA. Aunque las tasas son, para las tres tecnologías, mucho menores que las obtenidas en α, el rendimiento de las conexiones en este escenario es muy similar al del anterior. En la Figura 4.7.b, correspondiente al enlace UL, se observa, en cambio, el detrimento de 5G-NSA frente a LTE, obteniéndose un caudal útil mucho menor para la tecnología de quinta generación, si bien sigue siendo altamente superior al asociado a UMTS.

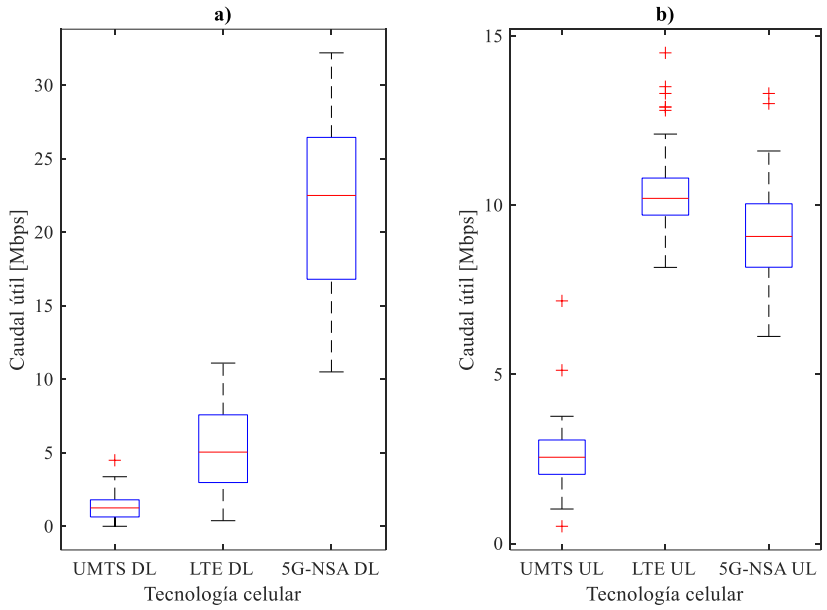


Figura 4.7. Dispersión del caudal útil en sentidos *downlink* (a) y *uplink* (b) de las tecnologías celulares analizadas en el escenario β: UMTS, LTE y 5G-NSA.

Capítulo 5

Conclusiones y líneas futuras

En este último capítulo se detallan las principales conclusiones que se han obtenido a lo largo del desarrollo del Trabajo. Se presentan también las diferentes líneas de investigación que se abren, en base al estudio realizado.

5.1. Conclusiones

Vistos los resultados analizados, se puede concluir que, utilizando la infraestructura pública desplegada actualmente, aún es temprano para obtener medidas que reflejen unas mejoras concluyentes de las prestaciones de la tecnología 5G, respecto a las generaciones anteriores.

A pesar de ello, todo parece indicar que se está avanzando en el sentido correcto. Con los despliegues actuales de las tecnologías 5G-NSA y 5G-DSS, que permiten compartir recursos entre LTE y NR, se facilitará la migración hacia el 5G-SA, reduciendo notablemente los costes.

Probablemente, cuando se despliegue y comercialice dicha arquitectura y se subaste la banda milimétrica (*mmWave*), pueda obtenerse un rendimiento óptimo de la infraestructura pública 5G, disminuyéndose notablemente las latencias y tasas de transferencia cuando se utilice tal tecnología.

No obstante, como se expone en [50], el porcentaje más alto del tiempo de ida y vuelta transcurre en la troncal de Internet, afectando de una manera ínfima el núcleo de la red móvil y la red de acceso radio. Por tanto, aunque en otras implementaciones y ubicaciones quizás podrían conseguirse resultados mejores, será utilizando una estación base privada, dentro de una red interna, cuando pueda llegarse a esas latencias del orden de 1 ms que se prometen con la tecnología de última generación.

Pese a esto, la conexión que se ha establecido en las ubicaciones α y β es suficientemente ágil para el buen desempeño de la aplicación implementada, confirmando su operabilidad y utilidad para múltiples funcionalidades, en el momento actual. Además, al estar desarrollada de forma que la conexión pueda ser establecida a través del módem, con cualquier tecnología que este permita, la arquitectura 5G-SA podrá utilizarse en cuanto esté operativa.

5.2. Líneas futuras

Tras las conclusiones obtenidas, se considera oportuno seguir dentro de este marco de investigación, de modo que se puedan comprobar los comportamientos del sistema bajo diferentes enfoques. A continuación, se detallan distintas líneas de investigación que han quedado abiertas y que se estima que podrían resultar de interés:

- Contenedores *Docker*. Incluir todos los servicios que han de ejecutarse en una misma máquina en un contenedor *Docker*, para automatizar el despliegue de todos los componentes del sistema.

- Implementación sobre QUIC. De un tiempo a esta parte, se ha venido implementando MQTT sobre el protocolo QUIC, ofreciendo un muy buen rendimiento [31]. Su uso es muy ventajoso, entre otros aspectos, para obtener latencias extremadamente bajas, por lo que la utilización de esta dupla de protocolos puede resultar muy atractiva para establecer comunicaciones similares en futuras propuestas. El módulo desarrollado en el proyecto ha sido programado en GO por las facilidades que ofrece este lenguaje para implementaciones de QUIC.

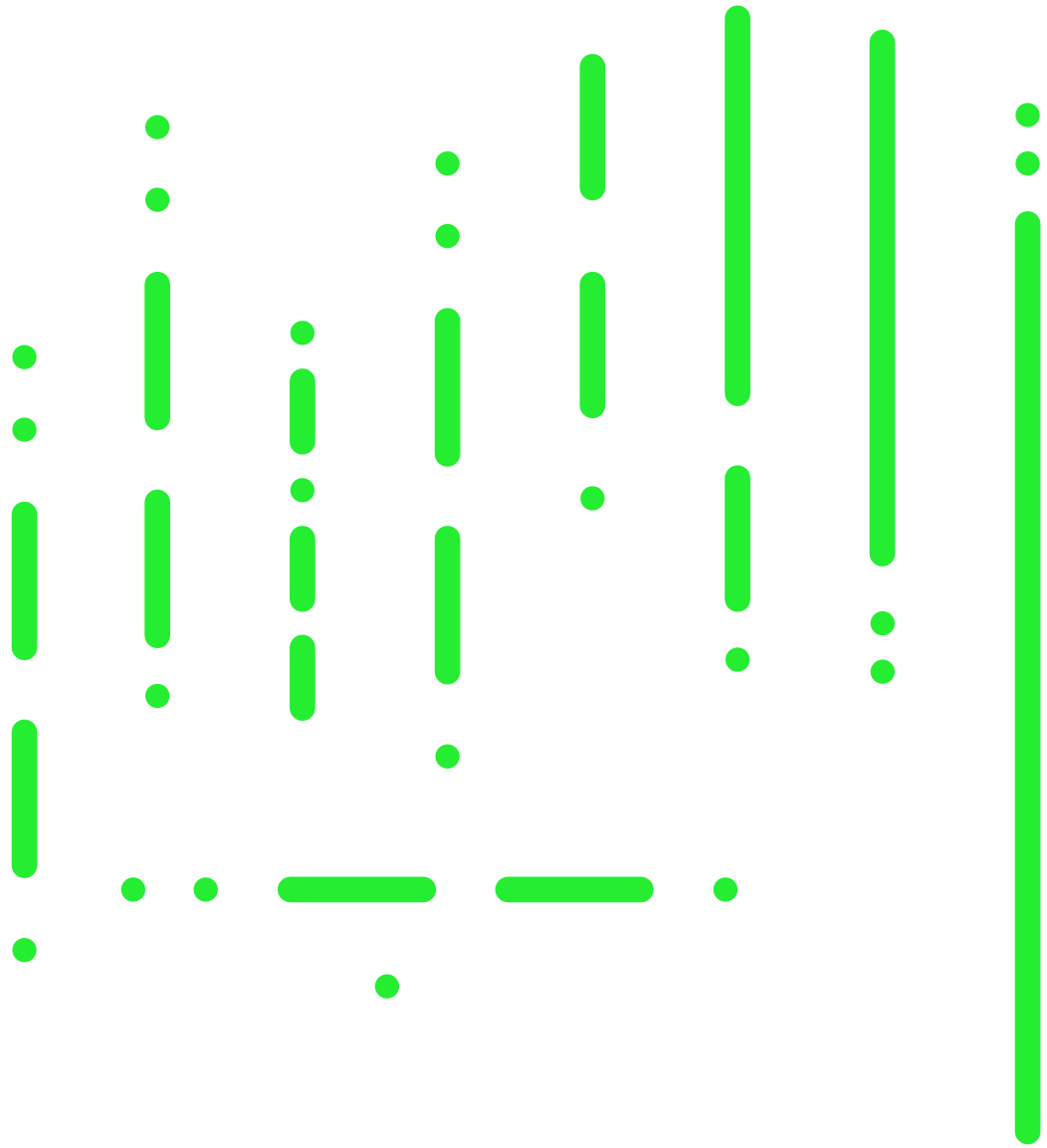
- Pruebas con protocolos de seguridad. La seguridad es cada vez más necesaria en las comunicaciones. Por tanto, será interesante dotar de seguridad al sistema, cifrando la información transmitida a lo largo de toda la sesión con los protocolos SSL/TLS, y comprobar cómo afecta a la comunicación en términos de latencia en el rendimiento de la aplicación.

- Pruebas con 5G-SA. Realizar pruebas del sistema con conectividad 5G *Stand Alone*, estableciendo una conexión entre el modem y una estación base 5G en que dicho modo esté habilitado, puede ser otro punto de interés para futuros estudios.

- Rendimiento sobre una solución de conectividad 5G privada. Evaluar las prestaciones de la tecnología 5G dentro de un entorno privado y analizar las mejoras que ofrece, respecto a la utilización de la infraestructura pública.

- Otros escenarios de despliegue. El sistema realizado puede tener numerosas aplicaciones inmersivas en el ámbito del entretenimiento, con contenidos de realidad virtual o realidad aumentada, así como en educación y terapia. De cara a una futura implementación de un brazo robótico conectado al servidor en el borde de la red, puede resultar muy beneficioso en terrenos como la Salud o en procesos industriales, ya que cumple con el nivel de precisión exigido en estos entornos.

Bibliografía



[1] I. Policy and T. Watch Division, “The Tactile Internet,” Aug. 2014, Accessed: Jul. 03, 2022. [Online]. Available: http://www.itu.int/ITU-T/techwatch

[2] Universidad Internacional de Valencia, “Evolución de la red de comunicación móvil, del 1G al 5G | VIU,” 2018. https://www.universidadviu.com/int/actualidad/nuestros-expertos/evolucion-de-la-red-de-comunicacion-movil-del-1g-al-5g (accessed Jun. 13, 2022).

[3] “3GPP specification Release version matrix.” https://www.3gpp.org/DynaReport/SpecReleaseMatrix.htm (accessed Jun. 13, 2022).

[4] M. Hermann, T. Pentek, and B. Otto, “Design Principles for Industrie 4.0 Scenarios,” in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan. 2016, pp. 3928–3937. doi: 10.1109/HICSS.2016.488.

[5] Thales Group, “Presentando la tecnología y redes 5G (definición, características, 5G vs 4G y casos de uso),” Nov. 14, 2019. https://www.thalesgroup.com/es/countries/americas/latin-america/dis/movil/inspiracion/5g (accessed May 30, 2022).

[6] European Telecommunications Standard Institute, “Release 15 (3GPP TR 21.915 version 15.0.0 Release 15),” Oct. 2019. Accessed: Jun. 14, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/121900_121999/121915/15.00.00_60/tr_121915v150000p.pdf

[7] J. C. López, “Banda milimétrica (mmWave): qué es y por qué este es el 5G tan rápido como la fibra óptica que nos han prometido las operadoras,” Mar. 19, 2020. https://www.xataka.com/moviles/banda-milimetrica-mmwave-que-que-este-5g-rapido-como-fibra-optica-que-nos-han-prometido-operadoras (accessed May 30, 2022).

[8] Mathworks, “5G Toolbox Simulación, análisis y pruebas de sistemas de comunicaciones 5G.” https://www.mathworks.com/products/5g.html (accessed May 30, 2022).

[9] European Telecommunications Standard Institute, “Release 16 (3GPP TR 21.916 version 16.0.0 Release 16),” 2021.

[10] I. Rahman *et al.*, “5G evolution toward 5G Advanced: an overview of 3GPP releases 17 and 18,” *Ericsson technology review*, Oct. 13, 2021.

[11] Qualcomm Technologies, “5 key technology inventions in 5G NR Release 17,” Apr. 13, 2022. https://www.qualcomm.com/news/onq/2022/04/5-key-technology-inventions-5g-nr-release-17 (accessed Jul. 23, 2022).

[12] T. Niwa, “Cellular IoT Module Market: Strong Demand, Supply Constraint, Possible slowdown in 2023, LTE Cat.1 rises,” *IoT Business News*, Feb. 04, 2022.

[13] Quectel Wireless Solutions Co., “RG50xQ&RM5xxQ Series AT Commands Manual 5G Module Series,” Nov. 2020, Accessed: Jun. 14, 2022. [Online]. Available: www.quectel.com

[14] European Telecommunications Standard Institute, “AT command set for User Equipment (UE) (3GPP TS 27.007 version 16.11.0 Release 16),” Apr. 2022, Accessed: Jun. 14, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/127000_127099/127007/16.11.00_60/ts_127007v161100p.pdf

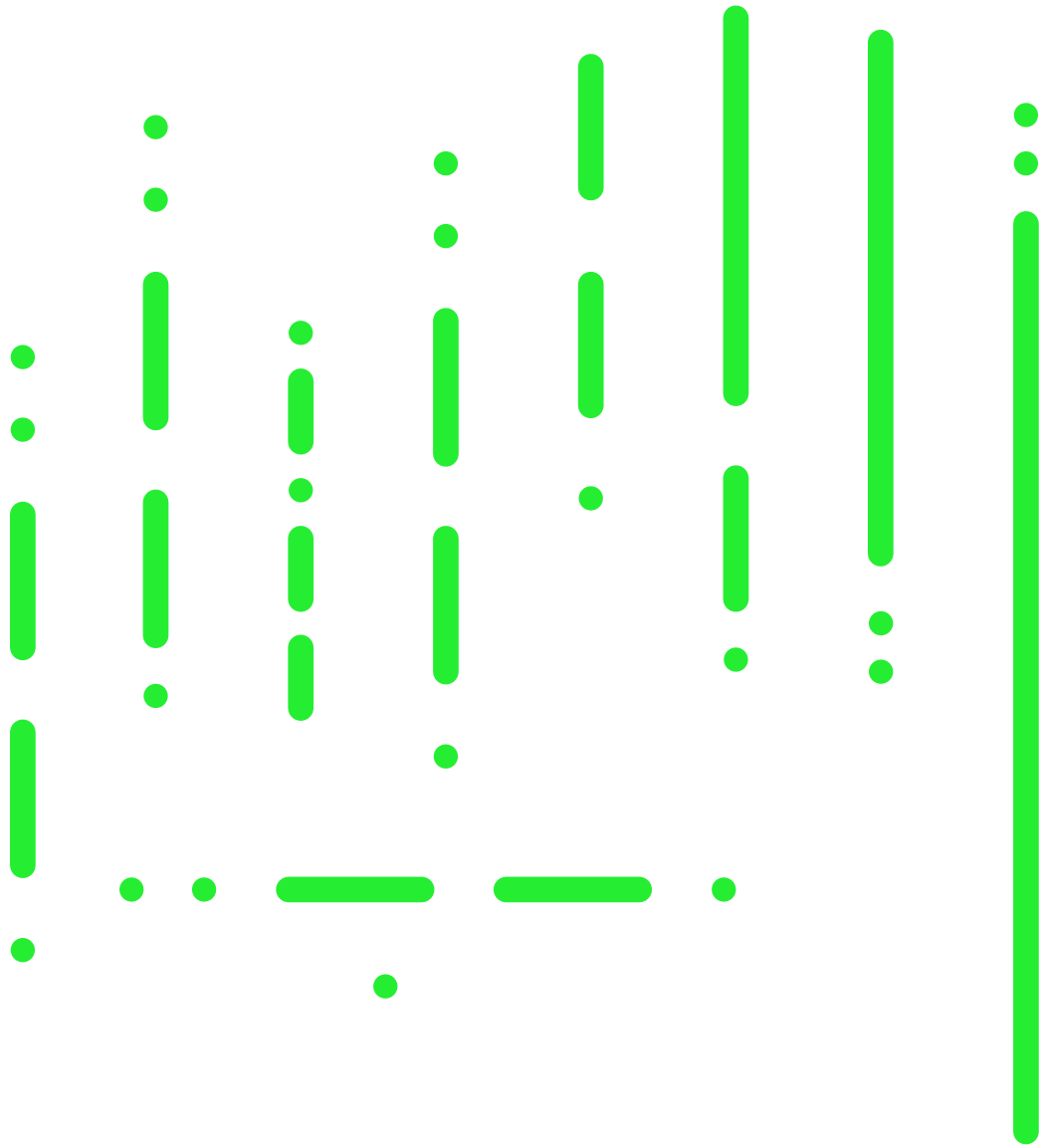
[15] A. Morgado, “Qualcomm Gobi devices in Linux based systems,” 2013, Accessed: May 31, 2022. [Online]. Available: http://www.3gpp.org/DynaReport/27007.htm

[16] A. Banks and R. Gupta, “MQTT Version 3.1.1,” *OASIS standard*, vol. 29, p. 89, 2014.

[17] "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0," *OASIS Standard*, Oct. 2012, Accessed: May 31, 2022. [Online]. Available: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf
[18] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," Jun. 2014. doi: 10.17487/rfc7252.
[19] J. Postel, "Transmission Control Protocol," Sep. 1981. doi: 10.17487/rfc0793.
[20] J. Postel, "Internet Protocol," Sep. 1981. doi: 10.17487/rfc0791.
[21] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," Jan. 1999. doi: 10.17487/rfc2246.
[22] "Introducción al protocolo AMQP." https://es.slideshare.net/GAMALIEL22MX/presentacion-amqp (accessed Jun. 13, 2022).
[23] "AMQP - the Advanced Message Queuing Protocol - CloudAMQP." https://www.cloudamqp.com/docs/amqp.html (accessed Jun. 13, 2022).
[24] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0," May 1996. doi: 10.17487/rfc1945.
[25] J. Postel, "User Datagram Protocol," Aug. 1980, doi: 10.17487/RFC0768.
[26] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security," Apr. 2006. doi: 10.17487/rfc4347.
[27] R. Gour, "4 Major IoT Protocols — MQTT, CoAP, AMQP, DDS | by Rinu Gour | Medium," 2018. https://medium.com/@rinu.gour123/4-major-iot-protocols-mqtt-coap-amqp-dds-46016897c3e9 (accessed Jun. 13, 2022).
[28] A. Venčkauskas, N. Morkevicius, V. Jukavičius, R. Damaševičius, J. Toldinas, and Š. Grigaliūnas, "An Edge-Fog Secure Self-Authenticable Data Transfer Protocol," *Sensors*, Aug. 2019, doi: 10.3390/s19163612.
[29] "MQTT vs CoAP, the battle to become the best IoT protocol," Oct. 21, 2019. https://www.pickdata.net/es/noticias/mqtt-vs-coap-mejor-protocolo-iot (accessed Jun. 13, 2022).
[30] "QUIC: A UDP-Based Multiplexed and Secure Transport," May 2021. doi: 10.17487/RFC9000.
[31] F. Fernandez, M. Zverev, P. Garrido, J. R. Juarez, J. Bilbao, and R. Aguero, "And QUIC meets IoT: performance assessment of MQTT over QUIC," in *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2020, pp. 1–6. doi: 10.1109/WiMob50308.2020.9253384.
[32] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Aug. 2018, doi: 10.17487/RFC8446.
[33] F. Fernández, M. Zverev, P. Garrido, J. R. Juárez, J. Bilbao, and R. Agüero, "Even Lower Latency in IIoT: Evaluation of QUIC in Industrial IoT Scenarios †," vol. 2020, 2021, doi: 10.3390/s21175737.
[34] "QUIC Loss Detection and Congestion Control," May 2021. doi: 10.17487/RFC9002.
[35] E. Sy, C. Burkert, H. Federrath, and M. Fischer, "A QUIC Look at Web Tracking," *Proceedings on Privacy Enhancing Technologies*, Jul. 2019, doi: 10.2478/popets-2019-0046.
[36] Ultraleap, "Tracking | Leap Motion Controller | Ultraleap." https://www.ultraleap.com/product/leap-motion-controller/ (accessed Jun. 01, 2022).

[37] Ultraleap, "Leap Motion Controller Datasheet." https://www.ultraleap.com/datasheets/Leap_Motion_Controller_Datasheet.pdf (accessed Jun. 01, 2022).
[38] Quectel, "5G EVB User Guide 5G/LTE-A Module Series Version: 1.2," 2021, Accessed: Jun. 02, 2022. [Online]. Available: https://www.quectel.com/wp-content/uploads/2021/04/Quectel_5G_EVB_User_Guide_V1.2.pdf
[39] Quectel, "Quectel RG50xQ Series," 2021, Accessed: Jun. 02, 2022. [Online]. Available: https://www.quectel.com/wp-content/uploads/2021/09/Quectel_RG50xQ_Series_5G_Specification_V1.5.pdf
[40] "Eclipse Mosquitto." https://mosquitto.org/ (accessed Jun. 07, 2022).
[41] "EMQX: Open-Source, Cloud-Native MQTT Broker for IoT." https://www.emqx.io/ (accessed Jun. 13, 2022).
[42] M. Barón, "Miguel Barón / eval5G_2022_miguelbaron · GitLab TLMAT UC," 2022. https://gitlab.tlmat.unican.es/mbaron/eval5g_2022_miguelbaron (accessed Jul. 26, 2022).
[43] "MQTT X: Cross-platform MQTT 5.0 Desktop Client." https://mqttx.app/ (accessed Jun. 07, 2022).
[44] Quectel Wireless Solutions Co., "5G EVB User Guide," Sep. 2019.
[45] Quectel Wireless Solutions Co., "RG50xQ&RM5xxQ Series AT Commands Manual 5G Module Series," Nov. 2020, Accessed: Jun. 14, 2022. [Online]. Available: www.quectel.com
[46] M. de A. E. y T. D. España, *Orden ETD/1449/2021, de 16 de diciembre, por la que se aprueba el Cuadro Nacional de Atribución de Frecuencias.* 2021. Accessed: Jul. 04, 2022. [Online]. Available: https://www.boe.es
[47] C. Valero, "Frecuencias y bandas 2G, 3G, 4G y 5G en España - operadores móviles," 2022. https://www.adslzone.net/operadores/en-detalle/frecuencias-moviles-espana/ (accessed Jul. 04, 2022).
[48] "CellTower Locator." http://www.cell2gps.com/ (accessed Jul. 23, 2022).
[49] Teltonika Networks, "Mobile Signal Strength Recommendations - Teltonika Networks Wiki," 2022. https://wiki.teltonika-networks.com/view/Mobile_Signal_Strength_Recommendations (accessed Jul. 20, 2022).
[50] B. Coll-Perales *et al.*, "End-to-End V2X Latency Modeling and Analysis in 5G Networks," Jan. 2022, doi: 10.48550/arxiv.2201.06082.

Anexos



Anexo A

Topic: /leap_motion/1/conf QoS: 0

```
{
  "tcset": {
    "delay": "",
    "loss": "",
    "delay_distro": "",
    "rate": "",
    "duplicate": "",
    "reordering": "",
    "corrupt": "",
    "device": ""
  },
  "qmi": {
    "set": "[UMTS|LTE|5GNR]"
  }
}
```

Figura A.1: Estructura de JSON de configuración de la red y del módem. Capturado con MQTT X [43].

Topic: /leap_motion/1/ping QoS: 0

```
{
  "rtt": 49,
  "time": 1656492397374724900
}
```

Figura A.2: Ejemplo de JSON de monitorización del ping. Capturado con MQTT X [43].

Topic: /leap_motion/1/status QoS: 0


```
{
  "status": {
    "manufacturer": "Quectel",
    "model": "RG500QEA_VH",
    "band-capabilities": {
      "bands": "wcdma-2100, wcdma-dcs-1800, wcdma-850-us, wcdma-800, wcdma-900, wcdma-1700-japan, w
cdma-850-japan",
      "lte-bands": "1, 2, 3, 4, 5, 7, 8, 12, 13, 14, 17, 18, 19, 20, 25, 26, 28, 29, 30, 32, 34, 3
8, 39, 40, 41, 42, 43",
      "lte-bands-extended": "1, 2, 3, 4, 5, 7, 8, 12, 13, 14, 17, 18, 19, 20, 25, 26, 28, 29, 30, 3
2, 34, 38, 39, 40, 41, 42, 43, 48, 66, 71"
    },
    "capabilities": {
      "max-tx-channel-rate": "50000000",
      "max-rx-channel-rate": "100000000",
      "data-service": "non-simultaneous-cs-ps",
      "sim": "supported",
      "networks": "umts, lte, 5gnr"
    },
    "operating-mode": {
      "mode": "online",
      "hw-restricted": "no"
    },
    "system-selection-preference": {
      "emergency-mode": "no",
      "mode-preference": "umts, lte, 5gnr",
      "disabled-modes": "none",
      "band-preference": "wcdma-2100, wcdma-dcs-1800, wcdma-850-us, wcdma-800, wcdma-900, wcdma-850
-japan",
      "lte-band-preference": "1, 3, 5, 7, 8, 18, 19, 20, 26, 28, 32, 34, 38, 39, 40, 41, 42, 43",
      "lte-band-preference-extended": "1, 3, 5, 7, 8, 18, 19, 20, 26, 28, 32, 34, 38, 39, 40, 41, 4
2, 43",
      "td-scdma-band-preference": "a, f",
      "roaming-preference": "any",
      "network-selection-preference": "automatic",
      "service-domain-order-preference": "cs-ps",
      "gsm-wcdma-acquisition-order-preference": "automatic",
      "usage-preference": "data-centric",
      "voice-domain-preference": "ps-preferred",
      "registration-restriction": "unrestricted",
      "acquisition-order-preference": "lte, 5gnr, umts"
    },
    "packet-service-status": "connected",
    "packet-statistics": {
      "tx-packets": "1733",
      "rx-packets": "1265",
      "tx-packets-dropped": "0",
      "rx-packets-dropped": "0",
      "tx-bytes-ok": "255545",
      "rx-bytes-ok": "223747"
    },
    "current-data-bearer-technology": {
      "network-type": "3gpp",
      "radio-access-technology": "none"
    },
    "channel-rates": {
      "current-tx-rate": "n/a",
      "current-rx-rate": "n/a",
      "max-tx-rate": "150000000bps",
      "max-rx-rate": "400000000bps"
    }
  }
}
```


Figura A.3: Ejemplo de JSON de monitorización del módem. Capturado con MQTT X [43].

Anexo B

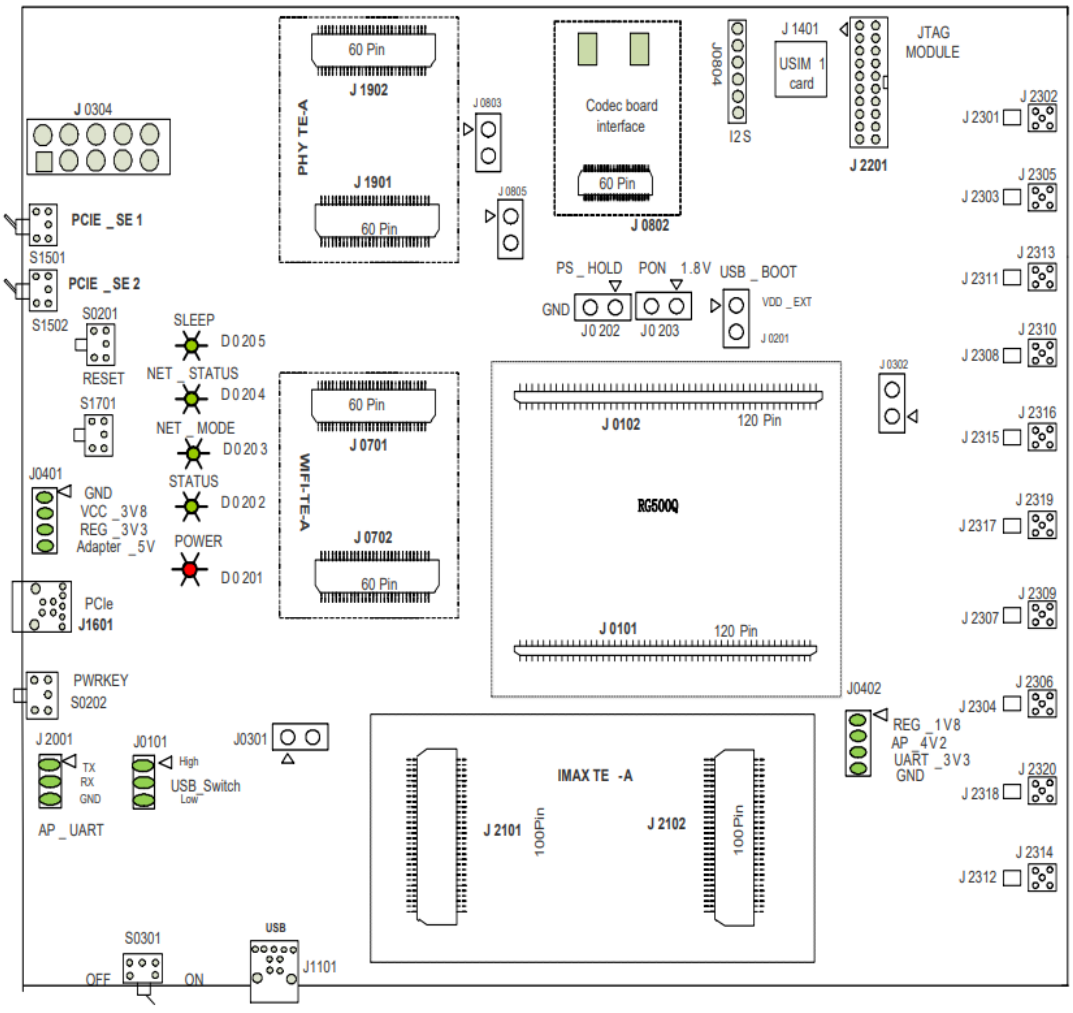


Figura B.1: Plano superior del 5G EVB Kit de *Quectel*.

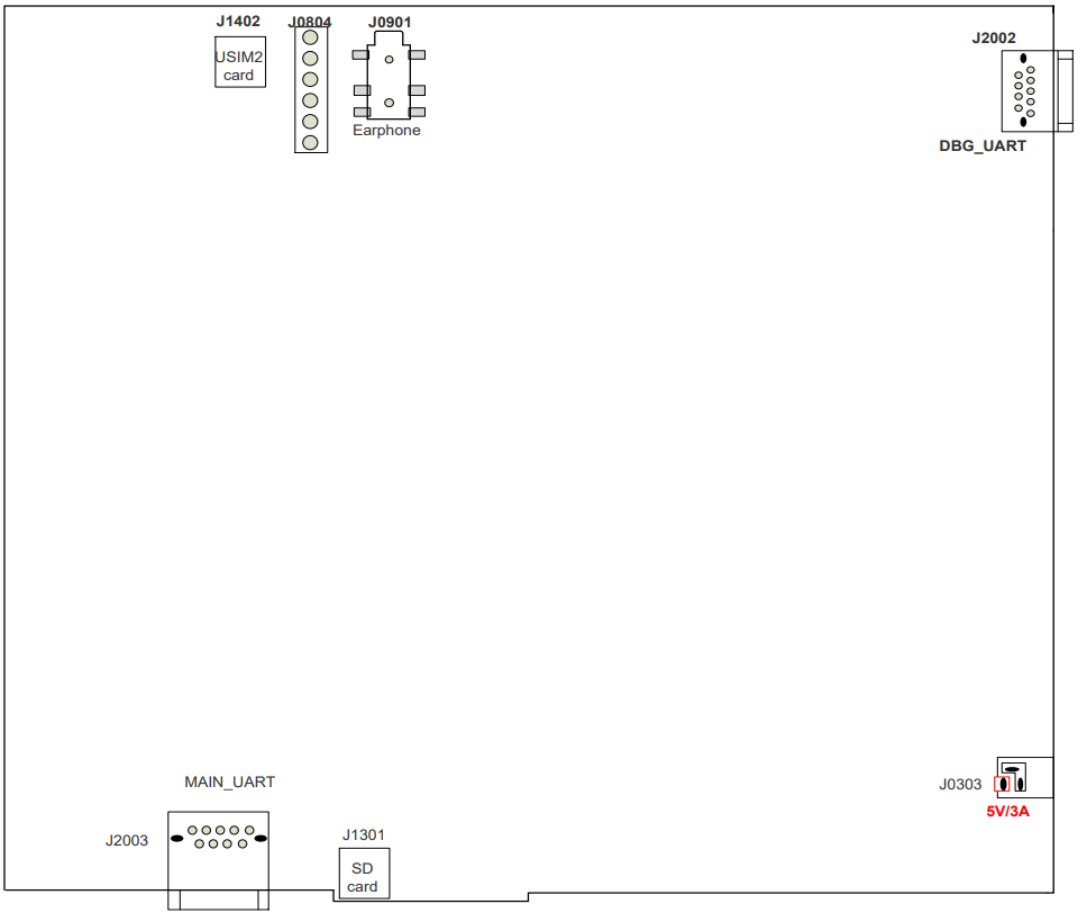


Figura B.2: Plano inferior del 5G EVB Kit de Quectel.

Anexo Código

```
package main
import (
    "fmt"
    "os"
    "github.com/timtadh/getopt"
    "os/signal"
    "os/exec"
    "log"
    "syscall"
    mqtt "github.com/eclipse/paho.mqtt.golang"
    "strconv"
    "strings"
    "time"
    "encoding/json"
)

var HelpMessage string = `
Data Conn version 1.0.0

Data Conn is an MQTT v3.1.1 client.

Usage:
    sudo go run DataConn.go [OPTION...]

Options:
    -h, --help                      display this help
    -i, --id                        set client identifier
                                    default: 1
    --modem                         set modem mode
                                     default: false
    -d, --debug                     debug
                                    default: false
    --src_host=<addr>               set source host address
                                    default: 127.0.0.1
    --src_port=<port>               set source host port number
                                    default: 1883 (no TLS)
    --src_topic=<topic>             set source topic
                                    default: /leap_motion/raw-data
    --dst_host=<addr>               set destination host address
                                    default: 127.0.0.1
    --dst_port=<port>               set destination host port number
                                    default: 1883 (no TLS)
    --dst_data_topic=<topic>        set destination topic of data
                                    default: /leap_motion/{id}/data
    --dst_ping_topic=<topic>        set destination topic of ping
                                    default: /leap_motion/{id}/ping
    --dst_cmd_topic=<topic>         set destination topic of cmd
                                    default: /leap_motion/{id}/cmd
    --dst_status_topic=<topic>      set destination topic of status
                                    default: /leap_motion/{id}/status
`
var Error = "\033[1;31m[ERROR] \033[0m"

type JSON_out struct {
    Status JSON_status `json:"status"`
}

type JSON_in struct {
    Tcset *JSON_tcset `json:"tcset"`
    Qmi *JSON_qmi `json:"qmi"`
```

```
}

type JSON_tcset struct {
    Delay string `json:delay`
    Loss string `json:loss`
    Delay_distro string `json:delay_distro`
    Rate string `json:rate`
    Duplicate string `json:duplicate`
    Reordering string `json:reordering`
    Corrupt string `json:corrupt`
    Device string `json:device`
}

type JSON_qmi struct {
    Set string `json:set`
}

type JSON_status struct {
    Manufacturer string `json:"manufacturer"`
    Model string `json:"model"`
    BandCapabilities S_BandCapabilities `json:"band-capabilities"`
    Capabilities S_Capabilities `json:"capabilities"`
    OperatingMode S_OperatingMode `json:"operating-mode"`
    SystemSelectionPreference S_SystemSelectionPreference `json:"system-
selection-preference"`
    PacketServiceStatus string `json:"packet-service-status"`
    PacketStatistics S_PacketStatistics `json:"packet-statistics"`
    CurrentDataBearerTechnology S_CurrentDataBearerTechnology `json:"current-
data-bearer-technology"`
    ChannelRates S_ChannelRates `json:"channel-rates"`
}

type S_BandCapabilities struct {
    Bands string `json:"bands"`
    LteBands string `json:"lte-bands"`
    LteBandsextended string `json:"lte-bands-extended"`
}

type S_Capabilities struct {
    MaxTxChannelRate string `json:"max-tx-channel-rate"`
    MaxRxChannelRate string `json:"max-rx-channel-rate"`
    DataService string `json:"data-service"`
    Sim string `json:"sim"`
    Networks string `json:"networks"`
}

type S_OperatingMode struct {
    Mode string `json:"mode"`
    HwRestricted string `json:"hw-restricted"`
}

type S_SystemSelectionPreference struct {
    EmergencyMode string `json:"emergency-mode"`
    ModePreference string `json:"mode-preference"`
    DisabledModes string `json:"disabled-modes"`
    BandPreference string `json:"band-preference"`
    LteBandPreference string `json:"lte-band-preference"`
    LteBandPreferenceExtended string `json:"lte-band-preference-extended"`
    TdScdmaBandPreference string `json:"td-scdma-band-preference"`
    RoamingPreference string `json:"roaming-preference"`
    NetworkSelectionPreference string `json:"network-selection-preference"`
    ServiceDomainPreference string `json:"service-domain-order-preference"`
    GsmWcdmaAcquisitionOrderPreference string `json:"gsm-wcdma-acquisition-
order-preference"`
    UsagePreference string `json:"usage-preference"`
    VoiceDomainPreference string `json:"voice-domain-preference"`
    RegistrationRestriction string `json:"registration-restriction"`
    AcquisitionOrderPreference string `json:"acquisition-order-preference"`
```

```
}

type S_PacketStatistics struct {
    TxPacketsOk string `json:"tx-packets"`
    RxPacketsOk string `json:"rx-packets"`
    TxPacketsDropped string `json:"tx-packets-dropped"`
    RxPacketsDropped string `json:"rx-packets-dropped"`
    TxBytesOk string `json:"tx-bytes-ok"`
    RxBytesOk string `json:"rx-bytes-ok"`
}

type S_CurrentDataBearerTechnology struct {
    NetworkType string `json:"network-type"`
    RadioAccessTechnology string `json:"radio-access-technology"`
}

type S_ChannelRates struct {
    CurrentTxRate string `json:"current-tx-rate"`
    CurrentRxRate string `json:"current-rx-rate"`
    MaxTxRate string `json:"max-tx-rate"`
    MaxRxRate string `json:"max-rx-rate"`
}

func execute_qmicli(cmdStr ...string) []byte {

    opts := []string{"--silent", "--device=/dev/cdc-wdm0", "--device-open-
proxy"}
    cmd := exec.Command("qmicli", append(opts, cmdStr...)...)
    // fmt.Println(cmdStr)
    out, err := cmd.Output()
    if err != nil {
        log.Fatal(err)
    }
    return out
 }

func parseResult(res []byte) []string {
    in := string(res)
    num := strings.Count(in, ": ")
    outslice := make([]string, 0)
    slice := strings.Split(in, "")
    x, i := 0, 0
    for i < num {
        newslice := make([]string, 0)
        aux := false
        for slice[x] != "\n" {
            if aux == true {
                if slice[x] != "'" && slice[x] != "\t" && (slice[x] != " " ||
(slice[x] == " " && (slice[x+1] != " " && slice[x-1] != " "))) {
                    newslice = append(newslice, slice[x])
                }
            }
            if (slice[x] == ":") && (slice[x+1] != "\n") {
                aux = true
                x++
            }
            x++
        }
        if aux == true {
            newstring := strings.Join(newslice[:], "")
            outslice = append(outslice, newstring)
            i++
        } else {
            x++
        }
    }
    return outslice
}
```

```
func get_status() string {
    fmt.Println("Modem Monitor")
    var message JSON_out
    var out []byte
    var value []string

    out = execute_qmicli("--dms-get-manufacturer")
    value = parseResult(out)
    message.Status.Manufacturer = value[0]

    out = execute_qmicli("--dms-get-model")
    value = parseResult(out)
    message.Status.Model = value[0]

    out = execute_qmicli("--dms-get-band-capabilities")
    value = parseResult(out)
    message.Status.BandCapabilities.Bands = value[0]
    message.Status.BandCapabilities.LteBands = value[1]
    message.Status.BandCapabilities.LteBandsextended = value[2]

    out = execute_qmicli("--dms-get-capabilities")
    value = parseResult(out)
    message.Status.Capabilities.MaxTxChannelRate = value[0]
    message.Status.Capabilities.MaxRxChannelRate = value[1]
    message.Status.Capabilities.DataService = value[2]
    message.Status.Capabilities.Sim = value[3]
    message.Status.Capabilities.Networks = value[4]

    out = execute_qmicli("--dms-get-operating-mode")
    value = parseResult(out)
    message.Status.OperatingMode.Mode = value[0]
    message.Status.OperatingMode.HwRestricted = value[1]

    out = execute_qmicli("--nas-get-system-selection-preference")
    value = parseResult(out)
    message.Status.SystemSelectionPreference.EmergencyMode = value[0]
    message.Status.SystemSelectionPreference.ModePreference = value[1]
    message.Status.SystemSelectionPreference.DisabledModes = value[2]
    message.Status.SystemSelectionPreference.BandPreference = value[3]
    message.Status.SystemSelectionPreference.LteBandPreference = value[4]
    message.Status.SystemSelectionPreference.LteBandPreferenceExtended =
value[5]
    message.Status.SystemSelectionPreference.TdScdmaBandPreference = value[6]
    message.Status.SystemSelectionPreference.RoamingPreference = value[7]
    message.Status.SystemSelectionPreference.NetworkSelectionPreference =
value[8]
    message.Status.SystemSelectionPreference.ServiceDomainPreference =
value[9]

message.Status.SystemSelectionPreference.GsmWcdmaAcquisitionOrderPreference =
value[10]
    message.Status.SystemSelectionPreference.UsagePreference = value [11]
    message.Status.SystemSelectionPreference.VoiceDomainPreference = value[12]
    message.Status.SystemSelectionPreference.RegistrationRestriction =
value[13]
    message.Status.SystemSelectionPreference.AcquisitionOrderPreference =
value[14]

    out = execute_qmicli("--wds-get-packet-service-status")
    value = parseResult(out)
    message.Status.PacketServiceStatus = value[0]

    out = execute_qmicli("--wds-get-packet-statistics")
    value = parseResult(out)
    message.Status.PacketStatistics.TxPacketsOk = value[0]
    message.Status.PacketStatistics.RxPacketsOk = value[1]
    message.Status.PacketStatistics.TxPacketsDropped = value[2]
```

```
    message.Status.PacketStatistics.RxPacketsDropped = value[3]
    message.Status.PacketStatistics.TxBytesOk = value[4]
    message.Status.PacketStatistics.RxBytesOk = value[5]

    out = execute_qmicli("--wds-get-current-data-bearer-technology")
    value = parseResult(out)
    message.Status.CurrentDataBearerTechnology.NetworkType = value[0]
    message.Status.CurrentDataBearerTechnology.RadioAccessTechnology =
value[1]

    out = execute_qmicli("--wds-get-channel-rates")
    value = parseResult(out);
    message.Status.ChannelRates.CurrentTxRate = value[0]
    message.Status.ChannelRates.CurrentRxRate = value[1]
    message.Status.ChannelRates.MaxTxRate = value[2]
    message.Status.ChannelRates.MaxRxRate = value[3]

    msg, err := json.Marshal(message)
    if err != nil {
        panic(err)
    }
    output := string(msg)
    fmt.Printf("%s\n", output)
    return output
}

var DataSource DataSourceConnection
var DataRemote DataRemoteConnection

type DataSourceConnection struct {
    Broker string
    Topic string
    Mqttc mqtt.Client
    Opts *mqtt.ClientOptions
}

func initSrcConn() {
    fmt.Println("SRC: Initializing MQTT connection")
    DataSource.Opts = mqtt.NewClientOptions()
// some default options
    DataSource.Opts.AddBroker(DataSource.Broker)
// broker --> "scheme://host:port". Scheme: tcp, ssl, ws
    DataSource.Opts.SetOnConnectHandler(connectHandler_source)
// set callback function to be executed if source connection established
    DataSource.Opts.SetConnectionLostHandler(lostHandler_source)
// set callback function to be exectuted in case of lost connection
    DataSource.Mqttc = mqtt.NewClient(DataSource.Opts)
// create client for source connection
}

var connectHandler_source mqtt.OnConnectHandler = func(client mqtt.Client) {
// executed when source connection established
    fmt.Println("SRC: Connected to MQTT broker")
    DataSource.Mqttc = client
    DataSource.Mqttc.Subscribe(DataSource.Topic, 1, subHandler_source)
// subscribe to source topic and set callback function
}

var subHandler_source mqtt.MessageHandler = func(client mqtt.Client, msg
mqtt.Message) {            // executed when message published on subscribed
topic
    token := DataRemote.Mqttc.Publish(DataRemote.Data_topic, 0, false,
string(msg.Payload()))       // publish message on remote data topic
    token.Wait()
}

var lostHandler_source mqtt.ConnectionLostHandler = func(client mqtt.Client,
err error) {           // executed when source connection is lost unintendedly
```

```
    fmt.Println("Connection to source server lost", err)
}

func mqtt_src_connect() {
    fmt.Println("SRC: Connecting to MQTT broker", DataSource.Broker)
    if token := DataSource.Mqttc.Connect(); token.Wait() && token.Error() !=
nil {                   // establish source connection
        panic(token.Error())
    }
}

type DataRemoteConnection struct {
    Broker string
    Data_topic string
    Ping_topic string
    Conf_topic string
    Status_topic string
    Modem bool
    Mqttc mqtt.Client
    Opts *mqtt.ClientOptions
    Ping_rtt int64
    Ping_request int64
}

func initRemConn() {
    fmt.Println("DEST: Initializing MQTT connection")
    DataRemote.Opts = mqtt.NewClientOptions()
// some default options
    DataRemote.Opts.AddBroker(DataRemote.Broker)
// broker --> "scheme://host:port". Scheme: tcp, ssl, ws
    DataRemote.Opts.SetOnConnectHandler(connectHandler_remote)
// set callback function to be called if remote connection established
    DataRemote.Opts.SetConnectionLostHandler(lostHandler_remote)
// set callback function to be exectuted in case of lost connection
    DataRemote.Mqttc = mqtt.NewClient(DataRemote.Opts)
// create client for remote connection
    DataRemote.Ping_rtt = 0
    DataRemote.Ping_request = 0
}

var connectHandler_remote mqtt.OnConnectHandler = func(client mqtt.Client) {
// executed when remote connection established
 fmt.Println("DEST: Connected to MQTT broker")
    DataRemote.Mqttc = client
    DataRemote.Mqttc.Subscribe(DataRemote.Ping_topic, 1,
subHandler_ping_remote)                     // subscribe to remote ping topic
and set callback function
    DataRemote.Mqttc.Subscribe(DataRemote.Conf_topic, 1,
subHandler_conf_remote)                     // subscribe to remote cmd topic
and set callback function

    if DataRemote.Modem == true {
        go func() {
            send_status()
            c := time.Tick(10 * time.Second)
            for range c {
                send_status()
            }
        }()
    }

    timerPing := time.NewTimer(1* time.Second)
    donePing := make(chan bool)
    go func() {
        <-timerPing.C
        send_ping()
// send first ping to remote broker
        donePing <- true
```

```
    }()
    <-donePing
}

func send_status() {
    msg := get_status()
    DataRemote.Mqttc.Publish(DataRemote.Status_topic, 0, false, msg)
}

func send_ping() {
    DataRemote.Ping_request = 0
    if DataRemote.Mqttc != nil {
        DataRemote.Ping_request = time.Now().UnixNano()
        msg, _ := json.Marshal(map[string]int64{"rtt": DataRemote.Ping_rtt})
        DataRemote.Mqttc.Publish(DataRemote.Ping_topic, 0, false, msg)
// publish rtt message on ping topic
    }
}

var subHandler_ping_remote mqtt.MessageHandler = func(client mqtt.Client, msg
mqtt.Message) {        // executed when message published on ping topic
    DataRemote.Ping_rtt = 0
    if DataRemote.Ping_request != 0 {
        DataRemote.Ping_rtt = (time.Now().UnixNano() -
DataRemote.Ping_request) / 1000000           // calculate ping_rtt
    }
    fmt.Println("DEST: Ping RTT =", strconv.FormatInt(DataRemote.Ping_rtt,
10))
    timer := time.NewTimer(1* time.Second)
    done := make(chan bool)
    go func() {
        <-timer.C
        send_ping()
// send ping to remote broker
        done <- true
    }()
    <-done
}

func execute_tccli(dev string, x ...string) {
    device := []string{dev}
    cmd := exec.Command("tcset", append(device, x...)...)
    // fmt.Println(cmd)
    cmd.Stdout = os.Stdout
    if err := cmd.Run(); err != nil {
        log.Fatal(err)
    }
}

func execute_nmcli(conn ...string) {
    opts := []string{"connection", "up"}
    cmd := exec.Command("nmcli", append(opts, conn...)...)
    if err := cmd.Run(); err != nil {
        log.Fatal(err)
    }
}

var subHandler_conf_remote mqtt.MessageHandler = func(client mqtt.Client, msg
mqtt.Message) {        // executed when message published on cmd topic
    var message JSON_in
    if err := json.Unmarshal([]byte(msg.Payload()), &message); err != nil {
        panic(err)
    }
    if message.Tcset != nil {
        delay := message.Tcset.Delay
        loss := message.Tcset.Loss
        delay_distro := message.Tcset.Delay_distro
        rate := message.Tcset.Rate
```

```
        duplicate := message.Tcset.Duplicate
        reordering := message.Tcset.Reordering
        corrupt := message.Tcset.Corrupt
        device := message.Tcset.Device

        fmt.Println("Setting traffic control")

        if device == "" {
            fmt.Println(Error + "It is mandatory to provide interface")
        } else {
            syscmd := []string{"--overwrite"}

            if delay != "" {
                syscmd = append(syscmd, "--delay", delay)
            }
            if loss != "" {
                syscmd = append(syscmd, "--loss", loss)
            }
            if delay_distro != "" {
                syscmd = append(syscmd, "--delay-distro", delay_distro)
            }
            if rate != "" {
                syscmd = append(syscmd, "--rate", rate)
            }
            if duplicate != "" {
                syscmd = append(syscmd, "--duplicate", duplicate)
            }
            if reordering != "" {
                syscmd = append(syscmd, "--reordering", reordering)
            }
            if corrupt != "" {
                syscmd = append(syscmd, "--corrupt", corrupt)
            }
            execute_tccli(device, syscmd...)
        }
    }
    if message.Qmi != nil {
        if DataRemote.Modem == true {
            fmt.Println("Setting current technology")
            if strings.Contains(message.Qmi.Set, "umts") == true {
                execute_qmicli("--nas-set-system-selection-preference=umts")
            }
            if strings.Contains(message.Qmi.Set, "lte") == true {
                execute_qmicli("--nas-set-system-selection-preference=lte")
            }
            if strings.Contains(message.Qmi.Set, "5gnr") == true {
                execute_qmicli("--nas-set-system-selection-preference=5gnr")
            }
            time.Sleep(time.Microsecond * 500)
            execute_nmcli("Movistar (Telefónica) Movistar (USB modems)")
        } else {
            fmt.Println(Error + "Modem mode not selected. Unable to set
technology")
        }
    }
}

var lostHandler_remote mqtt.ConnectionLostHandler = func(client mqtt.Client,
err error) {           // executed when remote connection is lost unintendedly
    fmt.Println("Connection to remote server lost", err)
}

func mqtt_rem_connect() {
    fmt.Println("DEST: Connecting to MQTT broker", DataRemote.Broker)
    if token := DataRemote.Mqttc.Connect(); token.Wait() && token.Error() !=
nil {                  // establish remote connection
        panic(token.Error())
    }
```

```
}

func enable_debug() {
// enable debug
    mqtt.ERROR =    log.New(os.Stdout, "[ERROR] ", 0)
    mqtt.CRITICAL = log.New(os.Stdout, "[CRIT]  ", 0)
    mqtt.WARN =     log.New(os.Stdout, "[WARN]  ", 0)
    mqtt.DEBUG =    log.New(os.Stdout, "[DEBUG] ", 0)
}

func main() {
    out := make(chan os.Signal, 1)
    signal.Notify(out, os.Interrupt, syscall.SIGINT)

    fmt.Println("Data Connection Manager v0.1")
    src_mqtt_host := "127.0.0.1"
    src_mqtt_port := "1883"
    src_topic := "/leap_motion/raw-data"
    dst_mqtt_host := "127.0.0.1"
    dst_mqtt_port := "1883"
    conn_id := "1"
    dst_data_topic   := "/leap_motion/" + conn_id + "/data"
    dst_ping_topic   := "/leap_motion/" + conn_id + "/ping"
    dst_conf_topic    := "/leap_motion/" + conn_id + "/conf"
    dst_status_topic := "/leap_motion/" + conn_id + "/status"
    modem := false
    debug := false

    _, opts, err := getopt.GetOpt(os.Args[1:], "", []string{"help",
"src_host=", "src_port=", "id=", "dst_host=",
        "dst_port=", "src_topic=", "dst_data_topic=", "dst_ping_topic=",
"dst_conf_topic=", "dst_status_topic=", "modem", "debug"})
    if err != nil {
        fmt.Fprintln(os.Stderr, Error + "Wrong parameters, check command
syntax", err)
        os.Exit(2)
    }

    tAux1, tAux2, tAux3, tAux4:= false, false, false, false

    for _, oa := range opts {
        switch oa.Opt() {
            case "-h", "--help":
                fmt.Fprintln(os.Stdout, HelpMessage)
                os.Exit(2)
            case "--src_host":
                src_mqtt_host = oa.Arg()
            case "--src_port":
                src_mqtt_port = oa.Arg()
            case "--src_topic":
                src_topic = oa.Arg()
            case "--dst_host":
                dst_mqtt_host = oa.Arg()
            case "--dst_port":
                dst_mqtt_port = oa.Arg()
            case "--dst_data_topic":
                dst_data_topic = oa.Arg()
                tAux1 = true
            case "--dst_ping_topic":
                dst_ping_topic = oa.Arg()
                tAux2 = true
            case "--dst_conf_topic":
                dst_conf_topic = oa.Arg()
                tAux3 = true
            case "--dst_status_topic":
                dst_status_topic = oa.Arg()
                tAux4 = true
            case "-i", "--id":
```

```
                conn_id = oa.Arg()
            case "--modem":
                modem = true
            case "-d", "--debug":
                debug = true
            default:
                fmt.Fprintln(os.Stderr, Error + "Unknown flag", err)
                os.Exit(2)
        }
    }

     if debug == true {
        fmt.Println("Debug enabled")
        enable_debug()
    } else {
        fmt.Println("Debug disabled")
    }

    fmt.Println("Modem mode is set to", modem)

    src_mqtt_broker := "tcp://" + src_mqtt_host + ":" + src_mqtt_port
    dst_mqtt_broker := "tcp://" + dst_mqtt_host + ":" + dst_mqtt_port

    fmt.Println("Local connection set to", src_mqtt_broker, "with id =",
conn_id)
    fmt.Println("Remote connection set to", dst_mqtt_broker, "with id =",
conn_id)

    if tAux1 != false {
        dst_data_topic = "/leap_motion/" + conn_id + "/data"
    }
    if tAux2 != false {
        dst_ping_topic = "/leap_motion/" + conn_id + "/ping"
    }
    if tAux3 != false {
        dst_conf_topic  = "/leap_motion/" + conn_id + "/conf"
    }
    if tAux4 != false {
        dst_status_topic  = "/leap_motion/" + conn_id + "/status"
    }

    DataSource = DataSourceConnection{Broker: src_mqtt_broker, Topic:
src_topic}
    DataRemote = DataRemoteConnection{Broker: dst_mqtt_broker, Data_topic:
dst_data_topic, Ping_topic: dst_ping_topic,
        Conf_topic: dst_conf_topic, Status_topic: dst_status_topic, Modem:
modem}

    initRemConn()
// Initialize remote connection (create client, set broker, set callback
function if connection detected...)
    initSrcConn()
// Initialize source connection

    go mqtt_rem_connect()
// goroutine for remote connection (first thread)
    go mqtt_src_connect()
// goroutine fot source connection (second thread)

    <-out
    DataRemote.Mqttc.Disconnect(0)
// disconnect from remote connection
    fmt.Println("\nClient disconnected from remote connection")
    DataSource.Mqttc.Disconnect(0)
// disconnect from source connection
    fmt.Println("Client disconnected from source connection")
    fmt.Println("Exit program")
}
```