



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO DE UN PLAN DE PRUEBAS PARA LA HABILITACIÓN DEL SERVICIO DE
RED NB-IOT

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

OCTAVIO NICOLÁS PÉREZ SILVA

PROFESOR GUÍA:
NAZRE EL HUREIMI FACUSE

MIEMBROS DE LA COMISIÓN:
ANDRÉS EDUARDO CABA RUTTE
JOSÉ IGNACIO GUERRA GÓMEZ

Este trabajo ha sido parcialmente financiado por Telefónica I+D

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: OCTAVIO NICOLÁS PÉREZ SILVA
FECHA: 2019
PROF. GUÍA: NAZRE EL HUREIMI FACUSE

DISEÑO DE UN PLAN DE PRUEBAS PARA LA HABILITACIÓN DEL SERVICIO DE RED NB-IOT

De acuerdo a estudios realizados por Cisco[9] se proyecta que al año 2022 existan más de 28,5 mil millones de dispositivos conectados a internet en un mundo donde la población se estima que alcance 7,6 mil millones[5]. Gran parte de estas conexiones no serán dispositivos móviles ni computadores, sino que objetos y servicios que hasta hace pocos años no demandaban acceso a internet. De este modo, surge la corriente llamada el Internet de las Cosas, más bien conocida como *Internet of Things (IoT)*.

El IoT engloba la conexión a internet de dispositivos que comúnmente no lo estaban. Esto es un problema para las redes actuales que no están optimizadas para ofrecer conexión a un gran número de dispositivos por celda, ni tampoco para optimizar el consumo energético. Además sus tasas de tráfico de datos son excesivas para la baja demanda del IoT. Surgen entonces las *Low Power Wide Area Networks (LPWAN's)*, tecnologías que se caracterizan por ofrecer grandes coberturas de red, bajas tasas de transferencia de datos, pero a un mínimo consumo energético. Una de ellas es *Narrow Band Internet of Things (NB-IoT)*, tecnología en la cual se basa el desarrollo del presente trabajo.

Telefónica Chile inició la habilitación del servicio NB-IoT, siendo Providencia, Región Metropolitana, uno de los sectores destinado para las pruebas. Pero parte del proceso de habilitación del servicio, requiere la realización de pruebas de cobertura y conexión que validen el correcto funcionamiento de la red. Con el apoyo de Telefonica I+D, se desarrolló un prototipo basado en el módulo Quectel BG96 capaz de conectarse a la red NB-IoT, y se implementó un servidor privado del proveedor Digital Ocean para la realización de pruebas de conexión.

En el presente trabajo se documenta el desarrollo del prototipo y servidor detallando en profundidad el proceso de diseño y las herramientas utilizadas en su implementación. Además, se presentan pruebas de validación realizadas en red 2G que demostraron el correcto funcionamiento del desarrollo. Las pruebas consistieron en la medición de RSSI y posición para una zona específica de la comuna de Maipú. A partir de los datos capturados, se generó un mapa de cobertura y un documento de respaldo. Se realizaron también pruebas de envío y recepción de paquetes con el servidor privado, obteniendo respuesta exitosa en todos los casos. Finalmente, se documentó la adaptación de las configuraciones para utilizar el prototipo en una red NB-IoT.

De este modo, se logró diseñar un plan de pruebas consistente en un prototipo-servidor para red NB-IoT y se validó de manera exitosa en red 2G, generando un importante apoyo para las futuras pruebas de habilitación del servicio NB-IoT, y abriendo también oportunidades para desarrollos futuros a partir del presente trabajo.

Tu mente es un borrador lleno de bocetos aspirantes a obras maestras. Y a pesar de que llevas una vida atestada de proyectos y sueños, indudablemente hubo un espacio para mí. Dedicado a quien jamás me negó su confianza, a quien llevó el rol de support a la vida real, a Manuel Segundo Pérez Silva, mi hermano.

Agradecimientos

Creo que el momento más difícil de este trabajo fue condensar en sólo una página el inmenso agradecimiento que siento hacia quienes estuvieron a mi lado durante este tiempo. Y aquellos que nuestros caminos se hayan bifurcado, sepan que en mi corazón no hay límites de páginas ni de líneas y ahí permanecerán eternamente. Por eso, agradezco de la forma más sincera:

A Manuel Pérez y Mónica Silva, mis padres. Sean conscientes de que todos los logros académicos que he conseguido son consecuencia del trabajo que ustedes hicieron desde que nací. Incluso, lo que pueda lograr en un futuro seguirá siendo mérito de ustedes. Me dieron la vida, y como si eso fuera poco, llevan veintiséis años regalándome sus vidas. A ustedes les debo más que un agradecimiento, a ustedes les debo todo lo que soy.

A Manuel Pérez, mi hermano. No me explico cómo has cimentado una confianza en mi que desborda los límites de mi propia fe. Y no es que padezca falta de amor propio, pero me has delegado e involucrado en tantos proyectos, que los agradecimientos escalan más allá del apoyo técnico que me brindaste en esta carrera.

A Nicolás Olave, mi amigo. Fuiste el gestor de la amistad más bonita que alguna vez pensé tener, y has sido la fuente diaria de aliento y confianza que necesité sobretodo este año. Mis agradecimientos por cada paso que he dado y en los que tú has estado, por contagiarme con tu locura y esas ganas imparables de robarte el mundo.

A esa parte de mi familia que ha permanecido hasta el final. A mi tía Mirna, tío Victor, a mi abuela Rosa, tío Eugenio. Ustedes han sido un apoyo muy importante en mi vida, y por mucho que las etapas y procesos generen cambios, no olvido aquellos períodos en que incondicionalmente estuvieron para mi y mi hermano.

A mis hermanos Cristian y Moisés, por esa confianza y orgullo que siempre han demostrado y que me hace sentir un *rockstar*. Valoro enormemente los inagotables esfuerzos que hacen por mantener a la familia unida.

Un agradecimiento especial a Hormi, por todo el apoyo académico brindado en mi paso por la universidad y a Ceci, mi compañera y amiga desde el primer día de clases, con quien compartí éxitos, frustraciones, y que juntos logramos sacar esto adelante.

Y a los responsables directos del desarrollo de este trabajo; porfe Nazre El Hureimi, José Guerra y profe Andrés Caba, les agradezco su apoyo, la confianza y la oportunidad entregada.

Tabla de Contenido

Introducción	1
1. Marco Teórico	3
1.1. Evolución de la telefonía móvil	3
1.2. Primera Generación (1G)	3
1.3. Segunda Generación (2G)	4
1.4. Tercera Generación (3G)	5
1.5. Cuarta Generación (4G)	5
1.5.1. Visión General	5
1.5.2. Mejoras 4G LTE	5
1.5.3. Limitaciones 4G LTE	6
1.5.4. Arquitectura red 4G	6
1.5.5. Bandas Operación LTE en Chile	7
1.6. Inicios del IoT	8
1.7. Low Power Wide Area Netwok	8
1.8. Narrow Band Internet of Things (NB-IoT)	9
1.8.1. Estandarización	9
1.8.2. Arquitectura Red NB-IoT	10
1.8.3. Modos de Trabajo	10
1.8.4. Bajo Consumo	11
1.8.5. Cobertura	12
1.8.6. Modos de Transmisión	12
1.8.7. Estructura de la Trama	12
1.8.8. Retransmisión	12
1.9. Potencia y calidad en la transmisión de señales	13
1.9.1. RSSI	13
1.9.2. SNR	13
2. Descripción del Hardware y Software utilizado	15
2.1. Hardware y Software utilizado en el Prototipo	15
2.1.1. Kit de Desarrollo UMTS & LTE EVB	15
2.1.2. Módulo Quectel BG96	18
2.1.3. Arduino Leonardo	18
2.1.4. Módulo SD	19
2.1.5. QNavigator V1.6.3	20
2.1.6. Arduino IDE V1.8.8	20

2.1.7.	GTK Term v1.0	21
2.1.8.	SD Card Formatter	22
2.2.	Hardware y Software para la implementación del servidor	23
2.2.1.	Servidor Digital Ocean	23
2.2.2.	Nginx	24
2.2.3.	Python/Flask	24
3.	Metodología de diseño, implementación y validación del plan de pruebas	25
3.1.	Diseño del Prototipo	25
3.2.	Implementación del prototipo	26
3.2.1.	Verificación del módulo usando QNavigator	26
3.2.2.	Verificación de la Conexión Serial	26
3.2.3.	Verificación del módulo SD utilizando Arduino Leonardo	26
3.2.4.	Conexiones	28
3.2.5.	Conexión a la red GSM	28
3.2.6.	Medición de RSSI de red GSM	29
3.2.7.	Obtención de coordenadas geográficas vía GPS	30
3.2.8.	Automatización del prototipo de muestreo	30
3.3.	Diseño del servidor	31
3.3.1.	Servidor de Echo de Quectel	32
3.3.2.	API's Públicas	32
3.3.3.	Diseño de requerimientos	32
3.4.	Implementación del servidor	32
3.4.1.	Elección y arriendo del servidor	33
3.4.2.	Preparación del entorno	35
3.4.3.	Instalación y configuración Nginx	37
3.4.4.	Instalación y configuración Flask	39
3.4.5.	Creación de la API de consulta	44
3.5.	Validación del Prototipo y Servidor en red GSM	45
3.5.1.	Toma de datos de posición y RSSI	45
3.5.2.	Visualización de datos mediante Mapa de Cobertura	46
3.5.3.	Creación de un archivo con los datos aislados	46
3.5.4.	Conexión y toma de datos del servidor	47
3.5.5.	Configuración del APN	48
3.5.6.	Apertura de socket TCP	49
3.5.7.	Envío de requerimientos GET	49
4.	Resultados y Análisis	51
4.1.	Prototipo final ensamblado	51
4.2.	Mediciones de cobertura y posición sobre red GSM	52
4.2.1.	Conexión a la red GSM	52
4.2.2.	Toma de datos de RSSI y posición en red GSM	53
4.2.3.	Extracción de datos de interés utilizando Python	53
4.2.4.	Visualización de los datos a través de un mapa de cobertura	54
4.3.	Pruebas de conexión al servidor mediante red GSM	56
4.3.1.	Prueba de Conexión vía Socket TCP	56
4.3.2.	Prueba mediante requerimiento GET	57

4.3.3. Prueba de Eco	57
5. Adaptación del trabajo para su uso en red NB-IoT	59
5.1. Adaptaciones para uso del prototipo en la medición de posición y RSSI . . .	59
5.2. Adaptaciones para pruebas de conexión al servidor	60
Conclusión	60
Bibliografía	62

Índice de Ilustraciones

1.1. Arquitectura red 4G LTE	7
1.2. Proyección de demanda de LPWAN's por sector económico[17]	9
1.3. Arquitectura red NB-IoT	10
1.4. Modos de operación NB-IoT [8]	11
1.5. Tecnologías ahorro energético en NB-IoT [10]	11
1.6. Estructura de la trama en downlink [10]	12
1.7. Estructura de la trama en uplink [10]	13
1.8. Cálculo del SNR en función de la potencia de señal y ruido en dB	14
2.1. Kit de Desarrollo UMTS & LTE EVB	16
2.2. Diagrama de Componentes Kit de Desarrollo	16
2.3. Módulo BG96	18
2.4. Pinout Arduino Leonardo[1]	19
2.5. Arduino Leonardo	20
2.6. Módulo SD	20
2.7. Programa QNavigator V1.6 sobre Windows 8.1	21
2.8. Arduino IDLE v1.8.8 sobre Linux Ubuntu v18.10	21
2.9. Programa GTK Term v1.0 sobre Linux Ubuntu 18.10	22
2.10. SD Card Formatter sobre Windows 8.1	22
2.11. Sistemas operativos disponibles para el <i>Droplet</i>	23
2.12. Capacidades y precios disponibles para el <i>Droplet</i>	24
3.1. Diseño de Funcionalidades del Prototipo	25
3.2. Verificación de conexión Serial	26
3.3. Diagrama de conexiones del prototipo	28
3.4. Códigos de bandas GSM para el módulo Quectel BG96	29
3.5. Información entregada por el comando AT+CSQ	30
3.6. Información entregada por el comando AT+QGPSLOC?	30
3.7. Creando un nuevo proyecto	33
3.8. Agregando un nuevo Droplet	34
3.9. Escogiendo características del servidor	34
3.10. Escogiendo características de la CPU	34
3.11. Escogiendo la ubicación física del servidor	35
3.12. Escogiendo el método de autenticación inicial	35
3.13. Correo electrónico confirmando el proceso y con la clave de inicio	35
3.14. Alerta por acceso desconocido al Host	36

3.15. Cambio de contraseña	36
3.16. Servidor Nginx habilitado	38
3.17. Servidor Nginx Web	39
3.18. Servidor Flask Web	41
3.19. Servidor Flask mediante WSGI	41
3.20. Servidor Flask Corriendo	43
3.21. Servidor Flask corriendo bajo Nginx	44
3.22. Zona de validación del prototipo	46
3.23. Comandos predeterminados en QCOM	48
3.24. Parámetros del comando AT+QICSGP	49
3.25. Parámetros del comando AT+QIOPEN	50
3.26. Parámetros del requerimiento GET	50
4.1. Prototipo final ensamblado	51
4.2. Archivo DATALOG con información de conexión 2G	52
4.3. Archivo DATALOG con datos brutos	53
4.4. Datos de Latitud, Longitud y RSSI	54
4.5. Archivo Excel con datos para graficar	55
4.6. Mapa de cobertura con datos de red 2G	55
4.7. Configuración de Punto de Acceso OK	56
4.8. Apertura de <i>socket</i> TCP OK	56
4.9. Prueba inicial de envío de requerimiento GET	57
4.10. Respuesta al requerimiento GET	57
4.12. Cierre del <i>socket</i> TCP	57
4.11. Pruebas de eco mediante requerimiento GET	58

Introducción

A la fecha, existen más de 25,7 millones de abonados a telefonía móvil en un país donde la población apenas supera los 18 millones de habitantes, logrando una penetración del sector móvil de un 135.6% al primer trimestre del año 2019[11]. Este nivel de participación de la telefonía es sólo una muestra del alcance que ha logrado el sector de las telecomunicaciones en los últimos 35 años.

De acuerdo a proyecciones realizadas por Cisco[9], al año 2020 se tendrá alrededor de 50 mil millones de dispositivos conectados a internet en un mundo donde la población se estima que alcance 7,6 mil millones de personas[5]. Esto implica que habrá 6,6 dispositivos conectados a internet por cada habitante en el mundo, cifras considerables si se toma en cuenta que en un hogar donde habitan 5 personas, habrá más de 30 dispositivos conectados a la red.

La tendencia mundial durante las últimas décadas ha sido mejorar la calidad de las conexiones a internet, centrandose los esfuerzos en lograr tasas de transmisión de datos cada vez más altas. En el año 1983 y con la aparición del primer teléfono móvil, se forjó lo que hoy se conoce como la primera generación (1G) de telefonía móvil, la cual alcanzaba escasas velocidades de transmisión de 2,4 Kbps. Treinta y cinco años después, se vive la cuarta generación (4G) ofreciendo velocidades que superan fácilmente los 100Mbps, y que en el corto plazo espera superar los límites de 1Gbps con la entrada comercial de la quinta generación (5G)[16].

Es inevitable pensar en qué momento se acrecentó con tal magnitud el acceso tecnológico de la población, y de qué manera las personas como seres individuales contribuyen a la cuota antes mencionada. Es entonces cuando toma relevancia el concepto de *Internet of Things* (en adelante IoT), el cual hace referencia a la tendencia de conectar objetos cotidianos y personas mediante el uso de redes de internet. IoT promete una red de interconexión de objetos como sensores, cámaras, electrodomésticos y elementos de toda índole, forjando el comienzo de lo que será un futuro cómodo e inteligente[8] de mano de la tecnología.

Dada la inminente masificación del IoT, es necesario tener claro cuáles son los requerimientos principales que demandará este fenómeno. Por un lado, la enorme cantidad de conexiones exige una red capaz de soportarlas, y por otro lado, se genera la necesidad de minimizar el consumo energético, de tal forma que el IoT pueda ser integrado a sectores industriales como las utilities, logística, control de stock, rastreo de contenedores, e incluso llegar a sistemas de edificación inteligente[17].

No obstante, los requisitos del IoT no se ven totalmente satisfechos con las características que actualmente ofrecen las redes, siendo necesaria la implementación de tecnologías que prioricen el bajo consumo energético sin sacrificar la disponibilidad de la red. Es acá donde nacen las llamadas *Low Power Wide Area Network* (LPWAN), redes de comunicación que pretenden solucionar los problemas de consumo y cobertura que no es posible resolver mediante los despliegues actuales.

Narrow Band Internet of Things (NB-IoT) es una tecnología LPWAN que opera bajo la arquitectura de la red 4G LTE. En la teoría, cumple con ser una tecnología de muy bajo consumo energético y de amplia cobertura, soportando conexiones masivas y dando solución a los problemas futuros del IoT. Actualmente se encuentra en fase de despliegue por parte de los principales operadores móviles de Chile, siendo Telefónica uno de los interesados en su habilitación.

De este modo, surge la necesidad de diseñar un plan que permita apoyar la habilitación del servicio mediante pruebas de conexión y medición de cobertura. Para esto, Telefónica I+D pone a disposición el Kit UMTS & LTE EVB de Quectel basado en el módulo BG96, capaz de conectarse a la red NB-IoT y a redes GSM.

El presente trabajo expone el diseño de un plan de pruebas para la habilitación de la tecnología NB-IoT a partir del diseño y construcción de un prototipo basado en dicha tecnología, y el montaje de un servidor privado.

En el capítulo 1 se presenta un marco teórico que entrega los conceptos y bases teóricas fundamentales para entender el desarrollo del trabajo. En el capítulo 2 se presenta en detalle el *hardware* y *software* utilizado en la implementación. Y en los siguientes capítulos se presenta el trabajo realizado en la implementación del prototipo, su validación y resultados de pruebas en red 2G, para finalmente concluir acerca de la utilidad y restricciones del prototipo para su uso en pruebas de habilitación de la tecnología NB-IoT.

Objetivo General

Diseñar y documentar un plan de pruebas para la habilitación del servicio NB-IoT, que permita realizar pruebas de conexión y cobertura a la red.

Objetivos Específicos

- Implementar un prototipo de comunicación basado en NB-IoT utilizando el módulo Quectel BG96
- Diseñar e implementar un sistema de medición de cobertura para el prototipo en un ambiente urbano de la Región Metropolitana
- Implementar un servidor que permita realizar y registrar pruebas de conexión
- Validar el prototipo mediante pruebas de medición de cobertura y conexión al servidor en redes 2G de la Región Metropolitana
- Documentar procedimientos y restricciones para el uso del prototipo en la red NB-IoT

Capítulo 1

Marco Teórico

En el siguiente capítulo se exponen las bases teóricas que sustentan el desarrollo del presente trabajo, comenzando con la evolución que ha tenido el sector de las comunicaciones móviles en los últimos 40 años, para luego revisar los actuales desafíos que enfrenta el área con el desarrollo de las *Low Power Wide Area Networks* (LPWAN) y el *Internet of Things* (IoT). El capítulo inicialmente se centra en entender la arquitectura de la red 4G LTE dado su importancia para el despliegue de la tecnología *Narrow Band-IoT* y luego profundiza en la estandarización de ésta a partir de los principales puntos establecidos por *Third Generation Partnership Project* (3GPP) en el *Release 13 y 14*[14].

1.1. Evolución de la telefonía móvil

Hacia el año 1980 fueron lanzados comercialmente los primeros teléfonos móviles en el mundo, los que operaban en una primitiva red de comunicaciones. Posteriormente, esto sería conocido como la primera generación de telefonía móvil (1G). La primera generación se basó en tecnología completamente análoga con capacidad sólo para transmisión de voz y con tecnología de acceso al medio mediante división de frecuencia (FDMA)[12]. Posteriormente, la arquitectura de la red de telecomunicaciones inició un proceso de digitalización dando paso a la segunda generación (2G) y evolucionando hasta alcanzar lo que hoy se conoce como la quinta generación (5G). Es necesario conocer las características distintivas de cada generación de telefonía para luego realizar una inspección profunda a la arquitectura de cuarta generación (4G), con el propósito de entender el contexto teórico que rodea el despliegue de la tecnología NB-IoT.

1.2. Primera Generación (1G)

La primera generación de telefonía se remota a los años 80, donde fueron introducidos al mercado los primeros teléfonos móviles en el mundo de la forma que los conocemos ahora.

Se basaron en la tecnología conocida como *Advanced Mobile Phone System* (AMPS), la cual ofrecía solamente el servicio de comunicación por voz[16]. El sistema AMPS usaba modulación en frecuencia y acceso al medio mediante división por frecuencia (FDMA) con canales de 30 KHz de capacidad, operando en la banda de frecuencia de 824- 894MHz[16] y logrando alcances de hasta 20km por celda[18]. El hecho de que la tecnología estuviese basada en protocolos análogos de modulación en frecuencia la hacía altamente susceptible a interferencias reduciendo la calidad de las llamadas. Además, por el mismo hecho de ser análoga, no permitía métodos de encriptación avanzada, teniendo bajas herramientas de seguridad[12]. Las principales especificaciones técnicas[16] se resumen a continuación:

- Tasa de datos de 2.4 kbps
- Uso de señal análoga
- Baja calidad de voz y duración de batería
- Baja Seguridad
- Baja eficiencia en el uso del espectro
- Handoff poco confiable

1.3. Segunda Generación (2G)

La segunda generación nace con la digitalización de las redes de telefonía, instaurándose el *Global System for Mobile Communications* (GSM). Esto permitió la integración de servicios de mensajería que se mantienen hasta hoy, como lo son el *Short Message Service* (SMS) o los *Multimedia Messaging Service* (MMS). Se implementó el acceso al medio por tiempo (TDMA) y por código (CDMA), mejorando la utilización del espectro con anchos de banda entre 20-200 kHz[16] y alcanzando velocidades de transmisión de 64 kbps. Posteriormente, la tecnología evolucionó en lo que se conoce como 2.5G, implementando la conmutación de paquetes y permitiendo con esto las primeras versiones de navegación web. Luego, una primitiva versión de 3G fue desarrollada mejorando las tasas de transmisión ofrecidas por GPRS, en el llamado *Enhanced Data-rates for GSM Evolution*. Los principales aspectos técnicos[16][12] se detallan a continuación:

- Tasa de datos de 64 kbps
- Uso de señal digital
- Calidad de voz y consumo energético mejorados
- Mejora de seguridad vía encriptación digital
- Al ser digital, necesita una señal fuerte para poder realizar la transmisión
- Aparece el *Suscriber Identity Module*, más conocida como tarjeta SIM

Aspectos mejorados en 2.5G

- Tasa de datos de 144 kbps
- Servicio de e-mail y navegación web vía GPRS/EDGE
- Modulación GMSK y 8PSK

- Con EDGE, comienzan las primeras implementaciones de 3G

1.4. Tercera Generación (3G)

El *Universal Mobile Telecommunications Service* (UMTS) o más conocido como 3G, logra mejoras considerables respecto a su predecesora. El acceso al medio mediante código es mejorado por acceso mediante código en *Wideband* (WCDMA). Con esto se logran velocidades de 2 Mbps en un comienzo con un ancho de banda de 15-20 MHz en la banda de 2100MHz. Posteriormente, se implementaron mejoras a la tecnología como el *High-Speed Packet Access* (HSPA y luego HSPA+). Existen diversas versiones de 3G, en las cuales no se profundizará por no ser el centro de interés de este trabajo, pero para más detalle se sugiere consultar el *Release 4 y 5* de 3GPP. Los aspectos más importantes[16] son:

- Velocidad 2Mbps (hasta 10Mbps para HSPA+)
- Transmisión de video vía streaming, juegos 3D
- Implementación de nuevos protocolos de seguridad
- Altos costos de implementación de arquitectura y pago de licencias
- Gran ancho de banda requerido

1.5. Cuarta Generación (4G)

1.5.1. Visión General

Uno de los aspectos más importantes que destaca a la cuarta generación, conocida como *Long Term Evolution* (LTE), es el quiebre producido en los servicios de voz respecto a las generaciones anteriores. La tecnología LTE es un sistema basado completamente en protocolo IP, dejando atrás la conmutación de circuitos para adoptar una arquitectura puramente *Packet Switching* como se ve en la figura 1.1. LTE fue desarrollado por 3GPP y por lo tanto es compatible con las tecnologías desarrolladas previamente en 3G y 2G. Además soporta una gran variedad de servicios basados en IP, destacando entre ellos la telefonía VoIP (*Voice over IP*)[13]

1.5.2. Mejoras 4G LTE

LTE destaca por el considerable aumento de las tasas de transmisión, alcanzado velocidad en *dowlink* de 100Mbps[12]. En 3G, el acceso al medio se realizaba mediante FDMA, lo que implicaba que las portadoras debían estar lo suficientemente separadas para evitar la interferencia entre portadoras cercanas. LTE adopta el acceso al medio para *dowlink* mediante *Orthogonal Frequency Division Multiple Access* (OFDMA) y con portadora única para *uplink*, *Single Carrier*(SC-FDMA), ambas utilizando portadoras con frecuencias ortogonales, lo que

garantiza una mínima interferencia de portadoras incluso cuando estén muy cercanas entre ellas. Gracias a este menor espaciado, es posible aprovechar de manera más eficiente el ancho de banda, logrando el mismo número de canales que FDMA con una menor utilización de este.

LTE también adopta diversas modulaciones como QPSK, 16QAM y 64QAM con diferentes anchos de bandas, pudiendo ser 1.4MHz, 3MHz, 5MHz, 10MHz, 20MHz[13].

1.5.3. Limitaciones 4G LTE

LTE se ve potenciado con la tecnología *Multiple Input Multiple Output* (MIMO) mediante la inclusión de dos antenas (MIMO 2x2) y de cuatro antenas (MIMO 4x4), lo que se conoce como *LTE Advance*. Mediante modulación 64QAM, MIMO 4x4 y utilizando un ancho de banda de 20MHz, se alcanzan velocidades de 343Mbps[12].

Además, al ser una tecnología basada en el protocolo IP, gran parte de su arquitectura se aprovecha de los sistemas de red ya existentes, logrando abaratar considerablemente los costos de implementación respecto a sus predecesores 3G y 2G[15].

1.5.4. Arquitectura red 4G

La Arquitectura de la red 4G consta de tres partes principales, el *Evolved UMTS Terrestrial Radio Access Network* (E-UTRAN), el *Evolved Packet Core* (EPC) y el *IP Services*[13].

- E-UTRAN: Corresponde a la interfaz aérea de la red LTE, la que involucra el *User Equipment* (UE) y el eNodeB.
- EPC: Es el encargado de mantener siempre disponible la conexión, y además ofrecer servicios como el *handover* y transporte de paquetes de voz
- IP Services: Es el encargado de ofrecer los servicios IP a los suscriptores.

En la figura 1.1 se muestra la arquitectura de la red LTE donde se aprecian las tres partes mencionadas. Además, se detalla a continuación las funciones principales[13] de cada estructura que conforma la red.

- ENodeB (*Evolved Node-B*): Es el nodo que se conecta directamente con el *User Equipment*. Corresponde a un híbrido entre la estación base y el controlador de radio que en 3G operaban por separados. Posee las antenas que emiten y reciben directamente las señales, con servicios de modulación-demodulación y corrección de errores. Además el controlador de radio evalúa la calidad de conexión administrando el *handover* entre celdas.
- PDN-GW (*Packet Data Network Gateway*): Es el encargado de dar a la red acceso hacia redes externas. Tiene la labor de filtrar cada paquete y consultar los servicios permitidos para cada usuario.
- MME (*Mobility Management Entity*): Servidor de señalización que realiza funciones de

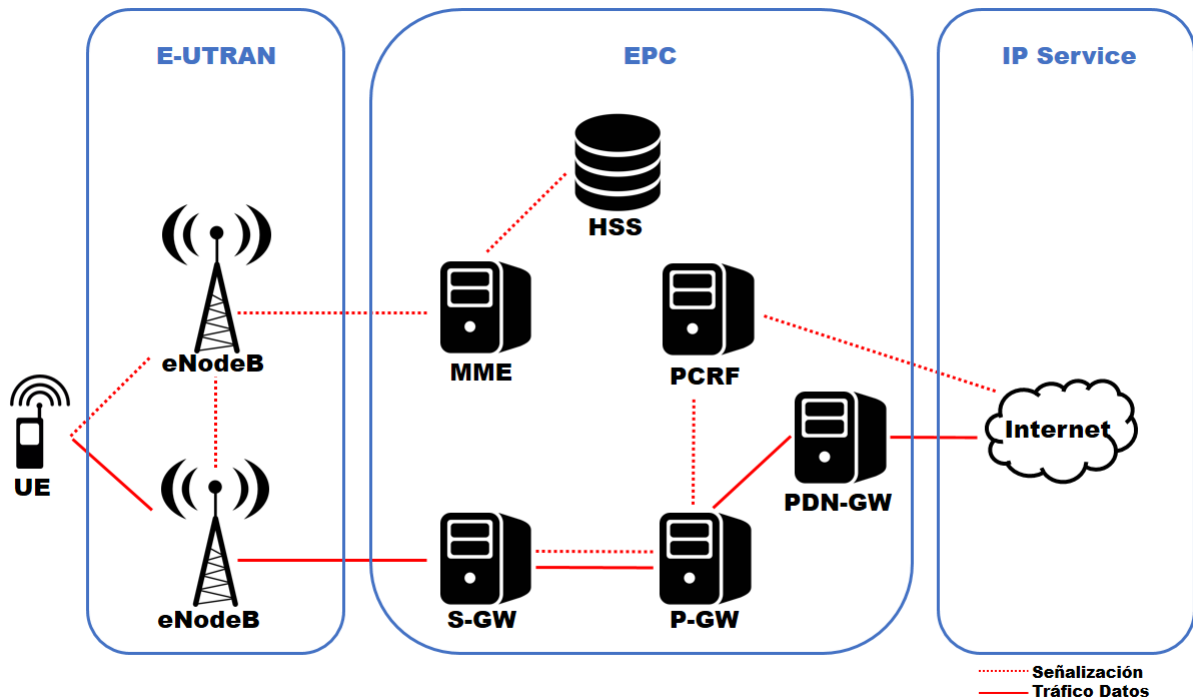


Figura 1.1: Arquitectura red 4G LTE

control y movilidad. Es el encargado de administrar llamadas o sesiones de entrada y de la autenticación del *User Equipment*.

- **S-GW (*Serving Gateway*):** Sistema encargado de gestionar el tráfico de usuario dando el tratamiento adecuado a los paquetes que garantice la calidad de servicio. Además, es el anclaje entre distintas redes 3GPP.
- **PCRF (*Policy and Charging Rules Functions*):** Sistema que mediante políticas de control de flujo se encarga de la tarificación y la calidad de servicio.
- **HSS (*Home Subscriber Server*):** Base de datos central de usuarios del sistema. Contiene los perfiles de usuario que incluye información acerca de los servicios y contenido al que cada usuario está autorizado a acceder.

1.5.5. Bandas Operación LTE en Chile

La tecnología LTE fue desplegada en Chile en la banda de 2600MHz. Compañías como Movistar, Entel, VTR y Claro se hicieron acreedoras de la licitación inicial comenzando a ofrecer el servicio en dichas frecuencias. Años después, se integró WOM con la licitación de la banda LTE AWS en 1700/2100MHz. La ventaja principal de utilizar bandas de alta frecuencia es que permiten la transmisión de información a una mayor tasa de bytes/s. Sin embargo, tienen la gran desventaja de que su alcance y capacidad de penetración es muy baja respecto a bandas de baja frecuencia, debido a las grandes pérdidas de energía por colisión propias de las ondas de alta frecuencia. Por esta razón, el 2015 se licitó por parte de Entel, Claro y Movistar, la banda 700MHz, lo que mejoró considerablemente la cobertura *indoor* y *outdoor* gracias al mayor alcance y penetración de la banda. Con esto, las frecuencias utilizadas para

el servicio LTE y su nombre estandarizado son:

- Banda 4 (1700/2100 MHz): WOM
- Banda 7 (2600 MHz) Entel, Movistar, Claro, VTR
- Banda 28 (700 MHz) Entel, Movistar, Claro

En Chile, y por parte del operador móvil Movistar, el servicio NB-IoT está siendo desplegado sobre la Banda 28 de la red LTE.

1.6. Inicios del IoT

Uno de los principios más importantes del IoT es lograr la interconexión no sólo de personas, sino también de dispositivos, tendencia conocida como *Machine Type Communication* (MTC)[8]. Para lograr un desarrollo exitoso del MTC no basta con minimizar los costos de producción de los dispositivos, también es necesario el despliegue de una red con la capacidad suficiente para albergar las billones de conexiones demandadas por éstos. Además, se espera tener una red inalámbrica con suficiente cobertura para alcanzar lugares alejados y a un consumo energético lo más bajo posible. Bajo estas demandas, comienza el desarrollo de diversas tecnologías de comunicación inalámbrica que apuntan a dar solución a los problemas de cobertura y consumo energético, es así como se da inicio al desarrollo de las *Low Power Wide Area Network* (LPWAN).

1.7. Low Power Wide Area Network

Las nuevas demandas de la industria junto a la creciente penetración tecnológica, han impulsado el desarrollo de sistemas de comunicaciones que sean capaces de satisfacer las nuevas necesidades del mercado. La tónica en el desarrollo de redes de telecomunicaciones durante los últimos años, ha sido aumentar la capacidad del ancho de banda y mejorar las técnicas de acceso al medio, con el fin de obtener tasas de transmisión de datos y latencias cada vez más bajas. Pero el sector de telecomunicaciones ya no sólo es demandado por las redes de telefonía celular, sino que la demanda se proyecta fuertemente a otros mercados[17]. Agricultura, utilities, logística, ciudades inteligentes, y todo lo que pueda ser controlado de manera remota, genera necesidades que apuntan a redes que puedan ofrecer amplia cobertura en lugares de difícil acceso, pero al mínimo consumo energético. De este modo, comienza el desarrollo de las *Low Power Wide Area Network* (en adelante LPWAN). La proyección de la demanda de conexiones LPWAN estima un aumento de casi diez veces la demanda actual, alcanzando los 3.5 billones de conexiones hacia el año 2025[17]. En la figura 1.2 se comparan los principales sectores y su demanda de LPWAN en los próximos 7 años.

Para dar solución a los requerimientos mencionados, 3GPP dedicó gran parte del *Release 13* a estandarizar dos nuevas tecnologías que serían posteriormente mejoradas en el *Release*

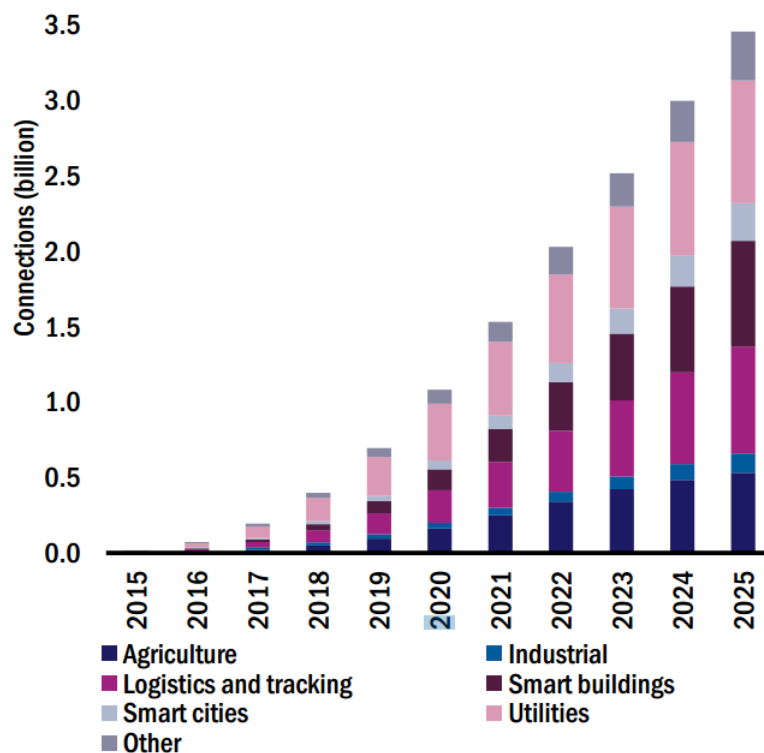


Figura 1.2: Proyección de demanda de LPWAN's por sector económico[17]

14[14]. La primera fue denominada LTE-MTC (LTE-M), introduciendo con esto la tecnología LTE Cat. 1, y la segunda, denominada *Narrow Band IoT* (NB-IoT) mediante el desarrollo de la tecnología LTE Cat. NB1. En adelante, el presente trabajo se centra en la revisión de los aspectos técnicos definidos para NB-IoT en el *Release 13* y *14*.

1.8. Narrow Band Internet of Things (NB-IoT)

1.8.1. Estandarización

NB-IoT es una tecnología LPWAN estandarizada por 3GPP el año 2016 en el *Release 13*, enfocada en la adquisición de datos por parte de aplicaciones inteligentes que demanden una baja tasa de transmisión. La característica más importante de la tecnología NB-IoT, es que admite conexiones masivas con un muy bajo consumo energético, ofreciendo al mismo tiempo una amplia cobertura bidireccional, tanto para el plano de señalización como para el plano de datos. Además, el hecho de que su despliegue sea en la arquitectura 4G LTE reduce considerablemente los costos de implementación de la tecnología, potenciado por el bajo costo de fabricación de los dispositivos.

1.8.2. Arquitectura Red NB-IoT

La arquitectura de la tecnología NB-IoT simplifica la arquitectura del EPC presente en la red LTE con el objetivo de cumplir con los requerimientos de tráfico estipulados en el *Release 13*[14]. Se integran las funciones del MME, SWG y PGW en un núcleo de red dedicado y llamado C-SGN. Se mantiene la interfaz S1-MME y se utiliza la interfaz S1-Lite optimizada para mensajes de control. Además, la interfaz S1-U ya no es requerida en esta arquitectura.

De esta forma, la arquitectura NB-IoT queda como se aprecia en la imagen 1.3:

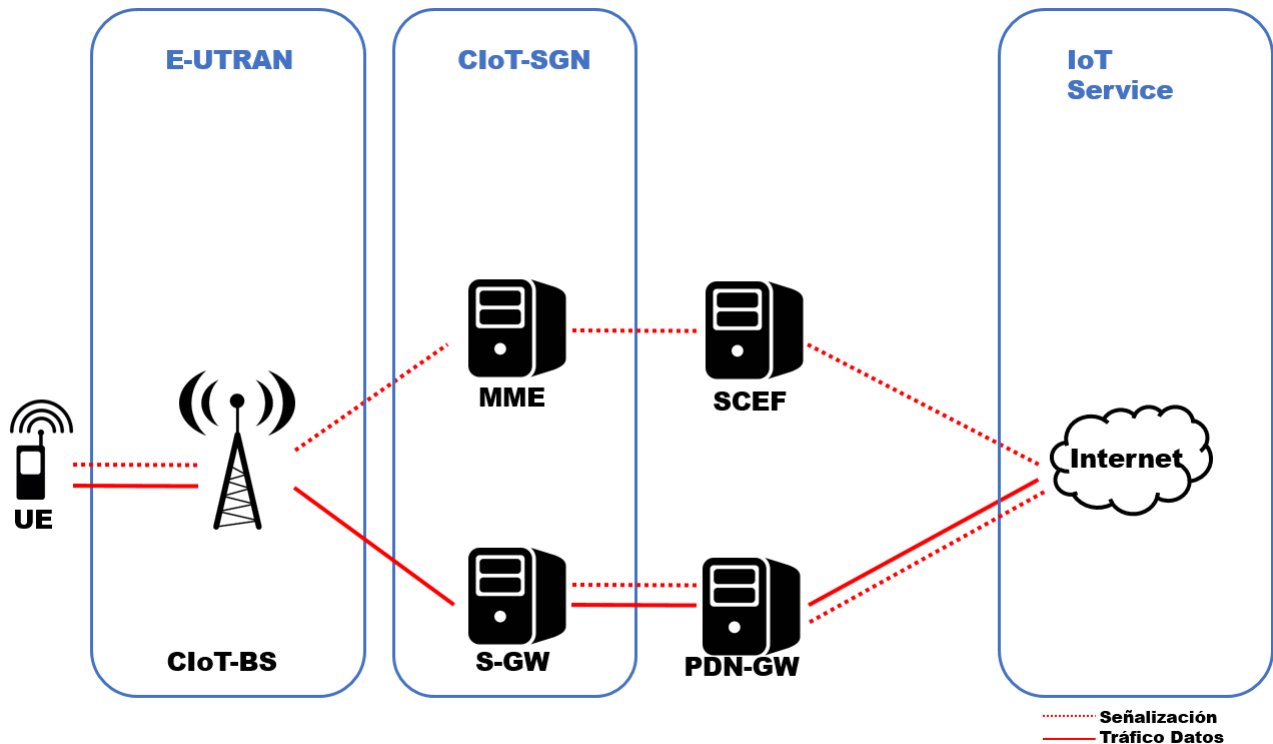


Figura 1.3: Arquitectura red NB-IoT

1.8.3. Modos de Trabajo

La tecnología NB-IoT puede utilizar el espectro de tres formas distintas, conviviendo junto a LTE de acuerdo a las siguientes maneras de despliegue:

- Modo Stand-Alone: Utiliza bandas de frecuencias independientes y que no se traslapan con la banda de frecuencia LTE
- Modo Guard-Band: Utiliza bandas de frecuencias en los bordes o límites de la banda de LTE
- Modo In-Band: Utiliza frecuencias de la banda de LTE, tomando un bloque de recurso de dicha banda para el despliegue de la tecnología.

En la Figura 1.4 se observa gráficamente los distintos modos de trabajo de NB-IOT.

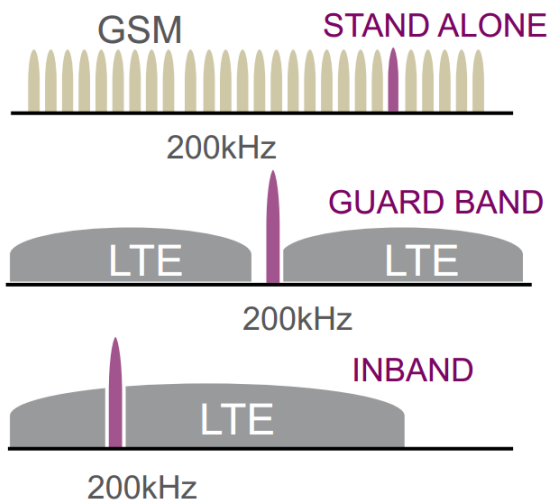


Figura 1.4: Modos de operación NB-IoT [8]

1.8.4. Bajo Consumo

Para lograr que la batería de los dispositivos se mantengan activas durante años, es necesario reducir el consumo energético al mínimo, por lo que NB-IoT integra sistemas de conexión temporal que permiten la desconexión del dispositivo por largos períodos de tiempo. Mediante el modo de ahorro de energía *Power Save Mode* (PSM) y recepción expandida discontinua *Expanded Discontinuous Reception*, (eDRX) se reduce considerablemente el consumo energético.

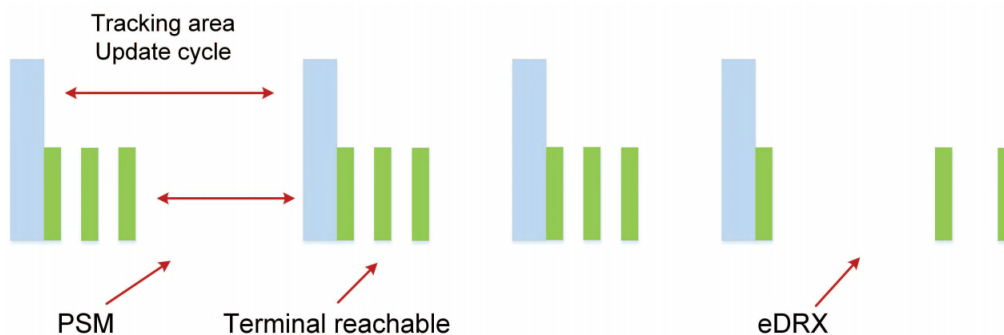


Figura 1.5: Tecnologías ahorro energético en NB-IoT [10]

De acuerdo con los datos simulados y expuestos en el TR45.820 de 3GPP[7], para una pérdida de acoplamiento de 164 dB y utilizando PSM y eDRX, una batería de 5Wh puede prolongar su vida útil por 12.8 años a una tasa de 1 mensaje diario de hasta 200 bytes.

1.8.5. Cobertura

De las simulaciones realizadas por 3GPP, se confirmó una ganancia de 164 dB, las cuales fueron realizadas en modo *in-band* y *guard-band*. Para obtener una mejora en la cobertura se implementó mecanismos como retransmisión y modulación en baja frecuencia QPSK. Además, está en estudio la implementación de modulación 16QAM. Por último, el límite de tolerancia para la latencia es de 10s, pero bajo condiciones de bajas pérdidas de acoplamiento puede soportarse latencias menores a 6s.

1.8.6. Modos de Transmisión

La implementación de NB-IOT está basada en la arquitectura de LTE, donde fueron hechas algunas modificaciones importantes que caracterizan a la tecnología. El ancho de banda de NB-IoT en capa física es de 200 KHz. Para el *downlink*, adopta modulación QPSK y acceso mediante tecnología OFDMA con portadoras de 15 KHz. En el *uplink*, la modulación es BPSK o QPSK con acceso mediante SC-FDMA, incluyendo portadoras simples y múltiples con anchos de 3.75 kHz y 15Khz. Para el caso de espaciado en 15 KHz, se definen 12 subportadoras, y para el caso de 3.75KHz quedan definidas 48 subportadoras, obteniéndose entonces una mayor cobertura para el espaciado en 3.75 KHz debido a la mayor densidad del espectro.

1.8.7. Estructura de la Trama

El *downlink* del eNodeB de NB-IoT soporta *E-Utran wireless Frame Structure* (FS1), el cual se muestra en la figura 1.6. En tanto el *uplink*, soporta FS1 para la portadora de 15 KHz, pero se define una nueva trama para la portadora de 3.75 KHz, que se muestra en la figura 1.7

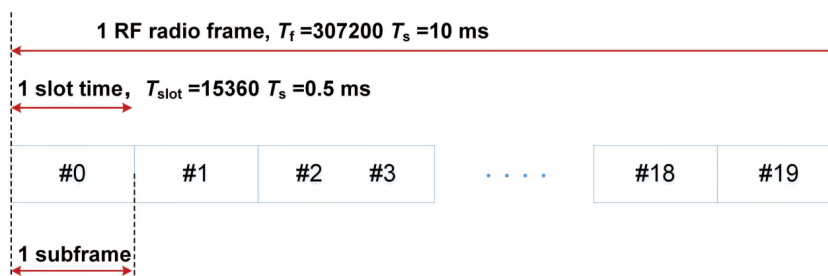


Figura 1.6: Estructura de la trama en downlink [10]

1.8.8. Retransmisión

NB-IoT adopta un mecanismo de retransmisión para obtener diversidad de ganancia en tiempo, y también modulación de bajo orden para mejorar el desempeño de la demodulación

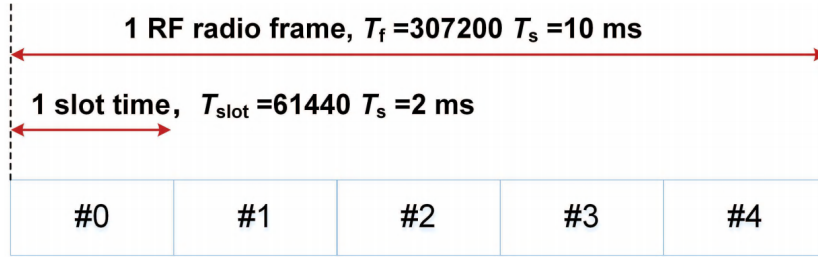


Figura 1.7: Estructura de la trama en uplink [10]

y cobertura. Todos los canales y portadoras soportan retransmisión de información, lo que fue especificado para cada canal y modulación por 3GPP en el *Release 14* [14]

1.9. Potencia y calidad en la transmisión de señales

1.9.1. RSSI

El *Received Signal Strength Indicator* (RSSI) es la cuantificación de la potencia de señal recibida, la que por practicidad se transforma a escala logarítmica y es expresada en dBm. La Escala comprende valores entre 0dBm y -120dBm, donde un valor cercano a 0 implica una señal más fuerte (1mW), y un valor cercano a -120 se interpreta como ausencia total de señal. EL RSSI es un indicador muy importante en telefonía móvil, ya que permite caracterizar la señal móvil de acuerdo a los siguientes intervalos:

- Excelente: $-70 \text{ dBm} < \text{RSSI}$
- Buena: $-85\text{dBm} < \text{RSSI} < -71\text{dBm}$
- Aceptable: $-100\text{dBm} < \text{RSSI} < -86\text{dBm}$
- Mala: $-119\text{dBm} < \text{RSSI} < -101\text{dBm}$
- Sin Cobertura $\text{RSSI} = -120\text{dBm}$

Sin embargo, el RSSI es sólo un indicador de la potencia recibida y no de la calidad de la señal, la que depende entre otros factores, del ruido presente en el canal. Para esto, se introduce el concepto de SNR.

1.9.2. SNR

El *Signal to Noise Ratio* (SNR) es un indicador que evalúa la señal a partir de la razón entre su potencia y la potencia del ruido que percibido en el canal, el que se expresa como:

$$SNR = \frac{P_{signal}}{P_{noise}}$$

Por simplicidad, conviene llevar esta expresión a escala logarítmica:

$$SNR_{dB} = 10 \log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$$

Aplicando propiedad del logaritmo de un cuociente, la ecuación se puede expresar como:

$$SNR_{dB} = 10 \log_{10}(P_{signal}) - 10 \log_{10}(P_{noise})$$

Lo que determina la expresión final para el SNR, correspondiente a la diferencia algebraica entre la potencia de la señal y del ruido en escala logarítmica.

$$SNR_{dB} = P_{signal_{dB}} - P_{noise_{dB}}$$

De este modo, el SNR corresponde a un valor positivo entre 0dB y 120dB, siendo los valores cercanos a 120dB indicadores de una mejor calidad respecto a 0dB, peor caso en el que la señal y el ruido tienen la misma potencia. Lo anterior, se puede ver gráficamente en la figura 1.8

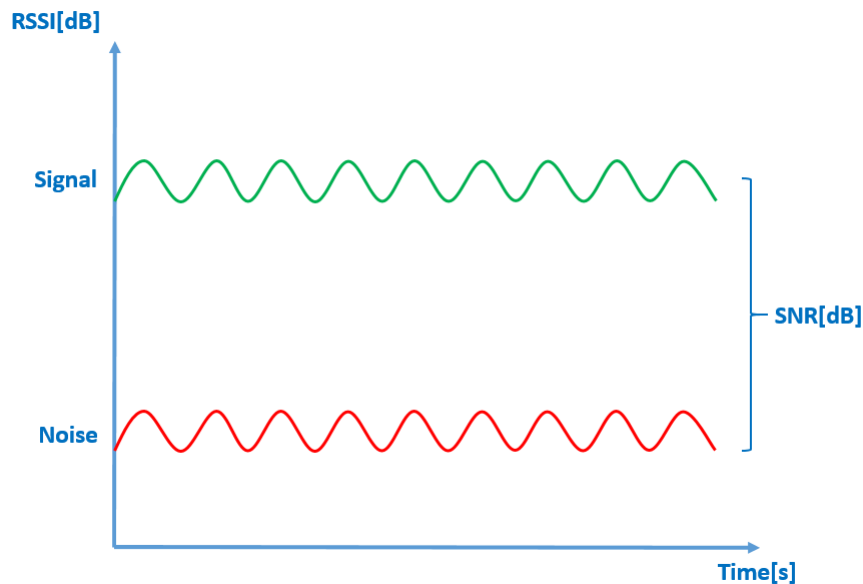


Figura 1.8: Cálculo del SNR en función de la potencia de señal y ruido en dB

Capítulo 2

Descripción del Hardware y Software utilizado

En el presente capítulo se describe en detalle el *hardware* y *software* utilizado en la implementación del plan de pruebas. En un primer subcapítulo se detalla los recursos utilizados para la construcción del prototipo de pruebas basado en el Kit de Desarrollo UMTS<E de Quectel, proporcionado en su totalidad por Telefónica I+D, junto a los dispositivos controladores y periféricos de adquisición personal. En el segundo subcapítulo se detalla los recursos utilizados en la implementación y configuración de un servidor privado, el cual fue arrendado al proveedor Digital Ocean.

2.1. Hardware y Software utilizado en el Prototipo

2.1.1. Kit de Desarrollo UMTS & LTE EVB

EL UMTS & LTE EVB Kit desarrollado por Quectel[6] permite probar aplicaciones y funcionalidades de la red 2G y 4G mediante el uso de los Módulos UMTS & LTE del mismo fabricante. El kit incluye como principal elemento una placa de desarrollo(ver figura 2.1), e incluye además diversos módulos que permiten ampliar las funcionalidades de acuerdo a las necesidades del usuario. A continuación se detalla los accesorios y funciones del kit, acompañados de un código de referencia que puede ser consultado en la figura 2.2.

Antena GPS

El kit incluye una antena GPS (*Global Position System*) que permite la georreferenciación mediante satélites. Los datos son entregados vía comando AT y de manera estandarizada en formato NMEA[2], lo que facilita el procesamiento de éstos al ser una cadena de texto con parámetros separados por comas.

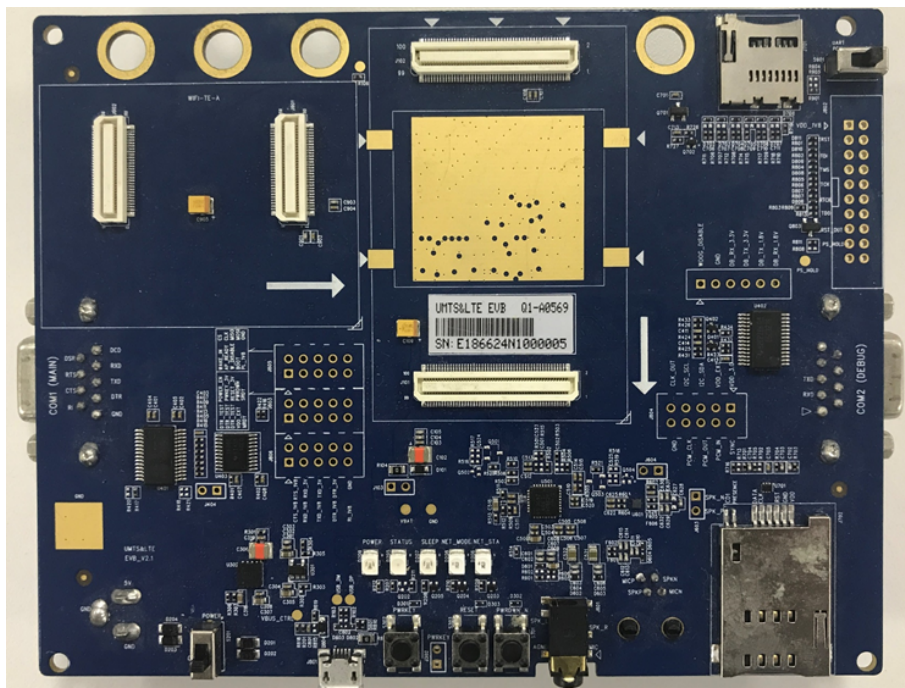


Figura 2.1: Kit de Desarrollo UMTS & LTE EVB

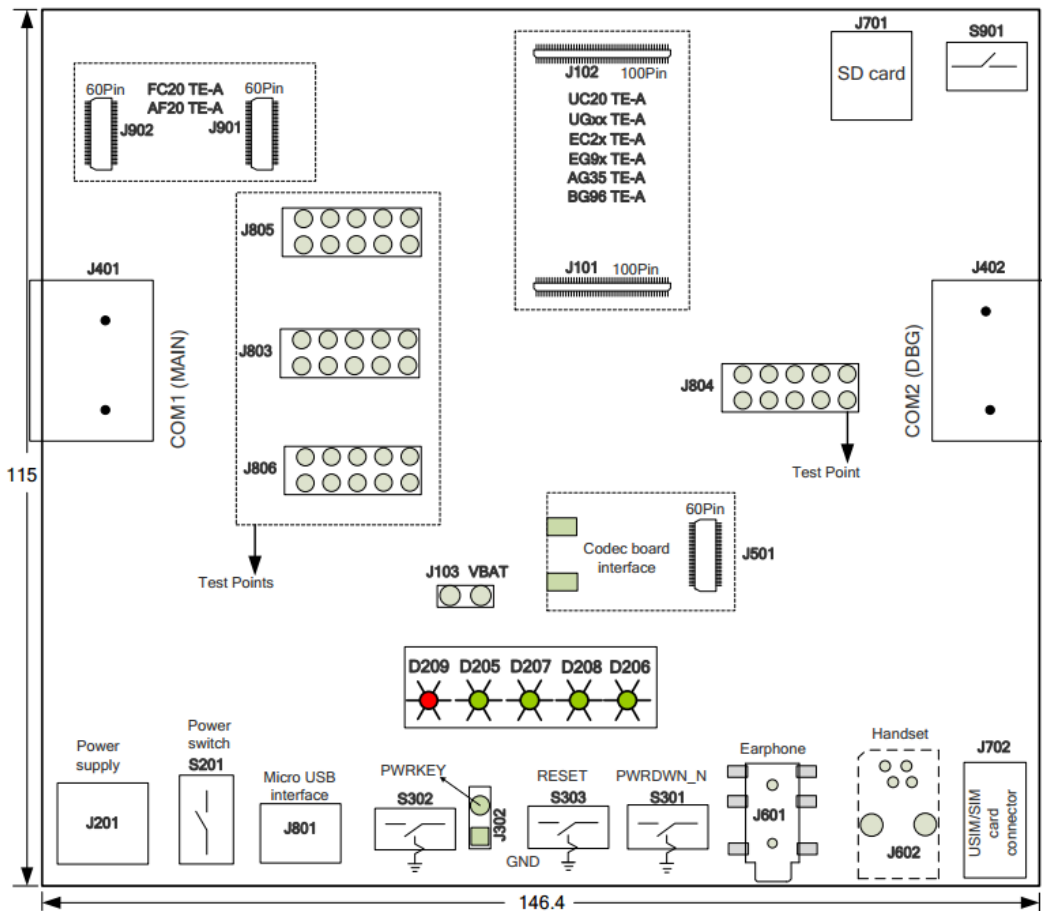


Figura 2.2: Diagrama de Componentes Kit de Desarrollo

Slot SIM (J702)

Posee un *Slot* para tarjeta SIM estándar, siendo compatible con tarjetas de cualquier proveedor de servicios de red móvil. Además es necesaria para la identificación del dispositivo en la red si se desea acceder a los servicios de datos ofrecidos por el proveedor.

Socket Módulos Quectel (J101/102)

La placa de desarrollo Quectel es compatible con diversos módulos en las categorías UTMS y LTE fabricados por el mismo proveedor. Los módulos Quectel son los que proveen la tecnología de conectividad, existiendo diversos modelos de acuerdo a los servicios de conexión ofrecidos. A continuación se detallan los módulos compatibles con la placa de desarrollo utilizada. La información técnica de ellos, puede ser consultada en la web oficial del proveedor[6]

- UC20
- UG96/UG95
- EC25/EC21/EC20/EC20 R2.0/EC20 R2.1
- EG91/EG95
- AG35
- BG96

Además es compatible con los módulos WiFi(J501):

- FC20 Series
- AF20

Puertos UART (J401/402)

La placa ofrece conexión Serial mediante el puerto principal UART1(J401) y el puerto de debug UART2(J402). Al mismo tiempo, es posible acceder al bus de comunicación mediante los pines de prueba RX/TX (J805) presentes en la placa, y que facilitan la conexión con microcontroladores y otros dispositivos de control compatibles.

Jack de Alimentación 5V (J201)

Mediante un jack genérico de 21mm y un interruptor ON/OFF (S201) es posible alimentar el kit de desarrollo con un voltaje continuo de 5V desde un adaptador AC/DC o desde una batería de 5V. Alternativamente, el Kit puede ser alimentado mediante el conector hembra micro USB (J801) con un voltaje de 5V.

Otras Funciones

El kit de desarrollo ofrece conexión de auriculares y micrófono través de un jack estéreo de 3.5mm(J601). También posee un *slot* para tarjeta micro SD(J702) y un puerto RJ22 para la conexión de un *Handset* telefónico(J602).

2.1.2. Módulo Quectel BG96

El módulo LTE BG96 Cat.M1/NB1 & EGPRS desarrollado por Quectel permite el acceso a la red con una velocidad máxima de descarga de 375 Kbps simétrica. Posee un bajo consumo de energía y es configurable mediante comandos AT para acceder a red LTE Cat.M1/NB1 o a la red 2G EGPRS. El módulo se conecta directamente en la placa a través del socket dispuesto para esto, y que además es compatible con los distintos módulos Quectel.

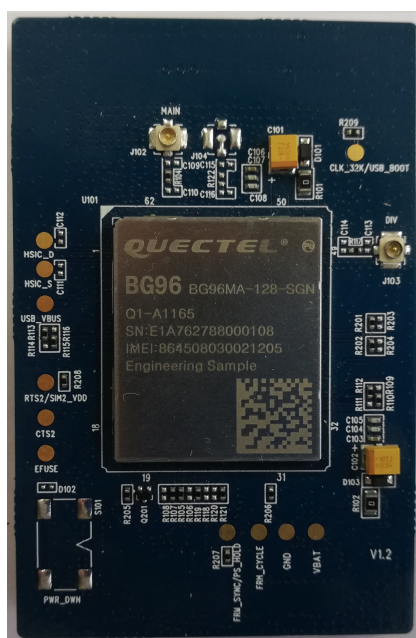


Figura 2.3: Módulo BG96

Una de las características importantes es que posee conexión directa para la antena LTE y para la antena GPS a través de dos conectores macho U.FL presentes en el módulo BG96. Gracias a este conector, es posible utilizar antenas genéricas e intercambiar de manera sencilla los accesorios necesarios para el desarrollo.

2.1.3. Arduino Leonardo

Arduino Leonardo es un dispositivo basado en el microcontrolador Atmega32u4[1]. Tiene 20 pines de entrada/salida digital, un oscilador de cristal de 16 MHz, un conector micro USB, un conector de alimentación genérico de 21mm, pines para un conector ICSP y un botón de reinicio. La principal ventaja de Arduino Leonardo respecto a versiones anteriores, es

que el microcontrolador ATmega32u4 tiene comunicación USB integrada, lo que le permite prescindir de un procesador secundario y ofrecer una conexión serial adicional al usuario mediante virtualización.

De acuerdo al *PinOut* de Arduino Leonardo (ver figura 2.4), los pines 0 y 1 corresponden a los pines RX y TX de la conexión Serial. Los pines SPI corresponden a la conexión de 6 pines aislados presentes en la placa, y también a través del pin 9 para SCLK, pin 10 para MOSI, pin 11 para MISO, y en el pin 4 el CS. Además, el microcontrolador puede alimentarse con 5Vcc en el pin Vin, o a través del jack hembra de 2.1mm con input de 7-12V.

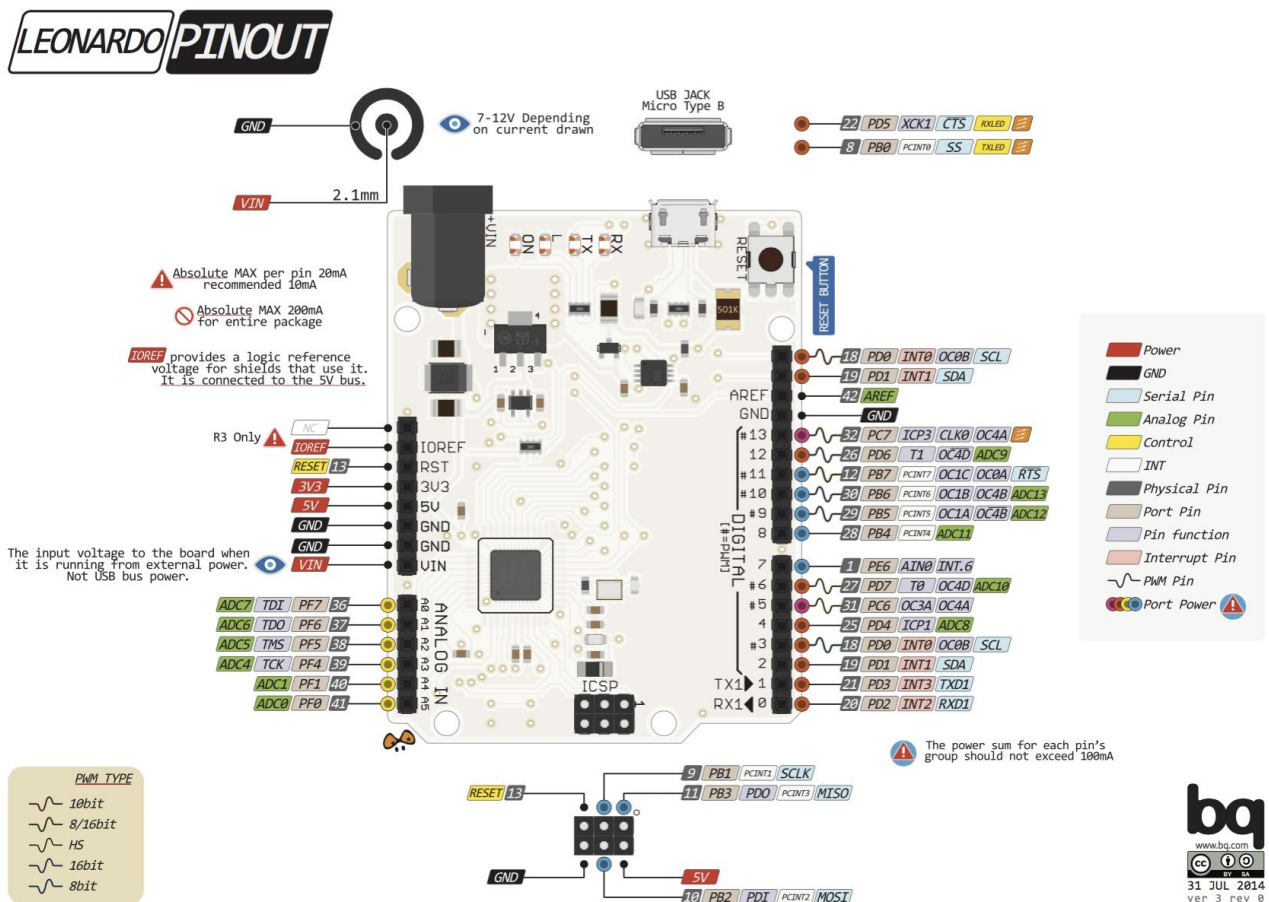


Figura 2.4: Pinout Arduino Leonardo[1]

2.1.4. Módulo SD

El módulo genérico para lectura y escritura de tarjetas micro SD se comunica a través del puerto SPI y se alimenta con 5Vcc. Permite la lectura y escritura de datos en una tarjeta micro SD, la que particularmente se escogió como una micro SD de 2G clase 4, en formato FAT16.

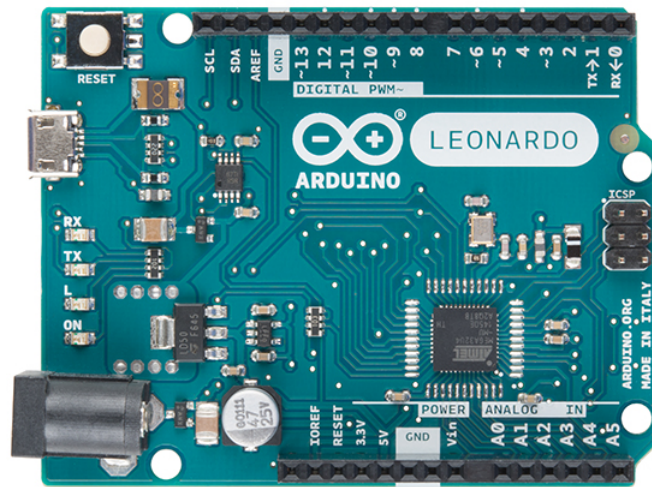


Figura 2.5: Arduino Leonardo

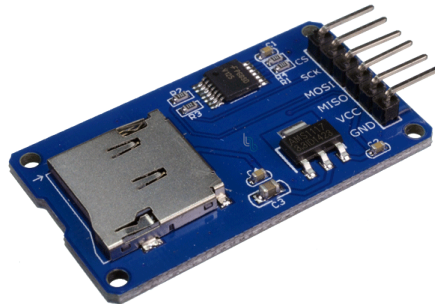


Figura 2.6: Módulo SD

2.1.5. QNavigator V1.6.3

El Kit EVB de Quectel, ofrece el software de configuración QNavigator v1.6.3. El programa es un entorno gráfico para el envío de comandos AT que simplifica la configuración de los módulos Quectel y las pruebas de conexiones para red y GPS.

Qnavigator es compatible con la mayoría de los kits de desarrollo Quectel, y permite trabajar con la red 2G, 3G, 4G y NB-IoT/LTE-M de acuerdo al módulo ensamblado en la placa

2.1.6. Arduino IDE V1.8.8

Arduino IDE permite programar el microcontrolador Atmega32u4 integrado en la placa Arduino Leonardo. A través de este software es posible cargar un archivo de instrucciones programadas en lenguaje C directamente en la memoria ROM del microcontrolador. El software ofrece la posibilidad de integrar librerías desarrolladas por terceros simplificando la integración de los diversos módulos. Por temas de compatibilidad de controladores de hardware, se decidió utilizar este software sobre Linux Ubuntu v18.10.

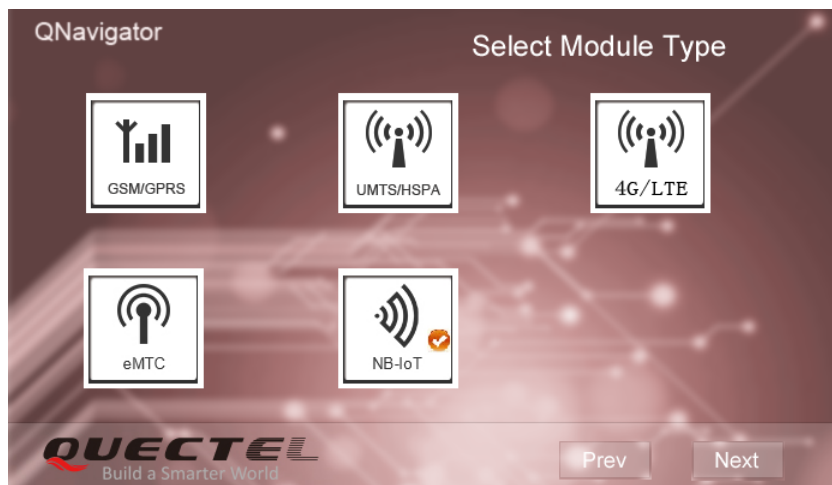


Figura 2.7: Programa QNavigator V1.6 sobre Windows 8.1

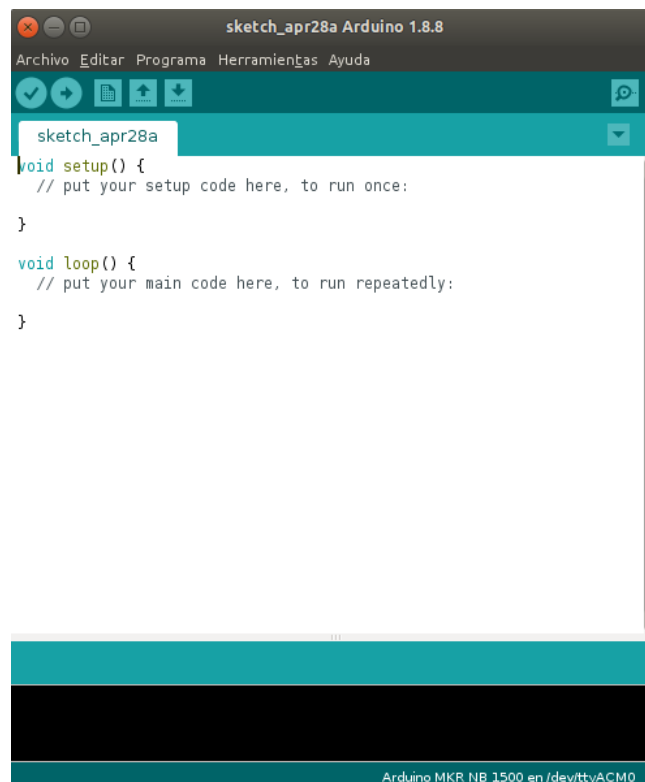


Figura 2.8: Arduino IDLE v1.8.8 sobre Linux Ubuntu v18.10

2.1.7. GTK Term v1.0

GTK Term es un software desarrollado para Linux Ubuntu que permite acceder a las conexiones seriales entre el computador y dispositivos externos. Ofrece envío y recepción de datos con el dispositivo conectado, mostrando en tiempo real la información que se trafica en el enlace. Es una excelente herramienta para verificar el funcionamiento de las conexiones, y al mismo tiempo detectar problemas de respuesta en los dispositivos.

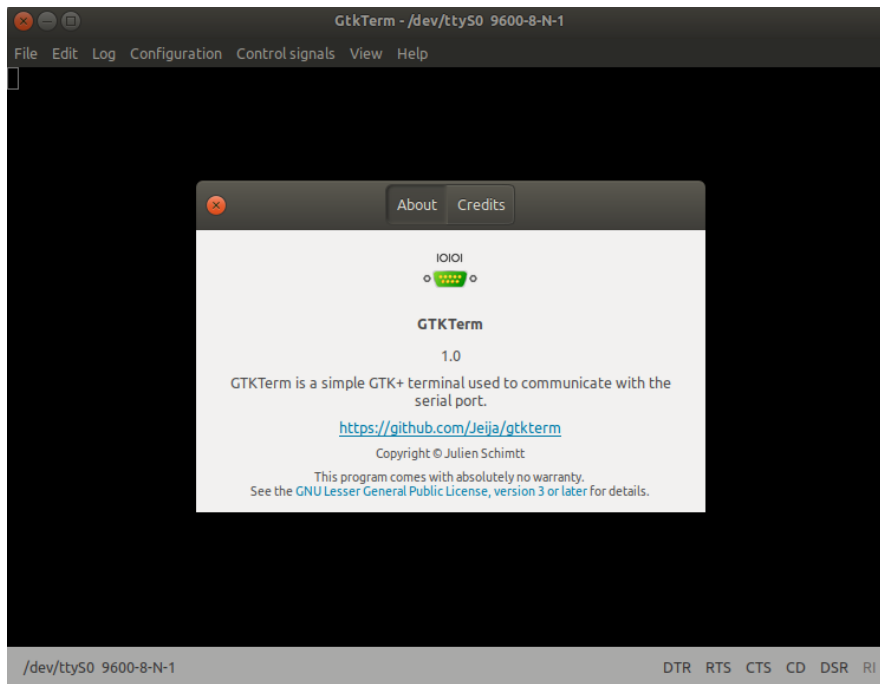


Figura 2.9: Programa GTK Term v1.0 sobre Linux Ubuntu 18.10

2.1.8. SD Card Formatter

SD Card Formatter es un programa para dar formato a tarjetas de memoria SD, SDHC y SDXC de acuerdo a la especificación del sistema de archivos SD creada por la *SD Asociación* (SDA). Ofrece una mejor compatibilidad de formato en comparación a las herramientas nativas de los sistemas operativos, a través el formato rápido en FAT16, altamente compatible con módulos de lectura y escritura SD.

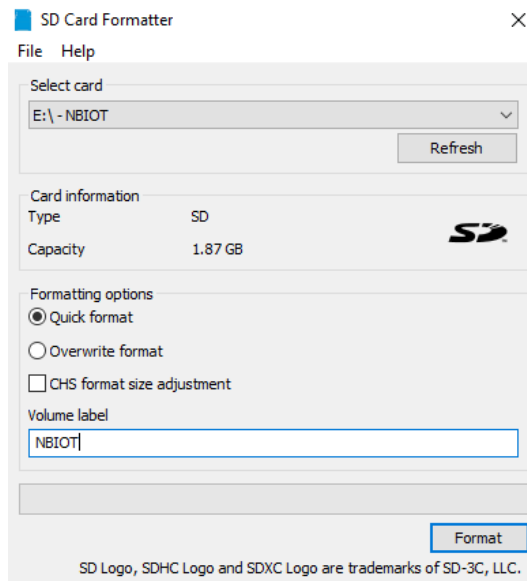


Figura 2.10: SD Card Formatter sobre Windows 8.1

2.2. Hardware y Software para la implementación del servidor

A continuación se describe el *software* utilizado en la implementación del servidor, correspondiente a las herramientas de configuración y programación del servidor y la API. El *hardware* utilizado no implica el uso de herramientas físicas, más bien apunta al concepto de servidor virtual, el cual fue arrendado al proveedor Digital Ocean.

2.2.1. Servidor Digital Ocean

DigitalOcean es un proveedor estadounidense de servidores virtuales privados lanzado el 2011 por Ben Uretsky en la ciudad de Nueva York[3]. DigitalOcean ofrece el servicio de arriendo de servidores, a los que denomina *Droplet*. A diferencia de los *web hosting*, el *Droplet* proporcionado es un servidor privado dado que la empresa no interviene en su instalación ni manejo, ofreciendo solamente el sistema operativo y repositorios correspondientes. De este modo, un usuario puede arrendar distintos *Droplets* y mediante tarjetas de red virtuales, crear una red privada local y administrar distintos servidores según la necesidad.

La ventaja de los servicios ofrecidos por DigitalOcean, y la razón por la que fue escogida para el desarrollo presente, es su flexibilidad para la elección de recursos y para el pago de éstos. Dentro de las opciones, ofrece diversos sistemas operativos y versiones de éstos, las que se muestran en la figura 2.11:

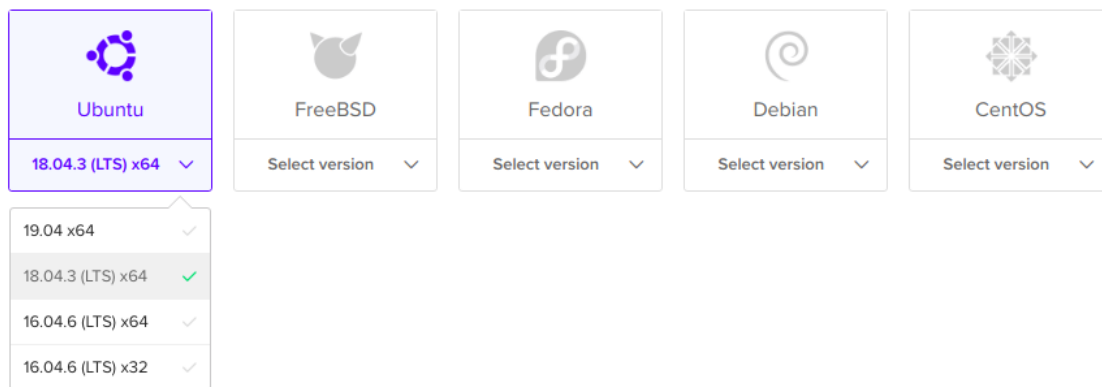


Figura 2.11: Sistemas operativos disponibles para el *Droplet*

Las capacidades del servidor también pueden ser escogidas según las necesidades de desarrollo, y en base a esto generar un plan de pago (ver figura 2.12) por el uso de servicio. Si bien, se ofrece un costo por uso mensual, el *Droplet* puede ser eliminado en cualquier momento pagando sólo el proporcional del costo según el período utilizado.

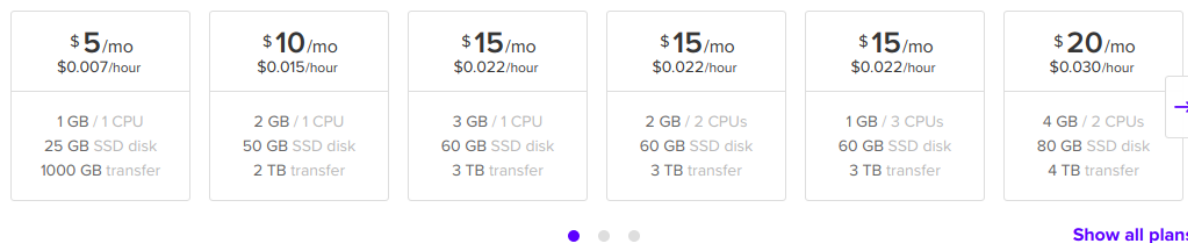


Figura 2.12: Capacidades y precios disponibles para el *Droplet*

2.2.2. Nginx

Nginx es un servidor web ligero y de alto rendimiento, que también puede ser usado como *proxy* para protocolos de correo electrónico[4]. Es software libre y totalmente compatible con sistemas *Unix*, en particular Linux Ubuntu. A pesar de existir Apache como competidor directo, se decidió utilizar Nginx pensando en la escalabilidad del prototipo y en los futuros desarrollos que pueden surgir a partir de éste, ya que de acuerdo a las condiciones actuales del diseño, no existían mayores requerimientos para realizar una elección específica. Algunos de los aspectos destacables de este servidor web son:

- Servidor de archivos estáticos, índices y autoindexado
- Soporte de HTTP y HTTP2 sobre SSL
- Servidores virtuales basados en nombre y/o en dirección IP
- Compatible con IPv6
- Tolerancia a fallos

2.2.3. Python/Flask

Flask es un *framework* simple y liviano escrito en Python y diseñado para crear aplicaciones web de manera rápida y sencilla. Está basado en la especificación WSGI de Werkzeug. Dado que no requiere herramientas ni librerías particulares, se le considera un *microframework*, razón por la que fue escogido para el desarrollo de este trabajo. Aunque los requisitos del servidor implementado no requieren de grandes funciones, Flask destaca por ofrecer características como:

- Servidor de desarrollo y depuración
- Soporte integrado para test unitario
- Envío de requerimientos RESTful
- Extensa documentación
- Tolerancia a fallos

Y por sobre todo, una de sus grandes ventajas es estar basado en Python, lenguaje de programación de uso sencillo y ampliamente documentado y soportado en la web.

Capítulo 3

Metodología de diseño, implementación y validación del plan de pruebas

3.1. Diseño del Prototipo

La frase previa a la implementación del prototipo, fue el diseño a nivel de bloques del sistema, el que debía satisfacer las necesidades del problema planteado. El objetivo principal del prototipo es obtener el valor de cobertura de la red NB-IoT para una ubicación específica. Teniendo la medida de cobertura y las coordenadas asociadas a esa medición, es posible realizar una caracterización de la red en términos de cobertura, que es uno de los objetivos del presente trabajo. Por lo tanto, el prototipo fue diseñado basado en los siguientes requerimientos:

- Obtener el valor de RSSI y las coordenadas geográficas asociadas a cada punto de medición
- Poseer un sistema de almacenamiento automático y extraíble para almacenar los datos
- Mantener un funcionamiento autónomo durante el tiempo que las pruebas lo requieran

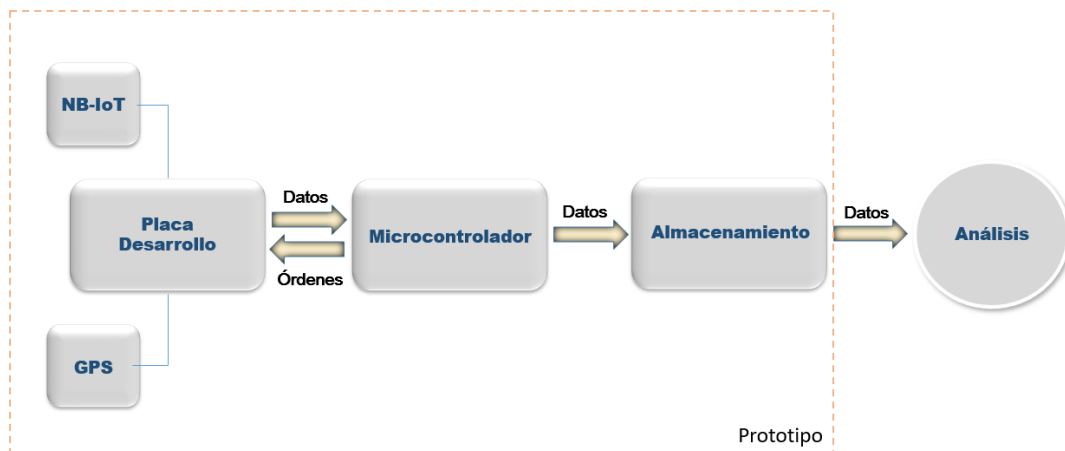


Figura 3.1: Diseño de Funcionalidades del Prototipo

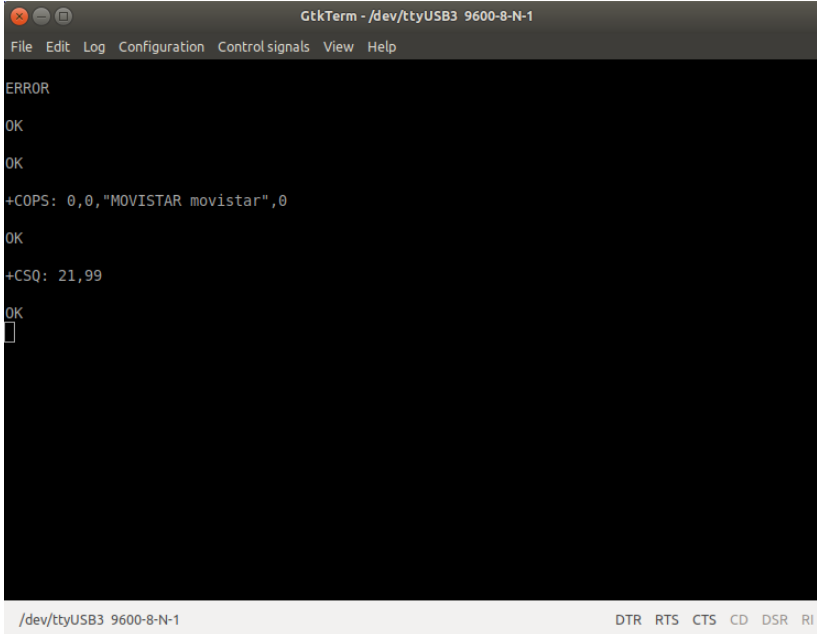
3.2. Implementación del prototipo

3.2.1. Verificación del módulo usando QNavigator

Mediante conexión USB, se conectó el Kit EVB de Quectel al computador y se accedió a su configuración utilizando el software QNavigator. Se configuró el *baudrate* en 9600, el puerto COM19 como canal de acceso al módulo BG96 y el puerto COM21 como canal de acceso al GPS.

3.2.2. Verificación de la Conexión Serial

Antes de realizar la conexión del módulo Quectel con el microcontrolador, se verificó el correcto funcionamiento de la conexión Serial utilizando GTK Term. Se entabló conexión utilizando el cable Serial-USB, conectando el puerto UART1 de la placa Quectel con el puerto USB del computador. Mediante GTK Term se ejecutaron los comandos AT de prueba, obteniendo la correcta respuesta por parte de la placa de desarrollo como se muestra a continuación:



```
GtkTerm - /dev/ttyUSB3 9600-8-N-1
File Edit Log Configuration Control signals View Help
ERROR
OK
OK
+COPS: 0,0,"MOVISTAR movistar",0
OK
+CSQ: 21,99
OK
█
/dev/ttyUSB3 9600-8-N-1 DTR RTS CTS CD DSR RI
```

Figura 3.2: Verificación de conexión Serial

3.2.3. Verificación del módulo SD utilizando Arduino Leonardo

Para verificar el correcto funcionamiento del módulo de almacenamiento de datos, se realizaron pruebas de lectura y escritura sobre una tarjeta Micro SD de 2GB a través de un módulo compatible con Arduino. Debido a que el módulo exige un correcto formato para la

tarjeta SD, fue necesario utilizar un software específico para el formateo de la memoria SD que eliminara los errores causados por el formato rápido de los sistemas operativos. Utilizando el programa SD Card Formatter, se le dio formato FAT16 y un tamaño de partición de 1.95GB. Utilizando la conexión SPI del Arduino, se realizaron pruebas de escritura y lectura de datos utilizando el siguiente código:

```
1
2 #include <SD.h>
3
4 File logFile;
5
6 void setup()
7 {
8     Serial.begin(9600);
9     Serial.print(F("Iniciando SD ..."));
10    if (!SD.begin(9))
11    {
12        Serial.println(F("Error al iniciar"));
13        return;
14    }
15    Serial.println(F("Iniciado correctamente"));
16 }
17
18
19 // Funcion que simula la lectura de un sensor
20 int readSensor()
21 {
22     return 0;
23 }
24
25 void loop()
26 {
27     // Abrir archivo y escribir valor
28     logFile = SD.open("datalog.txt", FILE_WRITE);
29
30     if (logFile) {
31         int value = readSensor;
32         logFile.print("Time(ms)=");
33         logFile.print(millis());
34         logFile.print(", value=");
35         logFile.println(value);
36
37         logFile.close();
38     }
39     else {
40         Serial.println("Error al abrir el archivo");
41     }
42     delay(500);
43 }
44 }
```

3.2.4. Conexiones

Una vez configurado el Kit EVB y verificado el correcto funcionamiento de la conexión serial, se procedió a implementar el sistema diseñado para la obtención de los datos de cobertura de manera automatizada. Tal como se detalla en el diagrama de la figura 3.3, la placa de desarrollo fue conectada con el microcontrolador via conexión serial(UART) utilizando los pines RX/TX de forma cruzada. Mediante el mismo protocolo de comunicación, el módulo BG96 se comunica con la placa de desarrollo a través del socket destinado para éste. El lector de tarjetas SD fue conectado al microcontrolador a través del puerto SPI. La antena del servicio LTE y del servicio GPS fueron conectadas directamente al módulo BG96 mediante el conector U.FL. Finalmente, tanto el Kit Quectel como el Arduino Leonardo, fueron conectados a una batería de litio de 5V como fuente de alimentación del prototipo.

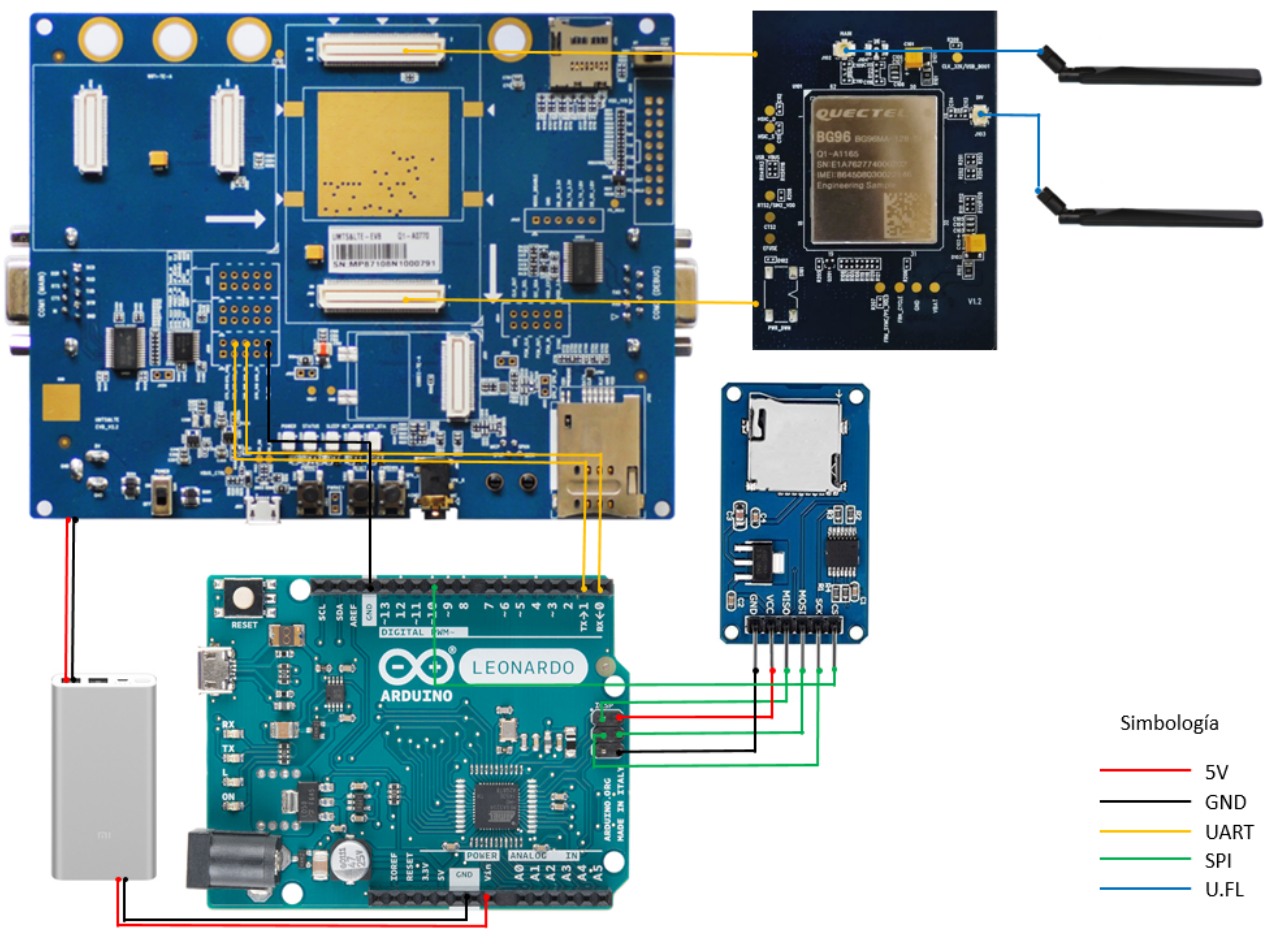


Figura 3.3: Diagrama de conexiones del prototipo

3.2.5. Conexión a la red GSM

La mejor opción para validar el prototipo en ausencia de la red NB-IoT, es mediante la toma de datos de cobertura en red GSM ya que el módulo Quectel BG96 también soporta

tecnología 2G. Para esto, se utilizó una tarjeta SIM Movistar con el objetivo de conectarse a la banda 1900 de la red GSM y se procedió a configurar el módulo de la siguiente manera:

```
AT+QCFG='nwscanseq',01           // Configura prioridad red GSM en la búsqueda
AT+QCFG='nwscanmode',1           // Configura en modo sólo GSM
AT+QCFG='band',1,8000000,8000000 // Configura en banda GSM 1900, NB 28, CatM 28
AT+QNWINFO                         // Solicita información de la red
```

De esta forma, se fuerza al módulo a conectarse únicamente y como primera opción a la tecnología de red seleccionada, en este caso GSM. Además se fuerza a al módulo a conectarse a la banda GSM 1900. Las bandas para NB-IoT deben ser configurada obligatoriamente en banda 28 ya que es la única banda sobre la cual opera en Chile la tecnología, y correspondiente al código <8000000>según el manual del proveedor. Dependiendo del proveedor, la banda GSM debe ser configurada de acuerdo a los siguientes códigos:

```
+QCFG: "band",<gsmbandval>,<catm1bandval>,<catnb1bandval>
<gsmbandval>
0          Sin Cambios
1          GSM 900MHz
2          GSM 1800MHz
4          GSM 850MHz
8          GSM 1900MHz
F          Cualquier Banda
<catm1bandval>
8000000   Banda 28
<catnb1bandval>
8000000   Banda 28
```

Figura 3.4: Códigos de bandas GSM para el módulo Quectel BG96

Una vez configurada la prioridad de conexión y las bandas deseadas, el módulo queda registrado en la red seleccionada y el comando AT+QNWINFO entrega la información de conexión, lo que permite verificar el correcto registro del dispositivo.

3.2.6. Medición de RSSI de red GSM

Una vez configurado los parámetros de conexión, es posible solicitar la información de la red correspondiente. En este caso, se obtiene el RSSI de la red a través del comando AT+CSQ, el que

entrega un valor para <rsi>y el <ber>de acuerdo al siguiente esquema de valores:

```
+CSQ:<rsi>,<ber>

<rsi>

0          -113dBm o menos
1          -111dBm
2...30     -109... -53dBm
31         -51dBm o más
99         desconocidos/no detectables

<ber>

Bit Error Rate del canal en porcentaje
```

Figura 3.5: Información entregada por el comando AT+CSQ

3.2.7. Obtención de coordenadas geográficas vía GPS

Las coordenadas geográficas se obtienen gracias a la antena GPS conectada al prototipo. Primero se debe iniciar el sistema GPS mediante al comando AT+QGPS=1, y una vez iniciado, es posible obtener las coordenadas de posición mediante el comando AT+QGPSLOC=2. Existen distintos formatos en los que puede ser solicitada la información, pero para este caso en particular el formato es el siguiente:

```
+QGPSLOC: <UTC>,<latitude>,<longitude>,<hdop>,<altitude>,<fix>,<cog>,<spkm>,<spkn>,<date>,<nsat>

<latitude>

Formato    (-)dd.ddddd
dd.ddddd  -89.99999-89.99999 (degree)

<longitude>

Formato    (-)dd.ddddd
dd.ddddd  -179.99999-179.99999 (degree)
```

Figura 3.6: Información entregada por el comando AT+QGPSLOC?

Como se puede observar en la Figura 3.6, la información entregada por el comando incluye hasta once parámetros relacionados con la geolocalización. Dado que el interés es obtener ubicación geográfica del RSSI almacenado, sólo son importantes el valor de <latitude>y <longitude>, los que deben ser filtrados posteriormente para su procesamiento.

3.2.8. Automatización del prototipo de muestreo

La obtención de datos de posición y de RSSI fue automatizadas mediante el microcontrolador del Arduino Leonardo. A través de éste se envían los comandos AT al módulo BG96 utilizando

la conexión serial. El módulo responde a través de la misma conexión serial al Arduino, quien mediante un módulo de lectura/escritura SD guarda los datos en una Micro SD. El proceso anterior, se programó tal como se presenta a continuación:

```
1
2 #include <SD.h>
3
4 char buffer[200];
5 int pos = 0;
6
7 void setup() {
8     Serial.begin(115200);
9     Serial1.begin(115200);
10    Serial1.println("AT+QGPS=1");
11    if (!SD.begin(10)) {
12        return;
13    }
14    memset(buffer, '\0', sizeof(buffer));
15 }
16
17 void loop() {
18     delay(100);
19     Serial1.println("AT+QGPSLOC=2");
20     delay(1000);
21     Serial1.println("AT+CSQ");
22     delay(1000);
23
24     while(Serial1.available()>0) {
25         char c = Serial1.read();
26         buffer[pos] = c;
27         pos++;
28         if (c == 10) {
29             File dataFile = SD.open("DATALOG.TXT", FILE_WRITE);
30             if (dataFile) {
31                 dataFile.println(buffer);
32                 dataFile.close();
33                 Serial.println(buffer);
34             }
35             memset(buffer, '\0', sizeof(buffer));
36             pos = 0;
37         }
38     }
39 }
```

3.3. Diseño del servidor

En el proceso de definir los requerimientos del servidor a implementar, se realizó un análisis de los servidores disponibles para la realización de pruebas. Estos corresponden al servidor de eco provisto por Quectel y a las Api's públicas de libre acceso disponibles en internet, de las cuales se escogió la Api de www.feriadosapp.com para su análisis.

3.3.1. Servidor de Echo de Quectel

Quectel pone a disposición de los usuarios un servidor alojado en la dirección **220.180.239.212** puerto **8009**, el que ofrece pruebas de conexión mediante socket TCP/UDP e intercambio datos mediante una función de eco, la que devuelve al cliente el mensaje enviado.

3.3.2. API's Públicas

Existen diversas API's en la web que permiten realizar requerimientos de tipo GET y pueden ser utilizadas para probar el funcionamiento de desarrollos privados. Una de estas API's es la ofrecida por www.feriadosapp.com, la que entrega información de los feriados anuales de Chile y la ley que los respalda, en formato *json*.

3.3.3. Diseño de requerimientos

Si bien las opciones disponibles para uso público ofrecen pruebas de requerimientos GET y la función eco, no existe posibilidad de obtener información respecto a los accesos realizados de manera exitosa o fallida. Por esta razón, se vuelve un requisito indispensable implementar un servidor que permita un total control de las conexiones entrantes y que al mismo tiempo, realice las funciones ofrecidas por el servidor de Quectel y la API de www.feriadosapp.com, ya que son necesarias para verificar el correcto funcionamiento de la red. Se definen entonces los requisitos del servidor privado declarando la necesidad de:

- API que responda a un requerimiento GET con un mensaje de prueba.
- API tipo eco, que responda a un requerimiento con el mismo texto ingresado.
- Registro de conexiones entrantes exitosas y fallidas.

3.4. Implementación del servidor

Considerando los requerimientos del servidor y el diseño planteado, un servidor de bajos recursos es suficiente para cumplir con las funciones a implementar, y por lo tanto su elección fue determinada principalmente por la optimización de costos.

Se escogió el Cloud de Digital Ocean por su modalidad de arriendo de servidores por hora, lo cual es una herramienta muy útil al momento de realizar pruebas y prototipaje, ya que los servidores pueden ser configurados, dados de baja, o ampliados con total libertad y pagando solamente las horas de uso. Además, ingresando una tarjeta de crédito al momento del registro el cobro se factura automáticamente.

3.4.1. Elección y arriendo del servidor

Antes que todo, se debe estar en posesión de una cuenta de usuario de Digital Ocean, la cual se puede crear en <https://www.digitalocean.com/>. Alternativamente es posible realizar el registro con una cuenta de Google o Github.

Habiendo ingresado a la cuenta de usuario y aceptado las condiciones de registro y pagos, se procedió a crear un nuevo proyecto en la pestaña izquierda como se muestra en la figura 3.7

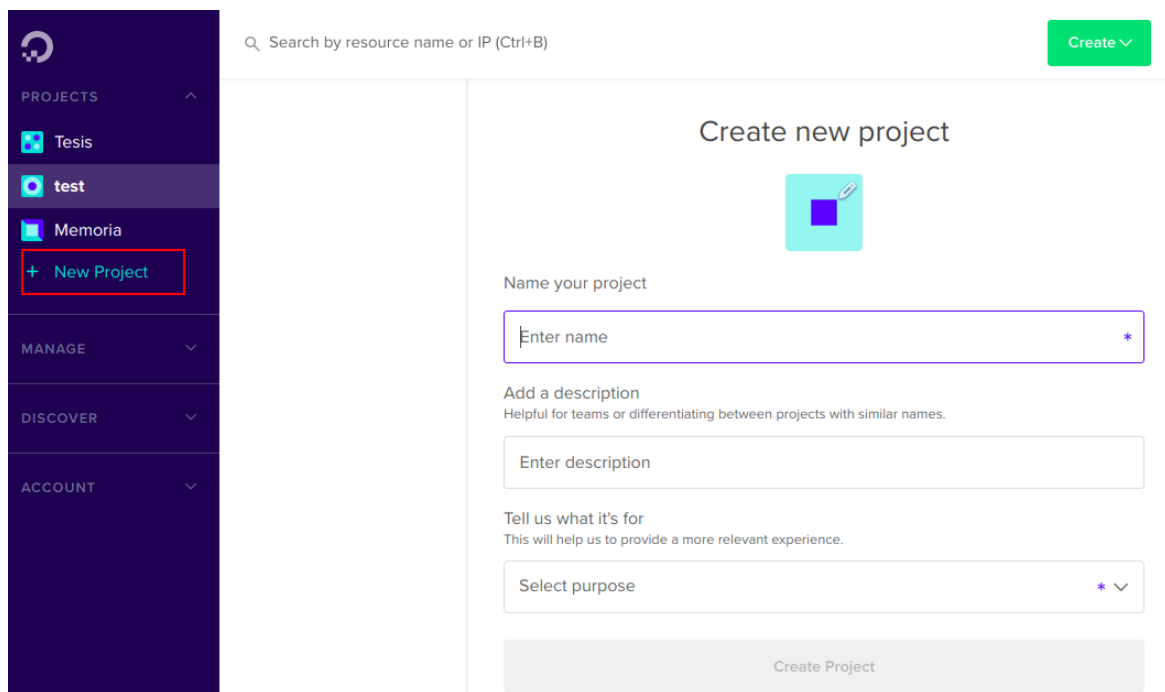


Figura 3.7: Creando un nuevo proyecto

Al hacer click en *New Project* se despliega un formulario para ingresar la información básica del proyecto, en este caso se nombró *Memoria*. El campo descripción y la ventana que se despliega tras la confirmación pueden ser omitidas.

Una vez creado el proyecto, este debiese aparecer en el panel izquierdo. Al seleccionarlo se despliega la opción *Get Started with a Droplet* como se muestra en la figura 3.8, lo que da inicio a la elección y configuración del servidor.

El primer paso es seleccionar el sistema operativo del servidor(ver figura 3.9), que en este caso se escogió como Ubuntu 18.04. A continuación, se seleccionó el plan *Standard*, 25GB de almacenamiento SSD y 1GB RAM para el servidor(ver figura3.10). Luego se seleccionó la ubicación física del servidor en San Francisco, ya que dentro de las otras opciones disponibles (ver figura 3.11) fue la que registró un menor *ping* para el proveedor de internet utilizado.

Finalmente, se escogió el método de autenticación inicial (ver figura 3.12 a través de clave auto-generada, la que fue enviada al correo electrónico ingresado al momento del registro.

Un email confirmando el proceso fue enviado al correo de registro tal como se muestra en la figura 3.13, con lo cual se validó la compra y además se obtuvo las claves para acceder al servidor mediante conexión SSH.

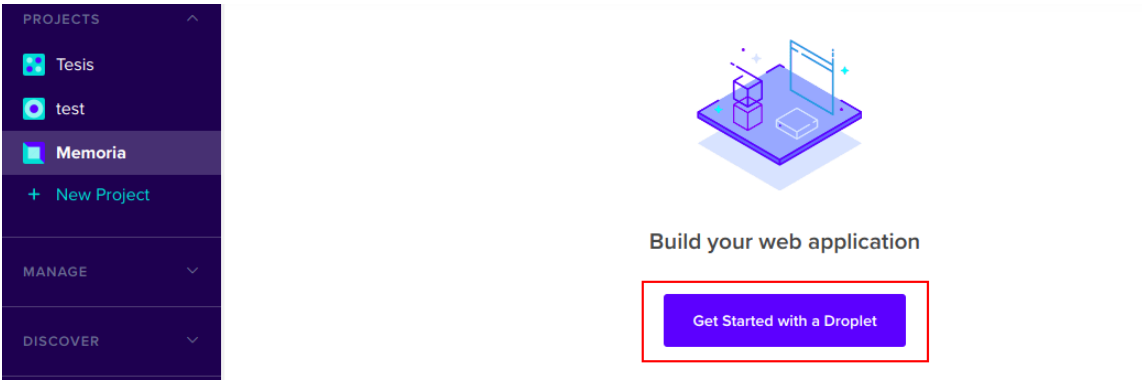


Figura 3.8: Agregando un nuevo Droplet

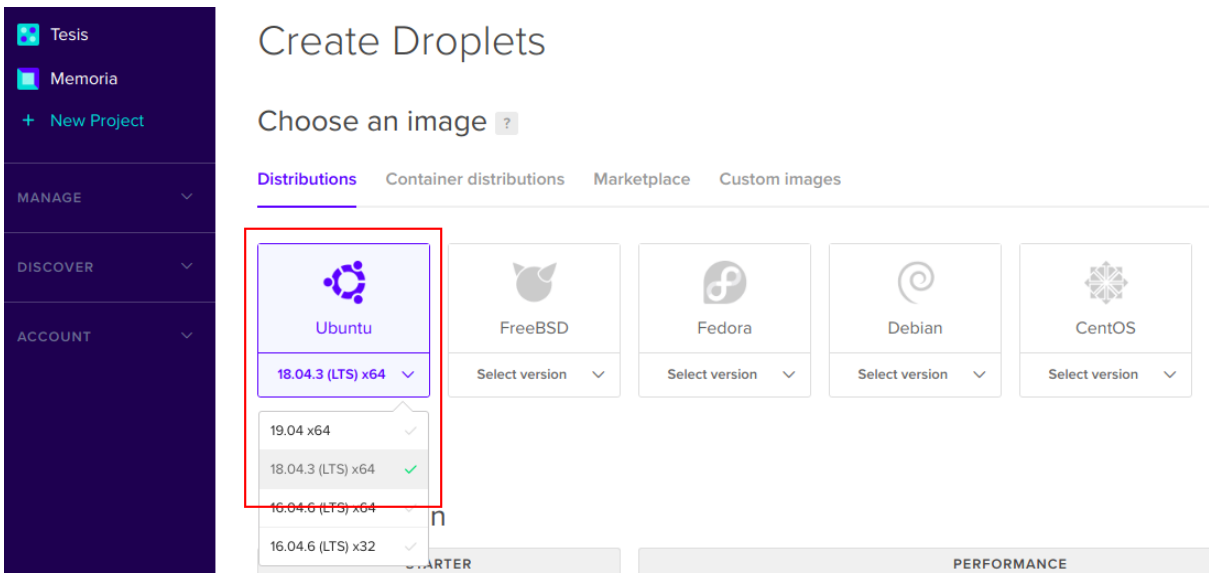


Figura 3.9: Escogiendo características del servidor

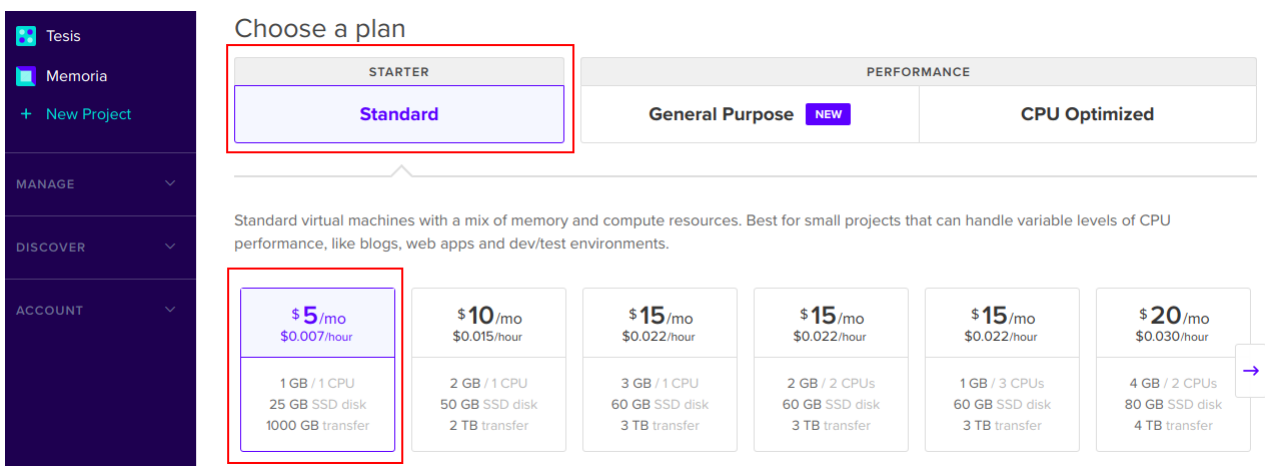


Figura 3.10: Escogiendo características de la CPU

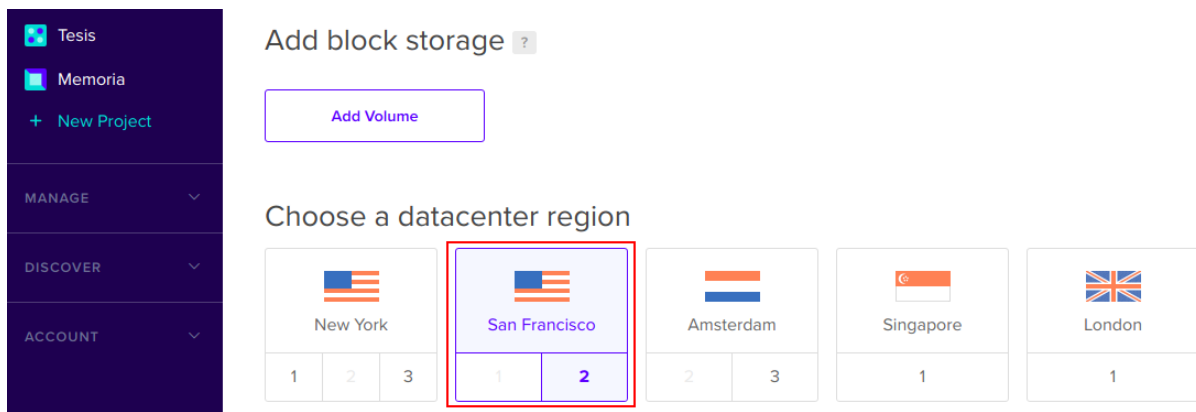


Figura 3.11: Escogiendo la ubicación física del servidor

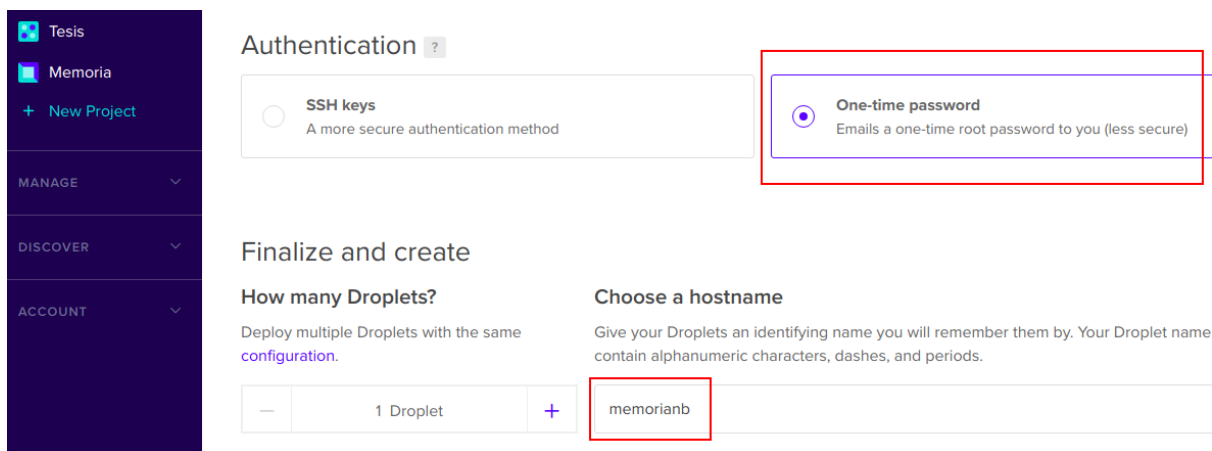


Figura 3.12: Escogiendo el método de autenticación inicial

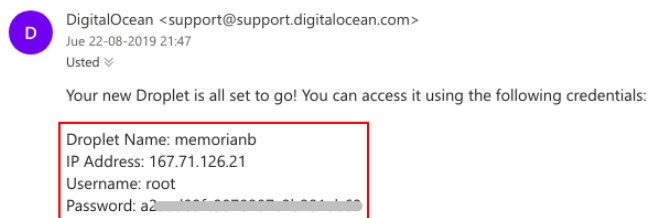


Figura 3.13: Correo electrónico confirmando el proceso y con la clave de inicio

3.4.2. Preparación del entorno

Para acceder al servidor se debe iniciar una conexión SSH, para lo que se utilizó el cliente propio de Ubuntu. En caso de usar Windows, se debe utilizar un cliente SSH externo como *PuTTY* o similar. La conexión se realizó abriendo un terminal y ejecutando el comando:

```
ssh root@167.71.126.21
```

Al ingresar el comando se realiza una solicitud de conexión en modo root con el servidor alojado en la ip ingresada, la que se concreta ingresando la contraseña enviada en el email de confirmación. Una vez establecida la conexión, se desplegó la siguiente ventana:

```
tabu@Tabu-Pc: ~
Archivo Editar Ver Buscar Terminal Ayuda
tabu@Tabu-Pc:~$ ssh root@167.71.126.21
The authenticity of host '167.71.126.21 (167.71.126.21)' can't be established.
ECDSA key fingerprint is [REDACTED]
Are you sure you want to continue connecting (yes/no)? [ ]
```

Figura 3.14: Alerta por acceso desconocido al Host

El mensaje desplegado es una advertencia que se generó debido a que es la primera vez que el cliente SSH accedió al Host solicitado. En las siguientes conexiones esta advertencia no volvió a aparecer.

Una vez aceptada la conexión, se realizó el cambio de contraseña ingresando una vez la clave autogenerada, y luego, ingresando dos veces una nueva contraseña (ver figura 3.15)

```
Last login: Fri Aug 23 01:56:08 2019 from 181.75.182.87
Changing password for root.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password: [ ]
```

Figura 3.15: Cambio de contraseña

Habiendo establecido la conexión vía SSH, se procedió a preparar el entorno realizando las configuraciones básicas al servidor. Primero se actualizaron los repositorios ejecutando:

```
# sudo apt-get update && apt-get upgrade
```

Además, se debe actualizar el archivo de sistema **host** agregando **167.71.126.21 memorianb** como tercera línea en el archivo. Para esto, se accedió mediante el editor de texto *Nano* ejecutando:

```
nano /etc/hosts
```

y se agregó la línea de código mencionada. Para salir del editor se debe ejecutar **ctrl+x** y para confirmar la edición **y+enter**. De este modo, el archivo resultante es el siguiente:

```
127.0.1.1 memorianb memorianb
127.0.0.1 localhost
167.71.126.21 memorianb

fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Luego, es necesario configurar la zona horaria del servidor, para lo cual se ejecutó el comando

tzdata. Tras ejecutarlo, se ingresó mediante las teclas de navegación la zona horaria como **América** y la ciudad como **Santiago**.

```
dpkg-reconfigure tzdata
```

Finalmente, es posible visualizar la correcta configuración de la zona horaria ejecutando:

```
date
```

3.4.3. Instalación y configuración Nginx

Para la instalación de Nginx, se debe cumplir con los siguientes requisitos:

- Servidor con Ubuntu 18.04 instalado
- Usuario *non-root* con privilegios *sudo*
- Zona horaria configurada y actualizada según 3.4.2

Para comenzar la instalación, es necesario crear el usuario *non-root* con privilegios *sudo*. Se creó un usuario nombrado **tabu** ejecutando:

```
# sudo adduser tabu  
# sudo adduser tabu sudo
```

Una vez creado el usuario se realizó el acceso SSH como se indicó en 3.4.2, y en adelante la instalación y configuración se realizó siempre desde dicho usuario.

Primero se instaló Nginx, ejecutando previamente la actualización de repositorios:

```
$ sudo apt update  
$ sudo apt install nginx
```

Nginx queda listado por defecto como un servicio con *UFW (Uncomplicated Firewall)* por lo que es importante agregarlo a la lista de excepciones del cortafuegos. Se consultó la lista de servicios mediante:

```
$ sudo ufw app list
```

Obteniendo como respuesta la lista de servicios bajo cortafuegos:

```
Available applications:  
Nginx Full  
Nginx HTTP  
Nginx HTTPS  
OpenSSH
```

Donde cada servicio corresponde a:

- Nginx Full: Este perfil abre el puerto 80 (puerto común para tráfico web no encriptado) y el puerto 443 (para tráfico web encriptado TLS/SSL)
- Nginx HTTP: Este perfil abre sólo el puerto 80
- Nginx HTTPS: Este perfil abre sólo el puerto 443

De este modo se habilitó el perfil Nginx HTTP típico, ya que no se configuró el SSL para servidor seguro. Se debe habilitar también la conexión OpenSSH, para lo que se ejecutó:

```
$ sudo ufw allow 'Nginx HTTP'  
$ sudo ufw allow 'Open SSH'
```

Se consultó el estado del cortafuegos para verificar la habilitación de las conexiones HTTP y SSH ejecutando:

```
$ sudo ufw status
```

Tras esto, el estado arrojó que el cortafuegos estaba deshabilitado. Se procedió entonces a habilitar el UFW mediante:

```
$ sudo ufw enabled
```

Finalmente, se consultó el estado del servidor *Nginx*:

```
$ systemctl status nginx
```

Con esto, el servidor quedó habilitado y la respuesta obtenida ante la consulta del estado de Nginx se observa en la figura 3.16

```
● nginx.service - A high performance web server and a reverse proxy server  
  Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: en  
  Active: active (running) since Wed 2019-08-21 23:51:02 -04; 13min ago  
    Docs: man:nginx(8)  
  Main PID: 2899 (nginx)  
    Tasks: 2 (limit: 1152)  
   CGroup: /system.slice/nginx.service  
           └─2899 nginx: master process /usr/sbin/nginx -g daemon on; master_pro  
             └─2902 nginx: worker process
```

Figura 3.16: Servidor Nginx habilitado

Finalmente, el servidor puede ser verificado ingresando en el navegador web la ip **167.71.126.21**, mostrando el mensaje que se observa en la figura 3.17

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Figura 3.17: Servidor Nginx Web

3.4.4. Instalación y configuración Flask

Para comenzar la instalación y configuración de Flask, se debe haber completado el proceso de instalación anterior, cumpliendo a este punto los siguientes requisitos:

- Servidor con Ubuntu 18.04 instalado
- Usuario *non-root* con privilegios *sudo*
- Nginx instalado y configurado según el punto 3.4.3.

Primero, se debe instalar los repositorios de Ubuntu necesarios para la correcta configuración. Se debe instalar el administrador de paquetes de Python3 pip y los paquetes de desarrollo necesarios para uWSGI, lo que se realizó ejecutando:

```
$ sudo apt update
$ sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev
python3-setuptools
```

Es necesario crear un ambiente virtual para aislar la aplicación de Flask del resto de los archivos de Python en el sistema y de este modo, evitar errores por acceso a los recursos de Python. Se procedió a crear un nuevo directorio en la dirección **/home/** moviendo el entorno al directorio actual mediante:

```
$ mkdir /memorianb $ cd /memorianb
```

Luego, se creó el ambiente virtual dentro del directorio actual, alojado en **/home/memorianb** y ejecutando:

```
$ python3.6 -m venv memorianbenv
```

De esta forma, se creó una copia del pip de Python en el directorio actual, el cual antes de comenzar cualquier instalación debe ser activado mediante:

```
$ source memorianbenv/bin/activate
```

Estando dentro del ambiente, se instaló **Wheel** procurando usar pip en vez de pip3, y en adelante se utilizó dicha sentencia para todas las instalaciones locales:

```
(memorianb) $ pip install wheel
```

Luego, se instaló **Flask** y **uWSGI** ejecutando:

```
(memorianb) $ pip install uwsgi flask
```

Una vez instalado Flask, se procedió a crear la primera aplicación de prueba. Para esto se creó un archivo **memorianb.py** en la carpeta del ambiente virtual utilizando el editor de texto nano mediante:

```
(memorianb) $nano /memorianb/memorianb.py
```

Se creó una aplicación que muestra un mensaje de bienvenida y se configuró el color de la fuente y tamaño;

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return <h1 style='color:blue'>Hello Mundito!</h1>"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Para el correcto funcionamiento de la aplicación, se debe permitir el tráfico por el puerto 5000 configurando el cortafuegos mediante:

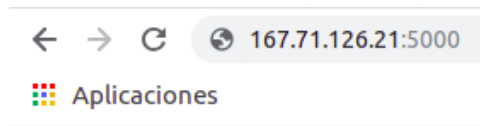
```
(memorianb) $ sudo ufw allow 5000
```

Finalmente, se ejecutó la aplicación con Python:

```
(memorianb) $ python memorianb.py
```

Mientras la aplicación esté siendo ejecutada por Python, es posible visualizar el mensaje de bienvenida ingresando la dirección IP del servidor y puerto del proceso como **167.71.126.21:5000**, de modo que se desplegó el mensaje mostrado en la figura 3.18

Una vez que Flask está funcionando correctamente en el puerto 5000, el siguiente paso es crear el archivo que servirá como punto de entrada para la aplicación diciéndole al server uWSGI cómo interactuar con ésta. De esta forma, se creó el archivo **wsgi.py** ejecutando:



Hello Mundito!

Figura 3.18: Servidor Flask Web

```
(memorianb) $ nano /memorianb/wsgi.py
```

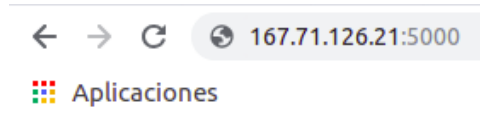
Se debe importar la instancia flask de la aplicación `memorianb.py`, escribiendo en el archivo `wsgi.py`:

```
from memorianb import app
if __name__ == "__main__":
    app.run()
```

Se especificó el puerto de entrada y el protocolo, en este caso el puerto es **5000**, y el protocolo es **HTTP**:

```
(memorianb) $ uwsgi -socket 0.0.0.0:5000 -protocol=http -w wsgi:app
```

En este punto, el servidor aún debería desplegar el mensaje de bienvenida en la IP y el puerto establecido, lo cual se verificó visitando a través del navegador la dirección **167.71.126.21:5000**. Se desplegó el mismo mensaje tal como se observa en la figura 3.19.



Hello Mundito!

Figura 3.19: Servidor Flask mediante WSGI

Una vez verificado el correcto funcionamiento del servidor, se desactivó el entorno virtual ejecutando:

```
(memorianb) $ deactivate
```

Si bien uWSGI está sirviendo correctamente a la aplicación, es necesario crear un archivo de configuración con algunos parámetros para robustecer el sistema. Se creó en el mismo directorio el

archivo **memorianb.ini**:

```
nano /memorianb/memorianb.ini
```

Y se escribió en el archivo las siguientes configuraciones:

```
[uwsgi]
module = wsgi:app

master = true
processes = 5

socket = memorianb.sock
chmod-socket = 660
vacuum = true

die-on-term = true
```

Para permitir que Ubuntu inicie automáticamente uWSGI y sirva la aplicación de Flask, se creó el archivo **memorianb.service** en la carpeta de sistema alojada en `/etc/systemd/system/` :

```
(memorianb) $ sudo nano /etc/systemd/system/memorianb.service
```

Y se configuró escribiendo las siguientes líneas:

```
[Unit]
Description=uWSGI instance to serve memorianb
After=network.target

[Service]
User=tabu
Group=www-data
WorkingDirectory=/home/tabu/memorianb
Environment='PATH=/home/tabu/memorianb/memorianbenv/bin'
ExecStart=/home/tabu/memorianb/memorianbenv/bin/uwsgi -ini
memorianb.ini

[Install]
WantedBy=multi-user.target
```

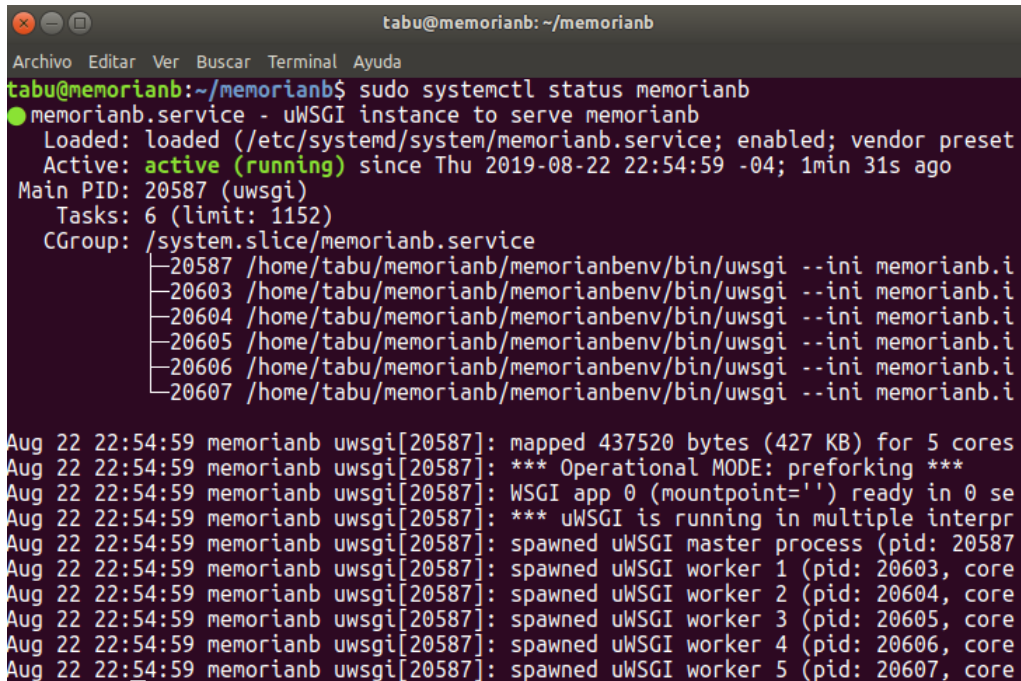
Finalmente, se inició y habilitó el servicio uWSGU ejecutando:

```
$ sudo systemctl start memorianb
$ sudo systemctl enable memorianb
```


Para verificar el estado del servidor y comprobar su correcto funcionamiento, se ejecutó:

```
$ sudo systemctl status memorianb
```

Mostrando como se observa en la figura 3.20 el proceso activo y corriendo perfectamente.



```
tabu@memorianb:~/memorianb
Archivo Editar Ver Buscar Terminal Ayuda
tabu@memorianb:~/memorianb$ sudo systemctl status memorianb
● memorianb.service - uWSGI instance to serve memorianb
   Loaded: loaded (/etc/systemd/system/memorianb.service; enabled; vendor preset
   Active: active (running) since Thu 2019-08-22 22:54:59 -04; 1min 31s ago
 Main PID: 20587 (uwsgi)
   Tasks: 6 (limit: 1152)
    CGroup: /system.slice/memorianb.service
            └─20587 /home/tabu/memorianb/memorianbenv/bin/uwsgi --ini memorianb.i
              └─20603 /home/tabu/memorianb/memorianbenv/bin/uwsgi --ini memorianb.i
                └─20604 /home/tabu/memorianb/memorianbenv/bin/uwsgi --ini memorianb.i
                  └─20605 /home/tabu/memorianb/memorianbenv/bin/uwsgi --ini memorianb.i
                    └─20606 /home/tabu/memorianb/memorianbenv/bin/uwsgi --ini memorianb.i
                      └─20607 /home/tabu/memorianb/memorianbenv/bin/uwsgi --ini memorianb.i

Aug 22 22:54:59 memorianb uwsgi[20587]: mapped 437520 bytes (427 KB) for 5 cores
Aug 22 22:54:59 memorianb uwsgi[20587]: *** Operational MODE: preforking ***
Aug 22 22:54:59 memorianb uwsgi[20587]: WSGI app 0 (mountpoint='') ready in 0 se
Aug 22 22:54:59 memorianb uwsgi[20587]: *** uWSGI is running in multiple interpr
Aug 22 22:54:59 memorianb uwsgi[20587]: spawned uWSGI master process (pid: 20587
Aug 22 22:54:59 memorianb uwsgi[20587]: spawned uWSGI worker 1 (pid: 20603, core
Aug 22 22:54:59 memorianb uwsgi[20587]: spawned uWSGI worker 2 (pid: 20604, core
Aug 22 22:54:59 memorianb uwsgi[20587]: spawned uWSGI worker 3 (pid: 20605, core
Aug 22 22:54:59 memorianb uwsgi[20587]: spawned uWSGI worker 4 (pid: 20606, core
Aug 22 22:54:59 memorianb uwsgi[20587]: spawned uWSGI worker 5 (pid: 20607, core
```

Figura 3.20: Servidor Flask Corriendo

El último paso en la configuración es decirle a Nginx en qué puertos debe escuchar. Se creó un nuevo bloque de configuración que reemplazó al bloque existente por defecto en la locación `/etc/nginx/sites-available/`, creando el archivo `memorianb`:

```
$ sudo nano /etc/nginx/sites-available/memorianb
```

Se señalaron los puertos de conexión y locación escribiendo en el archivo:

```
server {
    listen 80;
    listen [::]:80;

    location / {
        include uwsgi_params;
        uwsgi_pass unix:/home/tabu/memorianb/memorianb.sock;
    }
}
```

Para habilitar el bloque, se creó un *link* virtual del bloque de configuración hacia el directorio de sitios habilitados, alojado en `/etc/nginx/sites-enabled`:

```
$ sudo ln -s /etc/nginx/sites-available/memorianb /etc/nginx/sites-enabled
```

Para finalizar, se verificó la configuración en búsqueda de errores de sintaxis ejecutando:

```
$ sudo nginx -t
```

Dado que no arrojó ningún error, se reinició el proceso de Nginx ejecutando:

```
$ sudo systemctl restart nginx
```

Como las conexiones entrantes no usarán el puerto 5000, se bloqueó el tráfico en el cortafuegos y se permitió el acceso total a los servicios de Nginx mediante:

```
$ sudo ufw delete allow 5000  
$ sudo ufw allow 'Nginx Full'
```

Estando todo en orden, se verificó el acceso al servidor en el puerto 80 a través del navegador web, ingresando la IP:

167.71.126.21



Figura 3.21: Servidor Flask corriendo bajo Nginx

3.4.5. Creación de la API de consulta

Para cumplir los requisitos de diseño del servidor, se implementó una API para probar la conexión e intercambio de datos con el prototipo.

La primera ruta, devuelve un mensaje de prueba al hacer un requerimiento GET. Se implementó agregando las siguientes líneas de código al archivo **memorianb.py** alojado en el directorio **/home/tabu/memorianb/**:

```
@app.route("/consultar")  
def consultar():  
    return "Consultando Algo"
```

La segunda ruta, cumple la función de eco, por lo tanto al realizar una consulta se debe ingresar un texto como mensaje que será retornado al cliente como respuesta al requerimiento GET. Se agregó la siguiente función al archivo **memorianb.py** mencionado anteriormente:

```
@app.route('/echo/<msg>')
def echo(msg):
    return 'R: %s'% msg
```

En ambas funciones, se especificó la ruta de acceso en la sentencia **@app.route()**, por lo tanto la forma de acceder a éstas quedó definida como:

- **http://167.71.126.21/consultar**
- **http://167.71.126.21/echo/<insertar-mensaje>**

3.5. Validación del Prototipo y Servidor en red GSM

Para validar el sistema implementado, se realizaron pruebas al sistema prototipo-servidor con el objetivo de verificar su correcto funcionamiento.

Por un lado, se comprobó la capacidad el prototipo para conectarse a la red y capturar datos de manera autónoma. Debido a que aún no existe una red NB-IoT en la cual realizar pruebas, se realizaron de manera análoga en la red 2G para poder validar los comandos de configuración y la toma de datos posición y RSSI.

Por otro lado, se realizaron pruebas al servidor consistentes en la conexión, envío y recepción de paquetes vía requerimientos HTTP. Para esto, se usó el prototipo y el programa QCOM de Quectel para el envío de comandos AT vía conexión Serial.

3.5.1. Toma de datos de posición y RSSI

La toma de datos de posición y RSSI se realizó en la comuna de Maipú, Región Metropolitana. Se recorrió la zona sur de la intersección de Av. Pajaritos con Av. Las Torres, ya que es una zona que se aleja de las antenas Movistar que rodean el sector. En la Figura 3.22 se muestra el terreno recorrido y la posición de las antenas más cercanas. Aunque el objetivo de esta etapa es sólo comprobar el funcionamiento del prototipo, se aprovechó la instancia para para evaluar algún tipo de interferencia o relación con las antenas que pudieran repercutir en la calidad de los datos adquiridos, y en función de esto se definió la zona a recorrer.

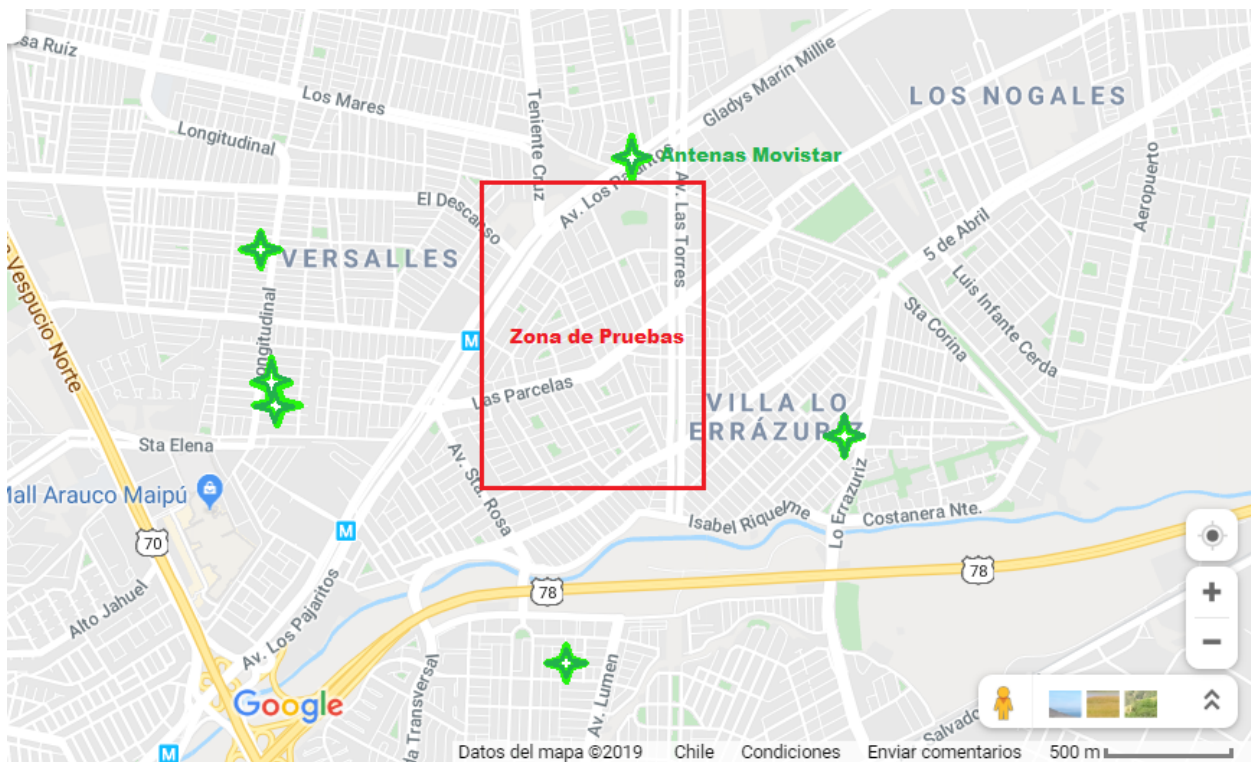


Figura 3.22: Zona de validación del prototipo

3.5.2. Visualización de datos mediante Mapa de Cobertura

Una forma interesante de caracterizar una red, es generar mapas de cobertura a partir de los datos de RSSI. *PowerMaps* es una extensión del software Excel de Microsoft Office que permite generar gráficos sobre mapas georeferenciados. Para esto, es necesario tener una base de datos con la información que se quiere estudiar y puntos geográficos asociados a éstos, los que pueden ser direcciones, locales, coordenadas, etc. La información debe estar organizada para ser interpretada fácilmente por Excel, por lo que fue necesario realizar un procesamiento inicial a los datos brutos entregados por el prototipo para generar un archivo `.txt` sólo con los datos de interés y separados por comas.

3.5.3. Creación de un archivo con los datos aislados

Para crear un archivo sólo con los datos necesarios para este trabajo, se escribió un *script* en Python que busca en el archivo `DATALOG.txt` la información correspondiente a la latitud, longitud y RSSI. Se recorrió el archivo mediante recurrencias de *strings*, volcando la información a listas temporales, las que finalmente son escritas en un archivo `output.txt`, separados por comas. El script utilizado se presenta a continuación:

```

1
2 entrada = open("DATALOG.TXT", "r")
3
4 latitudes = []
5 longitudes = []

```

```

6 coberturas = []
7
8 QGPSLOC = ""
9 CSQ = ""
10
11 for linea in entrada:
12
13     linea = linea.strip()
14
15     QGPSLOCBool = linea.find("+QGPSLOC:")
16     CSQBool = linea.find("+CSQ:")
17     ERROR = linea.find("ERROR:")
18
19     if (QGPSLOCBool == 0):
20         linea = linea.split(" ")
21         datos = linea[1].split(",")
22         latitudes.append(datos[1])
23         longitudes.append(datos[2])
24
25     if (ERROR == 5):
26         latitudes.append(-1)
27         longitudes.append(-1)
28
29     if (CSQBool == 0):
30         linea = linea.split(" ")
31         datos = linea[1]
32         coberturas.append(datos)
33
34 entrada.close()
35 salida = open("output.txt", "w")
36
37 i = 0
38 for i in range(0, len(coberturas)):
39
40     salida.write(str(latitudes[i])+","+str(longitudes[i])+","+str(coberturas[i])+"\n")
41
42 salida.close()

```

3.5.4. Conexión y toma de datos del servidor

Para validar la capacidad del prototipo de conectarse a un servidor y generar tráfico de datos, se realizaron pruebas vía socket TCP con el servidor implementado, lo que se realizó mediante comandos AT. Para el envío de comandos AT se debe utilizar un programa para realizar conexiones vía puerto Serial y como requisito, debe permitir el envío de Retorno de Carro <CR>y salto de línea <LF>, y también el envío de comandos de control, específicamente <Ctrl+Z>. Para esto, se utilizó el *software* QCOM de Quectel, siendo también útil el monitor serial de Arduino o GTK Term. Se priorizó el uso de este *software* por su ventaja de poder predeterminar series de comando AT en ASCII o Hexadecimal, y también exportar los datos a un archivo *.txt* con el registro de las pruebas realizadas. Tal como se observa en la figura 3.23, el rectángulo 1 muestra el <CR><LF>y <Ctrl+Z>guardados en código hexadecimal. El rectángulo 2 muestra activada la opción de agregar

<CR><LF>a cada comando AT.

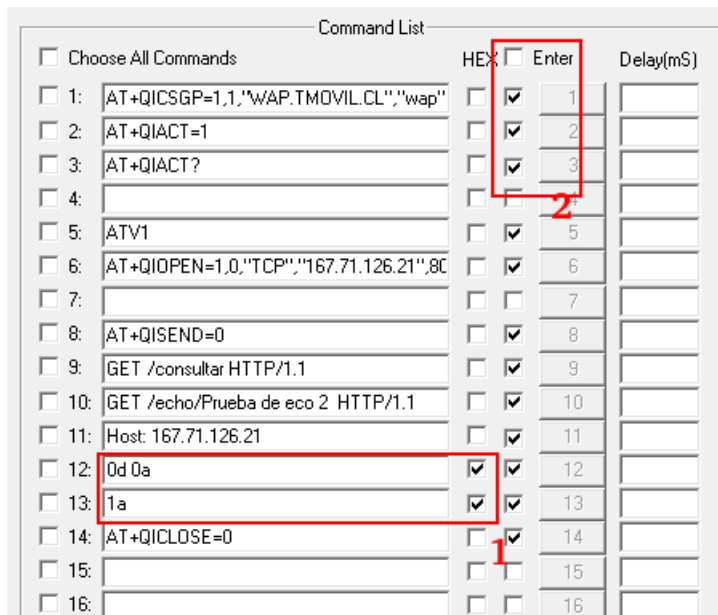


Figura 3.23: Comandos predeterminados en QCOM

3.5.5. Configuración del APN

Primero, se debe ingresar los datos del punto de acceso de la conexión de acuerdo al proveedor de servicios móviles. En este caso, se utilizó una tarjeta SIM de Movistar, cuyos datos son:

- APN: wap.tmovil.cl
- User: wap
- Pass: wap
- Auth: PAP

La configuración se realizó a través del comando **AT+QICSGP** de acuerdo a los parámetros descritos en la figura 3.24.

De esta forma, la serie de comandos AT para la configuración del APN quedó como:

```
AT+QICSGP=1,1,"WAP.TMOVIL.CL","wap","wap",1 // Configuración del APN
AT+QIACT=1 // Activa el perfil 1
AT+QIACT? // Solicita el estado del perfil
```

```

AT+QICSGP=<contextID>,<context_type>,<APN>,<username>,<password>,<authentication>

<contextID>

    1-16      Integer type

<context_type>

    1         IPV4
    2         IPV6

<APN>,<username>,<password>

    String type

<authentication>

    0         NONE
    1         PAP
    2         CHAP
    3         PAP or CHAP

```

Figura 3.24: Parámetros del comando AT+QICSGP

3.5.6. Apertura de socket TCP

Una vez configurado el APN, se tiene acceso a conexión a internet mediante red 2G, por lo que es posible iniciar la conexión con el servidor. Para esto, se abrió un *socket* **TCP** con el servidor alojado en **167.71.126.21** a través del puerto **80**. Se utilizó el comando **AT+QIOPEN**, que siguiendo la estructura especificada en la figura 3.25 corresponde a:

```

ATV1                                     // Formato de la respuesta

AT+QIOPEN=1,0,"TCP","167.71.126.21",80,0,1 // Activa el perfil

```

3.5.7. Envío de requerimientos GET

Una vez conectado al servidor mediante el socket TCP, se prepara el envío del requerimiento para probar la correcta conexión y funcionamiento del sistema. Para esto, se realizaron pruebas a las funcionalidades de la API a través de un requerimiento GET a **/consultar** y **/echo**. El requerimiento sigue la estructura de la figura 3.26.

```

Se prueba /consultar y de manera análoga /echo mediante el siguiente envío:
AT+QISEND=0                               // Responde con >
>GET /consultar HTTP/1.1                  // Finalizar con <CR><LF>
Host: 167.71.126.21                       // Finalizar con <CR><LF>
                                           // <CR><LF>
                                           // <CTRL+Z>

```

```

AT+QIOPEN=<contextID>,<connectID>,<service_type>,<IP>/<domain>,<remote_port>,<local_port>,<access>

<contextID>
    1-16      Integer type

<connectID>
    0-11      Integer type

<service_type>
    "TCP"     TCP connection as client
    "UDP"     UDP connection as client

<IP>/<domain>
    String type

<remote_port>
    0-65535   Remote port server, 80 HTTP

<local_port>
    0         Automatic

<access>
    0         Buffer access
    1         Direct push mode
    2         Transparent access mode

```

Figura 3.25: Parámetros del comando AT+QIOPEN

```

GET /<API> HTTP/1.1[enter]
Host: <ip> or <domain>[enter]
[enter]

<API>

    /consultar   Devuelve mensaje de prueba
    /echo/<msg>  Devuelve mensaje ingresado en <msg>

```

Figura 3.26: Parámetros del requerimiento GET

Capítulo 4

Resultados y Análisis

4.1. Prototipo final ensamblado

El primer resultado obtenido, es el prototipo definitivo, ensamblado y configurado para la toma de datos. Se implementó siguiendo la misma estructura planificada en la etapa de diseño, y cumpliendo todos los requerimientos definidos previamente. En la figura 4.1 se presenta el sistema completo, alimentado con una batería de 2000mAh y completamente autónomo.

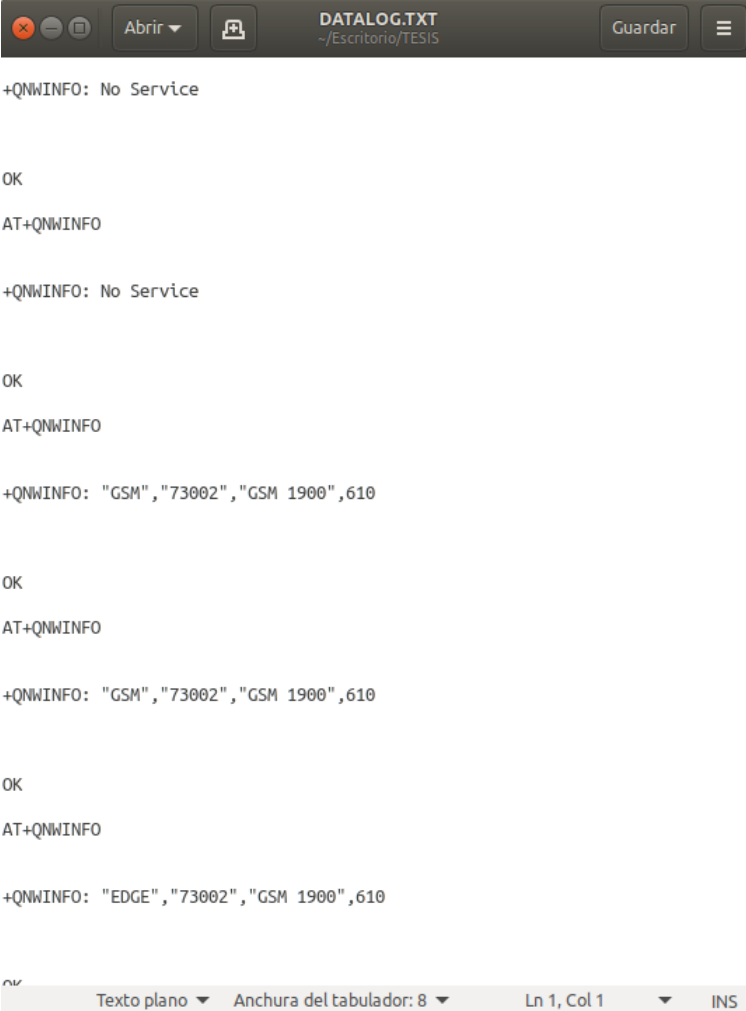


Figura 4.1: Prototipo final ensamblado

4.2. Mediciones de cobertura y posición sobre red GSM

4.2.1. Conexión a la red GSM

Tras la configuración y registro del módulo en la red GSM, se realizaron pruebas para verificar el éxito y la estabilidad de conexión, resultados que son presentados en la Figura 4.2



```
DATALOG.TXT
~/Escritorio/TESIS
Guardar

+QNWINFO: No Service

OK
AT+QNWINFO

+QNWINFO: No Service

OK
AT+QNWINFO

+QNWINFO: "GSM", "73002", "GSM 1900", 610

OK
AT+QNWINFO

+QNWINFO: "GSM", "73002", "GSM 1900", 610

OK
AT+QNWINFO

+QNWINFO: "EDGE", "73002", "GSM 1900", 610

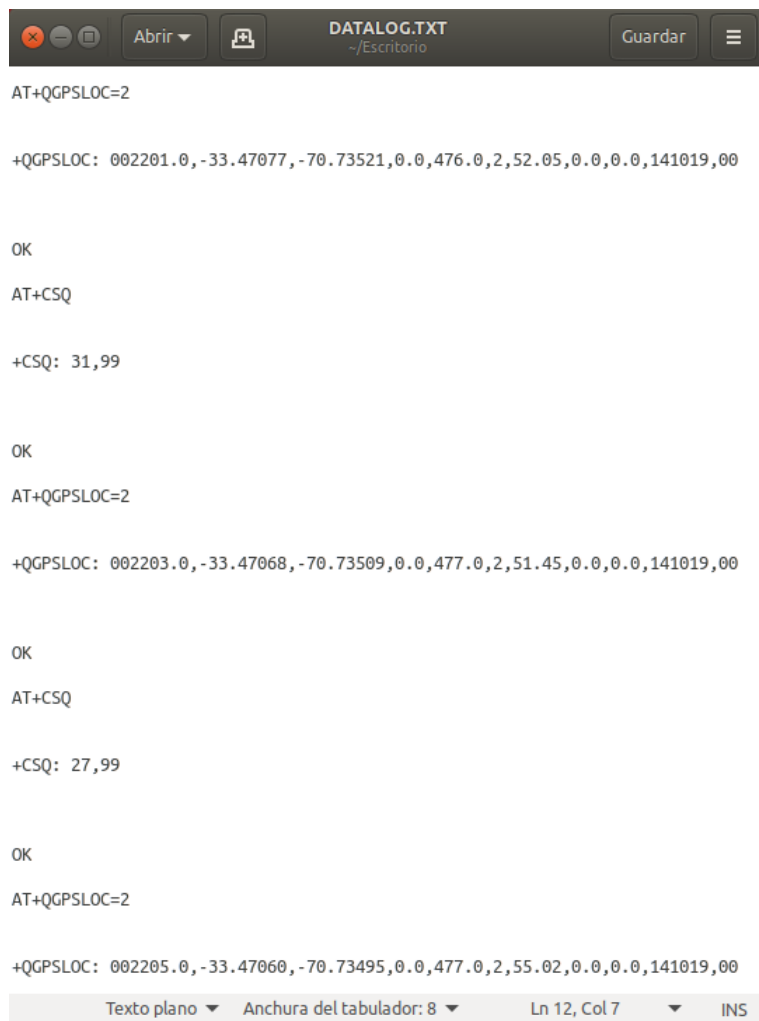
~
Texto plano  Anchura del tabulador: 8  Ln 1, Col 1  INS
```

Figura 4.2: Archivo DATALOG con información de conexión 2G

En la figura 4.2 se muestra la información de conexión realizada al prototipo, donde los datos corresponden respectivamente a <tecnología de acceso>, <operador>, <banda>, <canal>. Con estos se confirma que el módulo se conectó a la red GSM de Movistar, código de 73002, utilizando la banda 1900 y el canal 690. El envío de comandos se realizó a intervalos de 2 segundos, siendo importante destacar la iteración de la red observada en las dos conexiones iniciales fallidas, mostradas como **No Service**, y la iteración posterior entre GSM y EDGE de acuerdo a la disponibilidad de red. Se comprueba entonces la capacidad del prototipo para registrarse de manera autónoma a la red y mantener una conexión estable.

4.2.2. Toma de datos de RSSI y posición en red GSM

Habiendo recorrido la zona de prueba descrita en la sección anterior (ver figura 3.22), se extrajeron 369 iteraciones de consulta para posición y RSSI, las que se presentan a continuación:



```
AT+QGPSLOC=2

+QGPSLOC: 002201.0,-33.47077,-70.73521,0.0,476.0,2,52.05,0.0,0.0,141019,00

OK
AT+CSQ

+CSQ: 31,99

OK
AT+QGPSLOC=2

+QGPSLOC: 002203.0,-33.47068,-70.73509,0.0,477.0,2,51.45,0.0,0.0,141019,00

OK
AT+CSQ

+CSQ: 27,99

OK
AT+QGPSLOC=2

+QGPSLOC: 002205.0,-33.47060,-70.73495,0.0,477.0,2,55.02,0.0,0.0,141019,00
```

Figura 4.3: Archivo DATALOG con datos brutos

4.2.3. Extracción de datos de interés utilizando Python

Mediante el Script escrito en Python, se seleccionaron los datos correspondientes a las coordenadas geográficas Latitud/Longitud y el valor asociado del RSSI crando un archivo que contiene sólo los datos de interés (ver figura 4.4). Además, se eliminó los 10 primeros datos debido a la presencia de errores en la información de posición. Esta situación es completamente normal al iniciar el prototipo, ya que el módulo GPS demora 20 segundos aproximadamente en ajustar de manera correcta la localización y entregar los datos exactos.

```
-33.47487, -70.73148, 22,99
-33.47504, -70.73148, 25,99
-33.47522, -70.73147, 25,99
-33.47551, -70.73147, 25,99
-33.47572, -70.73146, 25,99
-33.47591, -70.73146, 19,99
-33.47609, -70.73146, 19,99
-33.47627, -70.73146, 19,99
-33.47655, -70.73149, 22,99
-33.47674, -70.73150, 22,99
-33.47691, -70.73150, 22,99
-33.47708, -70.73151, 22,99
-33.47735, -70.73151, 22,99
-33.47751, -70.73152, 22,99
-33.47768, -70.73151, 22,99
-33.47783, -70.73148, 22,99
-33.47810, -70.73147, 22,99
-33.47828, -70.73147, 22,99
-33.47846, -70.73148, 22,99
-33.47864, -70.73150, 22,99
-33.47889, -70.73150, 25,99
-33.47904, -70.73151, 25,99
-33.47913, -70.73152, 25,99
-33.47919, -70.73153, 25,99
-33.47930, -70.73157, 25,99
-33.47935, -70.73166, 25,99
-33.47942, -70.73178, 25,99
-33.47949, -70.73193, 25,99
-33.47955, -70.73208, 25,99
-33.47965, -70.73232, 22,99
-33.47971, -70.73246, 22,99
-33.47974, -70.73256, 22,99
-33.47977, -70.73266, 22,99
-33.47984, -70.73281, 22,99
-33.47987, -70.73285, 22,99
-33.47987, -70.73285, 22,99
-33.47987, -70.73285, 26,99
-33.47987, -70.73286, 26,99
-33.47990, -70.73294, 26,99
```

Figura 4.4: Datos de Latitud, Longitud y RSSI

4.2.4. Visualización de los datos a través de un mapa de cobertura

En la figura 4.5 se presenta la muestra de 359 datos de posición geográfica y RSSI obtenidos tras la el procesamiento de los datos brutos con Python. Como el objetivo es probar la funcionalidad del diseño para la extracción de datos y no realizar una caracterización de la red GSM, no fue necesario obtener una cantidad mayor de muestras, ya que los datos obtenidos son suficientes para probar la funcionalidad del prototipo.

Finalmente, en la figura 4.6 se presenta el mapa de cobertura generado a partir de la muestra de datos obtenidos en red 2G. Se observa que la cobertura se mantiene casi constante a lo largo de los 6 kilómetros de prueba, lo que es totalmente esperable para este tipo de tecnología, ya que presentan un decaimiento de potencia de señal mucho más bajo que la red LTE. No obstante, se presentó una zona en particular donde la cobertura cae a un factor de RSSI cercano a 10, equivalente a -93dBm de potencia, valor que bordea el concepto de mala cobertura. Dicha zona, es precisamente el área más alejada de las antenas Movistar, lo que podría explicar en cierta medida la pérdida de potencia, aunque también podría ser atribuido al alto tráfico de personas del sector de Av. Pajaritos.

Si se considera un muestreo a una velocidad de 30km/h , equivalente a 8.3m/s , y solicitando datos cada 2 segundos, se habrá recorrido 16.6m por cada dato obtenido. Esta distancia es acorde a la precisión del GPS, cercana a los 15m , y suficiente para tener un mapeo representativo.

	A	B	C	D		A	B	C	D
1	Latitud	Longitud	RSSI Factor		339	-33,46754	-70,72984	30	
2	-33,47253	-70,73406	25		340	-33,46738	-70,72956	31	
3	-33,47259	-70,73401	25		341	-33,46721	-70,72927	31	
4	-33,47262	-70,73408	25		342	-33,46704	-70,72897	29	
5	-33,47265	-70,73417	28		343	-33,46677	-70,72846	29	
6	-33,47269	-70,73426	28		344	-33,46658	-70,72811	29	
7	-33,47274	-70,73439	28		345	-33,46638	-70,72774	24	
8	-33,47283	-70,73461	28		346	-33,46618	-70,72739	24	
9	-33,47289	-70,73479	28		347	-33,46591	-70,7269	24	
10	-33,47296	-70,73495	28		348	-33,46576	-70,72663	24	
11	-33,47302	-70,7351	20		349	-33,46564	-70,72638	24	
12	-33,47308	-70,73523	23		350	-33,46551	-70,72615	24	
13	-33,47312	-70,73534	23		351	-33,46538	-70,72592	24	
14	-33,47318	-70,73548	23		352	-33,46519	-70,72558	24	
15	-33,47323	-70,73562	23		353	-33,46505	-70,72534	21	
16	-33,47327	-70,73577	23		354	-33,46494	-70,7251	21	
17	-33,47335	-70,736	23		355	-33,46482	-70,72486	21	
18	-33,47341	-70,73613	23		356	-33,46465	-70,72453	17	
19	-33,47344	-70,73618	31		357	-33,46454	-70,72433	17	
20	-33,47345	-70,73624	31		358	-33,46444	-70,72413	17	
21	-33,47342	-70,73634	31		359	-33,46434	-70,72393	17	
22	-33,47331	-70,73651	31		360				
23	-33,47323	-70,73665	29		361				

Figura 4.5: Archivo Excel con datos para graficar

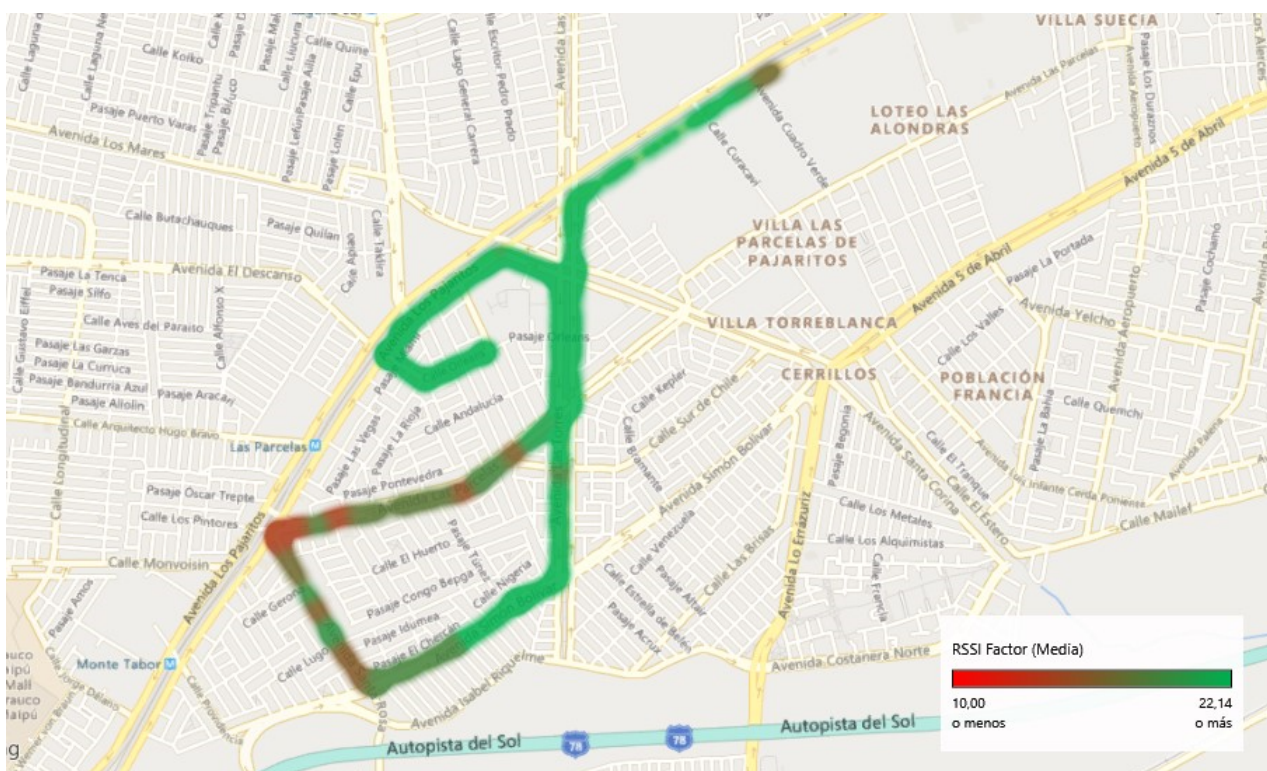


Figura 4.6: Mapa de cobertura con datos de red 2G

Con el fin de estudiar el efecto de la velocidad de desplazamiento, el tramo superior del mapa de realizó a una velocidad de 90km/h, y si bien se disminuyó el número de muestras por metro, el prototipo no perdió funcionalidad y continuó entregando datos libres de errores y totalmente consistentes.

Por otro lado, si bien no se presentó una caída significativa del RSSI en los 6km recorridos, se debe tener en cuenta que no es una conducta aplicable a la tecnología NB-IoT, ya que considerando su naturaleza LTE, presenta un gran decaimiento de potencia en función de la distancia a la antena.

Con todo lo anterior, se valida el correcto funcionamiento del prototipo al conectarse a la red y adquirir datos de RSSI y locación. Se valida también el algoritmo de configuración para acceder a la red, que en el caso de acceder a una red NB-IoT, debe ser modificado según los parámetros que se presentan en el capítulo 5.

4.3. Pruebas de conexión al servidor mediante red GSM

La segunda etapa del presente trabajo, contempló la realización de pruebas al servidor privado utilizando el mismo prototipo en red 2G. Los resultados de las distintas pruebas se presentan a continuación.

4.3.1. Prueba de Conexión vía Socket TCP

Primero, se configuraron los datos del APN de Movistar y los parámetros de conexión según se describió en el capítulo anterior, obteniendo la respuesta que se muestra en la figura 4.7.

```
AT+QICSGP=1,1,"WAP.TMOVIL.CL","wap","wap",1
OK
AT+QIACT=1
OK
AT+QIACT?
+QIACT: 1,1,1,"10.228.222.85"

OK
```

Figura 4.7: Configuración de Punto de Acceso OK

Como se puede observar en la figura 4.7, se logró registrar los datos de manera exitosa y activar el contexto para establecer la conexión. Con esto, se tiene acceso al tráfico de datos utilizando la red 2G de Movistar. Posteriormente, se abrió de forma exitosa un *socket* TCP con el servidor privado, obteniendo la respuesta que se aprecia en la figura 4.8.

```
ATV1
OK
AT+QIOPEN=1,0,"TCP","167.71.126.21",80,0,1
OK

+QIOPEN: 0,0
```

Figura 4.8: Apertura de *socket* TCP OK

4.3.2. Prueba mediante requerimiento GET

Una vez conectados a la red GSM de Movistar y mediante la conexión establecida de manera exitosa con el servidor, se inició el envío de requerimientos por parte del cliente para obtener respuesta del servidor. El envío comenzó ejecutando el comando **AT+QISEND=0**, el cual abrió de forma correcta (ver figura 4.9) una entrada de texto para ingresar los requerimientos.

```
AT+QISEND=0
> GET /consultar HTTP/1.1
Host: 167.71.126.21
```

```
SEND OK
```

Figura 4.9: Prueba inicial de envío de requerimiento GET

De este modo, se ingresó un solicitud a la API para obtener la información alojada en la ruta **/consultar**, obteniendo de forma exitosa la respuesta **consultando algo**, lo que se muestra en la figura 4.10 y corresponde a la información alojada en el servidor y recibida correctamente por el cliente.

```
+QIURC: "recv",0,187
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 02 Sep 2019 03:24:49 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 16
Connection: keep-alive
```

```
Consultando Algo
```

Figura 4.10: Respuesta al requerimiento GET

4.3.3. Prueba de Eco

De forma similar a la anterior, se realizaron dos pruebas de eco a la API en su ruta **/echo**, en la que inicialmente se escribió el mensaje **Prueba de eco 1** y posteriormente **Prueba de eco 2**. Ambas pruebas obtuvieron respuesta exitosa por parte del servidor, obteniendo el mensaje devuelta al cliente como se observa en la figura 4.11.

Finalmente, y dado que todas las pruebas resultaron exitosas, se procedió a cerrar la conexión mediante el envío del comando **AT+QICLOSE=0**, obteniendo una respuesta positiva (ver figura 4.12) y dando por finalizado el procedimiento de validación del servidor.

```
AT+QICLOSE=0
```

```
OK
```

Figura 4.12: Cierre del *socket* TCP

```
AT+QISEND=0
> GET /echo/Prueba de eco 1 HTTP/1.1
Host: 167.71.126.21
```

SEND OK

```
+QIURC: "recv",0,189
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 02 Sep 2019 03:25:28 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 18
Connection: keep-alive
```

R: Prueba de eco 1

```
AT+QISEND=0
> GET /echo/Prueba de eco 2 HTTP/1.1
Host: 167.71.126.21
```

SEND OK

```
+QIURC: "recv",0,189
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 02 Sep 2019 03:25:47 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 18
Connection: keep-alive
```

R: Prueba de eco 2

Figura 4.11: Pruebas de eco mediante requerimiento GET

Capítulo 5

Adaptación del trabajo para su uso en red NB-IoT

En el siguiente capítulo se presentan las adaptaciones que deben ser realizadas al prototipo para su uso en la red NB-IoT. Si bien, la mayoría de los comandos utilizados para la validación del plan de pruebas en red GSM son compatibles con el servicio NB-IoT, se creó el presente capítulo con el objetivo de entregar la documentación final y resumida para la realización de pruebas.

5.1. Adaptaciones para uso del prototipo en la medición de posición y RSSI

Se debe modificar la secuencia de búsqueda priorizando la red NB-IoT con código 03. Además, se debe configurar el modo de conexión para que solamente acceda a la red LTE. Para las bandas de acceso, se debe verificar su correcta configuración en banda 28 para NB-IoT, y en el caso de GSM y CatM, no adquieren relevancia pero se sugiere una configuración por defecto. Además se configura el `iotopmode` y `nbsibscramble` no presentes en GSM.

```
AT+QCFG='nwscanseq',03 // Configura prioridad NB-IoT en la búsqueda
AT+QCFG='nwscanmode',3 // Configura en modo sólo LTE
AT+QCFG='iotopmode',1 // Configura en banda GSM 1900, NB 28, CatM 28
AT+QCFG='nbsibscramble',0 // Configura NB Single Mode
AT+QCFG='band',1,8000000,8000000 //Configura Scramble 0-ON, 1-OFF
AT+QNWINFO // Solicita información de la red
```

Con esta configuración, el prototipo espera conectarse a la red NB-IoT como primera opción y buscando únicamente redes LTE. Al lograr una conexión exitosa, el código implementado en Arduino sigue siendo válido para la toma de datos de RSSI y localización mediante el GPS, sin necesidad de aplicar adaptaciones al sistema.

5.2. Adaptaciones para pruebas de conexión al servidor

Habiendo conectado el prototipo a la red NB-IoT, no es necesario realizar adaptaciones al servidor ni a la metodología de conexión. Por lo tanto, se debe seguir la misma estructura y serie de comandos detallada en el capítulo 3 y que se resume a continuación.

```
AT+QICSGP=1,1,"WAP.TMOVIL.CL","wap","wap",1 // Configuración del APN
AT+QIACT=1 // Activa el perfil 1
AT+QIACT? // Solicita el estado del perfil
ATV1 // Formato de la respuesta
AT+QIOPEN=1,0,"TCP","167.71.126.21",80,0,1 // Activa el perfil
AT+QISEND=0 // Responde con >
>GET /consultar HTTP/1.1 // Finalizar con <CR><LF>
Host: 167.71.126.21 // Finalizar con <CR><LF>
// <CR><LF>
// <CTRL+Z>
```

Conclusión

El *Internet of Things* demandará la existencia de redes que sean capaces de ofrecer una alta disponibilidad pero con un muy bajo consumo energético. NB-IoT es una tecnología con las cualidades suficientes para imponerse sobre otras tecnologías de bajo consumo, y más aún cuando los interesados son los grandes operadores del sector de las telecomunicaciones.

En el proceso de habilitación del servicio NB-IoT por parte de Telefónica Chile, surge la necesidad de realizar pruebas de laboratorio que permitan conocer las reales características de la tecnología, para lo cual el presente trabajo aporta con distintas herramientas, considerando que:

Se implementó un prototipo de comunicación utilizando el módulo Quectel BG96, compatible con redes 2G, Cat-M y NB-IoT, quedando documentado el proceso configuración para acceder a las redes 2G y NB-IoT.

Se diseñó e implementó un sistema para la toma de datos, capaz de almacenar datos de RSSI y localización en una tarjeta de memoria SD. Como el sistema es operado por un microcontrolador, la toma de datos queda completamente automatizada, otorgándole autonomía con una batería de Li-Po reemplazable y que permite su uso continuo.

Se diseñó e implementó un servidor privado alojado en la dirección IP 167.71.126.21 que permite conexiones vía socket TCP y el intercambio de datos mediante una API que ofrece respuesta a requerimientos de tipo GET en forma directa y en forma de eco.

Todo lo anterior, se validó mediante un ejemplo de uso en red 2G, con el que se tomó una muestra de datos y se generó un mapa de cobertura que muestra de manera gráfica los datos obtenidos. En la misma red, se realizaron pruebas de conexión exitosas con el servidor, logrando conexión, envío y recepción de datos.

Finalmente, se documentó el proceso de conexión, toma de datos, y configuración para adaptar su uso bajo el servicio de red NB-IoT, de tal forma que cualquier usuario pueda utilizar el sistema para los fines que fue diseñado.

En base a esto, se concluye que el diseño implementado cumple con las funcionalidades para ser utilizado en una red NB-IoT *outdoor*, sin embargo no es aplicable para una red *indoor*, ya que la precisión del GPS no es suficiente para desplazamientos menores a 10m. En su defecto, un módulo GPS de precisión puede ser implementado para poder generar mapas de RSSI en áreas reducidas. Otra opción, es caracterizar la red *indoor* enfocando las pruebas en la conexión, generando un estudio a partir de los tiempos de respuesta, conexiones exitosas y mensajes perdidos, pruebas que pueden ser realizadas usando el servidor implementado.

Se concluye también que la velocidad de desplazamiento no es determinante en la generación de un mapa de cobertura, logrando pruebas exitosas a más de 90km/h. No obstante, el muestreo ideal se calculó en 30km/h, velocidad que genera un desplazamiento equivalente a la precisión del GPS.

Además, del mapa de cobertura generado se evidenció una caída del RSSI al alejarse de las antenas emisoras, fenómeno que se debe considerar al trabajar con redes 4G o en particular con NB-IoT, ya que la distancia al nodo emisor afecta aún más estas redes que a las 2G.

También es importante mencionar que los códigos y configuraciones del sistema de toma de datos son perfectamente aplicables a otros módulos NB-IoT, y aunque en general los comandos AT están estandarizados, se debe considerar la adaptación de los comandos AT de acuerdo al proveedor.

Por último, se demostró la funcionalidad del servidor para responder a los requerimientos por parte del prototipo, lo que genera una herramienta de pruebas más allá de las necesarias para validar una red, siendo útil también en la creación de aplicaciones web o cualquier desarrollo que necesite pruebas de requerimientos HTTP.

En conclusión, se cumplió con el diseño de un plan de pruebas para la habilitación del servicio NB-IoT, el que consiste en un servidor privado para pruebas de conexión, y un prototipo versátil con capacidad para toma de datos de cobertura y posición en redes 2G y NB-IoT/Cat-M.

El plan de pruebas abarca el proceso de configuración inicial, conexión a la red, toma de datos de RSSI, generación de mapas de cobertura, montaje de un servidor, y pruebas de conexión. Todo el proceso de diseño e implementación fue validado en la red 2G y documentado para su uso en la red NB-IoT.

Trabajo Futuro

Dado que los alcances del presente trabajo involucraban sólo el diseño de un plan de pruebas a partir de la implementación de un prototipo y servidor, es posible realizar dos trabajos importantes con este sistema dependiendo de la evolución del proceso de habilitación de la red NB-IoT.

- En caso de habilitarse al menos una red NB-IoT *indoor*, se propone realizar pruebas de conexión con el servidor implementando con el objetivo de obtener una caracterización de la red de acuerdo a su respuesta de conexión, estabilidad y latencia. El servidor contiene un archivo de registro de todos los intentos de conexiones y requerimientos, lo cual puede ser útil para este fin
- En caso de habilitarse una red NB-IoT *outdoor*, se propone realizar una caracterización de la red en términos de cobertura, a partir mapas de RSSI que se pueden obtener con el prototipo y siguiendo los procedimientos detallados en el presente trabajo.

Bibliografía

- [1] Arduino, Arduino Leonardo with headers, 2019. <https://store.arduino.cc/usa/leonardo>. [Consultado: 23 septiembre 2019].
- [2] GPS World, What exactly is GPS NMEA data, 2017. <https://www.gpsworld.com/what-exactly-is-gps-nmea-data/>. [Consultado: 23 septiembre 2019].
- [3] Mike Lee, DigitalOcean reviews, WordPress hosting and customer support, 2019. <https://www.reviewplan.com/digitalocean-reviews-wordpress-hosting-customer-support/>. [Consultado: 23 septiembre 2019].
- [4] NGINX, Welcome to NGINX Wiki, 2017. <https://www.nginx.com/resources/wiki/>. [Consultado: 23 septiembre 2019].
- [5] Organización de las Naciones Unidas, Población, 2018. <http://www.un.org/es/sections/issues-depth/population/index.html>. [Consultado: 23 septiembre 2019].
- [6] Quectel, UMTS LTE EVB Kit, 2019. <https://www.quectel.com/product/umtsevb.htm>. [Consultado: 15 octubre 2019].
- [7] 3GPP. Tr 45.820 v2.1.0. August 2018.
- [8] W. Ayoub, A. E. Samhat, F. Nouvel, M. Mroue, and J. Prévotet. Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs standards and supported mobility. *IEEE Communications Surveys Tutorials*, pages 1–1, 2018.
- [9] T. Barnett, S. Jain, U. Andra, and T. Khurana. Cisco visual networking index (vni) complete forecast update. *Americas/EMEAR Cisco Knowledge Network (CKN) Presentation*, December 2018.
- [10] M. Chen, Y. Miao, Y. Hao, and K. Hwang. Narrow Band Internet of Things. *IEEE Access*, 5:20557–20577, 2017.
- [11] Subsecretaria de Telecomunicaciones. Sector telecomunicaciones primer trimestre 2019. mayo 2019.
- [12] O. T. Eluwole, N. Udoh, M. Ojo, C. Okoro, and A. J. Johnson. From 1g to 5g, what next? *IAENG International Journal of Computer Science*, August 2018.
- [13] Panagiota D. Giotopoulou. The evolution of mobile communications: Moving from 1G to 5G and from human-to-human to machine-to-machine communications. Master’s thesis, National and Kapodistrian University of Athens, 2015.

- [14] A. Hoglund, X. Lin, O. Liberg, A. Behravan, E. A. Yavuz, M. Van Der Zee, Y. Sui, T. Tirronen, A. Ratilainen, and D. Eriksson. Overview of 3gpp release 14 enhanced nb-iot. *IEEE Network*, 31(6):16–22, November 2017.
- [15] K. Abd Jalil, M.Hanafi Abd. Latif, and M. Noorman Masrek. Looking into the 4g features. *MASAUM Journal of Basic and Applied Sciences*, 1(2):249–253, September 2009.
- [16] Ms. Lopa and J. Vora. Evolution of mobile generation technology: 1G to 5G and review of upcoming wireless technology 5G. *International Journal of Modern Trends in Engineering and Research*.
- [17] Michele Mackenzie. LPWA Networks for IoT: World Trends and Forecast 2015–2025. *Research Forecast Report*, July 2016.
- [18] J. H. Schiller. *Mobile Communications*. Pearson Education Limited, 2nd edition, 2003.