Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Ciudad de México

Centro de Investigación en Microelectrónica y Biodiseño

Short Range and Cellular V2X Communication

Proyecto de ingeniería presentado por

Eloi Pascual Belda A01756116

Asesor:

Dr. Martin Rogelio Bustamante Bello

Sinodales:

Dr. Javier Izquierdo Reyes
Dr. Edgar Santoyo Castelazo
Dr. Renato Galluzi Aguilera

Ciudad de México, Nov, 2019

Índice de figuras

Índice de cuadros

Índice general

Capítulo 1

Introducción

En un ambiente cada día más conectado, el coche autónomo parece una realidad cada día más cercana. Son evidentes las repercusiones positivas que esto podría tener en el día a día de las personas y más concretamente en la seguridad. La necesidad de comunicación entre vehículos es uno de los diferentes problemas a solucionar para la creación de un vehículo autónomo, realidad cada día más cercana.

El actual avance en comunicaciones y más específicamente en redes de sensores ha derivado en tecnologías muy polivalentes que pueden ser utilizadas en comunicaciones entre vehículos (V2V) o en la comunicación entre estos vehículos y elementos que los rodean (V2X) tales como semáforos, farolas, radares, etc.

Concretamente el V2X (vehicle to X) pretende conectar los vehículos de manera inalámbrica con su entorno. Este entorno puede ser cualquier persona o infraestructura susceptible de tener conexión entre ellos de manera que estos nodos puedan conocer información relevante del vehículo como puede ser velocidad, ruta o cantidad disponible de combustible o también de la situación actual del entorno, como puede ser estado de la carretera, estado del tráfico o si se ha reportado algún accidente en la vía.

Como es lógico, todos estos datos analizados de manera correcta pueden transformarse en información realmente útil sobre aprovechamiento de la energía, optimización de rutas, prevención de accidentes, reducción de la contaminación y reducción de congestión del tráfico, entre otros. Debido a esto, diferentes empresas y entidades gubernamentales de todo el mundo han dedicado esfuerzos y recursos para el desarrollo de tecnologías y estándares que definan las necesidades y las direcciones que este concepto se centrará en abordar los próximos años.

La altas posibilidades de que la implantación de estas tecnologías se transformaran en resultados tan óptimos como reducción de decesos totales relativos a accidentes de tránsito, la reducción de retenciones y atascos en las ciudades y reducir tiempos de trayectos optimizándolos y así obtener una forma eficiente de reducir la contaminación (tanto ambiental como sonora) relativa al tráfico hacen de esta tecnología una de las realidades que cambiarán nuestro futuro para siempre, transformando nuestro mundo en una dirección más segura y responsable con el medio ambiente.

Para tomar conciencia de la verdadera magnitud que estos problemas suponen para la sociedad nos referiremos a algunos datos con la intención de que nos ilustren acerca del paradigma actual y la necesidad que se tiene de un cambio.

En cuanto a decesos totales, nos referimos al Instituto Nacional de Estadística y Geografía o por sus siglas INEGI, que se define como *"Un organismo público autónomo responsable de normar y coordinar el Sistema Nacional de Información Estadística y Geográfica, así como de captar y difundir información de México en cuanto al territorio, los recursos, la población y economía, que permita dar conocer las características de nuestro país y ayudar a la toma de decisiones."* El INEGI nos proporcionará los datos relativos a todos los accidentes registrados en los Estados Unidos Mexicanos de los últimos años.

Analizando los datos relativos al año anterior a la realización de este trabajo, es decir, el año 2018, en el INEGI se registraron un total de 238,594 accidentes entre vehículos automotor, área de acción directa de la comunicación V2X. Esto se corresponde a una media de más de 650 accidentes diarios. Estos accidentes se traducen en destrucción de recursos públicos como farolas, límites de vías, señales y también en retenciones y atascos que a su vez se traducen en nuevos accidentes.

Si se analizan estos datos más profundamente, se pueden aislar los accidentes que se produjeron en intersecciones. El número de accidentes en intersecciones se corresponde en el 90.265 % de los accidentes totales entre vehículos y esto equivale a un total de 214,973 accidentes en este entorno particular transformándose en un total de 386 fallecidos en 2018 y representando un 50.537 % de los decesos totales en accidentes entre vehículos de dicho año.

Se analiza este escenario en concreto ya que tiene relación directa con el proyecto que se presenta, ya que en las intersecciones se evidencia la necesidad de una comunicación entre los vehículos y su entorno, ya que con una comunicación adecuada se podrían prever las colisiones si se dispusiera de los datos de dirección, posición, velocidad, aceleración, etc.

Esto también nos lleva a observar uno de los dilemas que se abren con la comunicación entre

vehículos. La privacidad. Hablamos de enviar constantemente datos personales de los usuarios tales como ubicación o dirección a la que se dirigen. Si bien estos datos se tratarían con el objetivo de aumentar la seguridad vial de los usuarios, no se puede evitar ver que esta seguridad está subyugada a una invasión de la privacidad del usuario.

Para concluir con esta introducción, solo recalcar la necesidad de que esta tecnología de comunicación entre vehículos y entorno sea implementada en la mayoría de vehículos y de preferencia en todos, ya que para la optimización de esta red se debería de disponer de los datos de todos los vehículos que circulan por la vía y así ser capaz de realizar cálculos teniendo en cuenta el resto de vehículos, que podrían influir en las decisiones que debería tomar el vehículo.

Esto evidencia más la necesidad de un sistema global independiente del fabricante del vehículo, o al menos, sistemas que fueran compatibles entre ellos y interconectables al máximo para permitir que toda la información producida y potencia de cálculo tuvieran un aprovechamiento óptimo.

1.1. Problemática de la solución

La necesidad principal que motiva este proyecto es que actualmente no se dispone de una solución de bajo coste e independiente del fabricante del vehículo para la comunicación V2X.

Se pretende fabricar un dispositivo de bajo coste y usando herramientas de licencia de uso libre de manera que se pueda construir un prototipo funcional a precio reducido y totalmente independiente de una marca de vehículo o del fabricante.

Se puede añadir que las propuestas actuales no presentan soluciones híbridas de manera que se presente un sistema que refuerce el sistema de corto alcance, ya que si este funcionaba de manera individual se tienen nodos de información pero ninguna comunicación entre ellos o ninguna manera de extraer información entre estos.

Actualmente se dispone de una alta conectividad en sistemas de comunicación celular (4G, LTE, 5G) y aprovechar esta conectividad y estas altas velocidades de transmisión en beneficio de un sistema de comunicación de largo alcance en comunicaciones V2X podría significar interconexiones entre clústers de comunicaciones de corto alcance anticipando así problemas no tan prioritarios como colisiones, como podrían ser atascos o retenciones.

A esto debemos sumarle el valor añadido del impacto potencial que las comunicaciones entre vehículos y entorno tendrán en los próximos años. Nos encaminamos hacia un escenario

donde este tipo de comunicaciones van a avanzar rápidamente en un ambiente en constante evolución como es la conducción autónoma, y solucionar estos problemas con herramientas gratuitas y disponibles para cualquiera abre un gran abanico de posibles optimizaciones o actualizaciones futuras que fijen errores o simplemente implementen mejoras en el dispositivo sin coste alguno.

1.2. Objetivos

Como objetivo general del proyecto, se pretende generar el sistema de corto alcance descrito anteriormente y que cubra las necesidades demandadas para la correcta solución de la problemática descrita. También se buscará establecer las bases del protocolo de comunicación y mensajes a enviar a través de esta comunicación. Para finalizar, implementar también una puerta a la comunicación largo alcance, que permita establecer una comunicación con dispositivos remotos.

De manera específica, se marca como objetivo elaborar el prototipo con un coste reducido, de manera que este pueda ser accesible por cualquier usuario que quiera implementar este dispositivo en su vehículo. Siguiendo con esto, se pretende que los códigos y programas sean implementados con herramientas de uso libre que no supongan un coste adicional al prototipo.

1.3. Estado del arte

Para analizar el estado actual en el ámbito del V2X nos centraremos en dos aspectos principales. Observaremos los estándares sobre los que se soporta la tecnología existente y por otro lado observaremos algunos de los productos que se encuentran actualmente en el mercado.

1.3.1. Estándares

Los principales estándares que encontramos actualmente en el ámbito del V2X se podrían reducir a 802.11p, diseñado específicamente para comunicación vehicular y el protocolo ZigBee, orientado al internet de las cosas. A continuación explicaremos brevemente cada uno de estos protocolos y por qué es interesante analizarlos en el ámbito del V2X.

IEE 802.11P

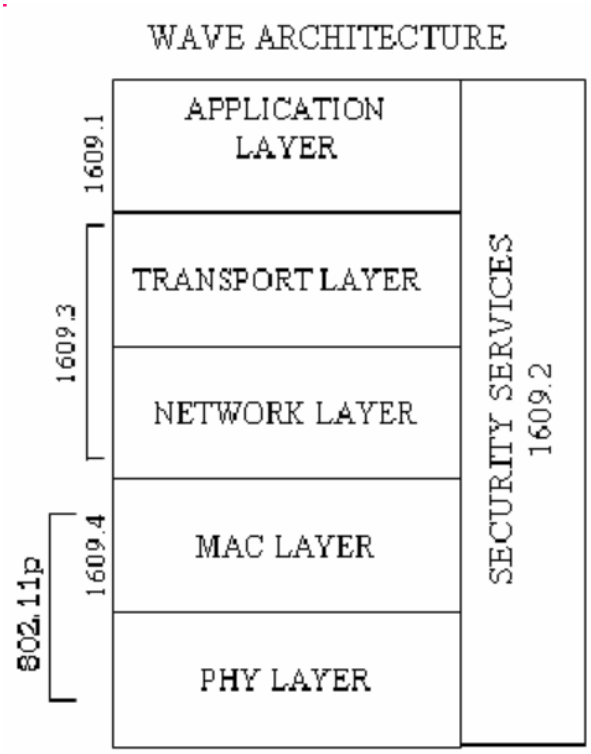


Figura 1.1: Arquitectura wave

También conocido como WAVE (Wireless Access in Vehicular Environments) es la estandarización amparada por el IEEE que abarca un conjunto de protocolos de acceso inalámbrico en entornos vehiculares. El objetivo de WAVE es proporcionar comunicación entre vehículos (V2V) o entre vehículos y infraestructura en un entorno donde se necesita un tiempo de envío y procesamiento muy rápido debido a que el entorno es constantemente cambiante.
El estándar IEEE 802.11p (derivado del estándar IEEE 802.11) describe el uso de las capas inferiores (la capa física y la capa básica MAC) de la pila WAVE. Las capas superiores, están definidas por el estándar IEEE 1609.

Ofrece la ventaja de que está específicamente diseñado para soportar las comunicaciones V2X.

ZigBee

Zigbee es el nombre de la especificación de un conjunto de protocolos de comunicación inalámbrica para su utilización en radiodifusión digital. Zigbee se basa en el estándar IEEE 802.15.4 sobre redes inalámbricas de área personal. Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y de bajo consumo.

Sus ventajas principales se resumen en su estructura de malla y en la facilidad de su integración en sistemas. Se puede implementar Zigbee con muy poca electrónica y esto lo hace especialmente útil en entornos como la domótica o el internet de las cosas (IoT).

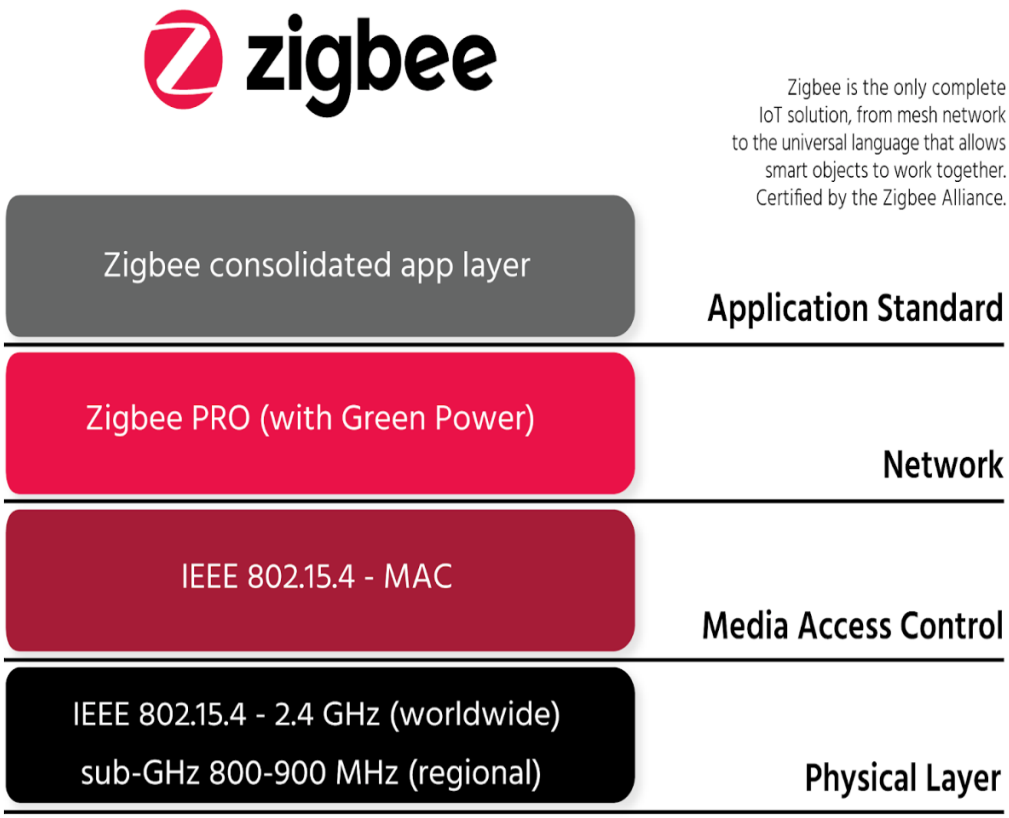


Figura 1.2: Capas sobre el está basado el protocolo Zigbee

1.3.2. Mercado

En cuanto a los productos del mercado, remarcar cómo la indústria orientada a IoT puede ser fácilmente adaptada a el concepto de V2X. Aquí debemos destacar Waspmote, que ejemplifica esta sinergia entre el sector del Internet of Things y V2X.

Waspmote

Comercializado por libelium, se trata de una solución integrada que brinda tanto comunicaciones por Zigbee como conexiones BlueTooth y WiFi. También incorpora salidas 4G que permitirían una conexión a internet. Se trata de una solución de coste medio que se podría utilizar o adaptar para la comunicación V2X.

También incorpora protocolos industriales tales como MODBUS o CAN bus que nos permiten interactuar con los datos de los sensores del vehículo.

Al tratarse de un módulo conectable al vehículo y con salidas de red, este dispositivo programado de la forma correcta podría encajar en una solución o un prototipo en el que basarse para determinar las necesidades de un dispositivo para la comunicación V2X.

1.4. Antecedentes

Para el desarrollo de este proyecto se ha partido de los proyectos que se realizaron en el 2017 y 2018 respecto a la comunicación vehicular.

Proyecto Noviembre 2017:

En el caso del proyecto realizado el primer semestre de 2017 cuyo objetivo era el diseño e implementación de una infraestructura de redes de telecomunicación enfocada a las comunicaciones automotrices la cual sirva como plataforma para desarrollar aplicaciones enfocadas a la seguridad, tráfico vial y comercial. Mostrando especial interés en los siguientes puntos:

1. Revisar los problemas de las comunicaciones inalámbricas C2X y C2C.
2. Investigar los antecedentes y protocolos de comunicación utilizados en las comunicaciones C2C.
3. Comparar las diferentes tecnologías utilizadas así como los dispositivos en existencia.
4. Simular el desempeño de la tecnología elegida en un entorno real simulado.
5. Probar el desempeño de la tecnología elegida a velocidades altas en tiempo real.
6. Desarrollar dos aplicaciones viales con base en la infraestructura propuesta.
7. Crear una infraestructura de comunicaciones Car to Car en México que esté estandarizada internacionalmente.

Y del cual extraemos a partir de las simulaciones realizadas en el mismo qué tecnologías son las más adecuadas para nuestra solución, quedando con mejores resultados el estándar 802.11p y Zigbee.

Proyecto Noviembre 2018:

En el segundo proyecto realizado se toma como punto de partida el del año 2017 poniendo a prueba los dos protocolos de comunicación (Zigbee y 802.11p), en un escenario real y simulado, para determinar la viabilidad y eficiencia de cada uno en la comunicación V2V y V2X. Centrándose específicamente en los siguientes puntos:

1. Migrar los avances de pruebas físicas logrados con el protocolo 802.11p a ZigBee.
2. Implementar transmisión y recepción de paquetes sobre protocolo ZigBee de forma física y 802.11p en simulación.
3. Definir un escenario para realizar pruebas de comunicación de forma física y simulada.
4. Determinar la mejor alternativa de comunicación para redes vehiculares, dependiendo de los resultados.

De este proyecto tomamos que aunque ambos protocolos son completamente distintos, ambos son aptos para la comunicación V2V y V2X. Realzando que el 802.11p está optimizado para la comunicación entre vehículos al contrario que Zigbee que está pensado para multitud de aplicaciones, y por tanto se adapta mejor para la comunicación vehicular.

1.5. Marco Teórico

En este apartado se procederá a explicar conceptos con los que se debe estar familiarizado para una comprensión más profunda de las decisiones que se han tomado para el diseño de la solución propuesta. Asimismo se introducirá vocabulario que será utilizado o citado en explicaciones más adelante. Este vocabulario lo agrupamos en diferentes bloques para facilitar la búsqueda en caso de utilizarse a modo de diccionario.

1.5.1. Difusión de mensajes

Unicast

Unidifusión, consiste en el envío de información desde un único emisor a un único receptor. Se necesita conocer la dirección del destinatario y añadirla al paquete para que los demás nodos no lo acepten como suyo.

Multicast

Multidifusión, consiste en el envío de información desde un emisor a un grupo de receptores. Esto se puede hacer de diferentes maneras, pero básicamente el concepto se basa en la pertenencia a un grupo y se envía mediante Unicast a ese grupo. Todos los nodos de ese grupo interpretan que son el destinatario.

Broadcast

Envío de información desde un emisor al resto de receptores de la red. No es necesario saber a quién se envía. El paquete es enviado de manera que cualquier receptor que lo reciba interpreta que es el destinatario.

1.5.2. Arquitecturas de red

Maestro/Esclavo

Arquitectura de red mediante la cual se establece una jerarquía entre los diferentes componentes de la red. El Esclavo solo puede enviar cuando el Maestro le habilita.

DigiMesh

Consiste en una topología de red en malla, en la cual cada nodo está conectado con el resto de nodos que le rodean. Cada nodo coopera en la transmisión de información. Ésta topología de red en malla proporciona los siguientes puntos fuertes:

- **Routing:** Un mensaje dirigido a un determinado nodo(receptor), es propagado a través de una ruta, saltando el mensaje de nodo en nodo hasta alcanzar su destino final.
- **Ad-hoc network creation:** Proceso automático por el que se crea una red entera de nodos sin ninguna intervención humana.
- **Self-healing:** detecta si uno o más nodos en la red no están ya disponibles y reconfigura la red para restablecer las rutas perdidas.
- **Peer-to-peer architecture:** no existe ninguna jerarquía, es decir que todos los nodos (dispositivos) son iguales en la red.
- **Route discovery:** en vez de mantener un mapa de red, las rutas son descubiertas y creadas únicamente cuando se necesitan.
- **Selective acknowledgments:** únicamente el destinatario (receptor) es quién contesta a las peticiones de establecer ruta.
- **Reliable delivery:** Confiabilidad en el envío de mensajes por medio de los ACK (acknowledgement).

Es una topología de red en malla desarrollada por la marca Digi para sus dispositivos que operan por medio de la tecnología Zigbee.

1.5.3. Tramas

El envío de mensajes se podría hacer directamente enviando el mensaje por la red, pero no es un envío eficiente ya que no tenemos forma de detectar errores en la generación o recepción del mismo, además de la imposibilidad de saber qué dispositivo ha enviado el mensaje en recepción. Por ello se ha optado por el envío de mensajes encapsulados en tramas.

Puesto que estábamos utilizando los dispositivos Xbee, estos dispositivos disponen de un modo de operación API que nos proporcionan una interfaz donde los datos se envían/reciben a través de la interfaz serie en forma de paquetes. Esto nos permite establecer comunicaciones complejas entre los dispositivos sin tener que definir nuestro propio protocolo.

Las tramas API que se envían tienen la siguiente estructura:

Start delimiter	Length		Frame data								Checksum
			API identifier	Identifier-specific Data							
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB	cmdID	cmdData							Single byte

Figura 1.3: Campos que conforman una trama API

- **Start delimiter:** Campo que determina el inicio de una trama, está conformado únicamente por un único byte de valor reservado (0x7E).
- **Length:** Especifica el número total de bytes que conforman los datos (Frame data) enviados en el paquete. Excluyendo de este los bytes de Start delimiter, Length y Checksum.
- **Frame data:** Este campo contiene la información que el dispositivo envía o recibe. La estructura de frame data depende del propósito de la trama API:
  - **API identifier** (cmdID): campo con el identificador de trama API. Los distintos identificadores los encontramos en las figuras 1.4 y 1.5
  - **Identifier-specific Data:** campo que contiene la información (datos).
- **Checksum:** es el último byte de la trama y nos ayuda a comprobar la integridad de los datos de la misma. Cálculo del checksum:
  - De una trama API (envío):
    1. Sumamos todos los bytes de la trama menos los tres primeros.
    2. De este resultado nos quedamos con los 8 bits menos significativos.
    3. Restamos esta cantidad a 0xFF.
  - Verificar checksum de una trama API (recepción):
    1. Sumar todos los bytes de la trama incluyendo el checksum (no incluir los tres primeros).

2. Si el checksum es correcto, los últimos dos dígitos de la derecha del resultado deben ser igual a 0xFF.

API frame names	API ID
AT Command	0x08
AT Command - Queue Parameter Value	0x09
TX Request	0x10
Explicit TX Request	0x11
Remote Command Request	0x17

Figura 1.4: Nombres e identificadores de las tramas API enviadas a un dispositivo (peticiones)

API frame names	API ID
AT Command Response	0x88
Modem Status	0x8A
Transmit Status	0x8B
Route information packet	0x8D
Aggregate Addressing Update frame	0x8E
RX Indicator (AO=0)	0x90
Explicit Rx Indicator (AO=1)	0x91
Data Sample Rx Indicator frame	0x92
Node Identification Indicator (AO=0)	0x95
Remote Command Response	0x97

Figura 1.5: Nombres e identificadores de las tramas API recibidas desde un dispositivo (respuestas)

Capítulo 2

Solución escogida

El coche autónomo definitivo necesita de un sistema eficaz de comunicación entre los vehículos, ya que esta comunicación nos permite anticiparnos a los sucesos, priorizar datos de sensores o obtener datos en tiempo real de escenarios futuros. Un sistema dual permite este nivel de confianza en la conexión y ofrece soluciones a los problemas de zonas lejanas.

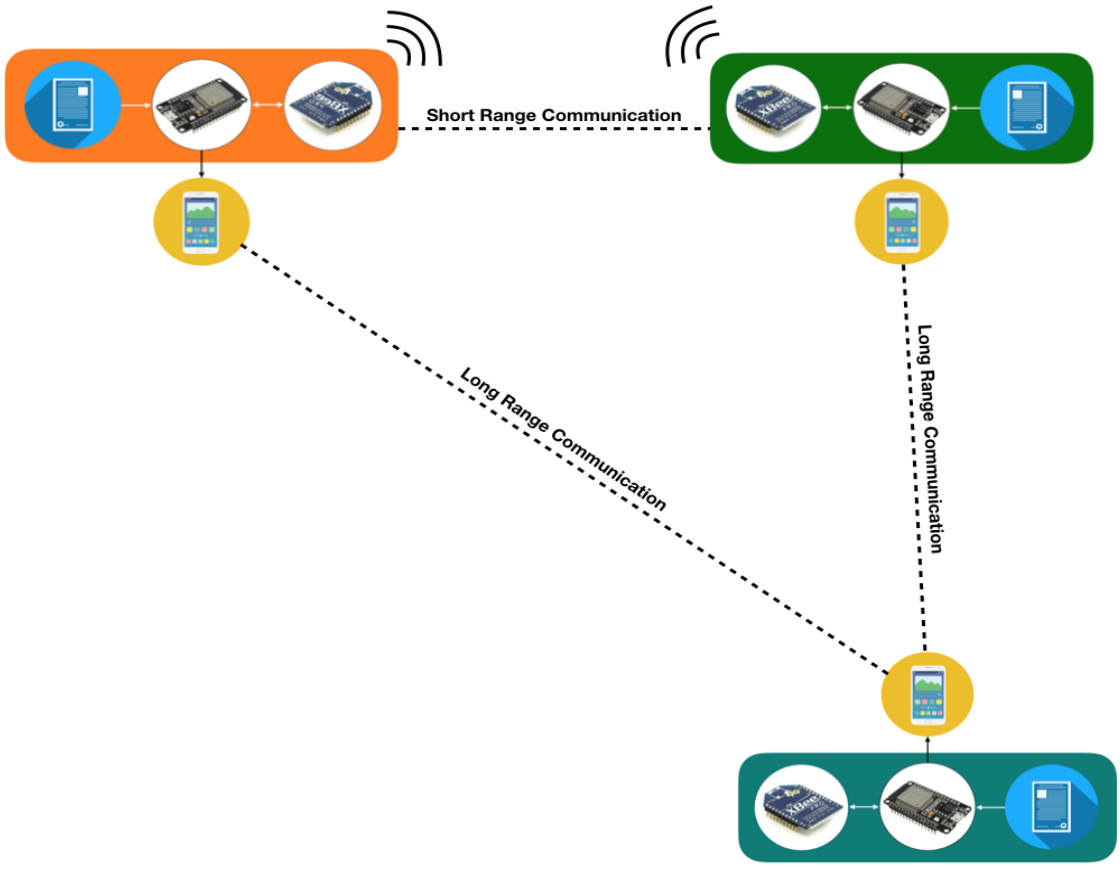


Figura 2.1: Esquema del sistema

Short Range and Cellular V2X communication propone un sistema de comunicación entre vehículos y entorno. Propone las capas físicas y de red necesarias para el intercambio de información. SRAC V2X está formado por dos sistemas de comunicación principales, uno de corto alcance y uno de largo alcance.

El sistema de comunicación de corto alcance (Short Range) se encarga de la comunicación entre los vehículos y su entorno. Por entorno definimos cualquier nodo de la red con el sistema de comunicación implementado. Esto nos permite interactuar con señales de tráfico, paneles informativos, semáforos o los sistemas deseados siempre que en estos implementemos el sistema de corto alcance. Las ventajas de este sistema son la capacidad de conectar con los actores vecinos de manera directa y rápida, permitiendo el intercambio de información de alta prioridad (detección de accidentes o excepciones) y poder convertirse cualquiera de estos en un repetidor que comparta la información con otros nodos vecinos las veces necesarias.

El sistema de comunicación de largo alcance (Cellular) permite exportar los datos relevantes de la comunicación de corto alcance para su análisis. Los datos son subidos a un servidor en internet, donde se almacenan para su posterior análisis. Una vez analizados estos datos podemos obtener información útil para otros vehículos o necesarios en otras zonas no alcanzables por la comunicación a corto alcance. Puede funcionar tanto como nexo entre comunicaciones de corto alcance, como transmisor general o como centro de procesamiento.

Mediante la simbiosis de estos dos sistemas se pretende otorgar de robustez y eficiencia a la comunicación entre vehículos para la prevención de accidentes y la recopilación de datos generales en vías y carreteras para uso estadístico. Con la sensorización adecuada se podrían prever potencialmente todos los accidentes por colisión o alcance entre vehículos, ya que estos podrían notificar las frenadas y maniobras antes de realizarlas y advertir a los coches colindantes.

En esta parte del proyecto nos centraremos en la parte de Short Range Communications (SRC) ya que es el primer paso para elaborar el sistema híbrido. Entendemos que esta parte de la comunicación debe ser la base en donde se establezcan los mensajes a enviar y las prioridades de estos.

En el siguiente semestre se abordará la parte de comunicación de largo alcance y la conclusión del proyecto de comunicación en su totalidad.

Una vez definida la funcionalidad que este dispositivo debe tener, seguimos el análisis para la solución del problema decidiendo los módulos que conformarán el sistema SRAC V2X.

Si se analizan detenidamente los requerimientos necesarios para la comunicación que deseamos establecer advertimos la necesidad de ciertos componentes básicos que cubrirán las demandas de la comunicación. Estos son una puerta a la comunicación de corto alcance, una puerta a la comunicación de largo alcance, un controlador que se encargue de coordinar el envío entre nodos y unos mensajes compartidos y entendibles por todos.

Capítulo 3

Desarrollo de la solución

Como ya hemos especificado anteriormente, el objetivo para este semestre es fijar la comunicación de corto alcance. El SRC se puede analizar desde diferentes puntos de vista. Para la mayor comprensión del proyecto creemos que la forma óptima es explicar los componentes que lo conforman y cómo estos interactúan con los demás.

Como ya se ha introducido en el planteamiento de la solución, el prototipo consta de varios componentes. El microcontrolador escogido será un módulo ESP-32 que ejercerá de coordinador entre las puertas de salida al exterior, a saber, un celular con conexión a Internet para la comunicación de largo alcance y un dispositivo Xbee para la comunicación de corto alcance. Además se explicarán los mensajes que se usarán y su proveniencia e importancia.

Procedemos a explicar estos componentes.

3.1. Mensajes

Como comentamos, los mensajes se necesitan para estandarizar el envío de datos. Para el desarrollo de estos se observaron diversos estándares, para acabar escogiendo los mensajes definidos en el estándar *ETSI TS 102 894-2 V1.2.1*. Este estándar nos brinda un conjunto de mensajes así como el formato en el que se deben transmitir, lo cual es muy útil para tener una base de la que partir para la creación de mensajes.

Se decidió por este estándar en concreto debido a la importancia de la entidad que lo emite, el Instituto Europeo de Estándares de Telecomunicación, o por sus siglas en inglés, ETSI. En concreto, el documento se trata de una Especificación Técnica (TS), que contiene los requisitos normativos y utilizada cuando el corto tiempo de lanzamiento al mercado, la validación y el mantenimiento son esenciales.

El estándar nos brinda el formato de los mensajes en ASN.1 (Abstract Syntax Notation One). Esta notación es una norma para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas. Esto nos permite implementar los mensajes directamente en el código del ESP-32 pasando los datos según nos indica el estándar.

En el anexo adjuntaremos el estándar que contiene todos los mensajes. De estos mensajes implementamos los 101 primeros (marcados como DE). A continuación explicaremos la estructura básica de estos mensajes.

- **Descriptive Name:** Indica un nombre para el mensaje que permita identificar rápidamente de cuál se trata.
- **Identifier:** indica el número de mensaje del que se trata.
- **ASN.1 representation:** Indica el formato en forma de norma ASN.1. Es importante porque esta será la parte en la que se indicarán los datos que deben ser pasados en los métodos de envío.
- **Definition:** Explica de qué se trata el mensaje. En el ejemplo de la figura 3.1 se puede ver cómo define los datos que se van a enviar y el significado que estos tienen.
- **Unit:** Especifica las unidades en las que se adjuntan los datos que se envía en el mensaje.
- **Category:** Clasifica el mensaje según la categoría a la que pertenezca.

A.19 DE_DeltaAltitude

Descriptive Name	DeltaAltitude
Identifier	DataType_ 19
ASN.1 representation	`DeltaAltitude ::= INTEGER {oneCentimeterUp (1), oneCentimeterDown (-1), unavailable(12800)} (-12700..12800)`
Definition	It defines an offset altitude with regards to a referred altitude value. It may be used to describe a geographical point with regards to a specific reference geographical position. Positive values are used for providing altitude offset above the reference position. For values equal or greater than 127,99 metres, the value shall be 12 799. Negative values are used for providing altitude offset below the reference position. When the information is unavailable, the value shall be set to 12 800. The DE is used in *DeltaReferencePosition* DF as defined in clause A.109.
Unit	0,01 metre
Category	GeoReference information

Figura 3.1: Mensaje sacado del estándar

3.1.1. Casos especiales

Debido al formato de trama utilizado en el ESP-32, ciertos mensajes se trasladan a otras direcciones, ya que sus originales se corresponden con tramas internas del XBee. Esto implica que no se pueden transmitir ya que el Xbee lo interpreta como órdenes y no lanza el mensaje al exterior. Estos casos son redirigidos por el mismo código de manera que no tenga influencia. Todo esto podemos verlo comentado en el código.

3.2. ESP-32 DEVKIT V1

El microcontrolador encargado de monitorear y dirigir todas las tareas del dispositivo necesitaría de tecnologías que permitieran una conexión con el módulo Xbee y el terminal Celular. El dispositivo ESP-32 DEVKIT V1 (a partir de este punto referido como ESP32) se trata de un microcontrolador programable en Arduino creado por DOIT que incorpora comunicación serie, WI-FI y Bluetooth. Este último será el encargado de enlazar el terminal Celular con el ESP32.

A lo largo de este apartado analizaremos los algoritmos utilizados en el ESP32 y explicaremos el funcionamiento básico de estos. Para facilitar una mayor comprensión se empezará explicando RTOS, que será determinante en la programación del prototipo y seguidamente los algoritmos diseñados para el desempeño de la comunicación.

3.2.1. Real Time Operating System (RTOS)

Una de las ventajas por las que el ESP32 es especialmente interesante para nuestro trabajo es porque permite la implementación de RTOS.

Es un sistema operativo en tiempo real (RTOS) kernel diseñado para sistemas embebidos. Es interesante para nuestro diseño puesto que los OS (Sistemas Operativos) se caracterizan por estar optimizados para la realización de muchas tareas complejas al mismo tiempo. Y ya que nuestro programa se caracteriza principalmente en el envío y recepción simultánea de datos (full dúplex), que mejor manera que poder tener 2 tareas independientes que se ejecutan en paralelo al mismo tiempo. Esto es posible gracias a la implementación de un RTOS en nuestro microcontrolador. Existen varios RTOS en el mercado, pero hubo uno que destacamos, freeRTOS que es compatible con el microcontrolador ESP32 además de estar bajo la licencia del MIT (licencia de software libre).

3.2.2. Algoritmos

Para la programación del microcontrolador se escogió programar 2 tareas utilizando freeRTOS con distintas prioridades que encargan de la transmisión y la recepción en paralelo.

Una vez terminado el proceso del Setup estas dos tareas se ejecutan dependiendo de la disponibilidad del procesador y de la prioridad asignada a cada una.

Procedemos a explicar las dos tareas con sus procesos y algoritmos individuales.

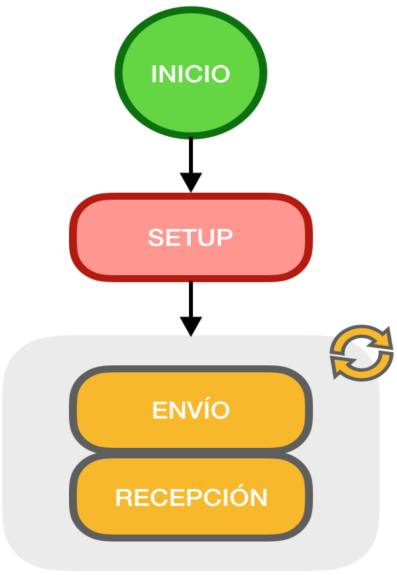


Figura 3.2: Diagrama programa del sistema

Una vez se enciende el ESP32, importamos las librerías e inicializamos las variables globales del sistema. Una vez hecho esto pasamos al setup del mismo.

En el setup a parte de inicializar los distintos elementos y variables que se utilizan en el programa, se crean las tareas del RTOS, en nuestro caso se crean dos tareas; "*TaskSend*" y "*TaskReceive*" que como su nombre indica se encargan de enviar y recibir respectivamente los distintos mensajes. Estas tareas contienen cada una un bucle infinito del que no salen nunca, es el TaskScheduler (Programador de tareas) el que determina qué tarea se ejecuta en función de su prioridad. Puesto que nuestra placa ESP32 dispone de 2 cores (núcleos), nos podemos permitir que cada una de las tareas la ejecute un núcleo, así que ambas tareas tienen la misma prioridad.

Envío

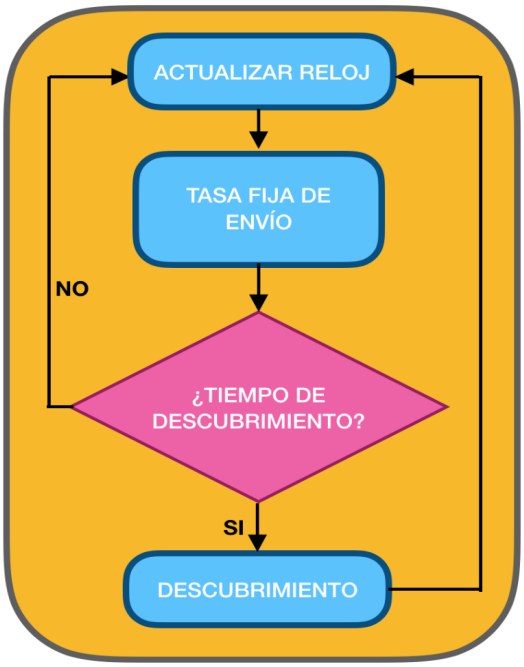


Figura 3.3: Diagrama de flujo de la tarea de envío

La tarea dedicada al envío de los mensajes sigue el esquema mostrado en la figura 3.3.

Se puede ver que consta principalmente de tres elementos, un envío de mensajes a una tasa fija, un envío de descubrimiento y un reloj.

La tasa fija de envío envía mensajes a una tasa escogida. Los mensajes enviados aquí serán los que estén normalizados por el estándar en el que nos basamos para la creación de estos. Paralelamente se va actualizando un reloj, encargado de contar hasta un tiempo prefijado al iniciarse el ESP32. Este reloj será el responsable de dar comienzo a la función de descubrimiento cada vez que vence el tiempo prefijado.

La función de descubrimiento no tiene un uso concreto en el desempeño actual del dispositivo, pero ofrece posibilidades para futuras configuraciones, como por ejemplo, saber periódicamente cuánta gente o quienes se encuentran en la red. Esto unido a un procesamiento podría derivar en la implementación futura de unicast o en la detección de caídas o accidentes por detección de pérdida de mensajes de descubrimiento.

Recepción

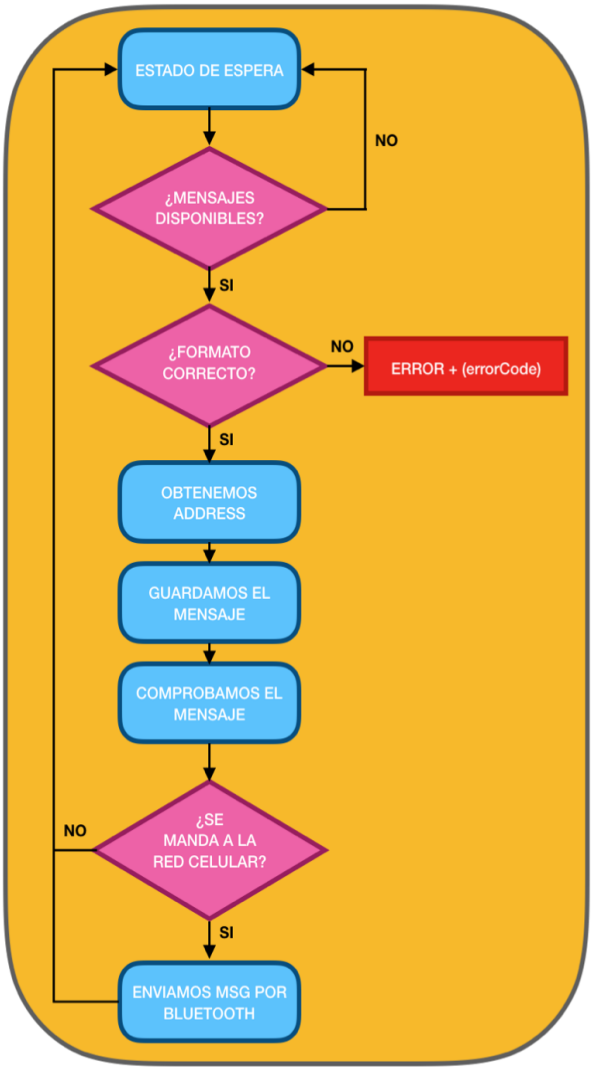


Figura 3.4: Diagrama de flujo de la tarea de recepción

El algoritmo de recepción es más complejo que el de envío. Una vez iniciada la tarea, el dispositivo espera cualquier mensaje que entre por el pin de recepción unido al Xbee. Se comprueba el formato de este y si se corresponde a los mensajes que esperamos recibir guardamos el mensaje y lo comprobamos. La subtarea encargada de comprobar el mensaje lo compara con los almacenados del estándar y lo decodifica. También nos informa de si éste debe ser enviado a la red celular y si lo es lo enviamos por ella a través del bluetooth. A continuación si los mensajes recibidos no se ajustan a una trama reconocible por el dispositivo, este nos informa de ello con un mensaje de error. Una vez terminados estos procesos, volvemos al estado inicial de espera de mensajes.

3.2.3. Códigos

Como ya se ha comentado, el microcontrolador se programó con el IDE de arduino y consta de diferentes tareas. Estas tareas se encargan de ejecutar los algoritmos explicados anteriormente. Todo el desempeño del código está comentado en el mismo de manera que éste pueda ser interpretado. En este apartado simplemente señalaremos las librerías utilizadas de manera que se entiendan las llamadas realizadas desde nuestro código principal. En nuestro programa se ha hecho uso de dos librerías:

- **BluetoothSerial.h:** librería que nos facilita la configuración y uso del sistema bluetooth integrado en el ESP32, es una librería que nos viene al incluir la placa (ESP32) en el IDE de Arduino.
- **XBee.h:** esta es una librería externa de un repositorio de github (ver referencias y anexos) que nos permite generar y decodificar las tramas API del Xbee de una forma sencilla.

3.3. Xbee

Por los requerimientos de la primera parte de este proyecto se necesita de una tecnología capaz de transmitir paquetes de información de forma constante en una red de arquitectura y nodos variables, donde cualquiera de estos nodos puede conectarse y desconectarse de manera rápida y sin ningún patrón.

Los módulos XBee son soluciones integradas que brindan un medio inalámbrico para la interconexión y comunicación entre dispositivos. Estos módulos utilizan el protocolo de red IEEE 802.15.4 para crear redes FAST POINT-TO-MULTIPOINT (punto a multipunto); o para redes PEER-TO-PEER (punto a punto). Fueron diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible. Por lo que básicamente XBee es propiedad de Digi basado en el protocolo Zigbee.

Debido a esto, el Xbee será el encargado de gestionar la comunicación a corto alcance. A continuación, pasaremos a explicar los parámetros configurados, justificando las opciones elegidas de manera que se pueda programar un dispositivo Xbee para su uso en esta aplicación a partir de modificar estos campos.

Los dispositivos utilizados en el proyecto se corresponden a la versión Xbee Series 3 a los que les añadimos unas antenas de una ganancia de 2.1dBi para aumentar el rango de alcance de nuestro dispositivo.

3.3.1. Configuración Xbee

Para la programación del dispositivo Xbee utilizamos el Software de libre acceso XCTU disponible desde la plataforma OnLine de Digi. El programa detecta los dispositivos Xbee que se conectan al puerto USB del ordenador y su firmware y dependiendo de la versión con la que trabajamos tenemos diferentes opciones de configuración.

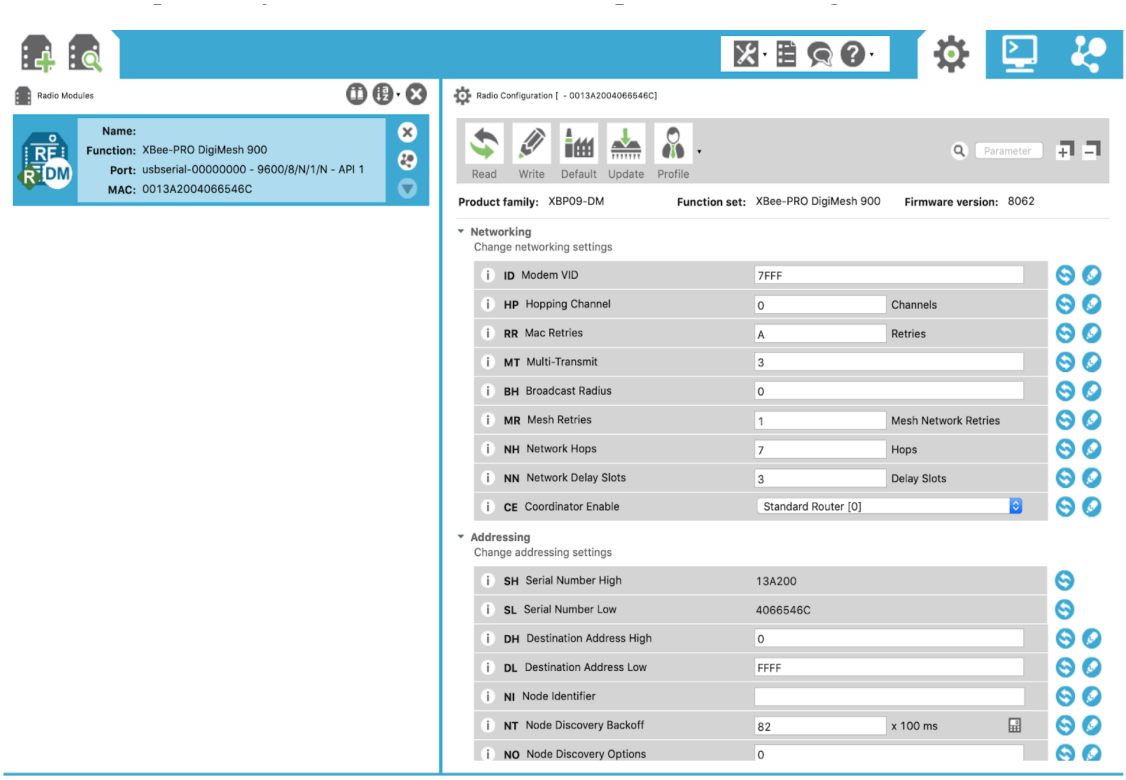


Figura 3.5: Pantalla Inicio XCTU

Una vez detectado el dispositivo se nos ofrecen los campos disponibles para modificar. Para el correcto proceso de programación de los Xbee es necesario modificar ciertos parámetros, estos implementan características diferentes a la comunicación y seguidamente explicaremos la configuración escogida y los parámetros modificados. Los campos modificados son:

- **Modem VID (ID):** Consiste en un identificador de red. Sólo los dispositivos con números de ID coincidentes pueden comunicarse entre sí. 0xFFF es el valor de fábrica. Escogimos 7FFF como valor aleatorio y configuramos todos los dispositivos con el mismo ID.
- **Encryption Enable (EE):** Habilita o inhabilita una encriptación AES de 128 bits. Actualmente no está activado en los dispositivos, pero se podría activar con el objetivo de proporcionar mayor seguridad al sistema.

- **AES Encryption Key (KY):** En caso de estar activado el Encryption Enable, el valor de este parámetro servirá de clave para la desencriptación AES.
- **API Enable (AP):** Selecciona el tipo de tramas que se usarán para la comunicación entre Xbee. En nuestro caso seleccionamos el modo API con caracteres de escape, ya que para las pruebas que realizamos priorizamos catalogar el rendimiento del Xbee y añadir caracteres de escape podría alargar el tiempo de procesamiento de las tramas.
- **API Options (AO):** Seleccionamos la opción DigiMesh que será la arquitectura que seguirá nuestra red.

Adjuntamos una imagen con los parámetros cambiados resaltados donde se puede apreciar el valor de configuración escogido para estos.

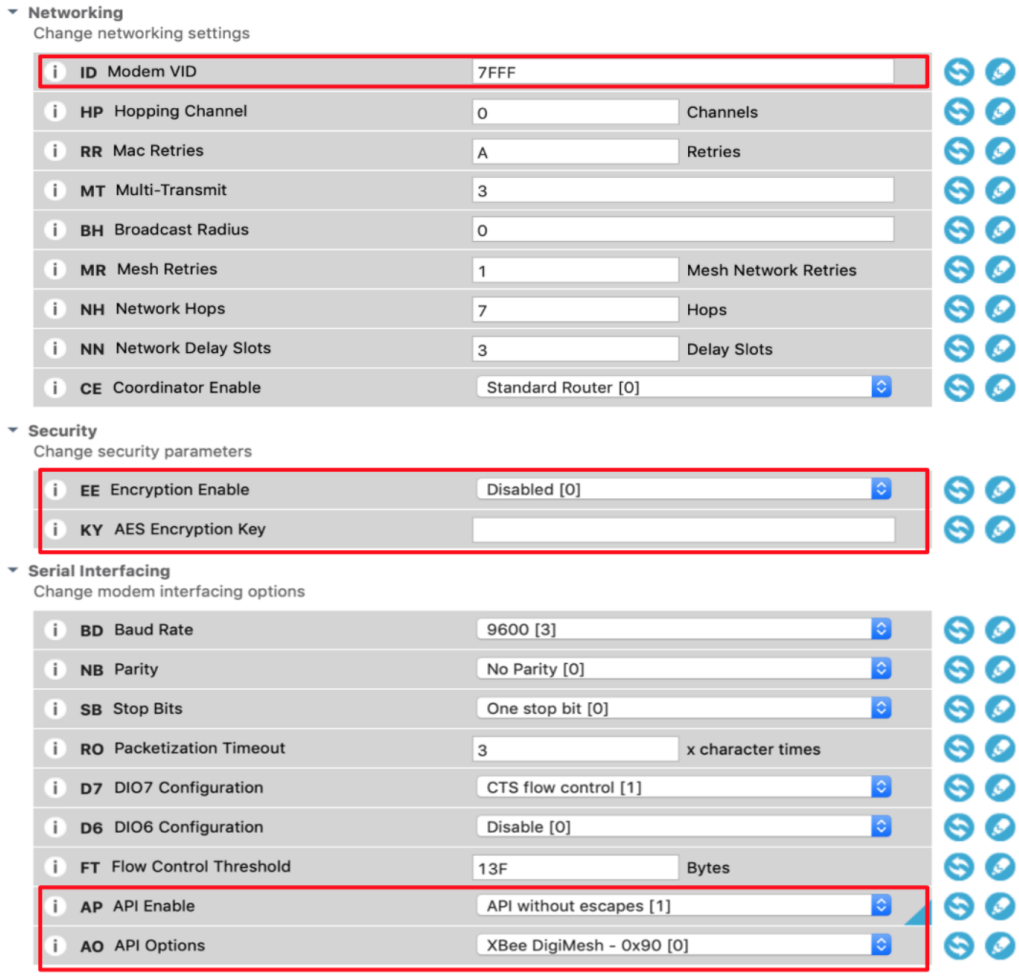


Figura 3.6: Parámetros configurados en el XCTU

3.3.2. Comunicación Xbee-ESP32

Para finalizar con el análisis del dispositivo Xbee, describiremos su comunicación con el ESP32. La comunicación entre estos dos se realiza mediante los pines de transmisión y recepción, TX2 y RX2 respectivamente. Utilizando como adaptador un módulo regulador que nos proporciona los pines de alimentación y masa y de transmisión y recepción.

La comunicación se lleva a cabo de manera serial. El ESP32 envía por el puerto TX2 los mensajes directamente encapsulados en la trama API, función que realiza la librería Xbee.h, y el Xbee se encarga de enviarlos al exterior por radiofrecuencia.

3.4. Terminal Móvil

Como ya se ha comentado, el terminal móvil será la puerta para la futura implementación de la subida de datos a la nube. Es por tanto un componente fundamental en el dispositivo que se situará en los vehículos. Aquí explicaremos el app programada para el móvil tanto en cuanto a funcionamiento como las herramientas utilizadas.

Empezando por las herramientas utilizadas, para programar la aplicación utilizamos la herramienta disponible de manera gratuita y onLine MIT App Inventor. Esta app nos permite programar aplicaciones por bloques e interactuar con dispositivos internos del celular como es el Bluetooth, necesario para la comunicación con el ESP32.

La app programada realiza dos funciones principales, en primer lugar, es la receptora de los mensajes del ESP32, posteriormente, envía esos mensajes a una hoja excell online que simula un servidor donde se almacenarían los datos para un posterior análisis de ellos.

Los bloques mostrados a continuación son los encargados de la parte de conexión al ESP32 mediante Bluetooth. Nos muestra una lista de los dispositivos posibles a los que conectarse y nos conecta al que seleccionemos mostrándonos un mensaje que nos indica si la conexión ha sido exitosa o si ha habido algún problema.

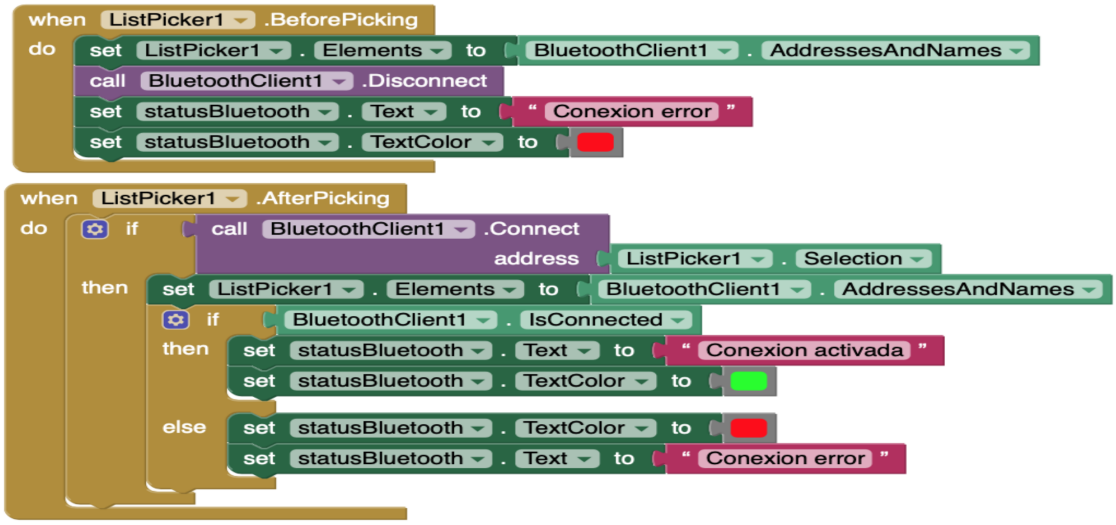


Figura 3.7: Código app encargado de la conexión bluetooth

Una vez conectado a un dispositivo Bluetooth actuará este apartado de bloques. Como se puede ver, si se está conectado a un dispositivo Bluetooth y hay Bytes disponibles para la recepción, entonces se reciben los Bytes disponibles y se almacenan en la variable global mensaje y se muestran por pantalla.

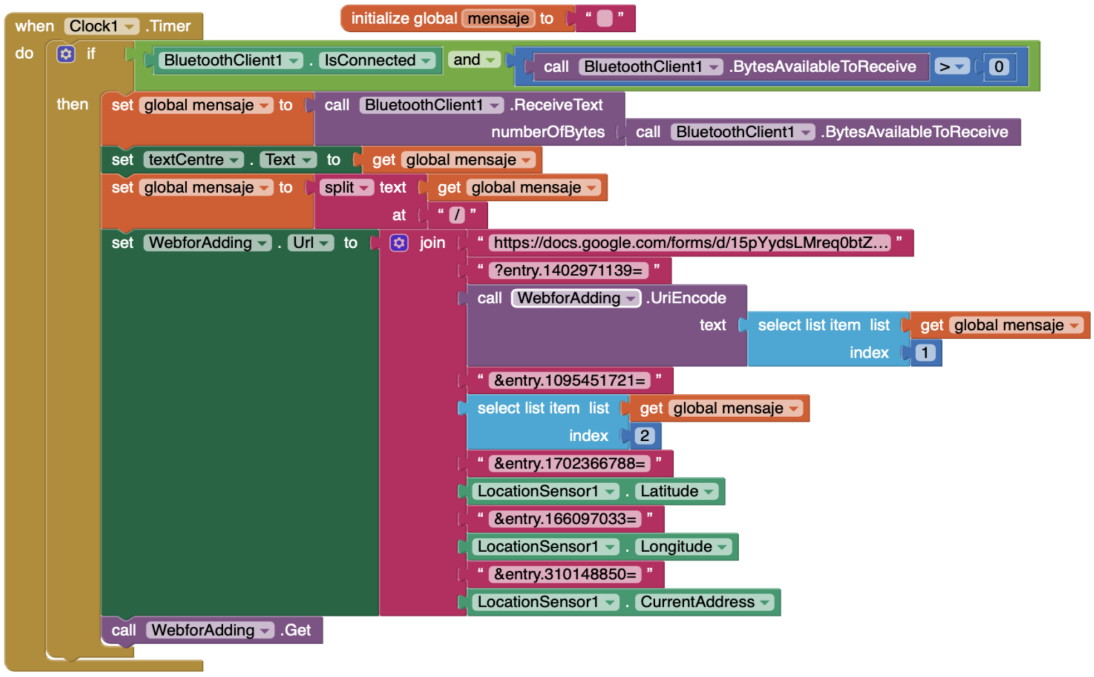


Figura 3.8: Código app encargado de la conexión a la nube

El apartado verde de mayor tamaño se encarga de enviar esos datos a la hoja drive excell. Esto lo hace enviando los datos mediante la conexión WiFi o de itinerancia de datos del teléfono a la dirección web que se le adjunte. Como extra enviamos también las coordenadas GPS del teléfono así como la dirección de la calle actual en la que se encuentra para poder preveer colisiones en un futuro, sacar información de seguimiento de rutas o detectar la posición de accidentes en cuanto sean reportados.

3.5. Prototipo

Una vez estos componentes estén programados y listos para ensamblar, se incorporan en una protoboard para probar su funcionamiento. En las siguientes imágenes se puede ver el prototipo completado que se usó para los experimentos y las pruebas.

3.6. Caracterización del dispositivo

Una vez implementadas y unidas estas tecnologías pasamos a realizar pruebas básicas de su funcionamiento con el fin de averiguar parámetros relevantes. En los siguientes apartados describiremos el diseño de los experimentos y analizaremos los resultados obtenidos.

3.6.1. Diseño de los experimentos

Se realizaron las pruebas del sistema en dos tipos de entorno. Primero mediante unas pruebas en el laboratorio donde se desarrolló el proyecto y unas segundas en el exterior, buscando simular un entorno real.

Entorno controlado (laboratorio)

Para la correcta caracterización de los dispositivos centramos los experimentos en dos bloques principales, tasa de comunicaciones y retardos. Esto nos ayuda a entender la eficiencia del sistema y las situaciones críticas a las que nos podemos enfrentar.

Tasa de comunicaciones y retardos

El objetivo de este experimento es obtener el retardo entre paquetes típico en base a diferentes parámetros. Los parámetros que vamos a analizar son tipo de comunicación, número de caracteres enviados y diferentes tasas de transmisión. Esto nos ayudará a hacernos una idea del rendimiento del dispositivo en diferentes situaciones y poder fijar futuros parámetros en los experimentos en escenarios más cercanos a la realidad.

Respecto al tipo de comunicación analizamos dos posibles escenarios, full duplex o transmisión en vacío (sin nada en el buffer).

Cabe destacar que una vez estuvieron estas pruebas, se realizaron unas nuevas aislando las tareas de envío y recepción en dos núcleos separados. Aquí podremos ver las prestaciones definitivas del sistema y podemos compararlas para ver las mejoras que implementa la utilización de un sistema multicore.

Pruebas de transmisión en vacío En estas pruebas se utilizaron 2 dispositivos en las cuales uno de ellos actuaba como emisor y uno actuaba como receptor. Se probaron diferentes tasas con un número fijo de caracteres para tratar de caracterizar el procesamiento del dispositivo receptor y las máximas velocidades de transmisión del dispositivo emisor. El número de caracteres fueron 5 ya que en estas pruebas no nos interesaba llegar a los casos extremos del buffer en cuanto a caracteres, solo averiguar su comportamiento de velocidad de envío y recepción y las tasas escogidas para esto fueron el envío de paquetes cada 250 ms, 125 ms, 50 ms, 30 ms y 5 ms.

A continuación se muestra la gráfica en la que se ven los retardos de llegada entre paquetes consecutivos. En este gráfico, el eje X nos indica el número de mensaje y el eje Y nos indica el valor del retardo temporal entre paquetes consecutivos en milisegundos.

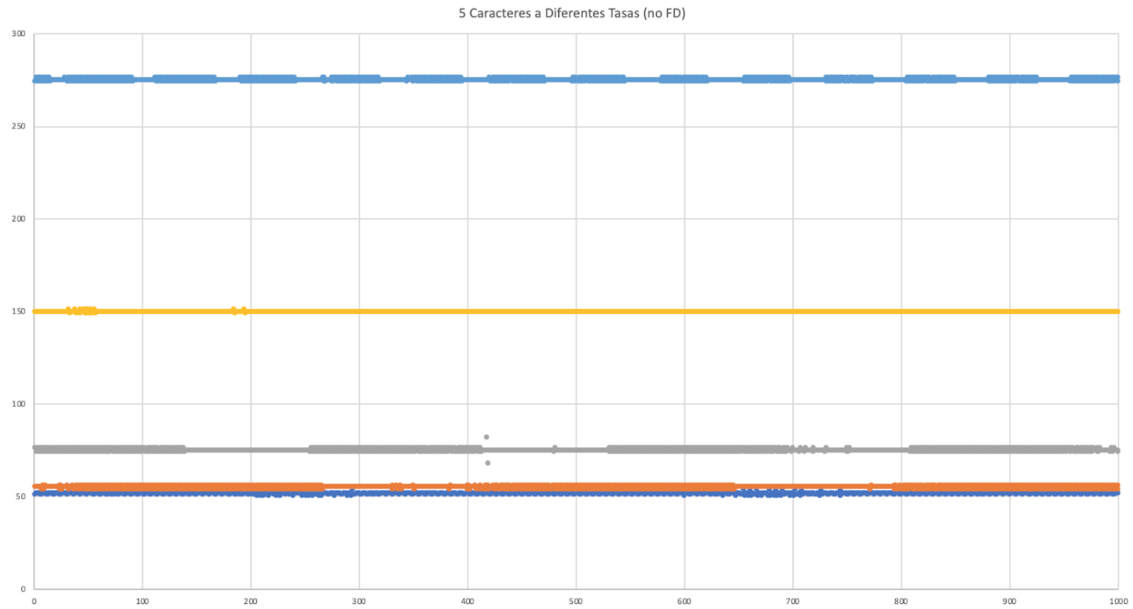


Figura 3.9: 5 caracteres diferentes tasas

El primer dato relevante que podemos extraer de estas transmisiones es el retardo que aparece debido al procesamiento de la trama. Aproximadamente es de unos 25 ms, tiempo en el que el ESP32 recorre el código. Una vez identificado esto podemos ver que se forman líneas casi perfectas en 275 ms, 150 ms, 75 ms, 55 ms y 51 ms.

En la siguiente ampliación se puede ver que las diferencias máximas entre retardos de mensajes son de 2 ms, variaciones mínimas y debido a que el buffer se autocompensa. Es decir, si un paquete es procesado más despacio, el siguiente será procesado más deprisa. Esto nos acompañará en todas las pruebas que realicemos, donde se podrá ver a simple vista que las líneas de retardo correctas tienen simetría horizontal desde su valor medio.

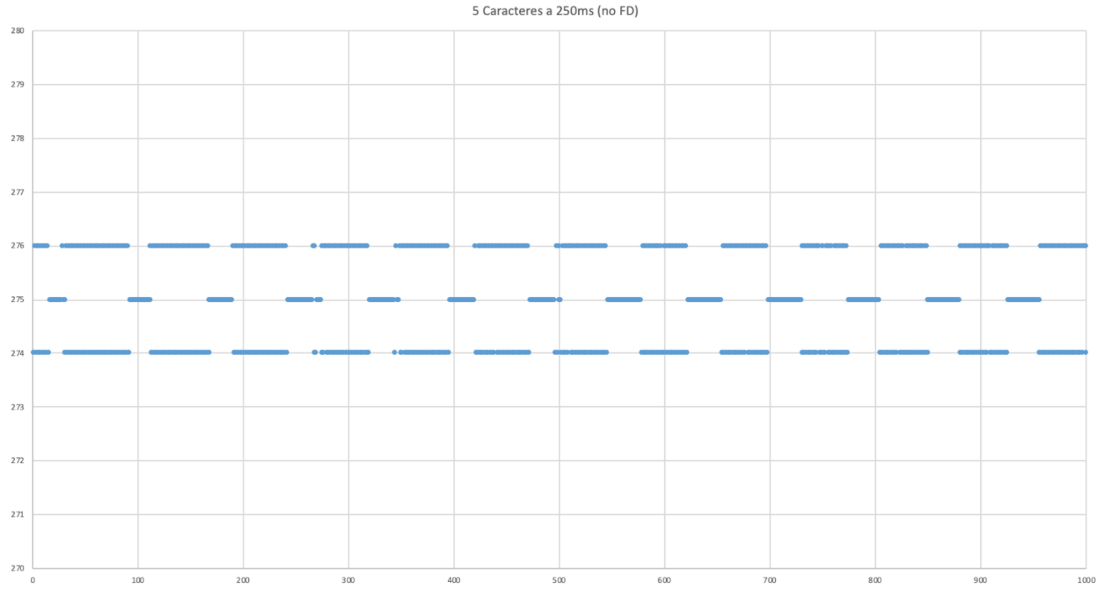


Figura 3.10: 5 caracteres a 250ms

Como podemos ver en el primer gráfico, donde se muestran todas las tasas superpuestas, el buffer de envío se satura y no permite el envío a velocidades mayores de 50 ms. Teniendo en cuenta el retardo de procesamiento y dejando un margen de seguridad de 5 ms para el buffer, asumimos que la máxima tasa de envío en nuestro programa es de 30 ms y se corresponde a 55 ms incluyendo la ejecución del programa.

Aparte de obtener las tasas de envío, con estas pruebas determinamos que si se tiene el buffer despejado se pueden obtener retardos mínimos en la recepción del mensaje, opción interesante si se aplica a entornos con gran tráfico de datos o momentos en los que no ocupamos el buffer ya que no enviamos información relevante, por ejemplo, si estamos aparcados pero todavía no nos incorporamos a la vía.

Pruebas de transmisión Full Duplex En estas pruebas realizamos comunicaciones en las que varios nodos transmiten a la vez. Esto nos acerca un poco más a una situación real, donde no solo importa recibir mensajes, sino tener el buffer disponible para enviarlos. Primero analizaremos las primeras pruebas, en las que se varían las tasas y el número de caracteres.

Este experimento brindará mucha información respecto a cómo se comporta el dispositivo en situaciones de tasas rápidas o de grandes cadenas de caracteres mientras además de recibir datos, los envía, es decir, generando retardos en el buffer debido a la ocupación generada al enviar.

En los siguientes gráficos se pueden ver los retardos consecutivos entre paquetes para diferentes tasas. Cada uno de estas gráficas se corresponde con un número diferente de caracteres. Estos serán 5 caracteres, 8, 12, 16 y 19. Las tasas de envío serán las mismas en ambos nodos y se corresponderán a 250 ms, 125 ms, 50 ms y 30 ms. Cabe señalar que las siguientes pruebas se realizaron utilizando un solo núcleo para poder comparar los resultados cuando se implementa la separación de tareas en dos núcleos diferentes.

- 5 caracteres

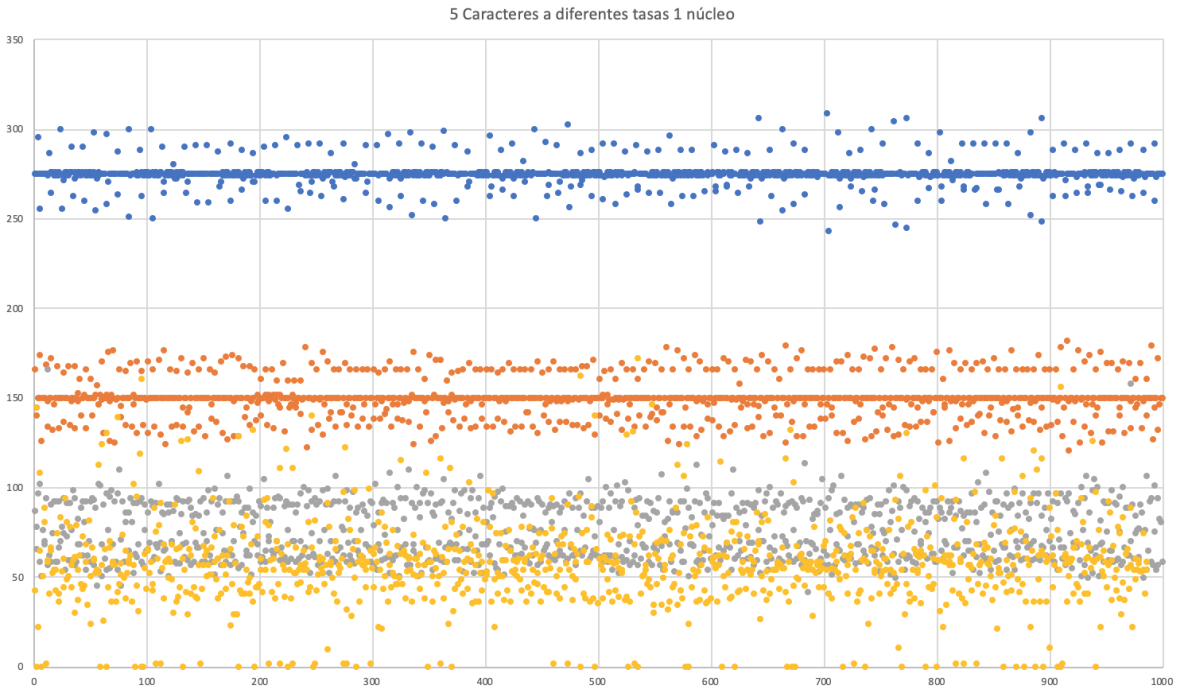


Figura 3.11: 5 caracteres a diferentes tasas 1 núcleo

- 8 caracteres

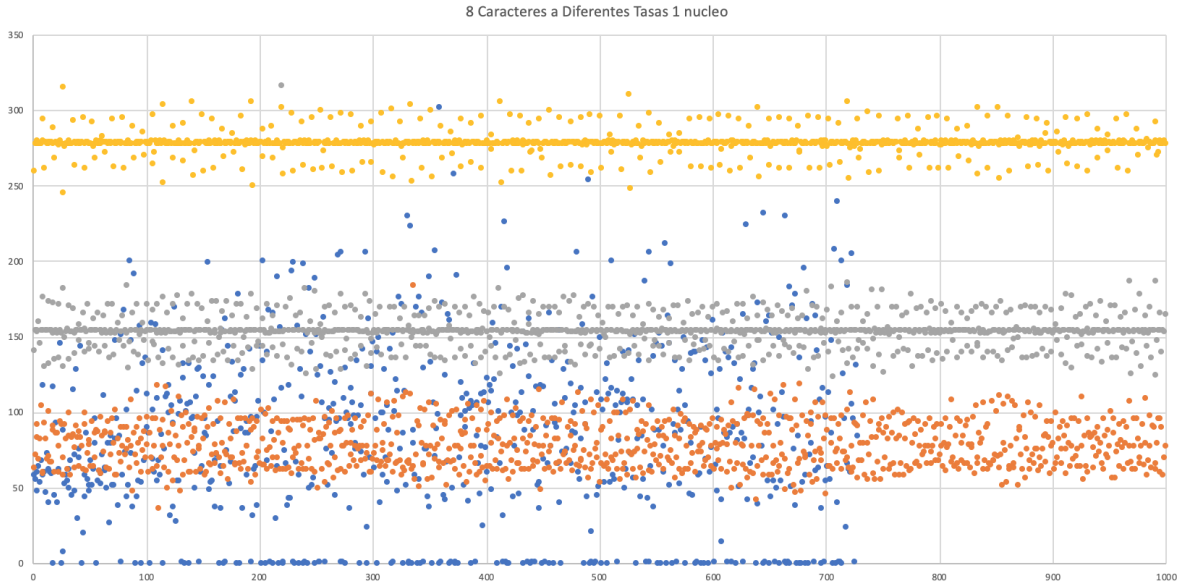


Figura 3.12: 8 caracteres a diferentes tasas 1 núcleo

- 12 caracteres

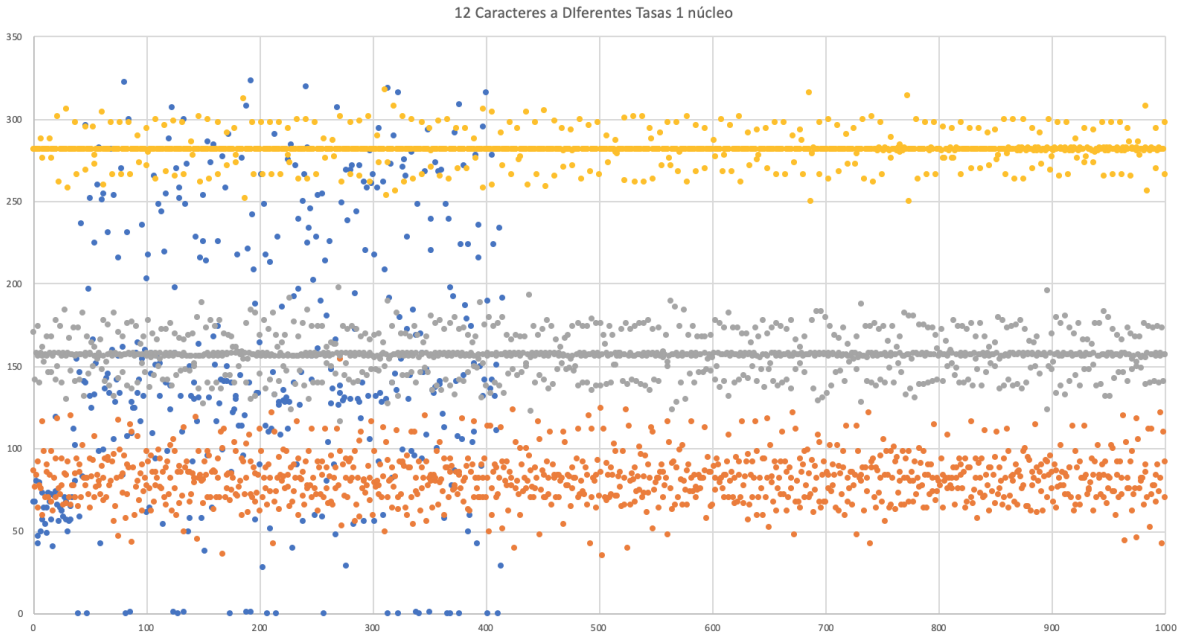


Figura 3.13: 12 caracteres a diferentes tasas 1 núcleo

- 16 caracteres

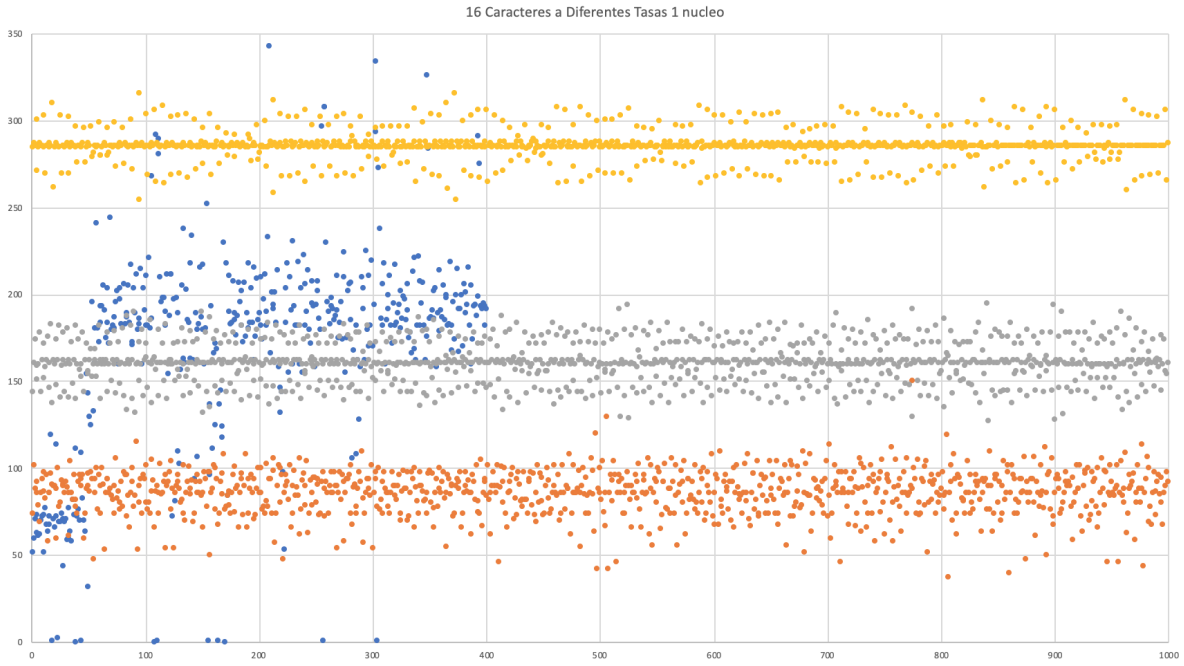


Figura 3.14: 16 caracteres a diferentes tasas 1 núcleo

- 19 caracteres

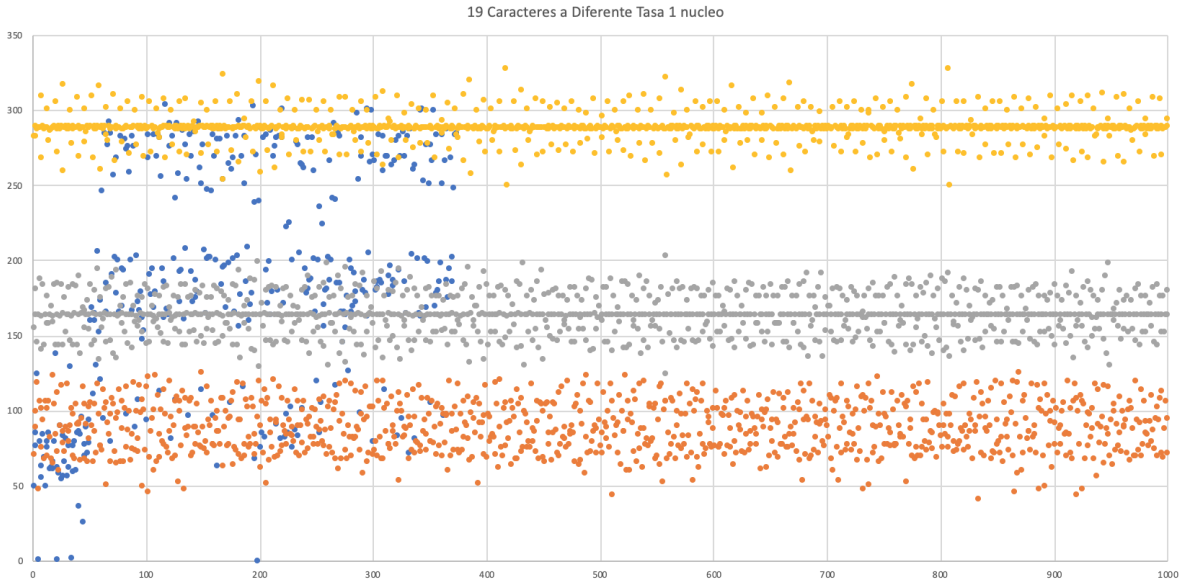


Figura 3.15: 19 caracteres a diferentes tasas 1 núcleo

Como podemos ver, estas son las gráficas que corresponden al experimento anteriormente descrito. En ellas se pueden ver claramente la tendencia de las dispersiones, que es la esperada. A mayor tasa, mayor dispersión.

Evidentemente esto ocurre en todas las gráficas y es debido a que a mayores tasas, menor es el tiempo entre paquetes, es decir, mayor es la ocupación del buffer y más irregular se vuelven los tiempos de envío y recepción. Esto provoca también retrasos a la hora de enviar los paquetes, que se suman a los retrasos comentados anteriormente de 25ms. Por tanto, nuestras tasas de transmisión vuelven a reducirse y además se suma la aleatoriedad de los retrasos en el buffer.

Se puede observar también que esto no solo depende de la tasa, sino también del número de caracteres. Si se presta atención a la tasa de envío de 125 ms (que se sitúa aproximadamente en los 160 ms), se puede observar la tendencia de aumentar la dispersión en el retardo de mensajes cuanto más larga es la trama.

Esto llega a un punto crítico en cuanto intentamos transmitir con la tasa de 30 ms, que aunque es soportada para tramas cortas como pueden ser tramas con un mensaje de 5 caracteres, se aprecian pérdidas en tramas más largas como las de 8 caracteres. Estas pérdidas son debidas a los descartes de paquetes ocurridos en el buffer debido a que está ocupado por la transmisión en ese momento y el paquete no se puede procesar. En 8 caracteres, las pérdidas ya son muy significativas (aproximadamente del 27 %), pero se vuelven insostenibles al aumentar el número de caracteres. Con 12 caracteres tenemos pérdidas del 58 % mientras que con 16 caracteres las pérdidas aumentan a un 60 %. Como es de esperar, las máximas pérdidas se corresponden con la máxima longitud testeada, que son 19 caracteres, donde se alcanza un 63 % de pérdidas.

Como veremos más adelante, esto justifica la utilización de un dispositivo capaz de llevar el procesado de la recepción de mensajes y el procesado del envío de mensajes en dos procesadores diferentes para así aumentar la eficiencia del sistema.

Siguiendo con esta línea, se implementaron las tareas de envío y recepción en dos núcleos separados con el objetivo de reducir la carga de procesamiento al dividirla en dos núcleos independientes. Una vez implementado, se realizaron las mismas pruebas mencionadas anteriormente.

Debido a que los resultados mejoraron notablemente, las pruebas solo se realizaron en 125 ms, 50 ms y 30 ms. La tasa de 250 ms no aportaba información ya que su dispersión era mínima debido a la mejora que supuso la utilización de los dos núcleos.

En los siguientes gráficos se podrán observar los retardos de dispersión y las características de formato del gráfico son iguales a los de los gráficos comentados en las pruebas anteriores.

- 5 caracteres

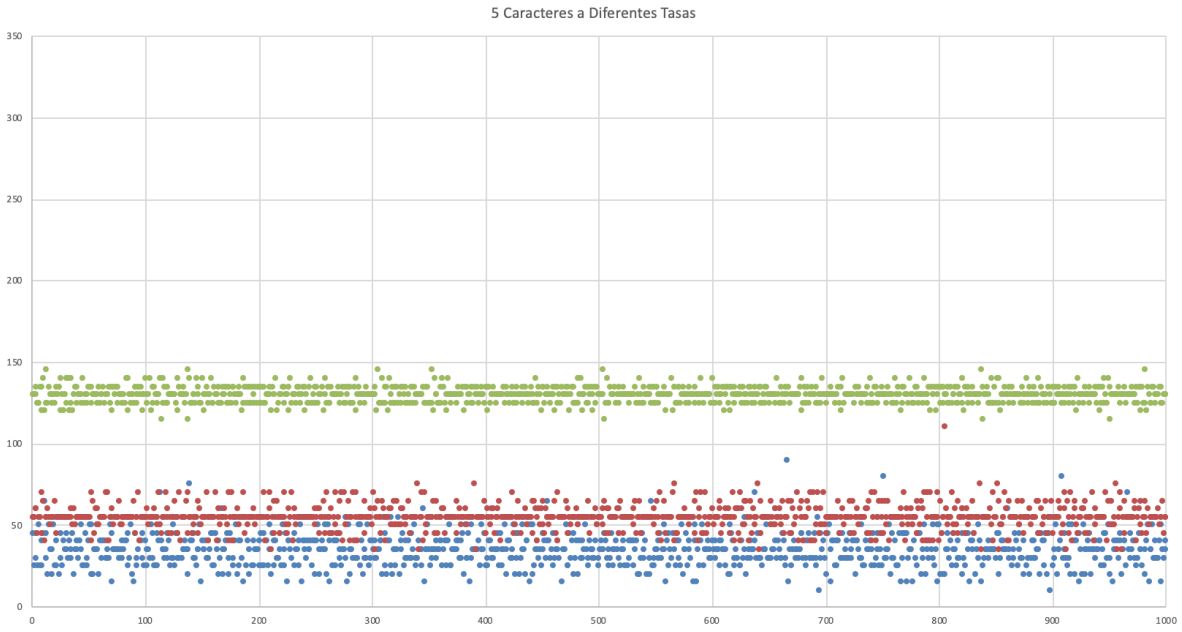


Figura 3.16: 5 caracteres a diferentes tasas 2 núcleos

- 8 caracteres

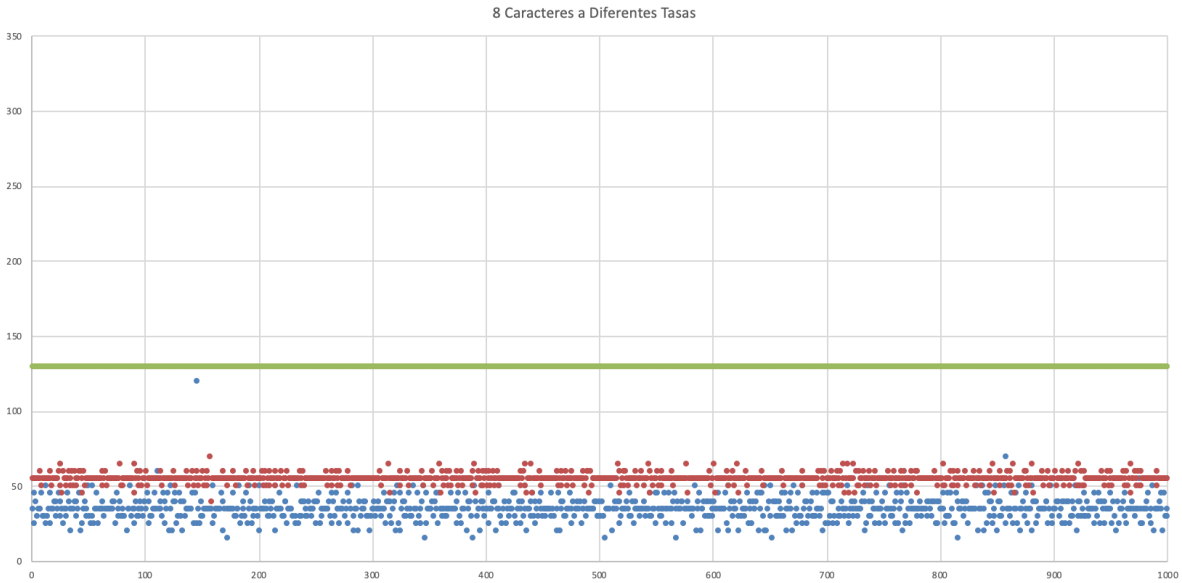


Figura 3.17: 8 caracteres a diferentes tasas 2 núcleos

- 12 caracteres

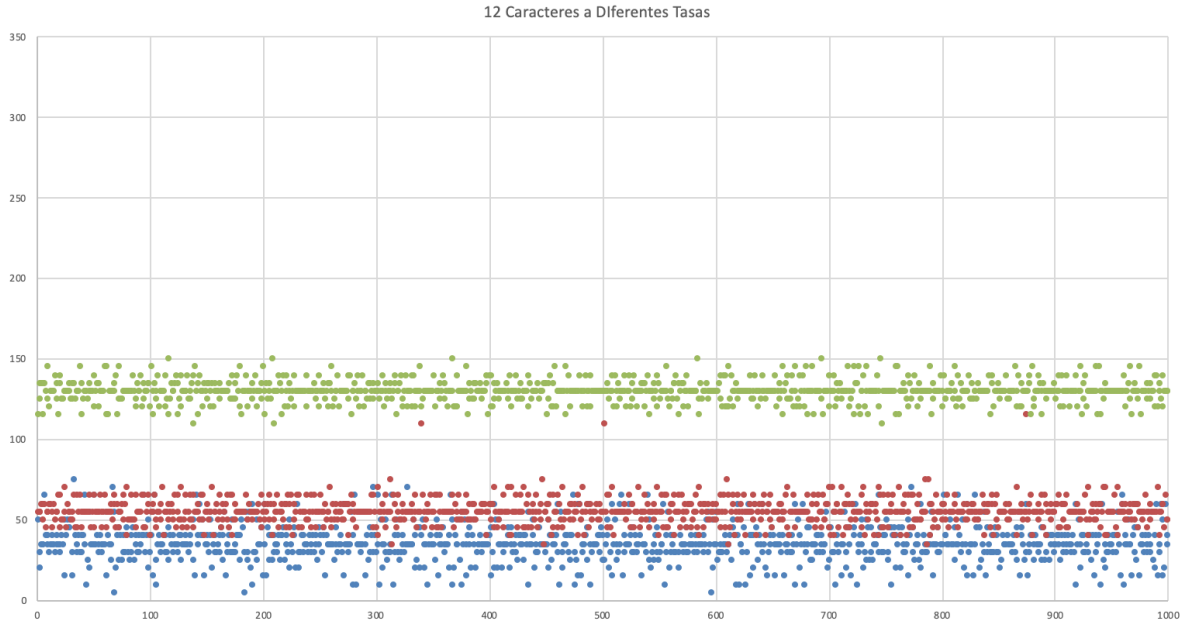


Figura 3.18: 12 caracteres a diferentes tasas 2 núcleos

- 16 caracteres

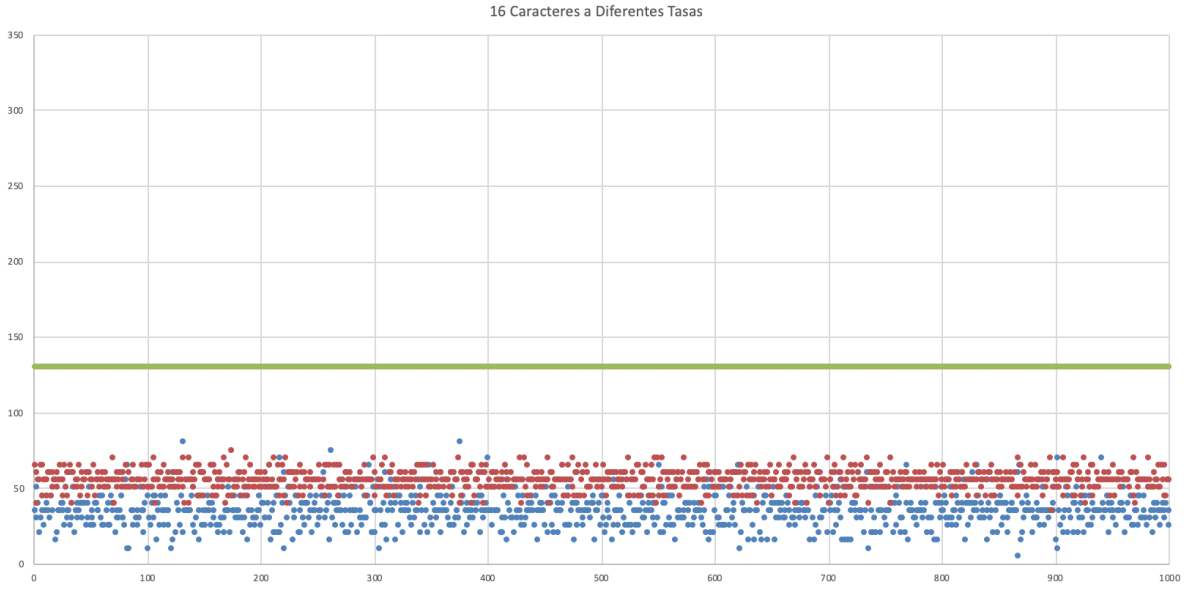


Figura 3.19: 16 caracteres a diferentes tasas 2 núcleos

- 19 caracteres

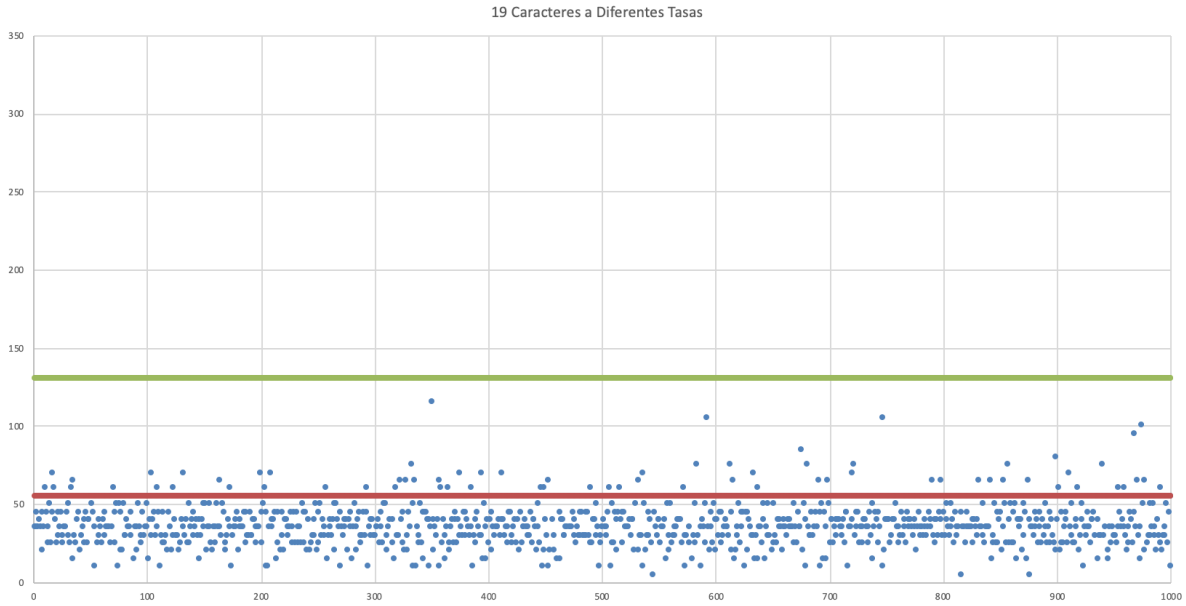


Figura 3.20: 19 caracteres a diferentes tasas 2 núcleos

Como se puede ver, esta vez los valores salen ordenados en pequeños intervalos regulares. Esto es debido a que la implementación en dos núcleos de las tareas requiere que estas tengan cierto retraso para dejar a los demás componentes (en este caso el Xbee) realizen sus tareas. El mecanismo es igual que un *delay()* de arduino y se llama *TaskDelay()*. Este *TaskDelay()* provoca que los retrasos ocurran de forma ordenada en intervalos que nosotros podamos controlar. En estas pruebas el *TaskDelay()* estaba fijado en un valor de 5 ms.

Aparte de dar tiempo a los componentes periféricos, el *TaskDelay()* es un parámetro utilizado para poner la tarea en suspensión para que el programador de tareas interno del ESP32 no entre en un bucle infinito, es decir hacer que las tareas se ejecuten y entren en un modo de reposo para el posterior reinicio de estas tareas. Esto tiene repercusión directa en la dispersión de los mensajes. Para encontrar el *TaskDelay()* óptimo para cada tasa se decidió hacer un barrido de diferentes tasas y diferentes *TaskDelay()* que nos sirviera de tabla de referencia de valores. El gráfico en cuestión es el siguiente.

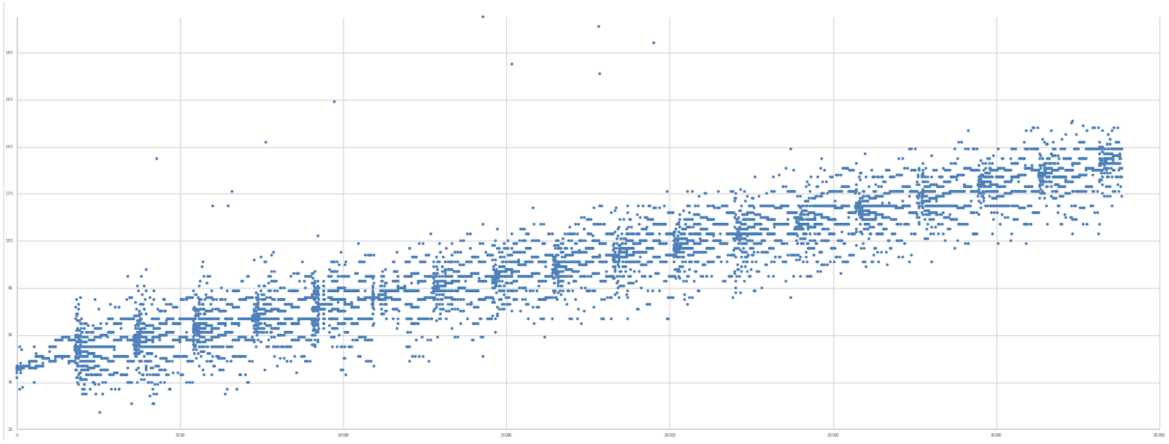


Figura 3.21: Diferentes tasas con diferentes vTaskDelay

Lo que se aprecia en este gráfico es el conjunto de mensajes enviados para comprobar el efecto del TaskDelay(). El proceso del programa fue el siguiente:

1. Se fija una tasa de transmisión.
2. Se fija el **TaskDelay()** en 1 ms.
3. Cada 200 mensajes se aumenta el **TaskDelay()** en 1 ms.
4. Cuando el **TaskDelay()** es 10 ms se envían los 200 mensajes y se aumenta la tasa en 5 ms.

Esto nos permite obtener 2000 mensajes de muestra por cada tasa, y dentro de estos 2000 mensajes, 200 mensajes de muestra por cada **TaskDelay()** entre 1 ms y 10 ms. A continuación vemos esto con más detalle.

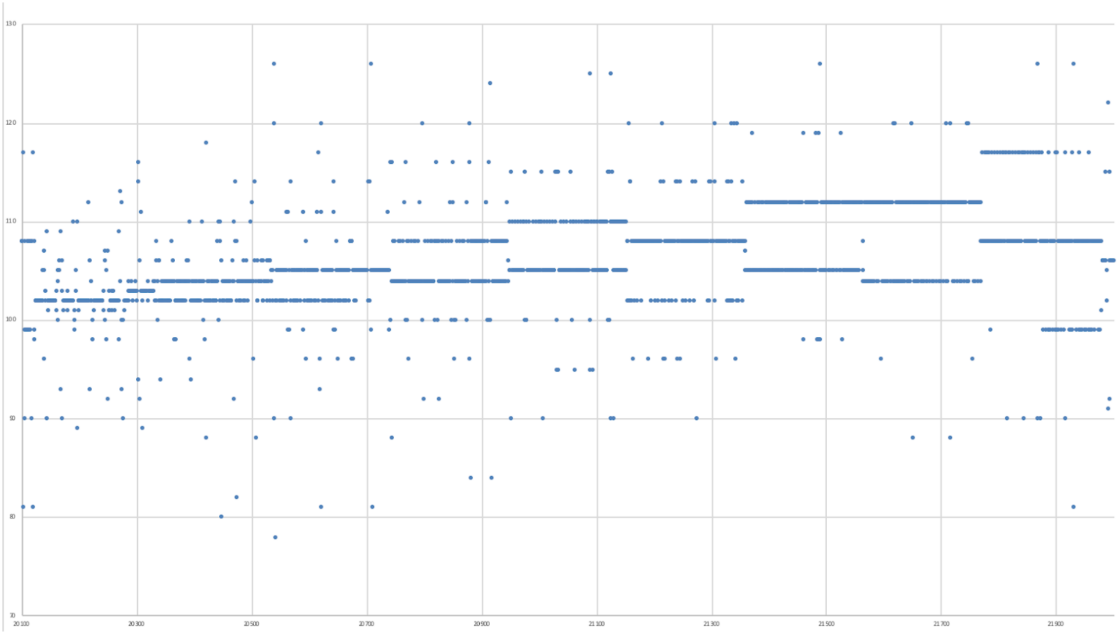


Figura 3.22: Zoom diferentes vTaskDelay a 100ms

Esta gráfica se corresponde con un zoom a la gráfica anterior para centrarnos en los mensajes 20000 al 22000 y así poder observar los diferentes TaskDelay(). Como vemos, los TaskDelay() menores a 4 ms generan una dispersión más descontrolada que los de valores superiores. Por tanto, si quisiéramos trabajar con 100 ms de tasa, deberíamos de escoger un TaskDelay() mayor a 4 ms.

Como hemos comentado anteriormente, para asegurar un valor medio de **TaskDelay()** que nos permitiera trabajar sin demasiada dispersión, se fija el valor de 5 ms.
A continuación realizaremos una comparativa de las gráficas unicore y multicore con la intención de evidenciar la necesidad del uso de un sistema RTOS para garantizar un buen desempeño en la comunicación entre dispositivos.

- 12 Caracteres en Unicore y en Multicore.

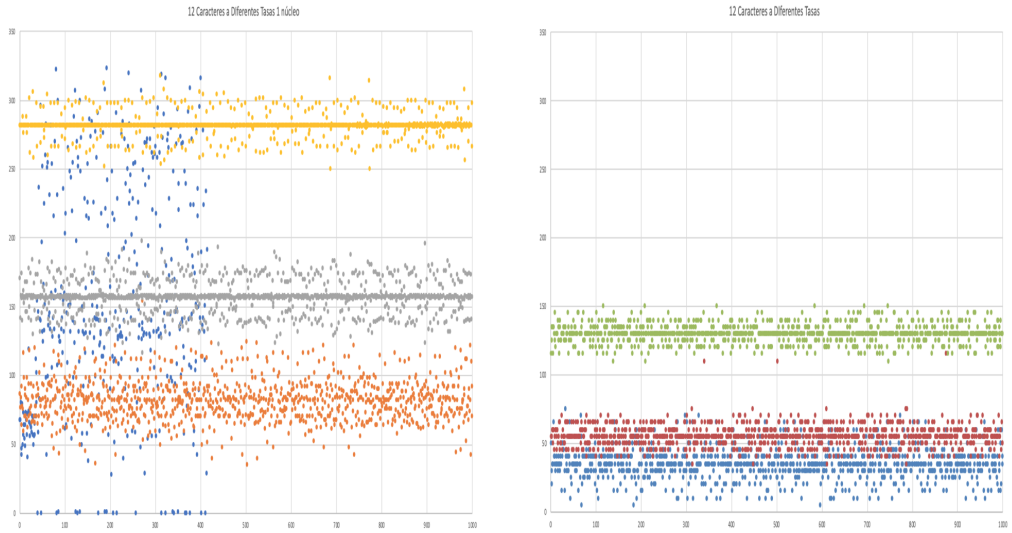


Figura 3.23: Comparación 12 carcteres unicore frente a multicore

- 19 Caracteres en Unicore y Multicore.

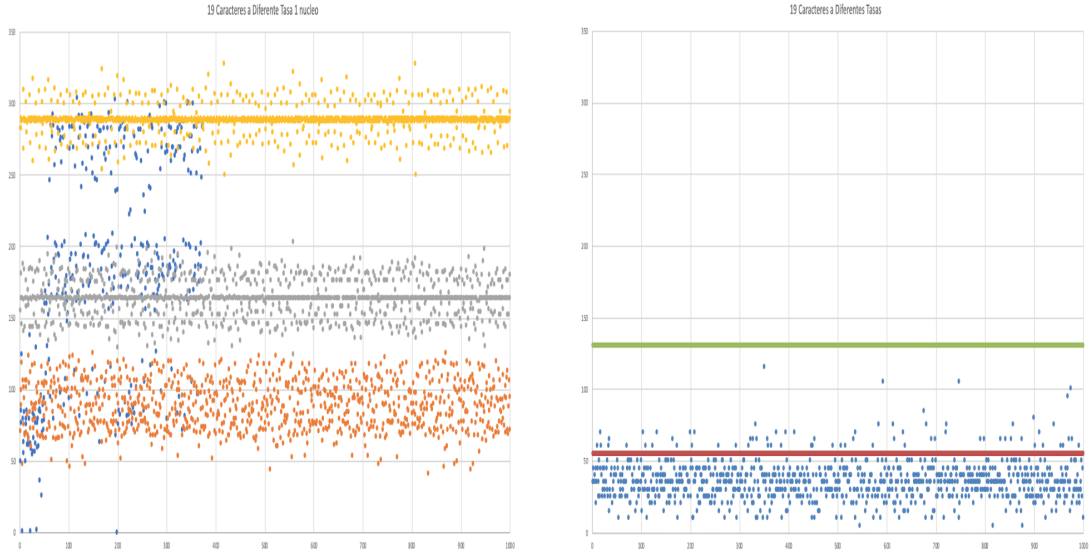


Figura 3.24: Comparación 19 carcteres unicore frente a multicore

Como podemos ver, al implementar las tareas en núcleos separados la dispersión disminuye y las tasas aumentan. Se muestran específicamente estos casos ya que 12 caracteres serán el tamaño medio que los mensajes escogidos tendrán y 19 caracteres ya que es el caso peor. Recalcar también que en ambos casos unicore teníamos pérdidas al intentar transmitir a 30 ms en Full duplex y que estos errores son corregidos por la implementación de las tareas en núcleos separados además de que alcanzamos tasas de envío reales de 30 ms de extremo a extremo.

Para finalizar, realizamos las pruebas con una red de cuatro dispositivos. Todos los nodos estaban en una comunicación full dúplex a una tasa fija compartida. A continuación adjuntamos unas gráficas que ilustran el resultado de las pruebas.

- Retardo entre paquetes por nodo emisor.

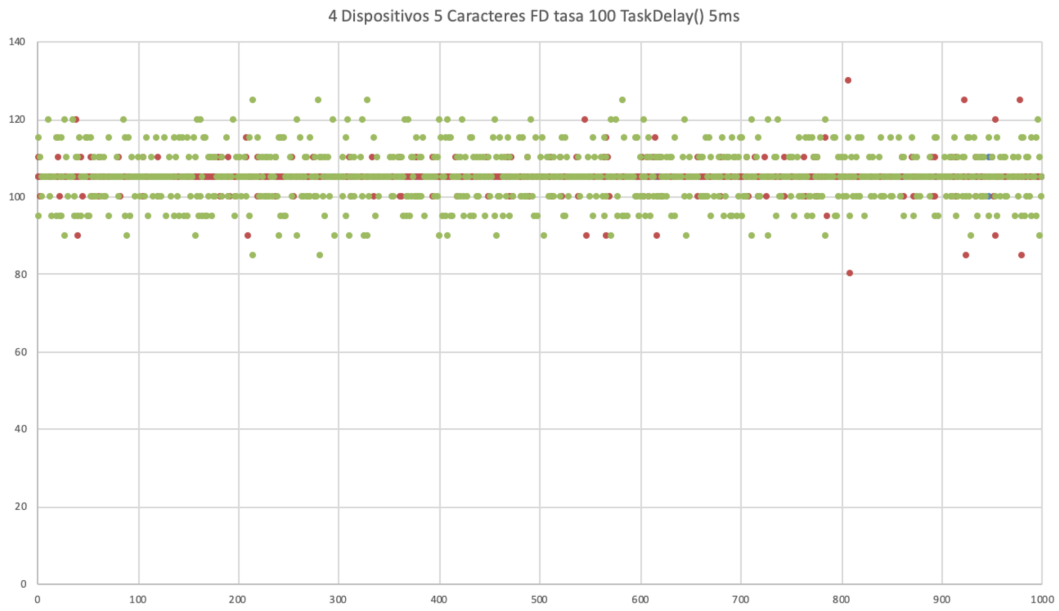


Figura 3.25: 4 dispositivos simultáneamente FD a tasa 100ms

- Retardo entre paquetes consecutivos en receptor.

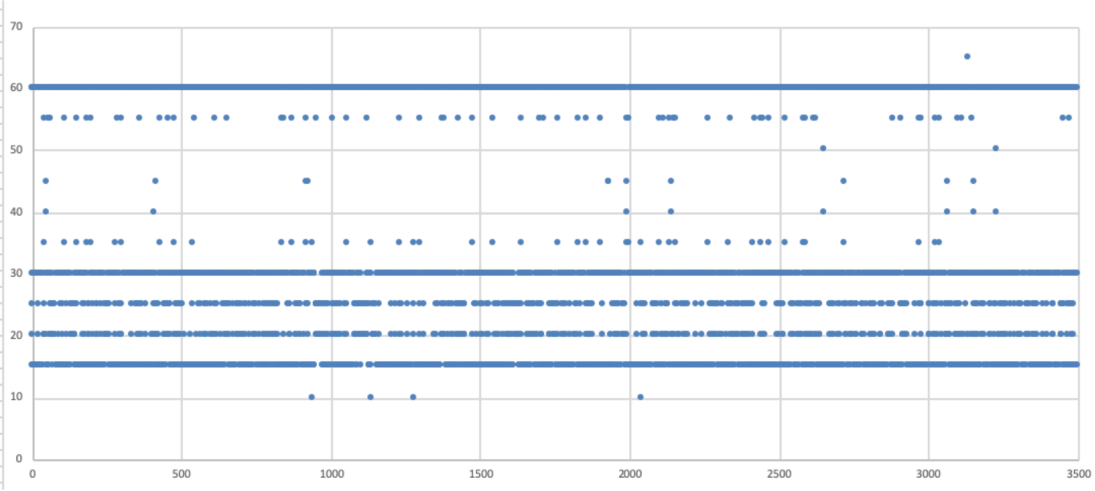


Figura 3.26: Retardo entre paquetes consecutivos 4 dipsositivos FD

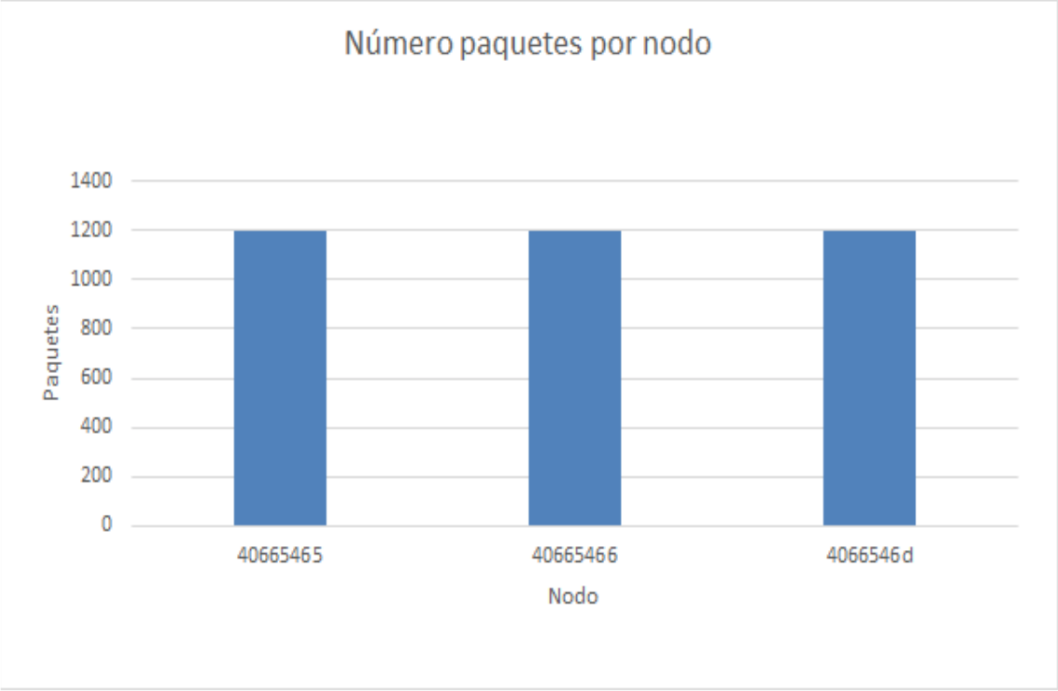


Figura 3.27: Número de paquetes por nodo

De la primera de estas tres gráficas obtenemos que los retardos máximos entre paquetes de un mismo nodo son de +-25 ms y solo se llega un 0,1 % a este nivel de retardo. La superposición de puntos es buena señal, ya que indica el comportamiento similar de los tres nodos. En la segunda podemos ver las llegadas de los paquetes y cómo estas se distribuyen de manera uniforme. La dispersión de los paquetes es muy baja y en la última gráfica podemos apreciar que no se descarta ningún paquete, lo cual indica que en ningún momento llegamos a saturar el buffer.

Esto confirmó que nuestros dispositivos ya estaban listos para ser probados en un entorno real, ya que los experimentos en ambiente controlado corroboraron su buen desempeño.

Entorno real (pruebas en vehículos)

En esta prueba se quiere comprobar el comportamiento que tiene nuestro sistema cuando los nodos están instalados en vehículos reales y estos están en movimiento.

Para la realización de esta prueba se necesitaba un espacio lo suficientemente grande como para que los vehículos pudieran circular sin problemas y a ser posible tenía que ser un sitio al aire libre para evitar tener rebotes de señales e interferencias. Por ello se eligió un aparcamiento del Campus de la universidad en el cual se distribuyeron 4 nodos de la siguiente manera, 1 estación fija (RSU) que estuvo activa durante toda la prueba y 3 estaciones móviles instaladas en vehículos que se iban a ir conectando una a una.

Configuración de los dispositivos:

- Envío/Recepción: 6 mensajes consecutivos a una tasa de 150 ms entre mensajes, y 2 mensajes consecutivos a una tasa de 300 ms. Esto se hizo para ver si se producían anomalías si de repente el sistema cambiaba su tasa de envío mientras los vehículos permanecían en movimiento. Comunicación Full-duplex.
- Mensajes enviados: 8 mensajes distintos escogidos del estándar.

Figura 3.28: Aparcamiento de Disney donde se realizaron las pruebas

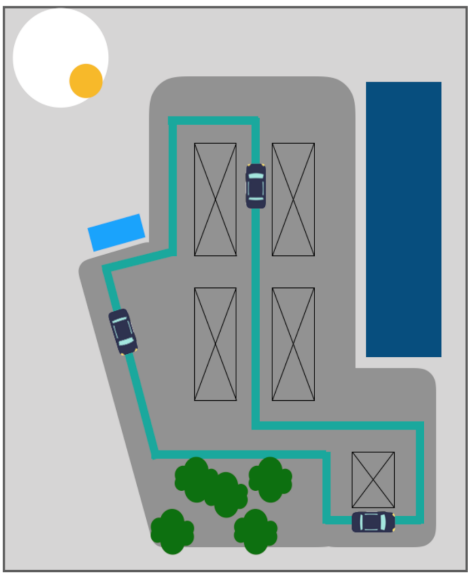


Figura 3.29: Plano explicativo circuito

Como se ha comentado el RSU se activó al inicio de la prueba y primero únicamente había un coche circulando por el circuito, después el segundo coche se incorporó y por último el tercero. De esta forma se deseaba comprobar el comportamiento de la red conforme se iban añadiendo nodos a la red.
Además en cada uno de los nodos se registraron los mensajes recibidos para después poder comprobar y analizar los resultados.

Toda la prueba se desarrolló como se esperaba excepto en el momento en el que por un problema técnico un nodo se quedó sin alimentación y por tanto se desconectó de forma inesperada de la red.
Al principio este hecho hizo que se replanteará el tener que repetir la prueba pero teniendo en cuenta que el nodo lo volvimos a conectar y seguimos registrando datos con el

resto de equipos se pudo comprobar cómo se comportaba la red ante este tipo de sucesos.

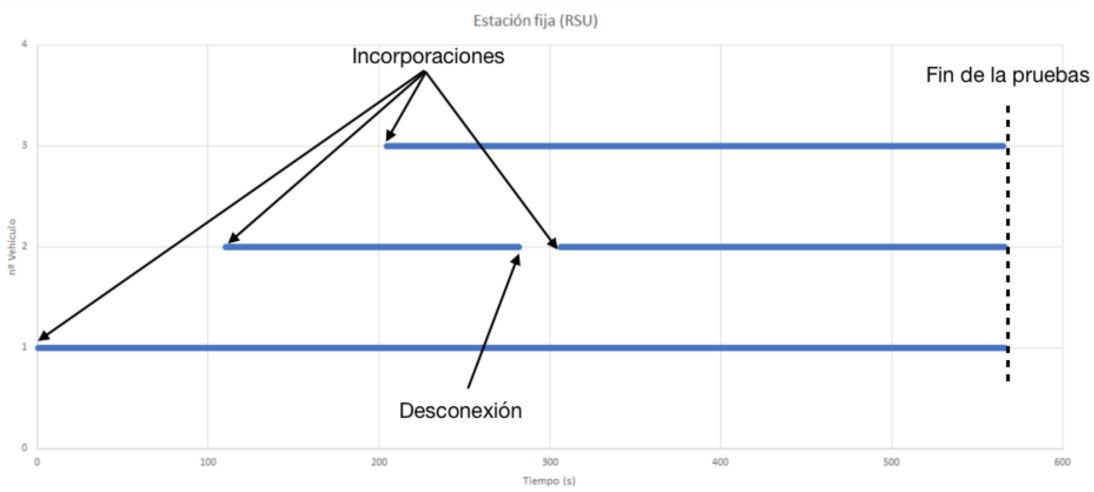


Figura 3.30: Recepción de mensajes en la RSU durante toda la prueba

En la gráfica se puede observar cómo al principio únicamente circula el vehículo 1 (id:4066546d), luego se incorpora el vehículo 2 (id:4066546c) y por último el vehículo 3 (id:4066546d). Además comprobamos que se produce una desconexión del vehículo 2 durante aproximadamente 30 segundos, tras los cuáles el dispositivo vuelve a conectarse a la red y empezar a emitir con normalidad.

Una vez analizado la respuesta de los dispositivos respecto al tiempo pasamos a comprobar otro parámetro muy importante como es la latencia entre mensajes recibidos.

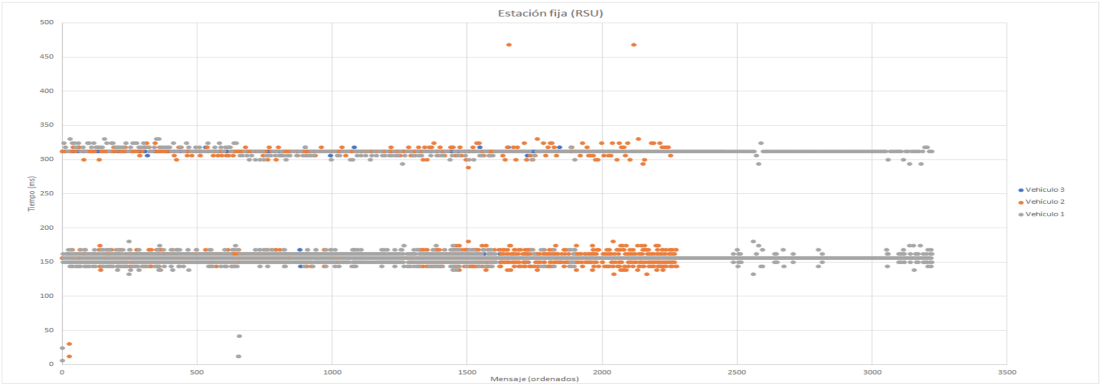


Figura 3.31: Retardo entre mensajes en RSU

Se observan claramente las dos tasas de transmisión posicionadas sobre los 150 y los 300 milisegundos como estaba programado. La dispersión aunque parezca que sea mucho mayor en la tasa de 150 ms en comparación con la de 300 ms, tenemos que tener en cuenta que estamos enviando un mayor número de mensajes consecutivos a tasa 150 ms (8 mensajes) que a 300 ms (2 mensajes), lo que provoca esa diferencia, pero si se enviara la misma cantidad de mensajes a ambas tasas las diferencias en dispersión entre tasas sería prácticamente nula. Además de que la dispersión no supone excepto en algún caso puntual, un aumento en la recepción de mensajes de más de un 20 % (25 ms) del tiempo estimado.

Por último pasamos a analizar la cantidad de mensajes perdidos durante la transmisión:

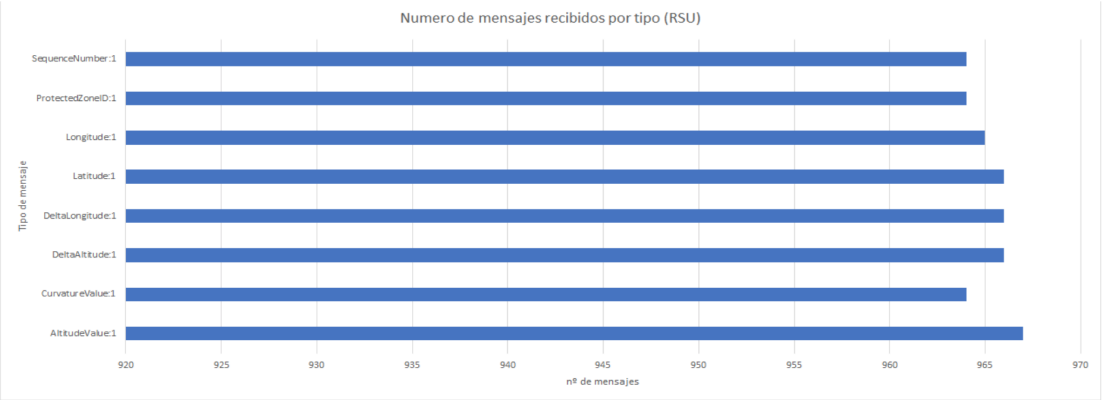


Figura 3.32: Cantidad de mensajes recibidos por tipo de mensaje en RSU

Mensaje	Cantidad
AltitudeValue:1	967
CurvatureValue:1	964
DeltaAltitude:1	966
DeltaLongitude:1	966
Latitude:1	966
Longitude:1	965
ProtectedeZoneID:1	964
SequenceNumber:1	964

Cuadro 3.1: Cantidad de mensajes por tipo.

Obtenemos que de los 8 tipos diferentes de mensajes enviados durante la transmisión, del que más recibimos son 967 y del que menos 964 por lo que hace que las pérdidas sean prácticamente nulas, ya que alguno de estos errores se produjeron al parar el finalizar la prueba

(desconexión de los nodos) o en la propia desconexión del nodo del vehículo 2.

Los datos de los otros nodos se encuentra en el repositorio, ya que no veo necesario incluirlos en este apartado ya que son información redundante para la comprensión de esta prueba.

Conclusión: una vez analizados todos los datos de la prueba en escenario real podemos determinar que los dispositivos se comportan de forma satisfactoria, ya que estas pruebas podrían haber pasado perfectamente por pruebas realizadas en el entorno controlado y por tanto podemos determinar que estos dispositivos no precisan de estar en zonas fijas sino que se pueden implementar en soluciones que necesiten de movilidad dentro del rango de la red.

Capítulo 4

Conclusiones

Una vez analizadas las pruebas realizadas al dispositivo nos damos cuenta de que dispone de una latencia baja y de una pérdida entre paquetes prácticamente nulas, que son características muy interesantes para la comunicación de corto alcance entre vehículos.

El problema viene conforme el número de nodos de la red comienza a incrementarse, lo que produce que comiencen a tener importancia la latencia y las pérdidas. El punto de inflexión a partir del cual la latencia y las pérdidas son insoportables se produce a partir de los 15/20 nodos en la red, lo que teniendo en cuenta el alcance del dispositivo hace que en muchas situaciones reales como atascos o circulación por ciudad se produzca dicho punto de inflexión superando los 20 nodos en la red.

Por lo que este dispositivo es ideal para comprobar los requerimientos que debe tener una red de este tipo (corto alcance), pero no es una solución lo suficientemente confiable como para comercializarlo como comunicación V2X.

Esto no quiere decir que el sistema se vaya a quedar como prueba o experimento piloto, una vez hemos visto el punto débil del sistema proponemos una serie de escenarios en los que sería viable su implementación y que incluso sería posible una comercialización del mismo.

En el caso de disponer de un entorno en el que tenemos controlado el número de nodos que existe en la red, la comunicación entre estos es posible consiguiendo resultados muy interesantes. Este tipo de entornos podrían ser entornos industriales como fábricas en las que se utilizan vehículos para realizar ciertas tareas y que cuando ocurre un accidente o problema en los mismos supone prácticamente la paralización de toda la actividad de la misma, lo que supone pérdidas tanto económicas como materiales además de un sobrecoste en el reinicio de algunas de ellas al tener que comenzar de nuevo el proceso (por ejemplo el encendido/enfriamiento paulatino de hornos).

Otra forma de no sobrepasar el número de nodos consiste en limitar el rango de la comunicación de largo alcance a distancias muy inferiores, cercanas a las del estándar 802.11p (estándar diseñado para V2X) comprendidas entre 40/50 metros en las que ya serían mucho menos probables las situaciones de saturación de la red por exceso de nodos en la misma. Se podría realizar utilizando antenas de mucha menos ganancia y realizando experimentos comprobando el comportamiento de la red variando el rango de alcance y cantidad de nodos en ella.

Además de todo esto hay que tener en cuenta que el coste físico del dispositivo es muy competitivo ya que se compone básicamente de un microcontrolador (ESP32) y el emisor de corto alcance (Xbee), lo que lo convierte en un dispositivo muy interesante en el caso de poder llegar a comercializarlo en un futuro. En la siguiente tabla podemos ver el precio estimado de un solo dispositivo (hay que tener en cuenta que si se comercializa a gran escala los precios serían mucho menores).

Cantidad	Producto	Precio	Total
1	XBP09-DMUIT-156	650$	650$
1	ESP32-DEVKITV1	159$	159$
1	PCB	250$	250$
1	Fuente alimentación	30$	30$
2	Mano de obra	100$	200$
Coste			1289$
1	Ganancia	100 %	1289$
Total			2578$
1	Impuestos	16 %	412.48$
Total con impuestos			2990.48$

Cuadro 4.1: Coste aproximado.

Capítulo 5

Dilema ético

El dilema ético que rodea la implantación de un sistema como el presentado a nivel global reside principalmente en la privacidad de los datos de los usuarios del sistema.

Para analizar en su profundidad el dilema ético debemos pensar en qué información estamos transmitiendo. Una parte fundamental de esta comunicación se basa en el hecho de conocer en todo momento la ubicación es los vehículos, algo que se entiende como algo privado y que afecta directamente al usuario, ya que normalmente estos solo comparten ubicación con personas de confianza. En este sistema, en todo momento el dispositivo está transmitiendo datos de ubicación, número de personas que viajan en el vehículo, el modelo de vehículo que se está utilizando, etc. y aunque solo fuese para usos de prevención de accidentes o mejora del desempeño de la comunicación autónoma, el usuario tendría que aceptar que estos datos fueran conocidos, alojados y analizados.

El alojamiento de estos datos implica que esa información privada del usuario está siendo conocida por alguien y con un análisis de esta información muchos datos como familia, vivienda, trabajo, etc. podrían ser conocidos.

Esto supone un claro riesgo por el daño potencial que provocaría cualquier actuación ilícita con estos datos y afecta a nuestro proyecto porque nuestro dispositivo es el encargado de brindarlos.

Capítulo 6

Líneas futuras

Este proyecto no se queda aquí, puesto que queda una parte muy importante que se continuará el siguiente semestre que consiste en el procesamiento de los datos, tanto de la comunicación de largo alcance como la de corto alcance.

La conexión de largo alcance actualmente se encuentra alojada en una URL direccionada a una hoja de cálculo de google (google sheet), en la que se guardan datos de la localización del vehículo (tanto latitud como longitud, además de la dirección postal), fecha en la que se recibe el mensaje y dirección del dispositivo que ha enviado el mensaje. Para un mejor almacenamiento y procesamiento de los datos de largo alcance se procederá a la instalación de una central de almacenamiento y procesamiento de datos en la nube, inicialmente se ha pensado en alojar un servidor en el laboratorio CIMB del edificio CEDETEC en el Campus Ciudad de México del TEC de Monterrey. Este servidor tiene que tener la capacidad de almacenar una gran cantidad de datos ya que va a recibir los datos del resto de dispositivos de la red, además de una gran potencia de procesamiento de los mismos y capacidad de comunicación rápida con los dispositivos deseados. Para conseguir una comunicación eficiente debemos conseguir una latencia lo más baja posible tanto en el envío y la recepción de mensajes, ya que es el punto más crítico en este sistema puesto que la potencia de cálculo únicamente depende de la tecnología utilizada pero la comunicación es más crítica ya que tiene que pasar por diferentes medios, eléctrico(internet) y aire(red celular) para llegar hasta su receptor y actualmente depende de la infraestructura aportada por las empresas de telecomunicaciones.

Puesto que el punto más crítico es el envío y recepción de mensajes, debemos pensar en alguna alternativa que nos permita bajar ese tiempo de latencia. Por ello es muy interesante el procesamiento de datos de corto alcance.

El procesamiento de estos datos es muy interesante ya que conseguimos bajar la latencia puesto que no es necesario enviar los datos a un servidor, que éste los procese y recibamos una

respuesta, ya que se envían directamente por la red de corto alcance a los RSU (Roadside Unit). Las cuales son unidades de recepción y envío de datos que se colocarán en sitios estratégicos de la vía. Estas unidades están compuestas por el mismo sistema de corto alcance que implementan el resto de vehículos, pero agregado a ellas llevan una unidad de procesamiento de datos que nos permite el procesar los datos recibidos y a partir de ellos determinar los diferentes avisos a los vehículos para evitar los posibles escenarios peligrosos.

Inicialmente se colocarán en las intersecciones ya que es el punto de las vías en el que más accidentes se producen como ya se ha expuesto en la introducción. A partir de los datos proporcionados por los vehículos (velocidad, aceleración, posición, etc.) se realizarán los cálculos necesarios para determinar el tipo de peligro (alcance o colisión) y enviar mensaje de aviso a los vehículos implicados por la red de corto alcance.

Nos centramos básicamente en el procesamiento de datos y envío de mensajes de aviso puesto que creemos que es una de las partes más interesantes a desarrollar actualmente ya que no existe ningún sistema similar implementados, además de que es la forma más eficiente de reducir los accidentes de vehículos.

Bibliografía

Wikipedia contributors. (2019d, 30 octubre). Real-time operating system. Recuperado 29 octubre, 2019, de https://en.wikipedia.org/wiki/FreeRTOS

Wikipedia contributors. (2019c, 9 octubre). Real-time operating system. Recuperado 1 noviembre, 2019, de https://en.wikipedia.org/wiki/Real-time_operating_system

Wikipedia contributors. (2019b, 21 octubre). Zigbee. Recuperado 29 octubre, 2019, de https://en.wikipedia.org/wiki/Zigbee

Wikipedia contributors. (2019a, 6 octubre). IEEE 802.11p. Recuperado 2 octubre, 2019, de https://en.wikipedia.org/wiki/IEEE_802.11p

Waspmote - Open Source Sensor Node for the Internet of Things — NB-IoT, Cat-M, ZigBee, Sigfox, LoRaWAN, 3G / 4G / GPRS — Libelium — Libelium. (s.f.). Recuperado 30 octubre, 2019, de http://www.libelium.com/products/waspmote/

Andrewrapp, A. user. (2016, 31 diciembre). xbee-arduino. Recuperado 1 noviembre, 2019, de https://github.com/andrewrapp/xbee-arduino

Digi. (2018). XBee-PRO 900HP/XSC RF Modules. agosto 20,219, de Digi Sitio web: https://www.digi.com/resources/documentation/digidocs/pdfs/90002173.pdf

arduinofanboy. (2018). ESP32 BLE + Android + Arduino IDE = AWESOME. agosto 20,2019, de instructables Sitio web: https://www.instructables.com/id/ESP32-BLE-Android-App-Arduino-IDE-AWESOME/

randomnerstutorials. (2018). esp32 dual core arduino ide. agosto 20, 2019, de randomnerstutorials Sitio web: https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/

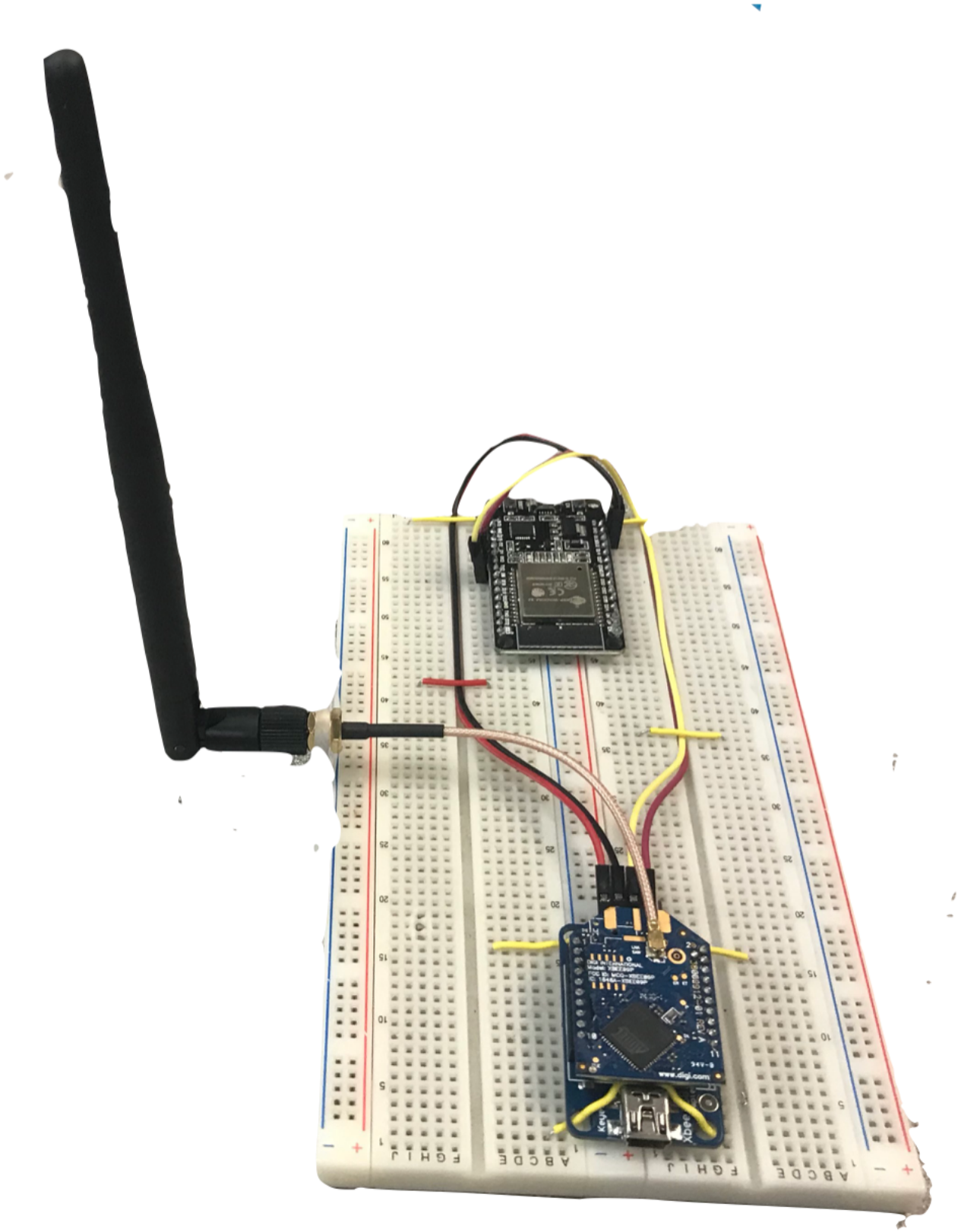
ETSI TS. (2014). ETSI TS 102 894-2. Francia: ETSI TS

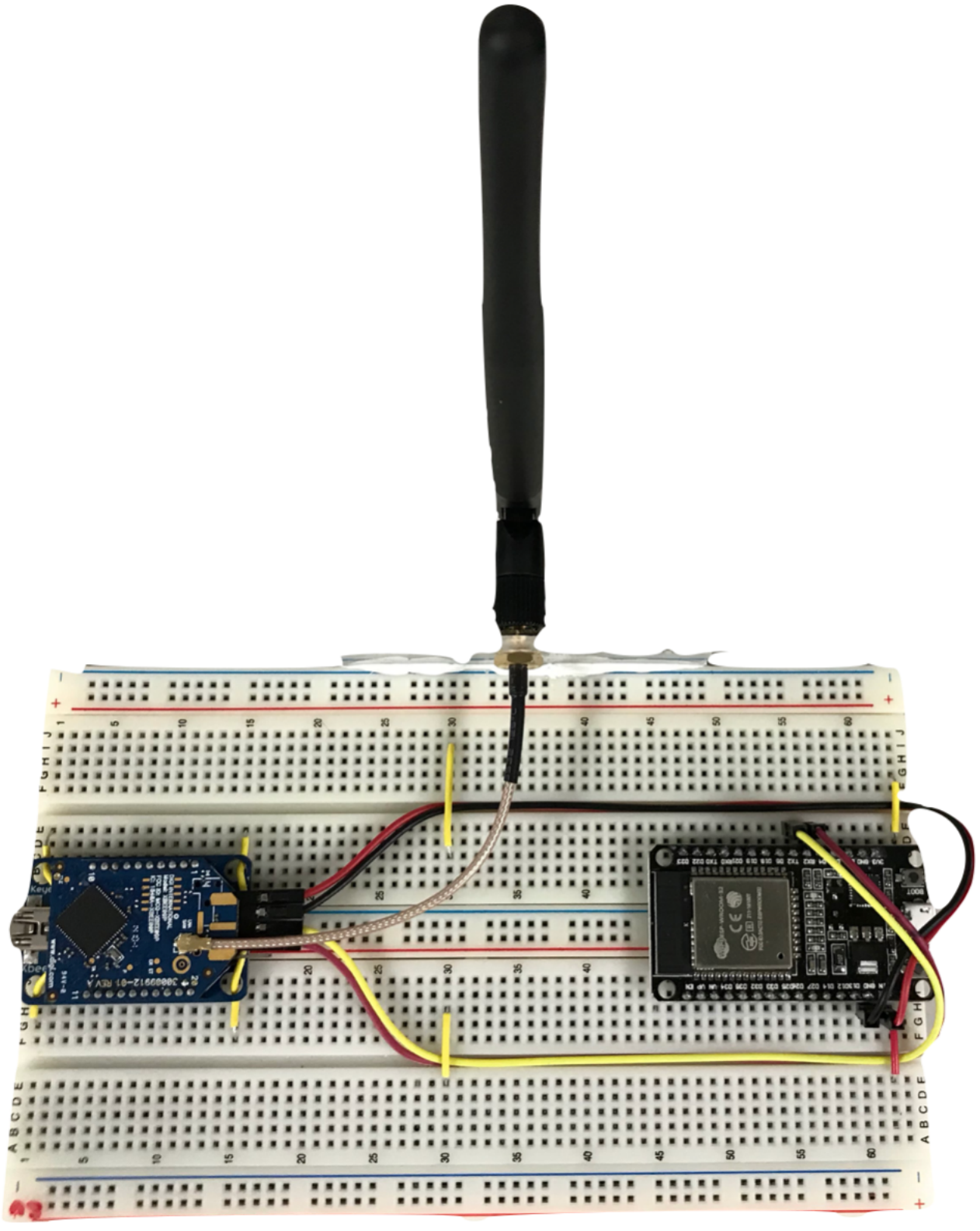
Digi. (2018). Set up a DigiMesh network. agosto 20, 2019, de Digi Sitio web: https://www.digi.com/resources/documentation/Digidocs/90001496/concepts/c_overview_network_setup.htm?TocPath=DigiMesh %20network %20setup %7C_____1

Capítulo 7

Anexos

7.1. AI: Prototipo





7.2. AII: Estándar ZigBee

1
2
3
4
5
6
7
8
9
10

ZigBee Specification

11
12
13
14
15
16

ZigBee Document 05-3474-21	
August 5, 2015	
Sponsored by: ZigBee Alliance	
Accepted by	ZigBee Alliance Board of Directors
Abstract	The ZigBee Specification describes the infrastructure and services available to applications operating on the ZigBee platform.
Keywords	ZigBee, Stack, Network, Application, Profile, Framework, Device Description, Binding, Security

17

18 August 5, 2015

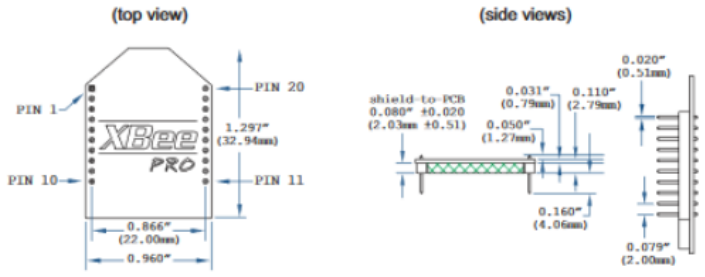
7.3. AIII: ETSI TS 102 894-2 V1.2.1

ETSI TS 102 894-2 V1.2.1 (2014-09)

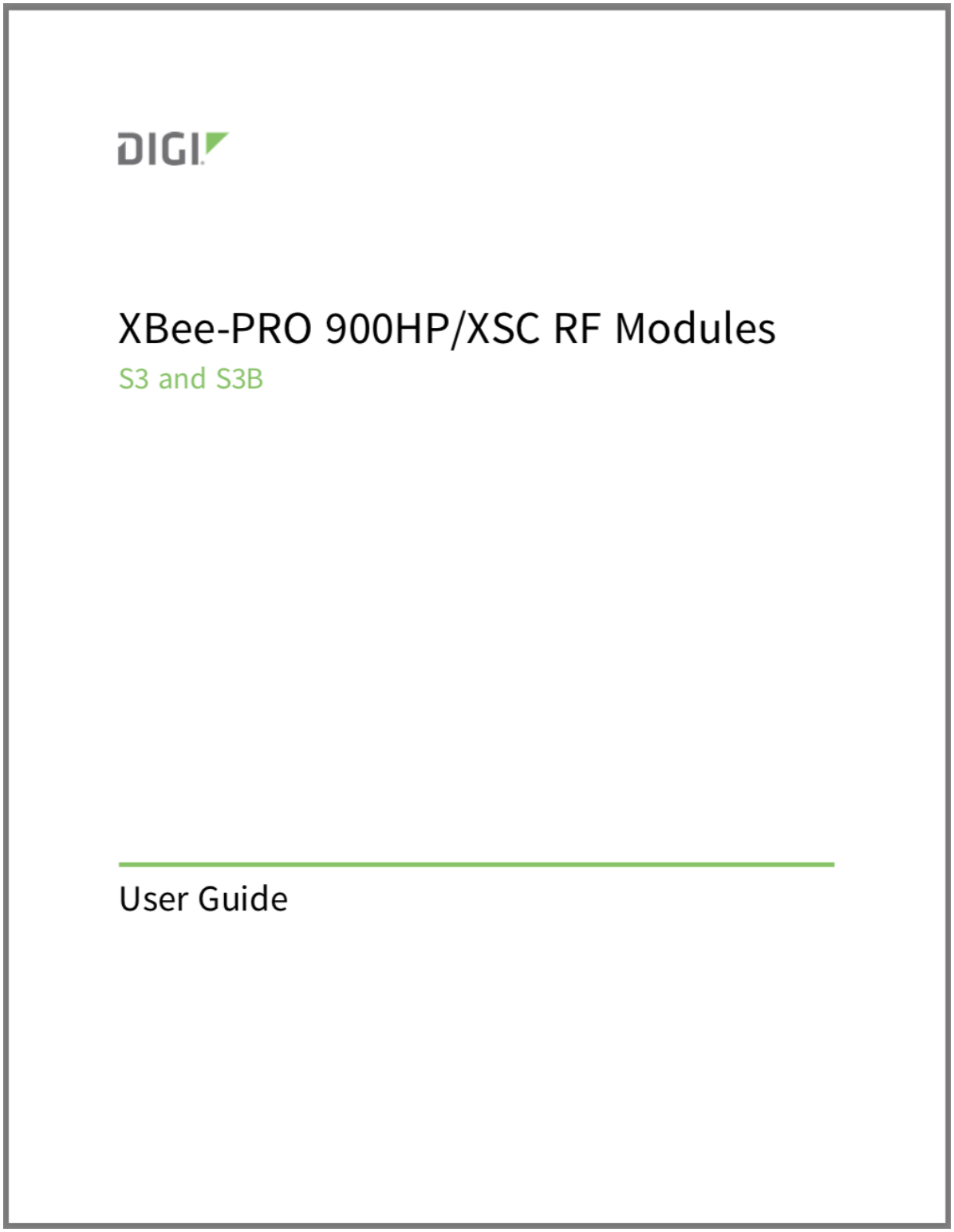
Intelligent Transport Systems (ITS);
Users and applications requirements;
Part 2: Applications and facilities layer
common data dictionary

7.4. AIV: Datasheet XBee

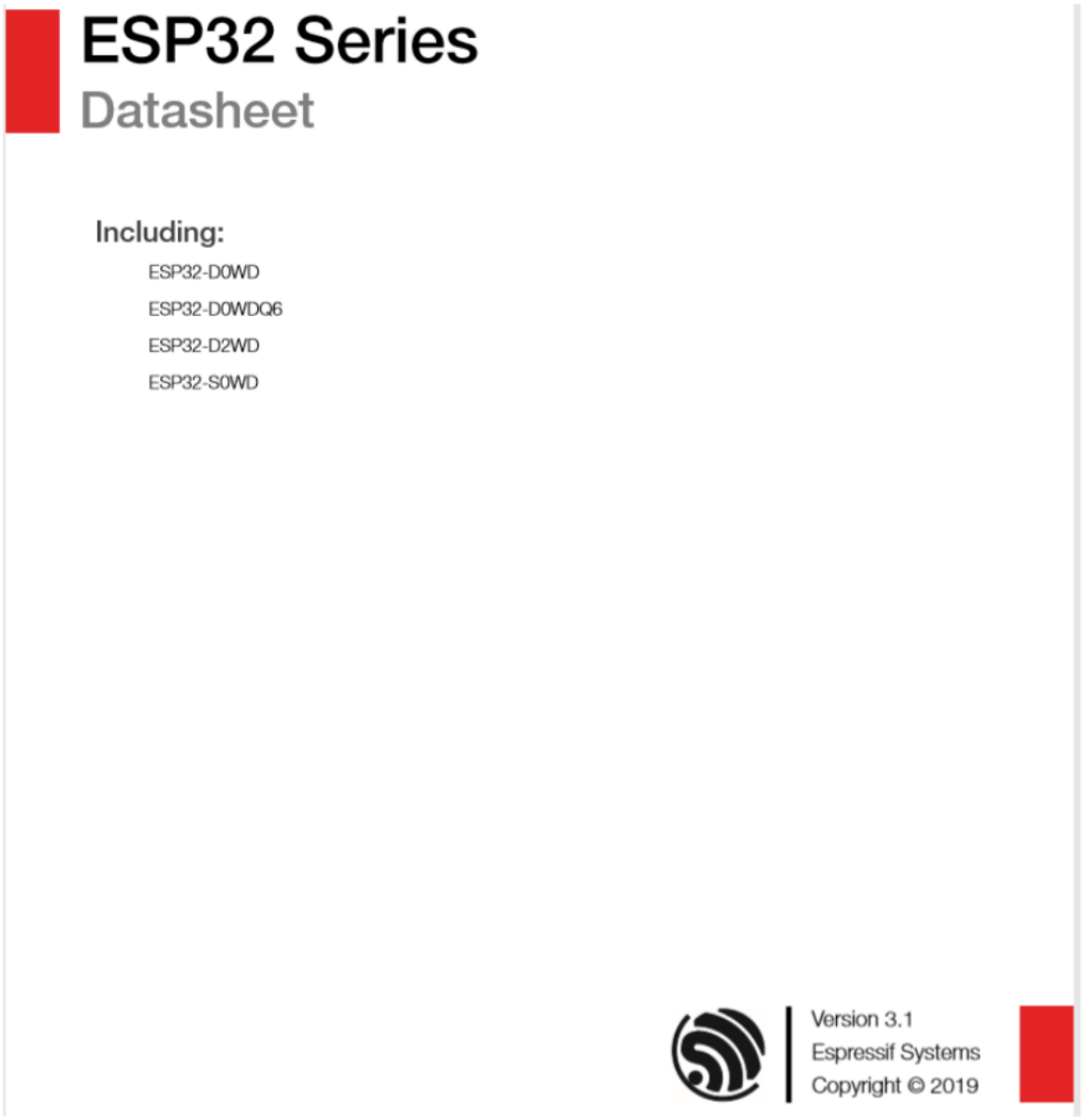
SPECIFICATIONS	Digi XBee-PRO® 900HP	Programmable Digi XBee-PRO® 900HP
HARDWARE		
PROCESSOR	ADF7023 transceiver, Cortex-M3 EFM32G230 @ 28 MHz; Programmable includes: Freescale MC9S08QE32	
FREQUENCY BAND	902 to 928 MHz, software selectable channel mask for interference immunity	
ANTENNA OPTIONS	Wire, U.FL and RPSMSA	
PERFORMANCE		
RF DATA RATE	10 Kbps or 200 Kbps	
INDOOR/URBAN RANGE*	10 Kbps: up to 2000 ft (610 m); 200 Kbps: up to 1000 ft (305 m)	
OUTDOOR/ LINE-OF-SIGHT RANGE*	10 Kbps: up to 9 miles (15.5 km); 200 Kbps: up to 4 miles (6.5 km) (with 2.1dB dipole antennas)	
TRANSMIT POWER	Up to 24 dBm (250 mW) software selectable	
RECEIVER SENSITIVITY	-101 dBm @ 200 Kbps, -110 dBm @ 10 Kbps	
FEATURES		
DATA INTERFACE	UART (3V), SPI	
GPIO	Up to 15 Digital I/O, 4 10-bit ADC inputs, 2 PWM outputs	
NETWORKING TOPOLOGIES	DigiMesh, Repeater, Point-to-Point, Point-to-Multipoint, Peer-to-Peer	
SPREAD SPECTRUM	FHSS (Software Selectable Channels)	
PROGRAMMABILITY		
MEMORY	N/A	32 KB Flash / 2 KB RAM
CPU/CLOCK SPEED	N/A	HCS08 / Up to 50.33 MHz
POWER		
SUPPLY VOLTAGE	2.1 to 3.6 VDC	2.4 to 3.6 VDC
TRANSMIT CURRENT	215 mA	229 mA
RECEIVE CURRENT	29 mA	44 mA
SLEEP CURRENT	2.5 uA	3 uA
REGULATORY APPROVALS		
FCC (USA)	MCQ-XB900HP	
IC (CANADA)	1846A-XB900HP	
C-TICK (AUSTRALIA)	Yes	
ANATEL (BRAZIL)	Yes	
IDA (SINGAPORE)	Yes	



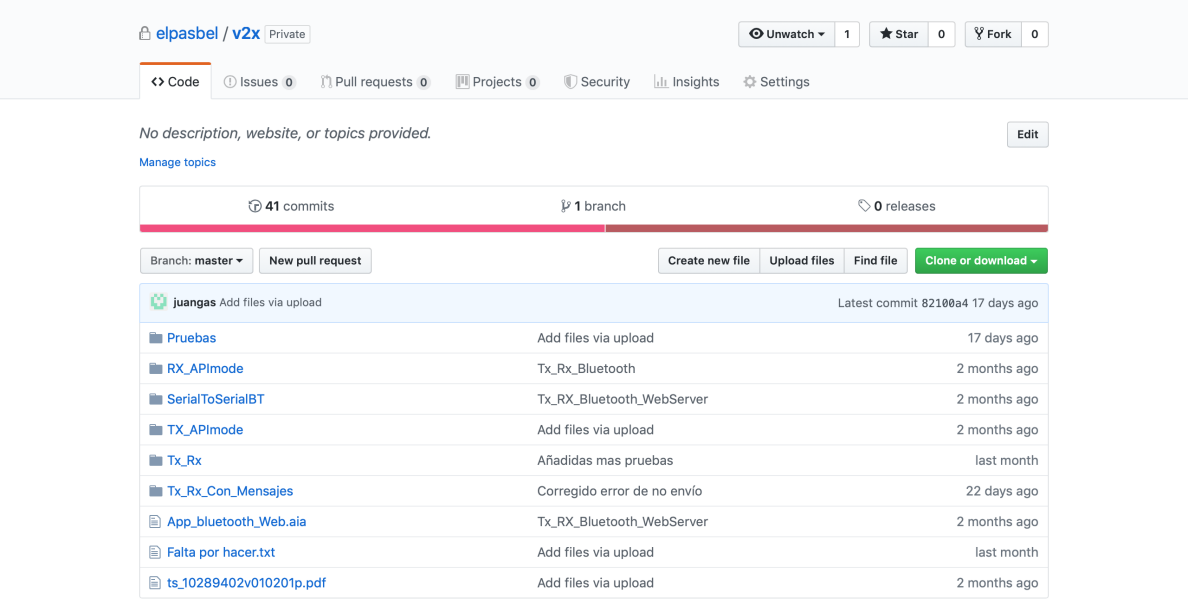
7.5. AV: Especificaciones XBee



7.6. AVI: Datasheet ESP-32 DEVKIT V1



7.7. AVII: Código y pruebas realizadas



Todos los códigos nombrados a lo largo de la memoria están disponibles en el siguiente repositorio de github: https://github.com/elpasbel/v2x

Aparte, también encontramos el catálogo de los mensajes así como los programas utilizados para la realización de las pruebas.

En este repositorio están alojadas también las hojas de cálculo originales con los datos reales, desde las cuales se obtuvieron las gráficas mostradas a lo largo de la memoria. Estos datos se corresponden a las pruebas descritas a lo largo de la caracterización del dispositivo.