Disseny i avaluació d'un canal de comunicacions en un bosc tropical

Treball Fi de Grau
realitzada a l'
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
per

Oriol Soldevila Gargallo

En compliment parcial
dels requisits per al Grau en
Enginyeria de Tecnologíes i Serveis de Telecomunicació

Director: Jordi Casademont
Barcelona, Date 06/2022

Índex

Índex de figures

Índex de taules

Abstract

Xprize Rainforest is a competition looking to enhance our understanding of the rainforest ecosystem. On behalf of the communications team, we seek to establish a reliable communication between a sensor placed in the forest and a base camp. To accomplish this, we evaluate two technologies: LoRa and IEEE 802.11ah. There is a theoretical evaluation of both technologies, followed by a set of tests regarding throughput and range. Due to the low throughput and other problems related to LoRa, 802.11ah is chosen as the most appropriate technology to establish the communication. We then follow with the creation of a relay prototype which is placed on a drone so that we can accomplish a more direct view with the drone and avoid losing coverage due to the forest. The relay tests are inconclusive because of interferences between channels.

Resum

Xprize Rainforest és una competició que pretén millorar el nostre coneixement sobre l'ecosistema del bosc tropical. Com a equip de comunicacions es pretén establir una comunicació fiable entre un sensor situat al bosc i un camp base. Per a fer-ho, s'avaluen dues tecnologies: LoRa i IEEE 802.11ah. Es fa una valoració teòrica d'ambdós seguida d'una sèrie de proves tant de *throughput* com de rang. Degut al baix *throughput* i a altres problemes de LoRa, s'arriba a la conclusió de que l'estàndard 802.11ah és el més adient per a establir la comunicació. Així, es proposa la creació d'un prototip de repetidor que es col·loca sobre un drone de manera que es pugui crear una visió més directa amb el sensor i així evitar que es perdi cobertura degut a la gran massa d'arbres. Les proves del prototip no són concloents degut a interferències entre canals.

Resumen

Xprize Rainforest es una competición que pretende mejorar nuestro conocimiento sobre el ecosistema del bosque tropical. Como equipo de comunicaciones se pretende establecer una comunicación fiable entre un sensor situado en el bosque y un campo base. Para ello, se evalúan dos tecnologías: LoRa y IEEE 802.11ah. Se hace una valoración teórica de ambos seguida de una serie de pruebas tanto de *throughput* como de rango. Debido al *throughput* bajo y otros problemas de LoRa, se llega a la conclusión de que el estándar 802.11ah es el más adecuado para establecer la comunicación. Así, se propone la creación de un prototipo de repetidor que se coloca sobre un drone de forma que se pueda tener una visión más directa con el sensor y evitar que se pierda cobertura debido a la gran masa de árboles. Las pruebas del prototipo no son concluyentes debido a interferencias entre canales.

Historial de revisions i registre d'aprovació

Revisió	**Data**	**Propòsit**
0	14/05/2022	Document creation
1	07/06/2022	Document revision
2	14/06/2022	Document revision
3	17/06/2022	Document revision

Nom	**e-mail**
Oriol Soldevila Gargallo	oriol.soldevila@estudiantat.upc.edu
Jordi Casademont	jordi.casademont@upc.edu
Josep Paradells	josep.paradells@entel.upc.edu
Josep Vidal	josep.vidal@upc.edu

Escrit per:		Revisat i aprovat per:	
Data	13/06/2021	Data	17/06/2022
Nom	Oriol Soldevila	Nom	Jordi Casademont
Posició	Autor	Posició	Supervisor

1 Introducció

Xprize Rainforest és una competició de 5 anys de duració que pretén millorar el nostre coneixement sobre l'ecosistema del bosc tropical. Els equips que competeixen han de dissenyar una tecnologia capaç d'identificar i catalogar la flora i la fauna de la zona. Es disposa d'un temps limitat per a explorar una zona de selva tropical, avaluar la biodiversitat i elaborar una visió que integri múltiples fonts de dades per tal de documentar l'estat de salut de la selva. La solució proposada consisteix en el desplegament d'un sensor en alguna zona del bosc tropical a certa distància d'un camp base. Com a part de l'equip de comunicacions, es pretén avaluar tecnologies capaces d'obtenir una comunicació fiable amb el sensor, en aquest cas, s'escullen LoRa i IEEE 802.11ah. Així, aquest projecte parteix de zero.

En quant a les especificacions del projecte, cal comentar que les condicions finals en les que es durà a terme la competició encara són desconegudes en la seva majoria. Encara es desconeix la distància entre el sensor i el camp base, si serà una zona de molta vegetació, o si hi haurà clarianes, etcètera. Així, els requeriments del projecte no són clars, ja que desconeixem la distància a la que haurem de transmetre, o el *throughput* que intentem assolir, entre d'altres. Per això s'avaluen tencologies dissenyades per a aplicacions diferents: LoRa està pensat per a establir comunicacions de llarg abast amb un *throughput* baix, mentre que l'estàndard 802.11ah pot assolir *throughputs* més alts a un rang inferior. Així, tenim un compromís entre *throughput* i distància.

A partir d'això podem concloure que l'objectiu principal del projecte és avaluar ambdós tecnologies per tal de poder determinar quins són els millors paràmetres a utilitzar. D'aquesta manera, quan les condicions de la competició siguin més clares i es determinin els requeriments a assolir, podrem adaptar-nos depenent dels resultats obtinguts per tal de proporcionar la solució més òptima.

1.1 Pla de treball

A mesura que es desenvolupava el projecte i es coneixia més sobre ambdós tecnologies i els mòduls utilitzats es van produir desviacions sobre el pla de treball inicial. D'aquesta manera, es van poder realitzar noves proves així com determinar les limitacions de cadascuna de les tecnologies. Això va facilitar el desenvolupament de noves solucions per a optimitzar el rang i el *throughput* en diferents situacions. Així, els *Work Packages* han quedat de la següent manera:

WP1: Aprendre sobre LoRa i 802.11ah	
Descripció	Búsqueda d'informació sobre les tecnologies utilitzades
Tasques	- Buscar informació teòrica de LoRa: banda freqüencial, altres treballs, throughput màxim, rang, limitacions, etcètera. - Buscar informació sobre IEEE 802.11ah i les seves diferències amb altres estàndards IEEE 802.11ah. - Aprendre sobre desenvolupament de software a Arduino i RaspberryPi.

Taula 1: *Work Package 1*

WP2: Arduino i LoRa	
Descripció	Aprendre com utilitzar la llibreria d'Arduino i desenvolupar els primers scripts
Tasques	- Aprendre com utilitzar les funcions de la llibreria i com modificar els seus paràmetres. - Crear els primers scripts client/servidor per a enviar i rebre informació entre els mòduls LoRa. - Simular una comunicació de tres nodes (sensor, camp base i ordinador). - Creació d'un algoritme de selecció automàtica d'Spreading Factor.

Taula 2: *Work Package 2*

WP3: Proves de LoRa	
Descripció	Fer les proves de LoRa per tal d'obtenir resultats que incloguin *throughput*, abast i fiabilitat, entre d'altres.
Tasques	- Aprendre més sobre Spreading Factor, limitacions de potència i freqüència, limitacions per país, etcétera. - Provar diferents paràmetres per veure quins funcionen millor en cada escenari.

Taula 3: *Work Package 3*

WP4: 802.11ah	
Descripció	Aprendre sobre IEEE 802.11ah i sobre els codis de GitHub del fabricant
Tasques	- Aprendre més sobre Spreading Factor, limitacions de potència i freqüència, limitacions per país, etcétera. - Provar diferents paràmetres per veure quins funcionen millor en cada escenari.

Taula 4: *Work Package 4*

WP5: Proves del 802.11ah	
Descripció	Provar IEEE 802.11ah en un escenari real
Tasques	- Desenvolupar varis scripts client/servidor amb diferents paràmetres per tal de determinar la configuració que funciona millor. - Provar una comunicació a 3 salts.

Taula 5: *Work Package 5*

WP6: Escollint la millor solució final	
Descripció	Determinar quina tecnologia s'adapta millor al nostre projecte o bé proporcionar una solució híbrida.
Tasques	- Avaluar els resultats i les restriccions del projecte de *throughput* i rang, entre d'altres. Així es pot determinar quina solució s'adapta millor al projecte. - Treballar en una possible solució híbrida fent ús de LoRa i 802.11ah alhora.

Taula 6: *Work Package 6*

1.2 Diagrama de Gantt

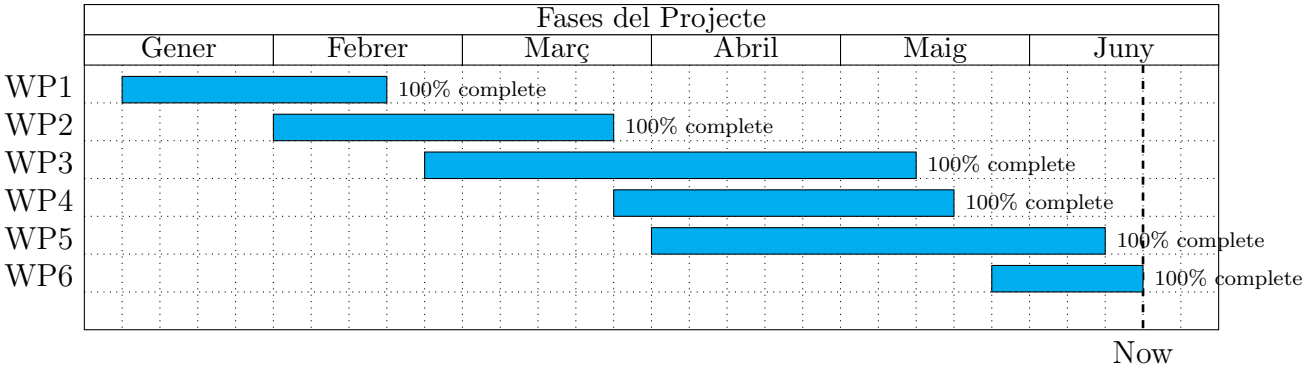


Figura 1: Diagrama de Gantt del projecte

2 State of the art

En aquesta secció s'introdueixen les tecnologies que han participat en el treball i s'explica el seu estat actual. El nostre projecte es pot considerar d'IoT (*Internet of Things*) i, com a tal, necessitem fer ús de les tecnologies adients que millor s'adaptin a aquest. Per això mateix es va decidir avaluar LoRa i 802.11ah, dos tecnologies emergents en el món de l'IoT.

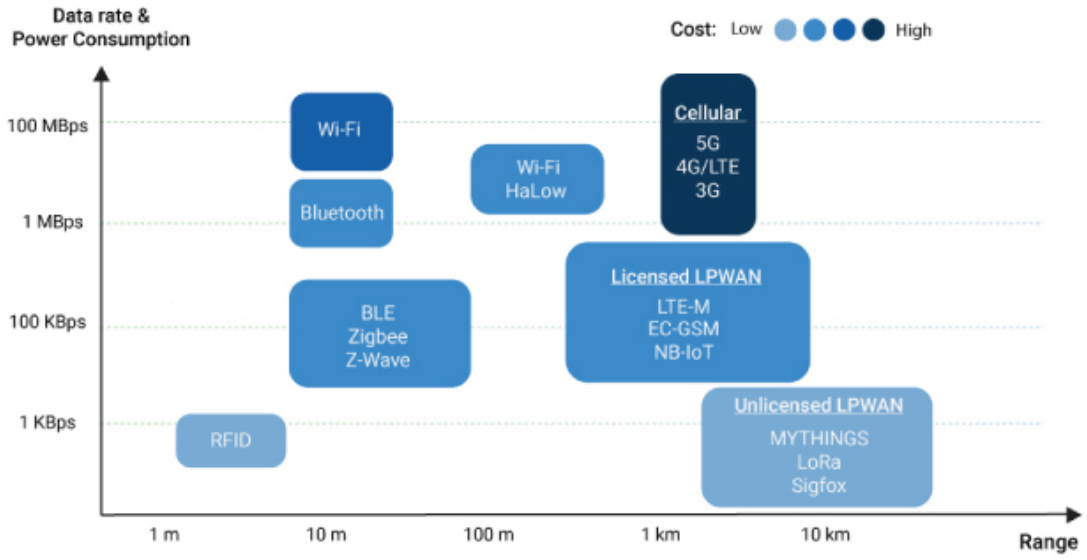


Figura 2: Comparació Tecnologies IoT.

Tal i com veurem més endavant, la figura 2, la qual hem obtingut de [1], no és del tot precisa. Tot i això, ens permet fer-nos una idea general de les tecnologies existents en el món del IoT així com algunes de les seves característiques. A més, ens permet observar que cadascuna té els seus punts forts i febles així com decidir quina seria la millor per aplicar al nostre projecte.

També cal comentar que la majoria de tecnologies d'IoT ofereixen un baix consum, així que la decisió sobre quina tecnologia utilitzar sol radicar en tres paràmetres: throughput, rang i cost.

Cal mencionar també la importància dels drones en aquest projecte, així com la necessitat d'establir una comunicació robusta amb aquests.

2.1 LoRa

LoRa és un tipus de modulació dissenyat per a LPWANs ("Low Power Area Networks"). El seu nom ve del terme "Long Range" degut a la seva capacitat per a transmetre a distàncies molt grans.

Tecnologia	Xarxa Inalàmbrica	Rang	Potència Tx
Bluetooth	WPAN	10m	2.5mW
Wifi	WLAN	50m	80mW
3G/4G	WWAN	5km	5000mW
LoRa	LPWAN	2-5 km (urbà) 5-15 km (rural) 15 km (LOS[1])	20mW

Taula 7: Comparació de tecnologies inalàmbriques més conegudes amb LoRa

La taula 7, obtinguda de [2], ens ofereix una comparativa de LoRa amb altres tecnologies conegudes. Igual que la majoria de dispositius dissenyats per a aplicacions IoT, consumeix extremadament poc i té un cost molt baix. La part negativa d'aquesta tecnología és que ofereix un *throughput* baix d'entre 0,3kbps i 5,5kbps (amb una amplada de banda de 125kHz).

Tot i que això és irrellevant per al nostre projecte, cal mencionar LoRaWAN; mentre que LoRa és la capa física, LoRaWAN és el protocol de nivell 2 que permet crear xarxes de dispositius LoRa fent ús de *gateways* LoRa. Es pot trobar més informació a [3].

LoRa està basat en CSS (Chirp Spread Spectrum), una modulació que utilitza un mètode de dispersió en freqüència.

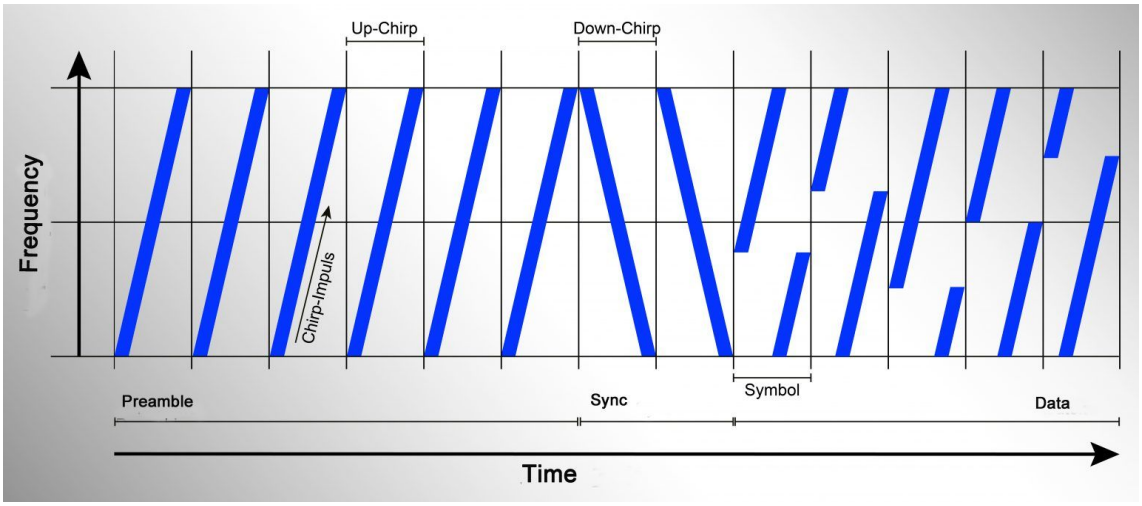


Figura 3: Pulsos *chirp* de LoRa

Si observem el paquet LoRa de la figura 3, obtinguda de [4], veuríem primer una sèrie de *Up-chirps* com a preàmbul, seguit de dos *Down-chirps* per sincronització, seguit finalment dels símbols amb les dades. La freqüència a la que comença cada símbol determina quin símbol és.

La quantitat de bits que es poden codificar per símbol ve donada pel SF (Spreading Factor). Existeixen 6 SF que van de SF7 a SF12. El SF és el que controla la velocitat

del pulsos *Chirp* i, per tant, determina la velocitat de transmissió. Tal i com veurem més endavant, tant el SF com l'amplada de banda ens ofereixen un compromís entre sensibilitat i *throughput*, de manera que augmentar el SF o l'amplada de banda significa reduir a la meitat el *throughput*. Així, aquests dos són els paràmetres en els que ens centrarem en avaluar en les nostres proves de camp.

La capacitat de LoRa per transmetre a grans distàncies així com el seu baix consum, fan d'aquesta tecnologia una eina molt adient per al nostre projecte a primera vista. Tot i això, cal avaluar també algunes desavantatges d'aquesta. Per exemple, LoRa opera a la banda no llicenciada ISM, que és accessible mundialment. Això fa que qualsevol persona pugui utilitzar aquestes freqüències, la qual cosa pot provocar interferències. Així, existeixen una sèrie de normes per a transmetre a aquesta banda freqüencial. A Europa (863 MHz - 870 MHz), per exemple:

- Es limita la potència de transmissió a 14dBm per al *Downlink* i a 27 *dBm* per a l'*Uplink* (només al canal de 869.525 MHz).
- Es limita el cicle de treball entre el 0,1% i l'1% depenent del canal. És a dir, només es permet transmetre durant 36 segons en una hora.
- Guany d'antena limitat a +2,15dBi.

Podem veure que LoRa també té punts febles si es consideren aquestes restriccions. Així, en cas que aquest projecte es dugui a terme en una altra zona, cal conèixer la regulació d'aquest per tal de determinar si és viable utilitzar LoRa.

2.2 802.11ah

L'estàndard IEEE 802.11ah sorgeix de la necessitat de crear una tecnologia similar a altres estàndards IEEE 802.11 compatible amb el món emergent de l'IoT. La intenció és poder oferir grans àrees de cobertura fent ús de moltes estacions associades a un punt d'accés.

Opera a freqüències baixes (sub 1 GHz) i està dissenyat per a oferir un millor rang que les tecnologies WiFi tradicionals (de 2.4 GHz i 5 GHz). A més, està dissenyat per a adreçar algunes limitacions d'altres estàndards 802.11 incorporant millores en la capa física i d'enllaç. Tot això fa que sigui una eina perfecte per a ser utilitzat en el món de l'IoT. Els autors de [5] proporcionen un bon anàlisi de la tecnologia, explicant la necessitat d'aquesta i fent una caracterització del seu rang i *throughput*.

Està pensat per a ser utilitzat amb canals d'1 MHz per tal d'oferir el màxim rang a *throughputs* reduïts. Tot i això, permet la unió de canals fins a amplades de banda de 16 MHz.

Pot oferir un rang de més de 1km a més de 5000 estacions amb un sol punt d'accés. Per a fer-ho, utilitza diferents MCS (*Modulation Coding Scheme*) que permeten modificar l'esquema de modulació utilitzat per tal d'oferir un compromís entre la robustesa i el *throughput*. A més, s'introdueix l'MCS10, que és essencialment un MCS0 però amb repetició mitjançant permutació de subportadores.

Proporciona millores a la capa MAC. Com que s'esperen trames amb poques dades s'utilitza un format de trama compacte per tal de millorar el rendiment. S'introdueix un Identificador d'Associació Jeràrquic (AID), un identificador de 13 bits que el punt d'accés proporciona a cada estació i que permet agrupar les estacions en nivells jeràrquics. Finalment, s'incorpora un mode d'estalvi d'energia que permet al punt d'accés i a una estació intercanviar trames durant una Oportunitat de Transmissió (TXOP) per tal de mantenir despertes les estacions durant breus períodes de temps, reduint així el consum d'energia.

En conclusió, IEEE 802.11ah ofereix una cobertura molt més amplia que altres estàndars IEEE 802.11 gràcies a la banda freqüencial utilitzada, així com a l'amplada de banda més estreta i la codificació més robusta. A més, el fet d'utilitzar una banda freqüencial per sota de 1 GHz també li proporciona robustesa contra l'atenuació per obstacles.

2.3 Drones

Actualment, els drones s'utilitzen per a aplicacions molt variades, des de l'ajuda al rescat de persones en llocs de difícil accés fins a la automatització de feines relacionades amb la agricultura, així com altres aplicacions industrials.

El que ens interessa realment en aquest projecte, però, és la millora de la comunicació amb els drones, així com la utilització d'un drone com a repetidor, per a poder cobrir distàncies més grans i tenir visió directa amb el sensor, de manera que puguem oferir *throughputs* més alts. En aquest sentit, ens podem referir a articles com [6] que expliquen com es podria fer ús de drones com a repetidors per a augmentar el *throughput* de les xarxes de 5G. Existeixen altres articles com [7], on s'explica com millorar l'efecte del multicamí en drones.

Com es pot veure, els drones són una tecnologia emergent amb gran quantitat d'aplicacions en molts sectors i encara tenen molt recorregut per tal d'aconseguir una comunicació que ofereixi robustesa i fiabilitat sense sacrificar el *throughput*.

3 LoRa

Les plaques que utilitzem estan basades en un ESP32 i compten amb un transceptor LoRa i una petita pantalla que controlem utilitzant la llibreria proporcionada pel fabricant Heltec.

Utilitzem l'IDE (*Integrated Development Environment*) proporcionat per Arduino per a desenvolupar, compilar i carregar el codi a les plaques.

El primer pas és importar la llibreria de Heltec, que ve amb codis d'exemple que utilitzem per a fer els primers tests de comunicació. Per això ens dirigim a **Preferencias/Additional Board Manager URL** i afegim:
"https://dl.espressif.com/dl/package_esp32_index.json". Després, ens dirigim a **Herramientas/Gestor de targetas** i seleccionem *esp32 by Espressif Systems*.

Després, utilitzant els codis d'exemple proporcionats per Heltec que trobem a **Archivo/Ejemplos/Heltec ESP 32 Dev-Boards/LoRa/** per a verificar que els mòduls funcionen correctament.

Utilitzem una API per a controlar LoRa [8] que ens proporciona funcions senzilles d'utilitzar per a enviar i rebre paquets, així com per a canviar diversos paràmetres. Els més rellevant són:

- Potència de transmissió: entre 2 dBm i 20 dBm.
- Spreading Factor: entre SF7 i SF12.
- Ample de banda (KHz): 62,5, 125, 250 i 500.

Cal mencionar que el mòdul de LoRa del que disposem no és capaç d'utilitzar amplades de banda inferiors a 62,5 kHz degut a que no és suficientment selectiu com per transmetre amb amplades de banda tant reduïts. Tot i això, el protocol LoRa en sí, admet amplades de banda inferiors.

3.1 Anàlisi inicial

Analitzem una sèrie de paquets de LoRa amb diferents *Spreading Factor* per veure les diferències entre aquests. Per a fer-ho utilitzem la eina CubicSDR per a processar i visualitzar els paquets. L'script utilitzat es troba a l'Annex A; la seva funció és rebre paràmetres de LoRa (SF, amplada de banda) per SPI i enviar paquets cada un segon. A més, calcularem el temps que es triga en enviar cada paquet.

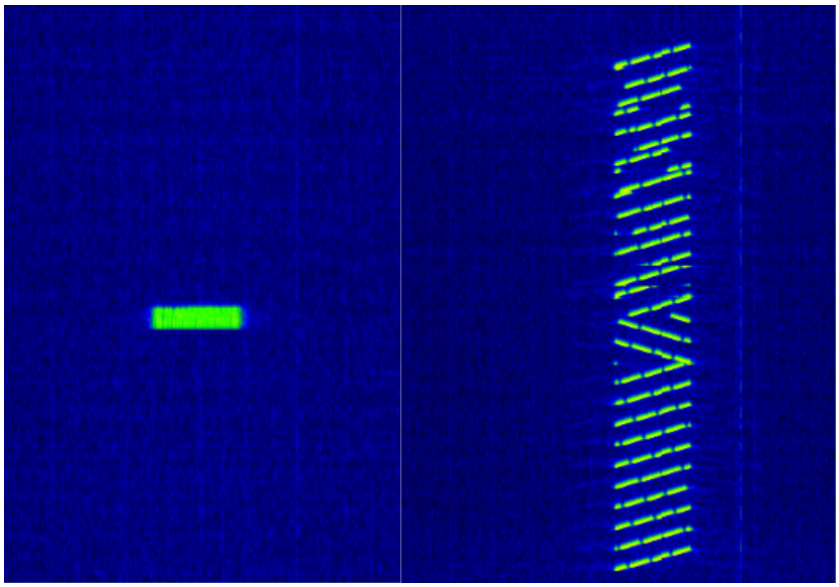


Figura 4: Visualització de paquets LoRa d'1 byte amb SF7 (esquerra) i SF12 (dreta).

Analitzant la figura 4, observem paquet amb SF7 és molt ràpid en temps i només veiem un petit pols, en canvi, amb SF12 el paquet està més espaiat en temps i podem distingir fàcilment cada *chirp*. Això queda reflectit clarament en el temps d'enviament de cada paquet, per SF7 és de 26 ms mentre que per SF12 és de 828ms.

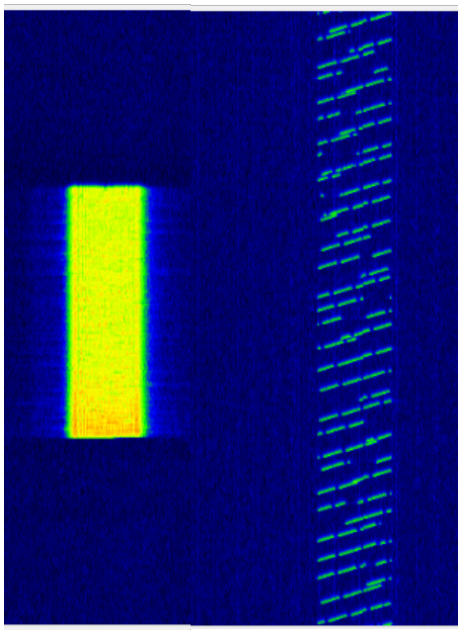


Figura 5: Visualització de paquets LoRa de 255 bytes amb SF7 (esquerra) i SF12 (dreta).

A la figura 5 enviem paquets de 255bytes (mida màxima permesa per LoRa) veiem que incrementa el temps d'enviament per a ambdós SF. Obtenim valors de 403ms per a SF7

i 7711ms per a SF12. Així, es triguen gairebé 8 segons en transmetre un paquet de mida màxima amb el *Spreading Factor* més alt, la qual cosa incrementa la probabilitat de que el paquet es corrompi per interferències externes. Tal i com veurem més endavant, aquesta diferència entre SF és el que determinarà el *throughput* i el rang màxim assolible per cada SF.

3.2 Proves *throughput*

Es proposa l'enviament de 100 paquets entre dos LoRa de manera que es pugui avaluar el *throughput* màxim que es pot aconseguir per a diferents valors de *Spreading Factor*, amplada de banda i mida de paquet considerant també possibles pèrdues de paquets. La finalitat és la generació d'un CSV que contingui els resultats de la prova.

El script complet es troba a l'Annex B. Segueix una sèrie de passos:

1. S'envien els paràmetres del test (*Spreading Factor*, amplada de banda, mida de paquet, potència de transmissió) des de l'ordinador al LoRa via SPI.
2. El transmissor envia els nous paràmetres al receptor i espera un ACK d'aquest.
3. Un cop el receptor ha rebut i actualitzat els paràmetres, s'inicia la prova de *throughput* on s'envien 100 paquets.
4. El receptor compta tots els paquets que va rebent per a obtenir el PER.
5. Quan acaba la prova, el transmissor notifica el temps de la prova al receptor i aquest envia els resultats de la prova a l'ordinador via SPI.
6. L'ordinador genera una nova entrada al CSV amb els resultats i els paràmetres de la prova.

L'últim pas per a la completa automatització del test de *throughput* és enviar de manera automàtica els paràmetres de cada test al transmissor. D'aquesta manera aconseguim fer un escombrat de tots els *Spreading Factor* i totes les amplades de banda amb diferents mides de paquet de manera automàtica.

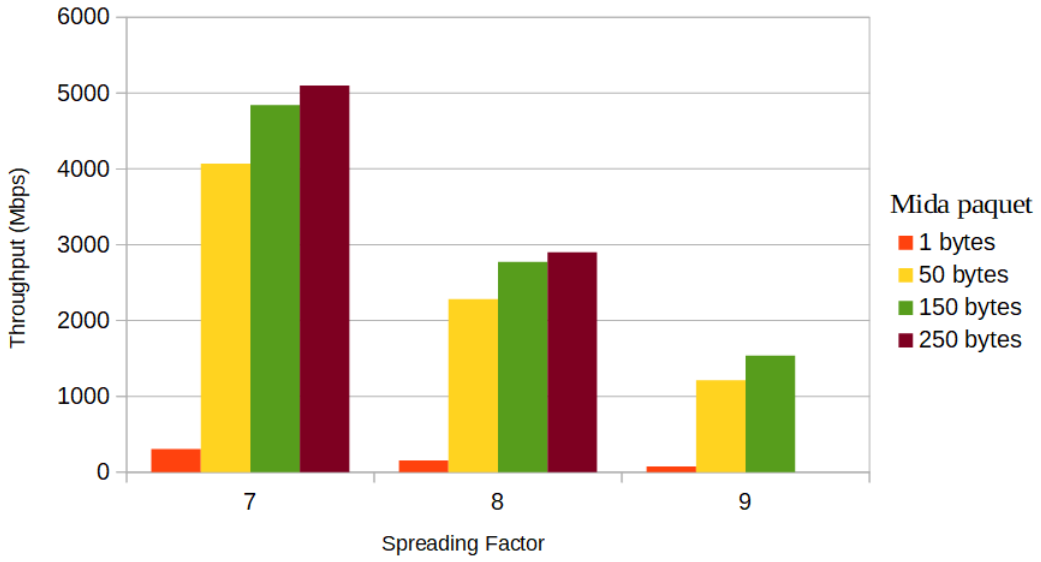


Figura 6: *Throughput* en funció de *Spreading Factor* més alts per a diferents mides de paquet amb una amplada de banda de 125KHz

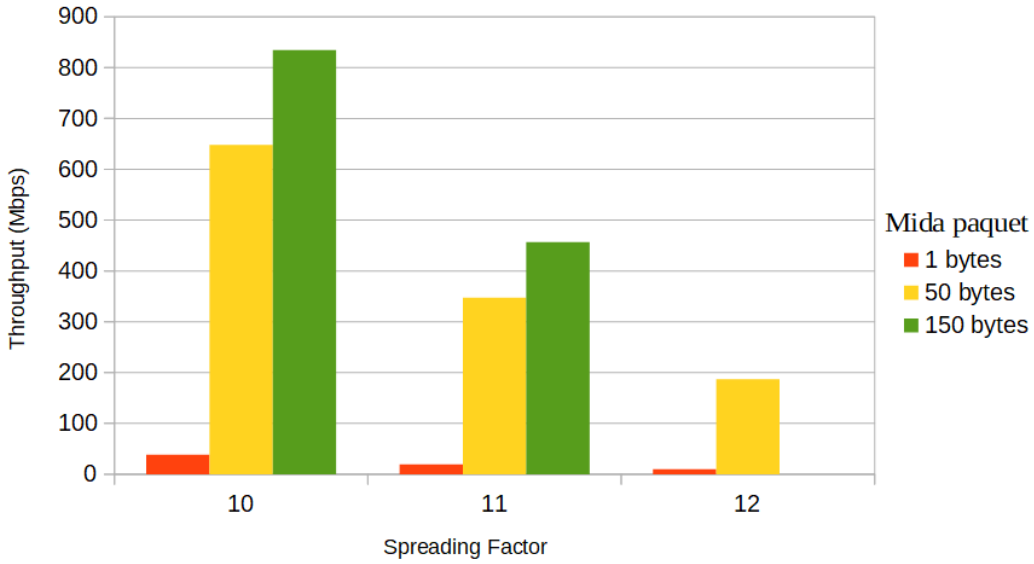


Figura 7: *Throughput* en funció de *Spreading Factor* més baixos per a diferents mides de paquet amb una amplada de banda de 125KHz

A les figures 6 i 7 podem veure el resultat de les proves per a una amplada de banda de 125KHz. A l'Annex C s'inclouen els resultats obtinguts per a les proves de *throughput* amb altres amplades de banda.

Analitzant els resultats podem veure que reduir el *Spreading Factor* redueix a la meitat el valor del *throughput* per a cada mida de paquet. També es denota la importància de la

mida dels paquets i com el fet d'utilitzar mides de paquets més grans augmenta de manera substancial el *throughput*. Tot i això, cal tenir en compte el fet que augmentar la mida de paquet provoca un increment del temps que es triga en generar i enviar un paquet, de manera que augmenta la probabilitat d'error d'aquest per interferències externes. Recordem que és important considerar les interferències externes ja que LoRa opera a la banda freqüencial ISM. A més, tal i com es va veure a l'anàlisi inicial de LoRa, els paquets amb *Spreading Factor* més elevats són molt costosos de generar (de l'ordre de varis segons) i, per tant, el fet de perdre un paquet podria ser desastrós per al *throughput*.

Cal comentar que algunes configuracions que ofereixen *throughput* alt veurem que el receptor només rep el 50% dels paquets degut a que s'envien els paquets massa ràpid i només en pot processar un de cada dos. Pot ser causa de la lentitud de processament de l'Arduino receptor i provoca que hi hagi *throughputs* una mica menors als mostrats a les gràfiques. Per tant, per a solucionar-ho, cal ajustar el temps entre paquets al transmissor depenent de la configuració que es vulgui utilitzar. També es pot observar com algunes mides de paquet no es van poder aconseguir amb *Spreading Factor* més alts degut a que no es rebien paquets.

Com es pot observar, veiem que a vegades tenim dificultats per transmetre grans quantitats de paquets seguits i els resultats no són gaire bons. A més, hi ha vegades en que l'Arduino receptor es queda aturat quan es transmeten molts paquets seguits. Tot això fa pensar que tant el protocol LoRa com el mòdul que estem avaluant no han estat dissenyats per a obtenir valors de *throughput* elevats, sinó per a altres aplicacions de IoT on es requereixi l'enviament de pocs paquets.

3.3 Proves rang

Es proposa l'enviament de paquets entre dos mòduls LoRa que van incrementant la seva distància progressivament fins a perdre la comunicació. A més, s'intenta obtenir el PER[2] per a diferents distàncies.

Els scripts utilitzat es troba a l'Annex D. El receptor simplement mostra per pantalla els paquets que rep. El transmissor admet una sèrie de funcions:

- Enviament constant de paquets amb un temps entre paquets determinat.
- Modificar el temps entre paquets.
- Realitzar una prova d'enviament de 10 paquets seguits.
- Enviament de nous paràmetres al receptor (*Spreading Factor*, amplada de banda).

El primer pas és decidir la configuració a utilitzar. Tal i com hem vist, els paràmetres que permeten obtenir un *link budget*[3] més alt són:

[2]PER(Packet Error Rate) defineix la relació entre el nombre de paquets erronis després del *Forward Error Correction*(FEC) i el nombre total de paquets

[3]*link budget* és la suma de guanys i pèrdues que experimenta un senyal en un sistema de telecomunicacions

- *Spreading Factor*: el més alt possible, SF12.
- Amplada de banda: el més petit possible. El mínim que permetia el nostre mòdul és 125KHz.
- Potència de transmissió: el més alt possible, 20dBm.
- Mida de paquet: el més petita possible, 1 byte.

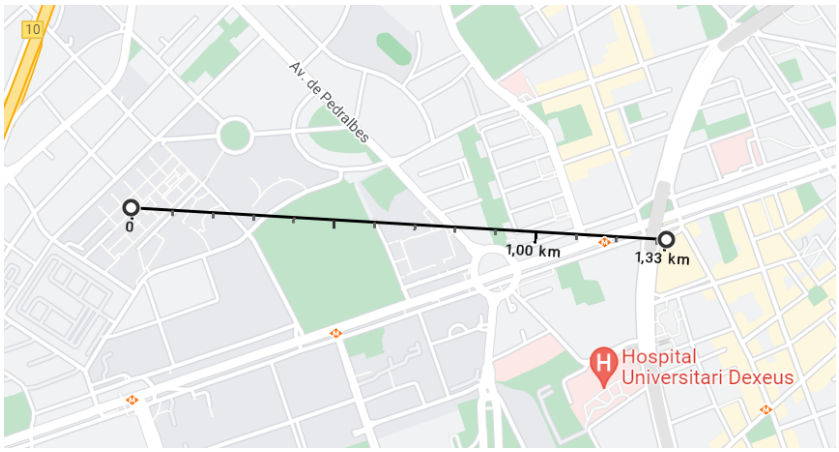


Figura 8: Màxima distància aconseguida amb LoRa en zona urbana.

Es pot apreciar a la figura 8 com la distància màxima aconseguida és de 1,33km. Cal mencionar que en aquest punt es rebien pocs paquets, i, per tant, s'aconseguia una comunicació molt poc fiable. També és necessari esmentar que ens trobàvem en un entorn urbà, amb bastants obstacles (edificis alts, arbres, etcètera) i segurament amb alguna interferència en la banda utilitzada.

La segona prova de rang es realitza en un entorn més rural, amb un dels dos LoRa col·locat sobre una muntanya amb bona visió perifèrica i l'altre allunyant-se progressivament d'aquest.

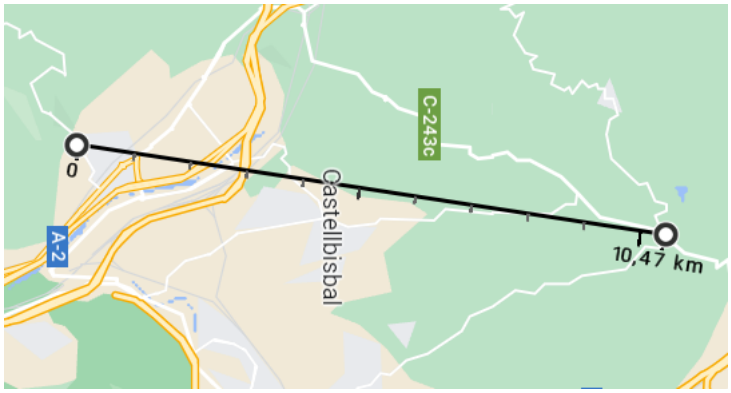


Figura 9: Màxima distància assolida amb LoRa en entorn rural

S'obté una comunicació fins als 10,47 km, tal i com es veu a la figura 9, moment en el qual es comencen a perdre paquets. Abans d'aquest punt es rebien tots els paquets de manera fiable. Cal notar que no hi havia visió directa entre els mòduls en cap moment i teníem molts arbres pel mig, sobretot a partir dels 4km de distància; tot i això, no es van observar afectacions.

Per a fer la prova de PER es proposa l'enviament de 10 paquets per a veure quants es reben correctament. El test es va realitzar des de posicions diferents, allunyant-nos fins a la distància màxima per a cada entorn.

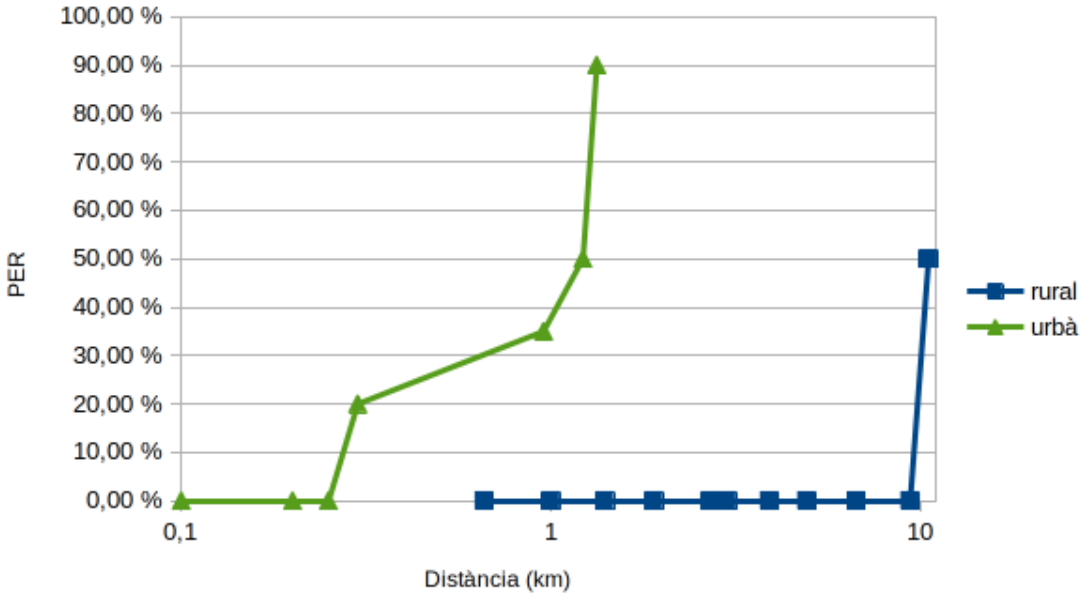


Figura 10: PER en funció de la distància en entorns rural i urbà

A la figura 10 veiem diferència entre els dos entorns. A la zona rural obtenim un PER del 0% fins que no ens aproximem a la distància màxima, moment en el qual s'obté un PER al voltant del 50%. En la zona urbana, en canvi, veiem com el PER es va incrementant gradualment conforme augmenta la distància, obtenint a la distància màxima un PER molt elevat.

Aquestes proves fan pensar que es dificulta molt la implementació de LoRa en entorns urbans degut a interferències externes que fan que el RSSI[4] no arribi al nivell mínim necessari. En aquestes zones les transmissions no són fiables i és probable tenir pèrdues inclús quan les distàncies entre els mòduls són petites.

També cal treure conclusions sobre el fet que en les proves en l'entorn rural el dispositiu transmissor estava col·locat sobre una muntanya, cosa que sembla facilitar molt la

[4]RSSI (Received Signal Strength Indicator) ens proporciona la potència del senyal rebuda

propagació fins a grans distàncies incrementant a més la fiabilitat. Aplicant-ho a la nostra solució: el simple fet d'elevar el mòdul del camp base per tal de tenir una visió perifèrica facilitaria molt la comunicació amb el sensor tot i que aquest estigués col·locat en una zona sense visibilitat directa, incrementant a més la fiabilitat i reduint el PER.

En conclusió, hem pogut igualar en les proves de camp els valors teòrics proposats per LoRa en l'entorn rural, quedant una mica per sota en l'entorn urbà. Així, queda demostrada la utilitat de LoRa en aplicacions IoT de llarg abast. Tot i això, sempre que s'implementi LoRa i es vulgui aconseguir una comunicació completament fiable, és necessari implementar ACK a nivell d'aplicació, tal i com s'ha fet als tests de *throughput*.

4 802.11ah

Comptem amb dos tipus de mòduls 802.11ah anomenats AHPI7292S i AHMB7292S. Més informació sobre aquests a [9] i [10]. La única diferència entre aquests és que el AHMB7292S està pensat per a ser utilitzat amb un adaptador *mikroBus* mentre que el AHPI7292S es connecta directament com a "Hat" a la RaspberryPi. Tots dos contenen el mateix SoC ("System on a chip") creat per Newracom, de manera que es pot utilitzar el mateix software per a ambdós, així que no fem distincions entre aquests.

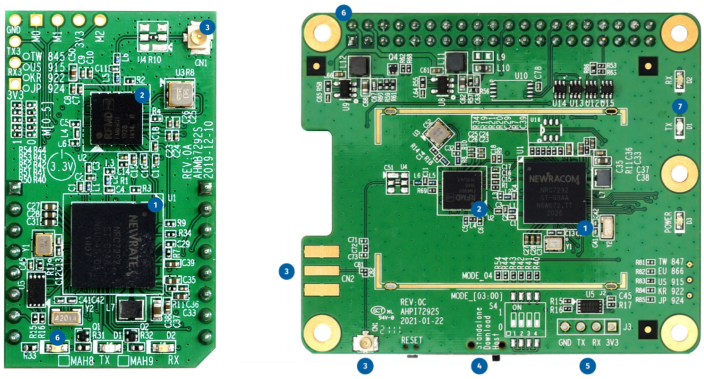


Figura 11: Mòduls AHMB7292S (esquerra) i AHPI7292S (dreta).

El mòdul permet 3 modes de funcionament: "Host", "Standalone" i "Download". Els modes "Standalone" i "Download" estan pensats per a utilitzar el mòdul de manera independent, és a dir, sense una placa hoste com podria ser una RaspberryPi. El mode "Host" és en el que ens centrarem. La configuració inicial de la RaspberryPi es troba a l'Annex E.

4.1 Descripció del software

La gran part del software proporcionat pel fabricant es troba a la carpeta `~/nrc_pkg/script`.

4.1.1 Script d'inici

Comencem explicant l'script que s'ens proporciona per a iniciar el mòdul en els diferents modes: AP, STA, RELAY, Sniffer, MP, MAP. A nosaltres només ens interessen els tres primers:

- AP (*Access Point*): inicia el mòdul com a Punt d'accés.
- STA (*Station*): inicia el mòdul com a client que permet connectar-se a un Punt d'accés.
- RELAY: inicia en els dos modes AP i STA alhora (cadascún en una interfície virtual diferent).

L'script admet 3 arguments: `./start.py [mode] [seguretat] [país]`. En el nostre cas posarem un 0 al paràmetre de seguretat ja que no volem utilitzar cap tipus de seguretat i anirem alternant el país entre `EU` o `US` per a utilitzar canals de la banda europea o americana respectivament. Necessitem fer ús de la banda americana conté canals de 1MHz, 2MHz i 4MHz mentre que la banda europea només permet canals de 1MHz i 2MHz.

Al directori `~/nrc_pkg/script/conf/etc/` trobem una sèrie de scripts de configuració utilitzats al script `start.py` així com un fitxer de configuració anomenat `CONFIG_IP`. En aquest fitxer podem especificar la configuració d'adreces IP que volem utilitzar per a cadascun dels modes d'inici. A més, es permet indicar si volem fer ús d'adreces estàtiques o DHCP. El propi script d'inici s'encarrega de llegir aquest fitxer i fer la configuració dels fitxers de DHCP i DNS.

Si ens dirigim a `~/nrc_pkg/script/conf/[país]/` trobarem els fitxers utilitzats per `hostapd`[5] i `wpa_supplicant`[6] per a iniciar el punt d'accés o el client, respectivament. Dins d'aquests fitxers trobem paràmetres com:

- `hostapd`: conté l'SSID i el canal que utilitzarà el punt d'accés.
- `wpa_supplicant`: conté l'SSID al qual ens volem connectar, així com les freqüències que volem escanejar per a buscar punts d'accés.

Si ens fixem en les freqüències especificades en aquests fitxers veurem que no són les de l'estàndard 802.11ah, ja que estàn al voltant dels 2,4GHz i els 5GHz. Això és degut a que, ambdós *daemons*, `wpa_supplicant` i `hostapd`, només suporten aquestes freqüències i no són capaços d'utilitzar l'estàndard 802.11ah per ells mateixos. Aquí és on entra el *driver*, que entre d'altres funcions, s'encarrega de mapejar les freqüències de 2,4GHz i 5GHz a les del 802.11ah.

Tot i que el fabricant ens proporciona tots aquests scripts, per a la solució final hem decidit crear el nostre script d'inici per tal de tenir més control sobre la configuració de cada mòdul. Tenim un script d'inici per al punt d'accés i un altre per al client. Els podem trobar a l'Annex F i veiem que es resumeixen en una sèrie de passos molt senzills:

1. Executem `clock_config.sh`, un script que proporciona el fabricant per a augmentar la velocitat del processador.
2. Es paren tots els processos existents de `hostapd`, `wpa_supplicant`, `dhcp` i `dnsmasq` i s'elimina el *driver* fent ús de `rmmod`.
3. S'insereix el *driver* indicant una sèrie de paràmetres. Veure [8] per a més informació sobre els paràmetres del *driver*.
4. Esperem uns segons a que es configuri el *driver* i aixequem la interfície *wlan0*.

[5] **hostapd**(*Host Access Point Daemon*) és un *daemon* per a GNU/Linux capaç de fer funcionar una targeta sense fil en un punt d'accés

[6] **wpa_supplicant** és un *daemon* per a implementar un sol·licitant (*supplicant*) IEEE 802.11, és a dir, per a connectar-se a un punt d'accés IEEE 802.11

5. Especifiquem algunes opcions fent ús de la comana `cli_app` explicada a la secció 4.1.2.
6. Iniciem `hostapd` o bé `wpa_supplicant` amb el seu fitxer de configuració corresponent depenent de si volem iniciar un punt d'accés o una estació, respectivament.
7. Finalment, assignem una adreça IP.

4.1.2 Command Line Application

La comana `cli_app` (abreviatura de *Command Line Application*) ens la proporciona el fabricant. Necessitarem executar les següents comanes per tal de compilar el fitxer i copiar el binari obtingut a `/usr/bin` per a poder utilitzar la comana des de qualsevol terminal.

```
$ cd /home/pi/nrc7292_sw_pkg/package/host/src/cli_app
$ make clean; make
$ mv ~/nrc_pkg/script/cli_app ~/nrc_pkg/script/cli_app.old
$ sudo cp ./cli_app ~/nrc_pkg/script/
$ sudo cp ./cli_app /usr/bin
```

Cal comentar que és necessari haver inserit el *driver* prèviament, ja que és el que permet la comunicació amb el mòdul via HSPI. Un cop fet això ja podem utilitzar la comana `cli_app`, la qual ens permet interactuar amb el mòdul 802.11ah per a modificar diversos paràmetres. Ens permet monitoritzar la qualitat del canal, o bé fer canvis a la capa física, així com obtenir estadístiques sobre la transmissió de paquets, entre d'altres.

4.2 Proves throughput i distància

Es proposa l'ús de l'eina `iperf` per a la avaluació del *throughput* per a diferents distàncies i escenaris. Les comanes que utilitzarem principalment seràn:

- Al servidor: `iperf3 -s`
- Al client: `iperf3 -c [IP] -u -b [BW]` on IP és la adreça IP del servidor i BW és l'ample de banda que volem assolir. S'utilitza la opció `-u` ja que es vol utilitzar UDP.

4.2.1 Rendiment en funció del MCS

La primera prova es realitza amb els dos mòduls col·locats molt a prop per tal d'aconseguir les millors prestacions i poder avaluar el *throughput* per cada MCS i per cada interval de guarda. Per a fer-ho, farem ús de la comana `cli_app`, que recordem que s'utilitza per a interactuar amb el mòdul i permet modificar diversos paràmetres. Per a obtenir el funcionament correcte, primer necessitem desactivar el `rc` (*Rate Control*) que permet la selecció automàtica de MCS en funció de les condicions del canal i que està activat en el mòdul per defecte. Després escollim el valor de MCS que desitgem utilitzar i finalment especifiquem l'interval de guarda que volem utilitzar. Cal notar que és necessari executar les comanes a ambdós RaspberryPi.

```
$ cli_app set rc off
$ cli_app test mcs [MCS]
$ cli_app set gi [long/short/auto]
```

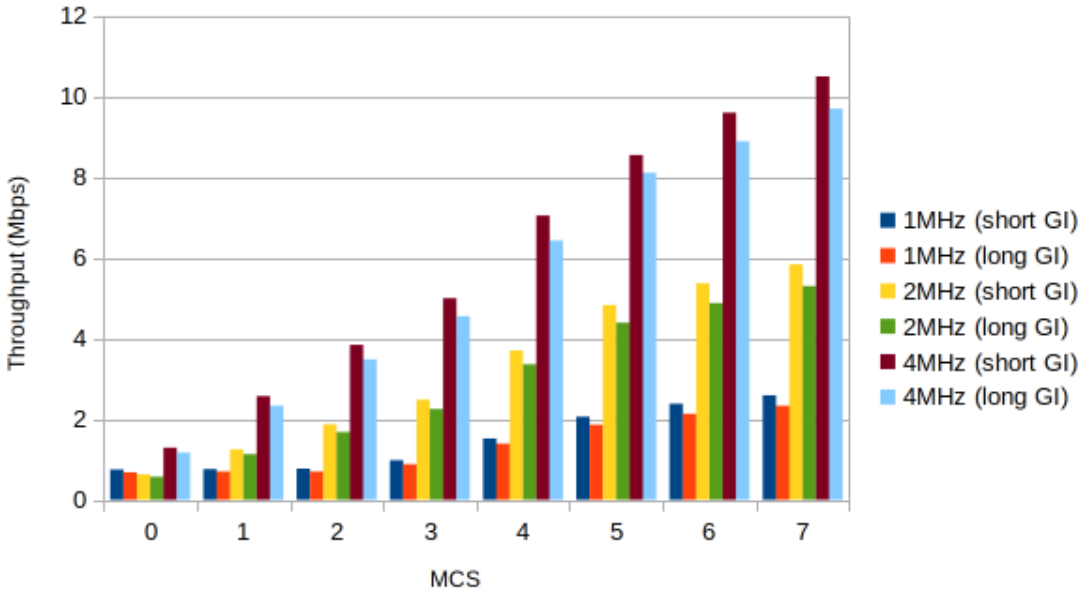


Figura 12: Rendiment en funció del MCS per a cada ample de banda

Obtenim els resultats que esperàvem, el *throughput* decreix quan reduïm el valor del MCS degut a que s'utilitzen modulacions més robustes però que no permeten codificar tants bits per símbol. A més, veiem que el rendiment pot assolir valors més alts quan augmentem l'amplada de banda. També es pot comprovar com l'interval de guarda afecta mínimament al *throughput*. Utilitzant l'interval de guarda llarg s'obté un 10% menys de *throughput* respecte l'interval de guarda curt per a tots els casos.

4.2.2 Rendiment en funció de la distància (LOS)

En aquest apartat es realitzen proves de *throughput* en funció de la distància. Les primeres proves es fan en un escenari on procurem que existeixi una visió directa entre els mòduls en tot moment, de manera que puguem comparar-ho posteriorment amb un escenari que s'ajusti més al nostre cas.

Utilitzarem un interval de guarda curt per tal d'obtenir el màxim *throughput*. No s'ha fet l'avaluació per a l'interval de guarda curt però es pot considerar que s'obtindria un 10% menys de *throughput*, tal i es va analitzar a la figura 12

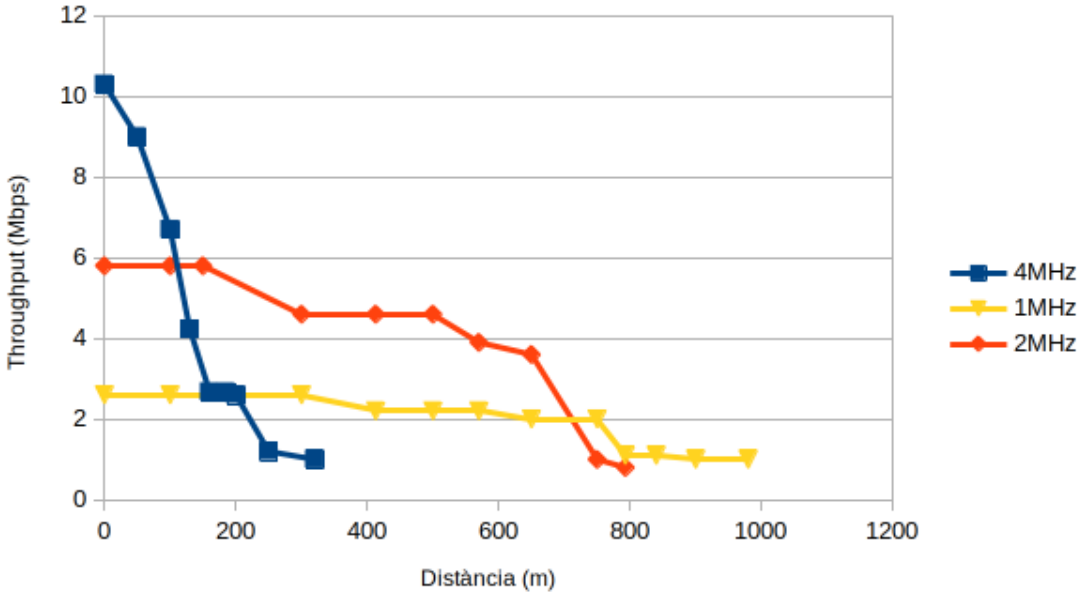


Figura 13: *Throughput* en funció de la distància per a les 3 amplades de banda disponibles

Els resultats que es mostren a la figura 13 s'han obtingut a partir de moltes proves diferents en diferents localitzacions i considerant només els resultats favorables en els que no teníem cap problema de *hardware*. És necessari comentar-ho ja que s'obtenien molts problemes de *hardware* de tant en tant. Alguns dies teníem velocitats de l'ordre dels kbps a distàncies de pocs metres que pujaven fins al màxim quan es col·locaven els mòduls al costat; això semblava indicar un problema de potència dels mòduls i les RaspberryPi però no es va poder trobar la causa ja que passava amb tots els mòduls, RaspberryPi i antenes provades. Cal comentar també que quan es col·loquen les antenes a terra el *throughput* també baixa als kbps inclús a pocs metres.

En conclusió, l'amplada de banda ens ofereixen un compromís entre distància i *throughput*. Cada amplada de banda té la seva distància òptima d'utilització. És per això que seria interessant poder canviar d'amplada de banda conforme la distància augmenta, per tal d'optimitzar el *throughput* al màxim. Podem fer-ho observant la figura superior i decidint en quins punts s'ha de reduir el valor de l'amplada de banda.

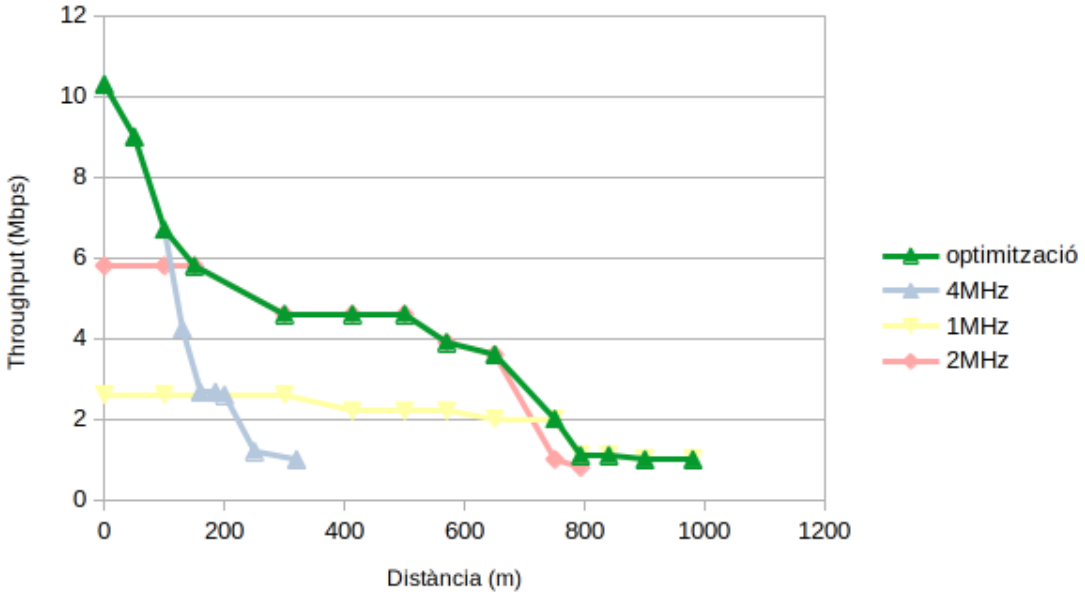


Figura 14: Optimització del throughput en funció de la distància modificant l'amplada de banda

A la figura 14 veiem la optimització del *throughput* modificant l'amplada de banda. Seria interessant comprovar si és possible implementar aquests canvis de canal quan la distància s'aproxima als llindars de cada ample de banda, ja que el nostre mòdul no és capaç de fer-ho per si mateix. Per a fer-ho, necessitem conèixer el temps que es necessita per a canviar de canal.

Per a canviar de canal necessitem modificar el fitxer de `hostapd` i tornar a executar el script d'inici `bootAPsetup.sh`. Mirem el temps que triga en executar-se aquest script afegint la línia `start=$(date +%s.%6N)` a l'inici i les línies al final:

```
end=$(date +%s.%6N)
duration=$(echo "scale=6; $end - $start" | bc)
echo $duration
```

Així, veiem que el script triga uns 4,9 s de mitja. Un cop modificat el canal, també necessitem conèixer el temps que triga el client en connectar-se al punt d'accés. Per a fer-ho, primer afegim el paràmetre `-f APlog.txt` a la comana de `hostapd` de l'script d'inici per tal de poder veure i analitzar els *logs* del *daemon*. Ara creem el següent script que s'encarrega de llegir els *logs* del *daemon*, comença el comptador quan el punt d'accés està actiu i para el comptador quan es connecta el client.

```
#!/bin/bash
#client_connecion_delay.sh

cat APlog.txt | grep AP-ENABLED
```

```
while [[ $? == 1 ]]
do
    cat APlog.txt | grep AP-ENABLED
done

start=$(date +%s.%6N)
cat APlog.txt | grep STA-CONNEC
while [[ $? == 1 ]]
do
    cat APlog.txt | grep STA-CONNEC
done

end=$(date +%s.%6N)
duration=$(echo "scale=6; $end - $start" | bc)
echo $duration
```

D'aquesta manera som capaços d'obtenir el temps que triga en iniciar-se el punt d'accés i el temps que triga en connectar-se el client. La suma dels dos és de 10,1 s aproximadament. Cal comentar que aquest temps es pot incrementar una mica si el client es troba més lluny degut al retard o a altres factors com la velocitat de la CPU, però ens proporciona una xifra orientativa sobre el temps que es triga en canviar de canal.

Com veiem, 10 s és un temps elevat per a un canvi de canal. Necessitem avaluar en quins casos és útil l'ús de les tres amplades de banda. Seria interessant generar un model per veure quant de temps ha d'estar el drone per sota de la distància llindar del canvi de canal de manera que s'optimitzi el *throughput* al màxim. També es podria desenvolupar un algoritme per tal de decidir quin és el millor moment per a fer el canvi de canal tenint en compte diversos paràmetres com interferències, visibilitat, temps des de l'últim canvi de canal, etcètera. Això queda una mica fora de l'abast d'aquest projecte i és per això que es deixarà per a un futur.

En conclusió, el temps que es triga en canviar de canal és massa elevat degut a que el mòdul del que disposem actualment no permet el canvi de canal dinàmic i, per tant, necessitem reiniciar el punt d'accés. Això fa que sigui molt més costós i requereixi de més temps, a més de dificultar la implementació en un escenari real.

4.2.3 Rendiment en funció de la distància (visibilitat reduïda)

En aquesta secció es vol obtenir el *throughput* fent ús de diferents configuracions i en un escenari més proper al nostre cas, és a dir, introduint una sèrie d'obstacles entre els mòduls. En aquest cas ens col·loquem en una zona amb arbres. Les proves es van fer per a un parell de distàncies fixades i amb el canal de 2 MHz, variant l'interval de guarda.

Analitzant l'interval de guarda curt veiem que s'obtenen *throughputs* molt inestables que fluctuen ràpidament entre el valor màxim i el mínim. Per exemple, a 160 m de distància es pot aconseguir 5,5 Mbps amb visió directa, però col·locant-nos darrere d'una filera d'arbres es produeixen baixades constants i un *throughput* inestable en general. El mateix passa si ens col·loquem més lluny, a 330 m, tot i que ara el *throughput* mai arriba al valor

màxim i fluctua entre valors més reduïts.

Si canviem a l'interval de guarda llarg veiem una gran millora, sobretot en quant a l'estabilitat del *throughput*; ja que tot i que no s'obtenen les velocitats màximes de 4,6 Mbps als 330 m, aquestes són molt més estables al volant d'un valor de 3 Mbps.

En conclusió, queda comprovada la eficàcia de l'interval de guarda llarg en presència d'obstacles. Degut a la estabilitat del *throughput* amb aquest interval de guarda, es poden proporcionar uns valors més objectius de *throughput* al projecte per tal de complir amb els requeriments. Així, es recomana l'ús d'aquesta configuració sempre que no es pugui garantir una visió directa entre els mòduls.

4.3 Configuració del repetidor i automatització de processos

Actualment no disposem de molta informació sobre la col·locació ni la distància a la que estarà ubicada el sensor. Per això, es proposa una configuració amb intenció de maximitzar tant el *throughput* com la distància en zones boscoses. Per a aconseguir-ho, es proposa la utilització d'un drone com a repetidor, és a dir, la comunicació entre el sensor i el camp base estaria formada per 2 salts passant pel drone. Amb això aconseguim minimitzar la massa d'arbres que el canal ha de travessar fins al sensor, augmentant d'aquesta manera la qualitat del senyal.

Tal i com hem vist a les proves de *throughput*, el fet de tenir visió directa entre els mòduls fa que es puguin aconseguir *throughputs* més estables i més elevats. Així, quan introduïm el drone podem aconseguir visió directa amb aquest i cobrir distàncies molt més elevades. Tot i que aquesta configuració introdueix més complexitat a la solució general, és molt necessària per tal d'obtenir canals més robustos i que ofereixin més rendiment.

La configuració inicial feia ús de la nova entitat que introdueix l'estàndard 802.11ah, el *Relay*(repetidor). El problema d'aquest mode de funcionament (en el cas del nostre mòdul) és que disposem d'una sola interfície ràdio, de manera que el repetidor només pot estar escoltant o transmetent alhora, la qual cosa redueix molt el rendiment. Per a solucionar-ho, és necessari la utilització de dos mòduls 802.11ah en una mateixa RaspberryPi o bé l'encadenament de dos RaspberryPi mitjançant un cable RJ45. Finalment vam optar per aquesta última ja que era la solució més senzilla i directa.

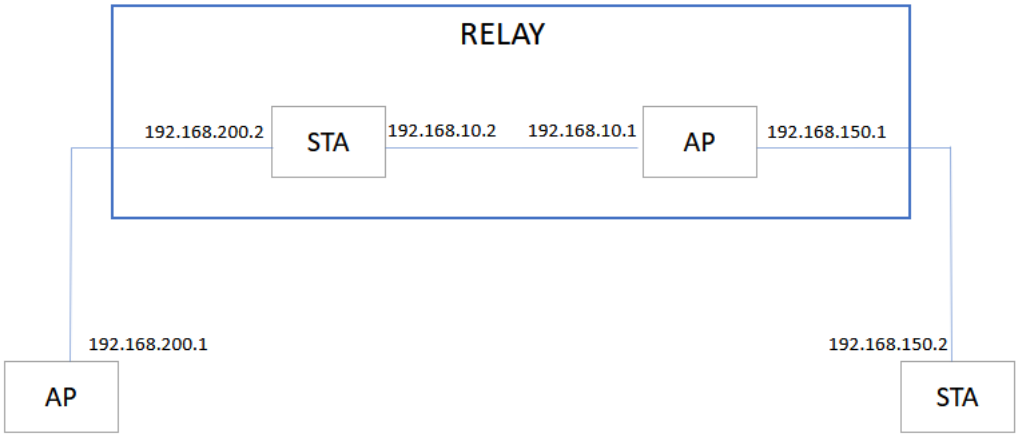


Figura 15: Configuració d'adreces IP de la solució final.

Observant la figura 15, veiem que és necessari la utilització de 4 RaspberryPi amb 4 mòduls 802.11ah. La RaspberryPi que està com a AP de terra es connecta a l'ordinador mitjançant un RJ45, mentre que la STA de terra seria el sensor. Per a iniciar cada AP o STA simplement utilitzarem l'script d'inici que hem creat. Ara només ens falta configurar les rutes correctament i les IP per la interfície *eth0*.

També necessitem automatitzar tot el procés de configuració de manera que s'executin totes les comanes automàticament quan engeguem les RaspberryPi. Per a fer-ho utilitzarem `cron`[7] i `crontab`[8]. Cal comentar que `cron` s'executa en el seu propi entorn (heretat del procés *init* que el genera quan l'ordinador s'engega) i, per tant, si volem que hereti les variables d'entorn del terminal necessitem afegir la línia `bash -lc` abans d'executar la comana que desitgem. Podem trobar tota la configuració per cada mòdul, així com altres scripts d'ajuda i d'automatització als Annexes G i H, respectivament.

[7] **cron** és un *daemon* que s'encarrega d'executar scripts a determinades hores o dies segons la hora configurada al sistema.

[8] **crontab** és un fitxer amb un format especial per a programar execucions d'scripts que **cron** és capaç d'interpretar

Figura 16: Prototip del repetidor

A la figura 16 veiem com queda el prototip del repetidor. Podem observar ambdós RaspberryPi unides mitjançant un RJ45, cadascuna amb el seu mòdul 802.11ah connectat a una antena.

4.3.1 Rendiment del repetidor

Tal i com s'ha comentat breument, el primer repetidor creat amb tres RaspberryPi no va funcionar correctament degut a que cada mòdul només compta amb una sola interfície ràdio. Això provoca que només pugui estar transmetent o rebent informació, la qual cosa feia disminuir el *throughput*. A més, la CPU de la RaspberryPi semblava ralentitzar-se molt després d'enviar grans quantitats d'informació pels dos canals i s'havia de reiniciar per tal de que tornés a funcionar correctament. Per això es va optar per la solució de fer un repetidor amb dues RaspberryPi i és la solució amb la que es realitzen les següents proves.

Les primeres proves es fan amb tots els mòduls molt a prop i utilitzant el canal de 2 MHz. Recordem que ara tenim dos canals 802.11ah diferents: un entre el punt d'accés i el repetidor i l'altre entre el repetidor i el sensor. És conegut que es poden produir interferències entre canals propers, així que escollim canals molt separats per tal d'intentar evitar-ho: un a la banda freqüencial europea (864 MHz) i l'altre a l'americana (909 MHz). Tot i això, si s'executa un `iperf3` als dos canals alhora, es veurà que un dels dos baixa molt el *throughput* (de l'ordre de kbps). De fet, utilitzem CubicSDR per tal de visualitzar si existeixen interferències i, efectivament, podem veure que, tot i estar molt separats els canals, es produeixen interferències entre ells. Fent les proves en exteriors segueixen existint les interferències.

Per tant, sembla que una separació entre canals de 45 MHz no és suficient per a combatre les interferències. Necessitem fer ús d'altres tècniques per a evitar-les:

- Provar altres mòduls 802.11ah per veure si són capaços de ser més selectius i no generar tantes interferències.
- Es poden utilitzar antenes directives en lloc d'omnidireccionals. Així aconseguim apuntar cap a l'objectiu amb més precisió, reduint les interferències.
- Fer ús de bandes freqüencials més allunyades. Per exemple, utilitzar la banda americana (902-928 MHz) i la xinesa (755 - 787 MHz).
- Utilitzar estàndards IEEE 802.11 diferents per a cadascun dels enllaços.

Aquesta última solució sembla la més plausible a simple vista ja que evitarien per complet les interferències. Es podria fer ús de 802.11ah en l'enllaç entre el drone i el sensor ja que és el més crític degut a la massa d'arbres que ha de travessar. Així, utilitzar la freqüència inferior a 1 GHz d'aquest estàndard és el més adient. Per a l'altre enllaç es podria utilitzar 802.11n. Ens podríem beneficiar del fet que la comunicació entre el camp base i el drone es produeix amb visió directa, de manera que no s'esperen atenuacions per obstacles, que solen ser crítiques en les bandes de 2,4 GHz i sobretot en la de 5 GHz.

4.4 Proves amb càmera

Els organitzadors de l'Xprize necessitaven un vídeo resumint el treball fet per part de cada competidor. Com a equip de comunicacions es va decidir gravar un vídeo en temps real de tot allò que veia el sensor, intentant demostrar la millora de *throughput* quan introduíem el drone com a repetidor.

La càmera utilitzada està dissenyada per a ser utilitzada amb una RaspberryPi, tal i com es pot veure a la figura 17. En el nostre muntatge, la càmera es col·loca al sensor i es visualitza el vídeo a l'ordinador destí, passant pel repetidor i per la RaspberryPi del camp base.

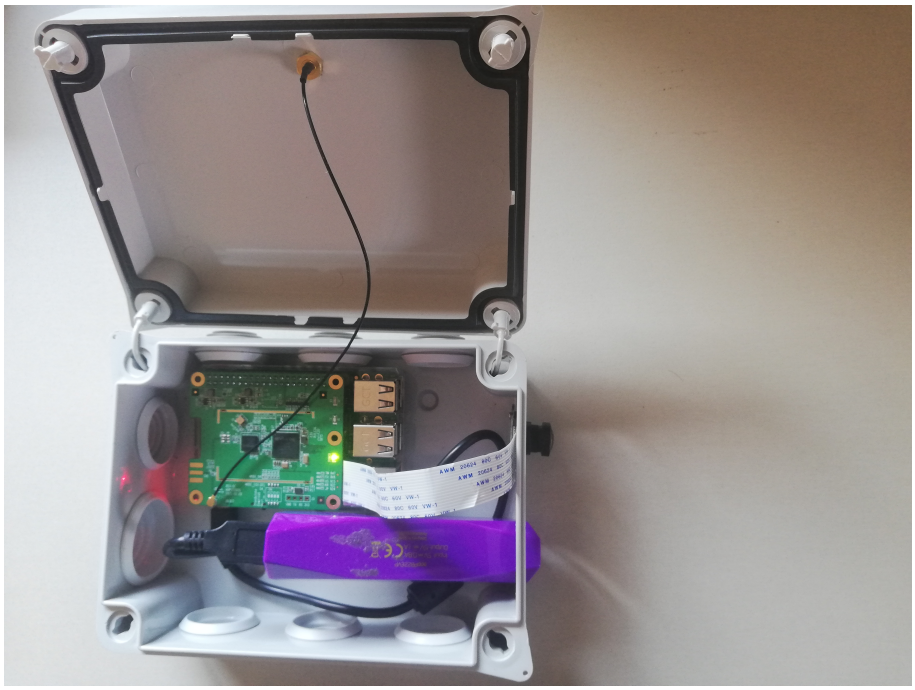


Figura 17: Prototip del sensor amb la càmera

Cal mencionar que és desconeixia la dificultat així com les eines necessàries per a transmetre vídeo en temps real. Així, es va haver de fer un gran treball de recerca previ per tal de trobar alguna eina adient per al nostre projecte. Finalment es va optar per `Gstreamer`, un *framework* multimèdia lliure útil per a la creació d'aplicacions multimèdia. A més del seu ús per a la reproducció o mescla d'àudio i vídeo, també facilita l'*streaming* en temps real. La eina funciona mitjançant la creació de *pipelines*, que es poden descriure com la concatenació de molts elements multimèdia (còdecs, multiplexors, conversors, filtres, etcètera) ordenats per a realitzar transformacions sobre els bytes d'àudio i vídeo.

`Gstreamer` és un *framework* molt extens i que requereix d'un gran coneixement per a utilitzar-lo correctament. Així, es va optar per la utilització de la eina que proporciona `Gstreamer` per a la ràpida creació de *pipelines* directament des de terminal, `gst-launch-1.0`. Aquesta eina ofereix més facilitat d'ús i no requereix de tant coneixement per a ser utilitzada. Tot i això, no és recomanable desplegar-la en una solució final ja que està pensada simplement com a eina de *debugging*.

Primerament es va intentar fer ús del còdec *hardware* que porta la RaspberryPi per defecte, però aviat ens vàrem adonar que introdueix molt retard (de l'ordre de varis segons). Després de provar diversos *pipelines*, les comanes que millor van funcionar van ser les següents:

```
SENSOR$gst-launch-1.0 libcamerasrc ! capsfilter
    caps=video/x-raw,width=640,height=480,format=NV12,framerate=30/1 !
    v4l2convert ! v4l2h264enc
    extra-controls="controls,repeat_sequence_header=1" !
```

```
    'video/x-h264,level=(string)4' ! h264parse ! rtph264pay ! udpsink
    host=10.42.0.1 port=5000
PC$gst-launch-1.0 udpsrc address=10.42.0.1 port=5000 caps=application/x-rtp !
    rtph264depay ! h264parse ! avdec_h264 ! autovideosink
```

Cal notar que s'utilitza "!" com a separador d'elements del *pipeline*. La primera comana s'executa a la RaspberryPi des d'on es vol retransmetre el vídeo (en el nostre cas el sensor), mentre que la segona s'executa a l'ordinador on es vol visualitzar el vídeo. Expliquem breument els elements del *pipeline* que s'executa al sensor:

1. Primer es captura el vídeo de la càmera com a bytes sense processar utilitzant `libcamerasrc`.
2. Ara s'utilitza `capsfilter` per a limitar la resolució i el nombre de *fps*, així com imposar el format dels píxels a NV12.
3. `v4l2h264enc` comprimeix els bytes utilitzant H.264, un còdec digital d'alta compressió molt adient per a la transmissió de vídeo en temps real. `h264parse` simplement reordena els bytes d'una manera que el còdec H.264 els pugui entendre i és necessari per al seu ús.
4. `rtph264pay` s'encarrega de codificar els bytes que surten del còdec en paquets RTP[9]
5. Finalment, `udpsink` s'encarrega de generar i enviar els paquets UDP a la direcció IP i port especificats.

Un cop s'executa la comana a la RaspberryPi, ja es pot començar a rebre a l'ordinador destí utilitzant el *pipeline* mostrat. Aquest s'encarregarà de fer els mateixos passos que el transmissor però a l'inrevés, és a dir, descodifica els paquets RTP, descomprimeix utilitzant H.264 i ens mostra el vídeo per pantalla. D'aquesta manera obtenim un vídeo en temps real amb molt poc retard, de l'ordre de pocs mil·lisegons.

Un cop trobat el *pipeline* ideal, és necessari analitzar la velocitat requerida per la transmissió de vídeo en temps real utilitzant diverses resolucions i *fps*; d'aquesta manera podrem determinar quina amplada de banda necessitem utilitzar, així com la distància màxima a la que podem transmetre el vídeo. Per a fer-ho, iniciarem el vídeo a la RaspberryPi i utilitzarem `Wireshark` per a capturar tots els paquets a la màquina destí. `Wireshark` permet obtenir gràfiques en temps real del *throughput* generat per una màquina, tal i com es pot observar a les figures 18, 19 i 20.

[9]RTP(Real Time Transport Protocol) és un protocol de nivell d'aplicació que defineix un format per a la transmissió de vídeo en temps real.

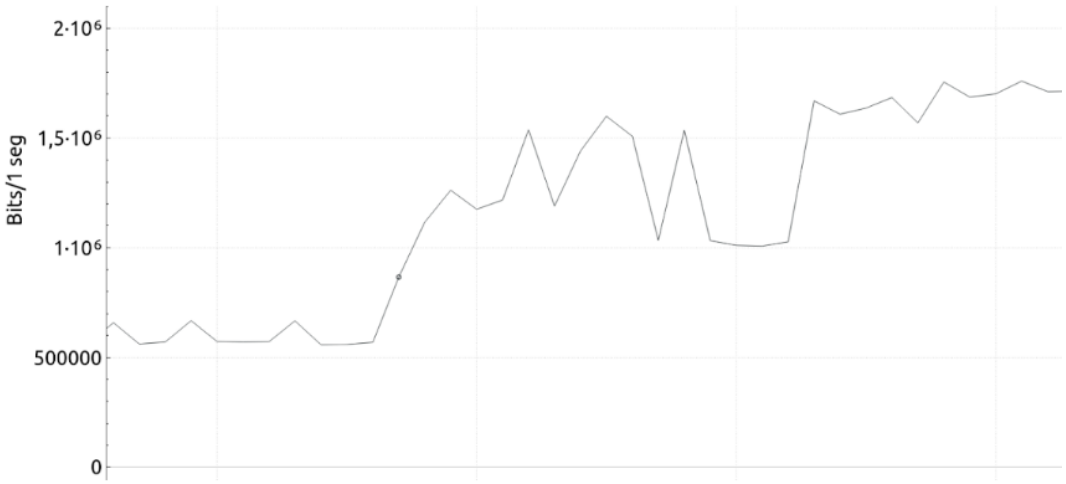


Figura 18: *Throughput* generat amb una resolució de 320x240 píxels i 30 *fps*

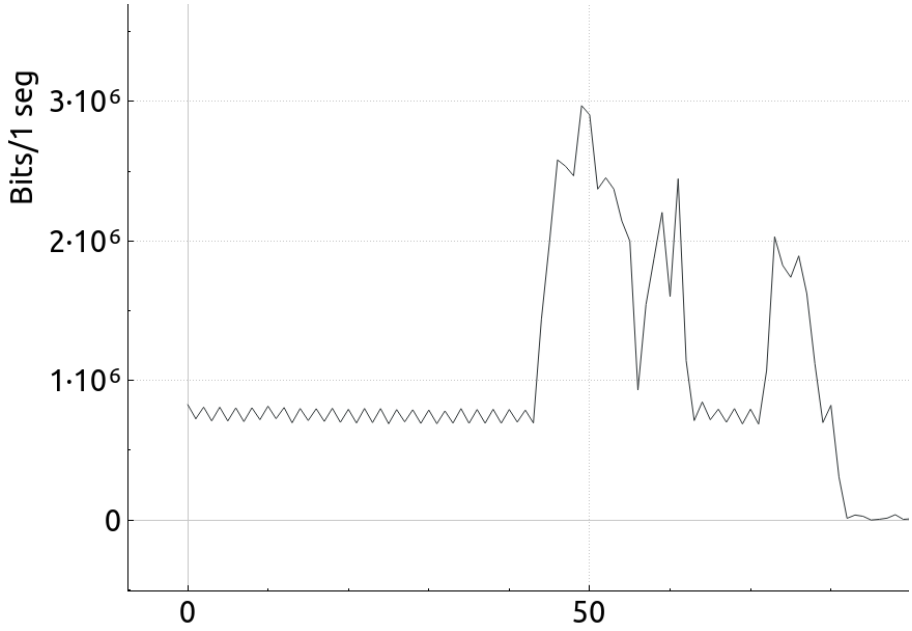
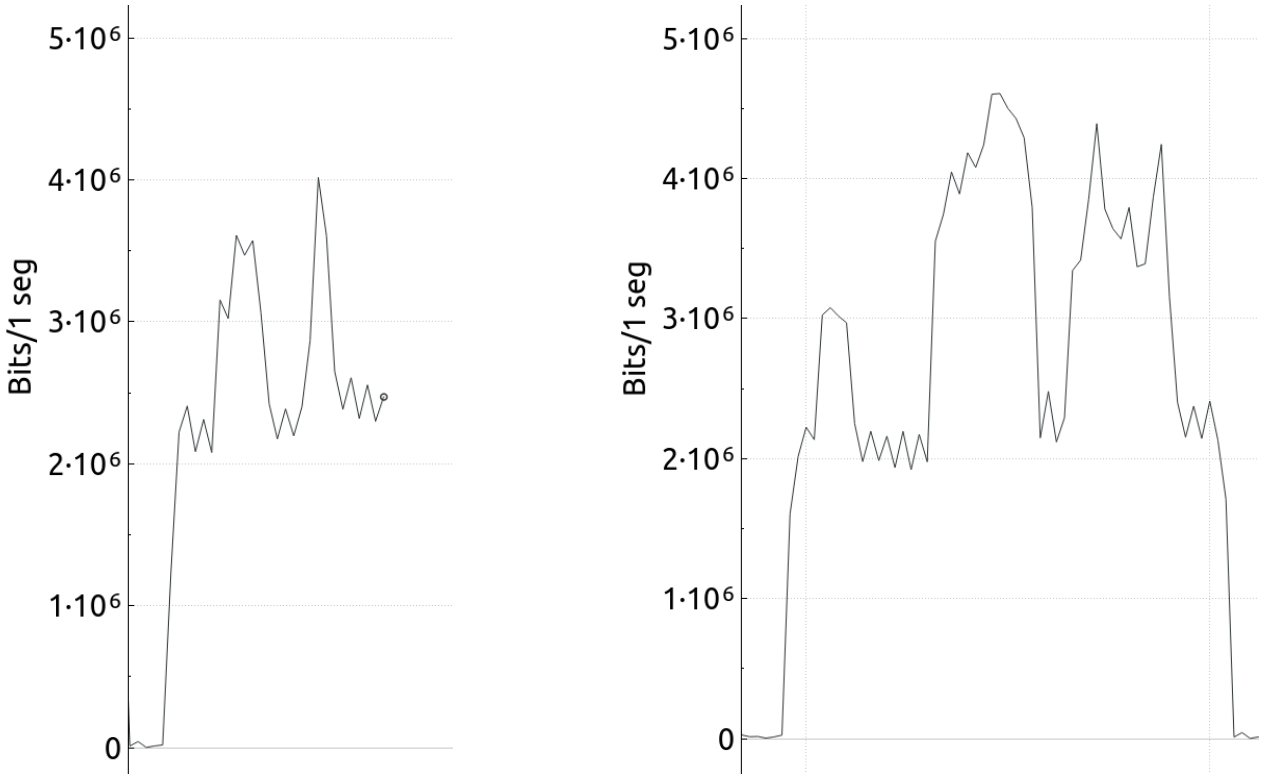


Figura 19: *Throughput* generat amb una resolució de 640x400 píxels i 30 *fps*



(a) 60 *fps* (b) 20 fps

Figura 20: *Throughput* generat amb una resolució de 800x600 píxels i diferents *fps*

Es pot observar clarament com incrementant la resolució incrementem de manera substancial el *throughput* generat. Augmentar el nombre de *fps*, però, no genera un augment tant notable del *throughput* necessari.

A les gràfiques superiors s'observa un *throughput* estable al voltant d'un valor que depèn de cada resolució, però també s'observen molts pics de *throughput* que l'incrementen en 2Mbps independentment de la resolució utilitzada. Això és degut a que quan la càmera obté una imatge que no varia massa, el *throughput* s'estabilitza, però quan es detecta molt de moviment, es produeixen aquests pics. Analitzem les gràfiques amb més detall a la Taula 8:

Resolució	**Throughput estable**	**Throughput màxim**
800x600	2 Mbps	4,75 Mbps
640x400	750 Kbps	3 Mbps
320x240	600 Kbps	1,8 Mbps

Taula 8: Throughput en funció de la resolució i del moviment de la càmera

Necessitem comparar els *throughputs* necessaris per a cada resolució amb el *throughput* assolible per cada amplada de banda de manera que puguem determinar quin hem d'utilitzar

en cada moment. Així, podem calcular la distància màxima assolible amb cada resolució i amplada de banda.

Resolució	4 MHz (m)	2 MHz (m)	1 MHz (m)
800x600	100	-	-
640x400	350	650	-
320x240	350	700	800

Taula 9: Distància màxima a la que podem transmetre amb cada amplada de banda en funció de la resolució utilitzada

Analitzant la Taula 9, veiem que el canal de 4 MHz només seria útil si utilitzem la resolució més alta i sempre que ens mantinguem per sota dels 100 m de distància. El canal de 1 MHz no és adient per a la transmissió de vídeo en temps real ja que només ens permetria utilitzar una resolució bastant dolenta. Així, la opció més bona seria utilitzar el canal de 2 MHz ja que és el més equilibrat: ens proporciona una qualitat de vídeo decent a una distància relativament bona.

Si a més es volgués realitzar una classificació d'espècies fent ús del vídeo en temps real que envia el sensor, seria necessari obtenir una resolució molt més alta de les analitzades. Això seria molt complicat per a un estàndard com el 802.11ah que no permet *throughputs* tant elevats amb les amplades de banda disponibles. Així, seria molt més interessant l'enviament d'imatges de més resolució fent ús de la càmera d'alta resolució que ofereix RaspberryPi.

Un cop analitzat el *throughput* necessari per a cada resolució, es va col·locar el el repetidor al drone per tal d'intentar obtenir el vídeo en temps real. Cal comentar que en aquest punt no s'havien fet encara les proves de *throughput*, de manera que es desconeixien la majoria de punts febles de la implementació. Així, s'obtenien *throughputs* baixos i pèrdues de senyal constants, sobretot quan el drone es posava a volar. Analitzant les causes *a posteriori*:

- Es desconeixien les baixades de *throughput* degut als moviments ràpids del drone.
- Es desconeixia la baixada de *throughput* i, en casos extrems, la pèrdua de senyal quan els mòduls es col·loquen a diferents altures.
- Es desconeixia la aparició d'interferències entre canals (inclús quan es deixen varis canals de guarda).

Podem comprovar, per tant, que les proves de camp van diferir molt de les teòriques i no es va poder obtenir un bon vídeo. Així, queda demostrat que establir una comunicació fiable amb un drone en moviment no és trivial i requereix de molta preparació i hardware dedicat. Per a millorar aquesta comunicació, tal i com s'ha proposat a les proves de *throughput*, seria necessari fer ús de millors antenes, així com millorar la orientació d'aquestes de manera que es pugui millorar el rendiment. A més, és possible que es necessiti utilitzar algun altre estàndard 802.11 en un dels dos canals per tal de combatre les interferències entre canals propers.

En conclusió, queda demostrada la diferència entre les proves realitzades en una situació ideal (visió directa o gairebé directa entre mòduls, estabilitat, etc.) de les proves realitzades en una situació més complexa, introduint a més l'element del drone en moviment. També queda demostrada la complexitat de la transmissió de vídeo en temps real i la necessitat d'oferir un *throughput* constant per tal d'obtenir un vídeo més agradable.

5 Pressupost

En aquesta secció es fa una estimació del pressupost necessari per a dur a terme aquest projecte.

Primer fem el càlcul del cost associat a recursos humans basant-nos en el cost d'un estudiant de pràctiques de l'ETSETB de 9 €/h i el temps total treballat: 23 setmanes · 20 h/setmana = 460 h. Per tant, el cost total queda: 460 h · 9 €/h = 4140 €.

Per a fer el càlcul del consum elèctric considerem un preu de 0,3 €/kWh. Necessitem tenir en compte el consum de l'ordinador portàtil i les RaspberryPi. El consum dels mòduls LoRa és negligible.

- Ordinador: Suposem que sempre s'utilitza en totes les hores de feina, que es tradueix en 460h. Un ordinador consumeix 200 W aproximadament, és a dir, 200 W · 460 h = 92 kW. Això es tradueix en un cost total: 92 kW · 0,3 €/kWh = 27,6 €.
- RaspberryPi: només s'utilitzen en les proves, que aproximem en la meitat del temps total, és a dir, 230h. Una RaspberryPi consumeix 2,5W. Considerant que utilitzem 4 RaspberryPi alhora i fent els mateixos càlculs que amb l'ordinador obtenim un cost total: 4 · 2,5 W · 230 h · 0,3 €/kWh / 10000 = 0,69 €.

Així, el cost elèctric és de: 27,6 + 0,69 = 28,29 €

També es té en compte l'amortització de l'ordinador utilitzat fent ús d'un coeficient lineal màxim anual aplicat a equips per al tractament d'informació i sistemes i programes informàtic, que és del 26%. A més, tenim en compte el PVP de l'ordinador de 750 €i suposarem un total de 160 dies treballats.

Concepte	**Cost**
Salari	4140 €
Amortització ordinador	85,48 €
Electricitat	28,29 €
RaspberryPi	6 u · 75 € = 450 €
RJ45	2 u · 7 € = 14 €
Commutador L2 D-LINK	25 €
mòdul 802.11ah (AHPI7292S)	3 u · 57 € = 171 €
mòdul 802.11ah (AHMB7292S)	5 u · 50 € = 250 €
APIMB	5 u · 12 € = 60 €
Antena	4 u · 9 € = 36 €
Connector SMA-IPEX	8 u · 6 € = 48 €
TOTAL	**5307,77 €**

Taula 10: Costos estimats del projecte

6 Impacte mediambiental

Aquest TFG es realitza en el context de l'Xprize Rainforest i dins d'un equip que participa en aquesta competició i, com a tal, és necessari el compliment de les normes proposades en aquesta. Pel que fa a les restriccions sobre l'impacte mediambiental, la competició és molt estricta i no és permeten activitats que puguin causar danys a la zona on es desenvoluparà aquesta. Així, la solució proposada ha de concordar amb aquestes normes i, per tant, cap de les proves fetes a generat cap impacte negatiu notable sobre el medi.

Tot i això, si considerem les emissions de CO2 derivades del consum d'electricitat, podem trobar de manera objectiva l'impacte mediambiental generat pel projecte. El consum d'energia calculat a l'apartat anterior és de 92 kW per a l'ordinador i de 2,3 kW per a les RaspberryPi. Així, fent ús del coeficient proporcionat per [11], obtenim un valor de 23,57 kg de CO2.

7 Conclusions

En aquest document s'ha iniciat l'avaluació de dues tecnologies (LoRa i IEEE 802.11ah) que puguin satisfer la necessitat d'establir una comunicació amb un sensor en un entorn de bosc tropical. Els resultats ens han indicat que LoRa pot cobrir distàncies molt elevades amb una alta fiabilitat en entorns rurals lliures d'interferències, però també hem pogut veure que no és un protocol dissenyat per a la transmissió de grans volums de dades, sinó per a l'enviament de pocs missatges curts diaris. En quant al 802.11ah, hem fet un anàlisi a fons dels mòduls proporcionats per tal d'obtenir resultats de *throughput* en funció del rang per a diversos escenaris i avaluant totes les amplades de banda disponibles.

S'ha treballat en un prototip de repetidor fent ús de dos canals 802.11ah i incorporant una càmera al sensor per tal de generar vídeo en temps real. Els resultats han mostrat la complexitat d'implementar la solució i han desvelat els problemes dels mòduls amb els que treballem, que han resultat ser poc fiables i susceptibles al més mínim canvi.

Aquest document només planteja un anàlisi inicial de les tecnologies, però queda molt feina per tal d'avaluar d'on provenen els problemes de potència obtinguts amb els mòduls 802.11ah. Així, és necessari esperar a la sortida de nous mòduls 802.11ah que incorporin noves millores i siguin més fiables per tal de poder fer comparatives.

Com a feina futura, també és necessari millorar la solució del repetidor. Com a primer pas, s'hauria de valorar com es poden evitar les interferències provocades per la poca separació entre canals del 802.11ah al repetidor. Començant per la millora de les antenes, fent ús d'unes més directives. A més, també seria interessant fer ús de dues tecnologies IEEE 802.11ah, una per a cada canal del repetidor. Finalment, s'hauria de proporcionar una solució més neta del repetidor, fent ús d'una sola RaspberryPi amb dos mòduls 802.11ah. Per a aconseguir-ho es podria utilitzar un dels mòduls en mode *Host*, comunicant-se amb la RaspberryPi via HSPI, i l'altre en mode *Standalone*, comunicant-se amb la RaspberryPi via UART.

Per acabar, tot i que no s'ha proposat al document, seria interessant treballar en una solució que incorpori ambdós tecnologies, LoRa i 802.11ah. És a dir, tenir LoRa com a canal de control per a la transmissió de petits missatges que sabem que arribaran amb alta probabilitat, i mantenir un canal 802.11ah per a la descàrrega de molta informació des del sensor.

Bibliografia

[1] Isabelle Sourmey. The impact of the communication technology protocol on your iot application's power consumption. https://www.saftbatteries.com/energizing-iot/impact-communication-technology-protocol-your-iot-application%E2%80%99s-power-consumption.

[2] Eric B. Lora i lorawan. https://lora.readthedocs.io/en/latest/.

[3] LoRa Alliance. Lorawan. https://lora-alliance.org/.

[4] Moko Smart. ¿cuál es la tecnología detrás de la frecuencia lora? https://www.mokosmart.com/es/lora-frequency/.

[5] Victor Banos, Muhammad Shahwaiz Afaqui, Elena Lopez, and Eduard Garcia. Throughput and range characterization of ieee 802.11ah. *IEEE Latin America Transactions*, 15(9):1621–1628, 2017.

[6] Edgar Arribas, Vincenzo Mancuso, and Vicent Cholvi. Fair cellular throughput optimization with the aid of coordinated drones. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 295–300, 2019.

[7] Lukas Marcel Schalk and Dennis Becker. The impact of multipath propagation on cooperative traffic conflict detection among drones. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, pages 1–7, 2019.

[8] Newracom. Repositori de software per a la utilització del mòdul 802.11ah. https://github.com/newracom/nrc7292_sw_pkg/.

[9] alfa. Ahm7292s. https://www.alfa.com.tw/products/ahmb7292s.

[10] alfa. Ahpi7292s. https://www.alfa.com.tw/products/ahpi7292s?variant=36473961283656.

[11] ceroco2. Cálculo de huella de carbono por consumo eléctrico. https://www.ceroco2.org/calculadoras/electrico.

Appendices

A Scripts d'anàlisi inicial de LoRa

Es mostra l'script utilitzat per a fer l'anàlisi inicial de LoRa. Es reben uns paràmetres via Serial de l'ordinador en format "BW-SF-mida" i s'envia un paquet per a ser capturat i analitzat amb CubicSDR.

```
# analisi_inicial.ino

#include "heltec.h"

//LoRa Settings
#define BAND  868E6
int SF = 8;
int BW = 250E3;
int tx_power = 10;
int packet_size;
String packet;

//Throughput test Variables
unsigned long startTime;
unsigned long T=0;


void LoRaSetup() {
  LoRa.setTxPower(tx_power);
  LoRa.setSignalBandwidth(BW);                          //7.8E3, 10.4E3, 15.6E3,
     20.8E3, 31.25E3, 41.7E3, 62.5E3, 125E3,250E3, and 500E3.
  LoRa.setSpreadingFactor(SF);                          //6 to 12. If a spreading
     factor of 6 is set, implicit header mode must be used to transmit and
     receive packets.
  delay(1000);
}


void parseNewSettings() {
  String s = Serial.readString();
  Serial.println(s);

  int cr1 = s.indexOf("-");
  int cr2 = s.indexOf("-", cr1+1);
  int cr3 = s.length()-1;

  BW = s.substring(0, cr1).toInt();
  SF = s.substring(cr1+1, cr2).toInt();
  packet_size = s.substring(cr2+1, cr3).toInt();
```

```
}


void setup() {
   Serial.begin(115200);

   while (!LoRa.begin(BAND, true)) {
    Serial.println("...");
    delay(500);
   }
   LoRaSetup();
}


void loop() {

  if (Serial.available() > 0) {
    //Get new LoRa settings from Serial
    parseNewSettings();

    //Create the new packet for the test
    packet = "";
    for (int i=0; i<packet_size; i++){
      packet+="A";
    }

    LoRaSetup();

    startTime = millis();
    LoRa.beginPacket();
    LoRa.print(packet);
    LoRa.endPacket();
    T = millis() - startTime;
    Serial.println(T);
  }

  delay(10);

}
```

B Scripts per les proves de *throughput* de LoRa

Els scripts utilitzats per a les proves de *throughput* de LoRa programats utilitzant l'Arduino IDE. Consten d'un transmissor ("lora_throughput_sender_ACK"), un receptor (("lora_throughput_receiv
i dos scripts de `python`. Un script de `python` ("receiver_to_csv.py") s'utilitza per a enviar informació al transmissor sobre els paràmetres de les proves mentre que el segon script

("sender_parameters.py") s'utilitza per a rebre els paràmetres de les proves del receptor.

```
// lora_throughput_sender_ACK.ino
#include "heltec.h"

//LoRa Settings
#define BAND  868E6
int SF = 8;
int BW = 250E3;
int tx_power = 10;
int packet_size;
String packet;

//Throughput test Variables
boolean isTestRunning = false;
int counter;
unsigned long startTime;
unsigned long T=0;
int res;


void LoRaSetup() {
   //LoRa.setFrequency(BAND);
   LoRa.setTxPower(tx_power, PA_OUTPUT_PA_BOOST_PIN); //2 to 20 for
      PA_OUTPUT_PA_BOOST_PIN, and 0 to 14 for PA_OUTPUT_RFO_PIN
   LoRa.setSignalBandwidth(BW);                       //7.8E3, 10.4E3, 15.6E3,
      20.8E3, 31.25E3, 41.7E3, 62.5E3, 125E3,250E3, and 500E3.
   LoRa.setSpreadingFactor(SF);                       //6 to 12. If a spreading
      factor of 6 is set, implicit header mode must be used to transmit and
      receive packets.
   delay(1000);
}


void parseNewSettings() {
  String s = Serial.readString();
  Serial.println(s);

  int cr1 = s.indexOf("-");
  int cr2 = s.indexOf("-", cr1+1);
  int cr3 = s.indexOf("-", cr2+1);
  int cr4 = s.length()-1;

  tx_power = s.substring(0, cr1).toInt();
  BW = s.substring(cr1+1, cr2).toInt();
  SF = s.substring(cr2+1, cr3).toInt();
  packet_size = s.substring(cr3+1, cr4).toInt();
}
```

```
void notifyRcv() {
  LoRa.beginPacket();
  LoRa.println("SET");
  LoRa.println(tx_power);
  LoRa.println(BW);
  LoRa.println(SF);
  LoRa.println(T);
  LoRa.endPacket();
}


void setup() {
   Serial.begin(115200);

   while (!LoRa.begin(BAND, true)) {
    Serial.println("...");
    delay(500);
   }
   LoRaSetup();
}


void loop() {

   //THROUGHPUT TEST
   if (isTestRunning) {
    if (counter != 100) {
      LoRa.beginPacket();
      LoRa.print(packet);
      res = LoRa.endPacket();
      counter++;
    }
    else {
      //STOP TEST
      T = millis() - startTime;
      Serial.println(T);
      isTestRunning = false;
    }
   }
   else {

     //LORA SETTINGS FROM SERIAL
     if (Serial.available() > 0) {
       //Get new LoRa settings from Serial
       parseNewSettings();

       //Create the new packet for the test
```

```
      packet = "";
      for (int i=0; i<packet_size; i++){
       packet+="A";
      }

      //Notify Receiver of the new Settings and wait for ACK
      String ack;
      do {
       Serial.println("Sending new settings");
       notifyRcv();
       LoRa.sleep();
       delay(1500);
       LoRa.receive();
       for(int i=0; i<10000; i++) {
         int packetSize = LoRa.parsePacket();
         ack = "";
         if (packetSize) {
           for (int i=0; i<packetSize; i++){
            ack += (char) LoRa.read();
           }
           Serial.println(ack);
           if (ack == "ACK") {
             break;
           }
         }
         delay(1);
       }
      } while (ack != "ACK");

      //Apply new Settings
      LoRaSetup();
      delay(3000);

      //Reset variables for a new test
      isTestRunning = true;
      counter = 0;
      startTime = millis();
    }
  }
}
```

```
# lora_throughput_receiver_ACK.ino

#include "heltec.h"

String packet ;
```

```
//LoRa SETTINGS
#define BAND  868E6
int SF = 8;
int BW = 250E3;
int tx_power = 10;

//Throughput test Settings
int counter = 0;
float T;


void LoRaSetup() {
   //LoRa.setFrequency(BAND);
   LoRa.setTxPower(tx_power, PA_OUTPUT_PA_BOOST_PIN);
   LoRa.setSignalBandwidth(BW);
   LoRa.setSpreadingFactor(SF);
   delay(1000);
}


void LoRa_rcv_newSettings() {
  delay(2000);
  LoRa.beginPacket();
  LoRa.print("ACK");
  LoRa.endPacket();

  int cr1 = packet.indexOf("\n");
  int cr2 = packet.indexOf("\n", cr1+1);
  int cr3 = packet.indexOf("\n", cr2+1);
  int cr4 = packet.indexOf("\n", cr3+1);
  int cr5 = packet.length()-1;

  tx_power = packet.substring(cr1+1, cr2).toInt();
  BW = packet.substring(cr2+1, cr3).toInt();
  SF = packet.substring(cr3+1, cr4).toInt();
  T = packet.substring(cr4+1, cr5).toFloat();

  LoRaSetup();
}



void setup() {
  Serial.begin(115200);

  while (!LoRa.begin(BAND, true)) {
    delay(1000);
  }
```

```
  LoRaSetup();
  LoRa.receive();
}


void loop() {

  int packetSize = LoRa.parsePacket();

  if (packetSize) {
    //Create packet from raw bytes
    packet = "";
    for (int i = 0; i < packetSize; i++) {
      packet += (char) LoRa.read();
    }
    Serial.println(packet);
    if (packet.substring(0,3).equals("SET")) {
      //send ACK to sender and parse new settings
      LoRa_rcv_newSettings();

      //send results via Serial
      Serial.println(counter);
      Serial.println(T);
      Serial.println(tx_power);
      Serial.println(BW);
      Serial.println(SF);
      Serial.println(LoRa.packetRssi());
      Serial.println(LoRa.packetSnr());

      LoRa.receive();
      counter = 0;
    }
    else {
      counter++;
    }

  }
}
```

```
# receiver_to_csv.py

import serial
import csv

ard = serial.Serial("COM8", 115200)
```

```
aux = False

while(True):
    #Test parameters
    N = 100        #n packets sent
    counter = 0    #n packets received

    #print all incoming packet until a SET packet
    str = ard.readline().strip().decode('utf-8')
    while True:
        if str == "SET":
            break
        else:
            packet_size=len(str)
        str=ard.readline().strip().decode('utf-8')

    for x in range(5):
        print(ard.readline().strip().decode('utf-8'))


    #Get parameters after test
    count = int(ard.readline().strip().decode('utf-8'))
    T = float(ard.readline().strip().decode('utf-8'))/1000

    #Append to CSV
    if aux:
        pps = N/T
        bps = pps * 8 * packet_size
        frequency = "868E6"
        packet_loss_ratio = (N-count)/N

        fields = [round(T,4), packet_loss_ratio, round(bps,4), round(pps,4),
            packet_size, SF, BW, tx_power, frequency, SNR, RSSI]

        with open("serial\TestResults\TestWithACK.csv", "a") as file:
            writer = csv.writer(file)
            writer.writerow(fields)

        print("Write to CSV")

    aux = True

    #Get parameters before test
    tx_power = ard.readline().strip().decode('utf-8')
    BW = ard.readline().strip().decode('utf-8')
    SF = ard.readline().strip().decode('utf-8')
    RSSI = ard.readline().strip().decode('utf-8')
    SNR = ard.readline().strip().decode('utf-8')
```

```
# sender_parameters.py

from time import sleep
import serial

arduino = serial.Serial("COM3", 115200)

#P-BW-SF-L (Potencia - Amplada de banda - Spreading Factor - Mida de paquet)
settings = [
    b'10-62500-7-1-',b'10-125000-7-1-',b'10-250000-7-1-',b'10-500000-7-1-',
    b'10-62500-7-50-',b'10-125000-7-50-',b'10-250000-7-50-',b'10-500000-7-50-',
    b'10-62500-7-150-',b'10-125000-7-150-',b'10-250000-7-150-',b'10-500000-7-150-',
    b'10-62500-7-250-',b'10-125000-7-250-',b'10-250000-7-250-',b'10-500000-7-250-',
    b'10-62500-8-1-',b'10-125000-8-1-',b'10-250000-8-1-',b'10-500000-8-1-',
    b'10-62500-8-50-',b'10-125000-8-50-',b'10-250000-8-50-',b'10-500000-8-50-',
    b'10-62500-8-150-',b'10-125000-8-150-',b'10-250000-8-150-',b'10-500000-8-150-',
    b'10-62500-8-250-',b'10-125000-8-250-',b'10-250000-8-250-',b'10-500000-8-250-',
    b'10-62500-9-1-',b'10-125000-9-1-',b'10-250000-9-1-',b'10-500000-9-1-',
    b'10-62500-10-1-',b'10-125000-10-1-',b'10-250000-10-1-',b'10-500000-10-1-',
    b'10-62500-11-1-',b'10-125000-11-1-',b'10-250000-11-1-',b'10-500000-11-1-',
    b'10-62500-12-1-',b'10-125000-12-1-',b'10-250000-12-1-',b'10-500000-12-1-',
    b'10-62500-9-50-',b'10-125000-9-50-',b'10-250000-9-50-',b'10-500000-9-50-',
    b'10-62500-10-50-',b'10-125000-10-50-',b'10-250000-10-50-',b'10-500000-10-50-',
    b'10-62500-11-50-',b'10-125000-11-50-',b'10-250000-11-50-',b'10-500000-11-50-',
    b'10-62500-12-50-',b'10-125000-12-50-',b'10-250000-12-50-',b'10-500000-12-50-',
    b'10-62500-9-150-',b'10-125000-9-150-',b'10-250000-9-150-',b'10-500000-9-150-',
    b'10-62500-10-150-',b'10-125000-10-150-',b'10-250000-10-150-',b'10-500000-10-150-',
    b'10-62500-11-150-',b'10-125000-11-150-',b'10-250000-11-150-',
    b'10-500000-11-150-',
    b'10-62500-12-150-',b'10-125000-12-150-',b'10-250000-12-150-',b'10-500000-12-150-',
    b'10-62500-9-250-',b'10-125000-9-250-',b'10-250000-9-250-',b'10-500000-9-250-',
    b'10-62500-10-250-',b'10-125000-10-250-',b'10-250000-10-250-',b'10-500000-10-250-',
    b'10-62500-11-250-',b'10-125000-11-250-',b'10-250000-11-250-',b'10-500000-11-250-',
    b'10-62500-12-250-',b'10-125000-12-250-',b'10-250000-12-250-',b'10-500000-12-250-'
]

for set in settings:
    arduino.write(bytes(set))
    str = arduino.readline().strip().decode('utf-8')
    while str != "ACK":
        print(str)
        str = arduino.readline().strip().decode('utf-8')
    print(str)
    print(arduino.readline().strip().decode('utf-8'))
    sleep(1)

arduino.close()
```

C Proves de *throughput* de LoRa

Es mostren les gràfiques obtingudes fent ús dels scripts proporcionats a l'Annex B per a amplades de banda diferents a la que ve per defecte (125 KHz).

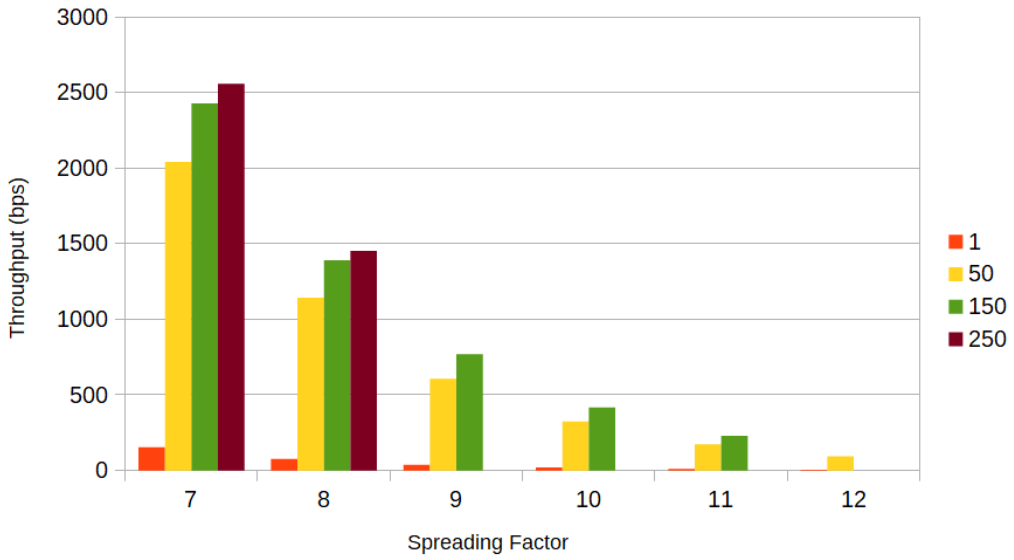


Figura 21: *Throughput* en funció de *Spreading Factor* més alts per a diferents mides de paquet amb una amplada de banda de 62,5 KHz

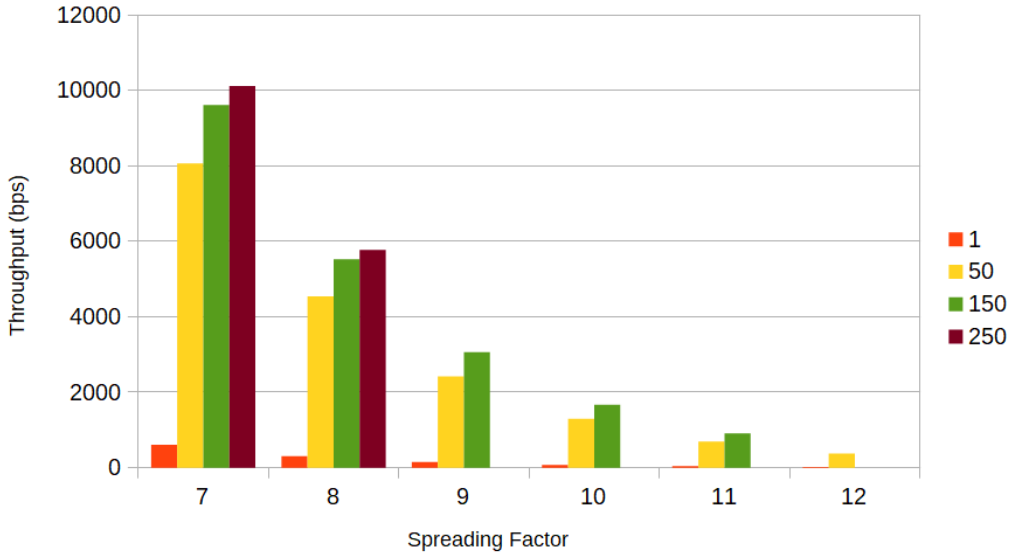


Figura 22: *Throughput* en funció de *Spreading Factor* més alts per a diferents mides de paquet amb una amplada de banda de 250 KHz

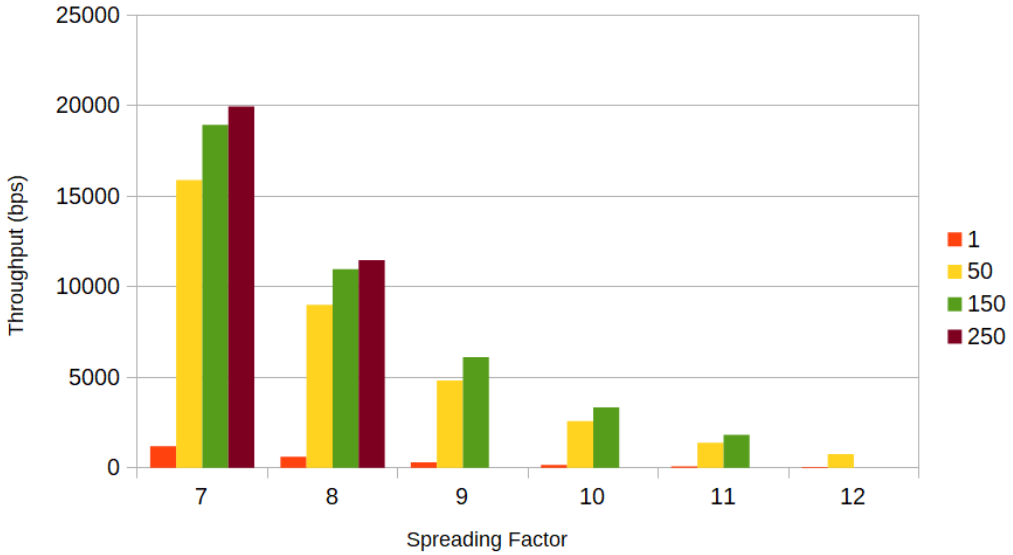


Figura 23: *Throughput* en funció de *Spreading Factor* més alts per a diferents mides de paquet amb una amplada de banda de 500 KHz

D Scripts per les proves d'*abast* de LoRa

Els scripts utilitzats per a les proves d'abast de LoRa programats utilitzant l'Arduino IDE. Consten d'un transmissor ("lora_range_test_sender.ino") i un receptor (("lora_range_test_receiver.ino")) que mostra els paquets rebuts per pantalla.

```
# lora_range_test_sender.ino

#include "heltec.h"

//LoRa Settings
#define BAND   868E6
int SF = 12;
int BW = 125000;
int tx_power = 20;

boolean sending = false;
int counter = 0;
int delay_packets = 1000;

unsigned long startTime = 0;
unsigned long toa = 0;


void setup() {
   Serial.begin(115200);
```

```
   while (!LoRa.begin(BAND, true)) {
    Serial.println("...");
    delay(500);
   }
   LoRa.setFrequency(BAND);
   LoRa.setTxPower(tx_power, PA_OUTPUT_PA_BOOST_PIN);
   LoRa.setSignalBandwidth(BW);
   LoRa.setSpreadingFactor(SF);
   delay(1000);
}


void loop() {

  if (Serial.available() > 0) {

    String serial_string = Serial.readString();
    int cr1 = serial_string.indexOf("-");
    int cr2 = serial_string.length();
    SF = serial_string.substring(0, cr1).toInt();
    BW = serial_string.substring(cr1+1, cr2).toInt(); //7.8E3, 10.4E3, 15.6E3,
        20.8E3, 31.25E3, 41.7E3, 62.5E3, 125E3,250E3, and 500E3

    if ( SF==7 || SF==8 || SF==9 || SF==10 || SF==11 || SF==12) {

      sendSettings();

      LoRa.setSpreadingFactor(SF);
      LoRa.setSignalBandwidth(BW);

      delay(2000);
    }
    else if (serial_string == "S") {
      Serial.println("STOP");
      counter = 0;
      sending = false;
    }
    else if (serial_string == "R") {
      Serial.println("Sending packets...");
      sending = true;
    }
    else if (serial_string == "T") {
      Serial.println("Starting 10 packet test");
      counter = 0;
      while (counter < 10) {
        Serial.print(counter);
        LoRa.beginPacket();
        LoRa.write((uint8_t) counter);
```

```
        LoRa.endPacket();
        counter++;
        delay(delay_packets);
      }
    }
    else if (serial_string == "D") {
      if (delay_packets == 1000) {
        delay_packets = 5;
      }
      else {
        delay_packets = 1000;
      }
      Serial.print("Setting delay between packets to ");
      Serial.println(delay_packets);
    }
    else {
      Serial.print("Invalid");
    }

  }

  if (sending) {
    Serial.print("Sending new packet ");
    Serial.println(counter);
    startTime = millis();
    LoRa.beginPacket();
    LoRa.write((uint8_t) counter);
    LoRa.endPacket();
    toa = millis() - startTime;
    Serial.println(toa);
    Serial.println("Packet sent");

    counter ++;
    if (counter == 10) {
      counter = 0;
    }
  }

  delay(delay_packets);
}


void sendSettings() {
  Serial.print("Sending settings: SF=");
  Serial.print(SF);
  Serial.print(" BW=");
  Serial.println(BW);
  LoRa.beginPacket();
```

```
  LoRa.println(SF);
  LoRa.println(BW);
  LoRa.endPacket();
}
```

```
# lora_range_test_receiver.ino

#include "heltec.h"

String packet ;
int pack;

//LoRa SETTINGS
#define BAND  868E6
int SF = 12;
int BW = 125000;
int tx_power = 20;

int counter = 0;


void rcv_newSettings() {
  int cr1 = packet.indexOf("\n");
  int cr2 = packet.length()-1;

  SF = packet.substring(0, cr1).toInt();
  BW = packet.substring(cr1+1, cr2).toInt();

  LoRa.setSpreadingFactor(SF);
  LoRa.setSignalBandwidth(BW);
}



void setup() {
  Serial.begin(115200);
  Heltec.begin(true /*DisplayEnable Enable*/, false
      /*Heltec.Heltec.Heltec.LoRa Disable*/, false /*Serial Enable*/, true
      /*PABOOST Enable*/, BAND /*long BAND*/);

  Heltec.display->init();
  Heltec.display->flipScreenVertically();
  Heltec.display->setFont(ArialMT_Plain_10);
  Heltec.display->clear();

  Heltec.display->drawString(0, 0, "INIT");
  Heltec.display->display();
```

```
  delay(2000);


  while (!LoRa.begin(BAND, true)) {
    delay(500);
  }

  LoRa.setFrequency(BAND);
  LoRa.setTxPower(tx_power, PA_OUTPUT_PA_BOOST_PIN);
  LoRa.setSignalBandwidth(BW);
  LoRa.setSpreadingFactor(SF);
  delay(1000);

  LoRa.receive();
  Serial.println("hola");
}


void loop() {

  int packetSize = LoRa.parsePacket();

  if (packetSize) {

    Heltec.display->clear();

    if (packetSize == 1) {

      pack = (int) LoRa.read();
      Heltec.display->drawString(0, 0, "Received packet");
      Heltec.display->drawString(0, 10, String(packetSize,DEC) + " bytes");
      Heltec.display->drawString(0, 20, String(pack, DEC));
      Heltec.display->drawString(0, 30, String(counter,DEC));
      Heltec.display->display();

      counter++;
    }

    else {
      packet = "";
      for (int i = 0; i < packetSize; i++) {
        packet += (char) LoRa.read();
      }

      int cr1 = packet.indexOf("\n");
      int cr2 = packet.length();
```

```
      SF = packet.substring(0, cr1).toInt();
      BW = packet.substring(cr1+1, cr2).toInt();

      if ( SF==7 || SF==8 || SF==9 || SF==10 || SF==11 || SF==12) {

        LoRa.setSpreadingFactor(SF);
        LoRa.setSignalBandwidth(BW);

        Heltec.display->drawString(0, 0, "Received new settings");
        Heltec.display->drawString(0, 10, "SF=" + String(SF,DEC) + " BW=" +
            String(BW,DEC));
        Heltec.display->display();
      }
    }
  }

  delay(5);
}
```

E Configuració RaspberryPi per a utilitzar el mòdul 802.11ah

La configuració de la RaspberryPi comprèn una sèrie de passos. Primer de tot necessitem gravar el RaspberryPi OS a una tarjeta SD; per a fer-ho podem utilitzar la eina oficial *RaspberryPi Imager*, seleccionant *Raspberry Pi OS*. Actualment, el mòdul suporta RaspberryPi 3 model B/B+ i RaspberryPi 4 model B.

Accedim a *Preferences/Raspberry Pi Configuration*, habilitem SSH, SPI i Serial Port i deshabilitem Serial Console. Ara és necessari executar una sèrie de comanes per tal d'instal·lar uns paquets i actualitzar el kernel a la última versió disponible.

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install raspberrypi-bootloader raspberrypi-kernel
$ sudo reboot
$ sudo apt install raspberrypi-kernel-headers
$ sudo apt install vim iperf3
```

Per a utilitzar el mòdul, desactivem el Wifi i Bluetooth per defecte de la RaspberryPi. Per a fer-ho afegim les següents línies a `/boot/config.txt` i quan reiniciem haurà desaparegut la interfície wlan0:

```
dtoverlay=disable-bt
dtoverlay=disable-wifi
```

Hem de desactivar també *User mode SPI* per tal de poder inserir correctament el *driver*

que ens permetrà comunicar-nos amb el mòdul. Per a fer-ho, necessitem crear manualment un fitxer amb extensió *dts*:

```
#disable-spidev-overlays.dts

/dts-v1/;
/plugin/;
/{
    compatible = "brcm,bcm2835", "brcm,bcm2708","brcm,bcm2709";

    fragment@0 {
        target = <\&spi0>;

        __overlay__ {
            status = "okay";

            spidev@0 {
                status = "disabled";
            };
            spidev@1 {
                status = "disabled";
            };
        };
    };
};
```

Un cop fet això, compilem utilitzant `dtc`(Device Tree Compiler) i col·loquem el binari a la carpeta `/boot/overlays`.

```
$ dtc -I dts -O dtb -o disable-spidev.dtbo disable-spidev-overlay.dts
$ sudo cp disable-spidev.dtbo /boot/overlays
```

Finalment, afegim una nova línia a `/boot/config.txt` i podem comprovar que, efectivament, els dispositius spi han desaparegut:

```
dtoverlay=disable-spidev
```

El següent pas és instal·lar `hostapd`, que és el *daemon* que permet generar *Access Points*. A més, necessitem el·liminar el fitxer per defecte de `wpa_supplicant` (fent un *backup*). Recordem que `wpa_supplicant` és un client de xarxes 802.11.

```
$ sudo apt install hostapd dnsmasq
$ sudo mv /etc/wpa_supplicant/wpa_supplicant.conf
    /etc/wpa_supplicant/wpa_supplicant.conf.old
```

Necessitem afegir la línia `mac80211` al final del fitxer `/etc/modules` per tal de poder carregar el *driver* més endavant. Finalment, necessitem afegir al fitxer

/etc/modrpobe.d/raspi-blacklist.conf:

```
blacklist brcmfmac
blacklist brcmutil
```

Per acabar, ja podem descarregar el repositori de GitHub per tal de compilar el *driver* i ja estarà tot llest per a començar a utilitzar el mòdul 802.11ah:

```
$ cd ~
$ git clone https://github.com/newracom/nrc7292_sw_pkg.git
$ cd nrc7292_sw_pkg/package/host/src/nrc
$ make
$ cp -b nrc.ko ~/nrc_pkg/sw/driver
```

F Scripts d'inici del mòdul IEEE 802.11ah

Aquests scripts ens permeten iniciar un mòdul en mode AP o STA un cop configurada la RaspberryPi. Cal mencionar que és necessari modificar la línia [IP] per la adreça IP que es vulgui posar.

```
#!/bin/bash
#startAP.sh

if [ "$#" -ne 1 ]; then
   echo "Specify Country"
   exit 1
fi

country=$1

sudo /home/pi/nrc_pkg/script/conf/etc/clock_config.sh

sudo killall -9 wpa_supplicant
sudo killall -9 hostapd
sudo rmmod nrc
sudo systemctl stop dhcpcd
sudo systemctl stop dnsmasq

echo "Inserting driver nrc.ko"
sudo insmod /home/pi/nrc_pkg/sw/driver/nrc.ko sudo insmod
    /home/pi/nrc_pkg/sw/driver/nrc.ko hifspeed=20000000 spi_bus_num=0
    spi_cs_num=0 spi_gpio_irq=5 spi_polling_interval=0 fw_name=uni_s1g.bin
    power_save=0 auto_ba=1 disable_cqm=1 listen_interval=1000 credit_ac_be=40
sleep 4

sudo ifconfig wlan0 up
if [ $? -ne 0 ]; then
```

```
   echo "Can't get wlan0 up";
   exit $?
fi

cli_app set txpwr 20
cli_app set gi short
cli_app set config 1 1 7

echo "starting hostapd on $country"
sudo hostapd -B /home/pi/nrc_pkg/script/conf/$country/ap_halow_open.conf

sudo ip addr add [IP]/24 dev wlan0
```

```
#!/bin/bash
#startSTA.sh

if [ "$#" -ne 1 ]; then
   echo "Specify Country"
   exit 1
fi

country=$1

sudo /home/pi/nrc_pkg/script/conf/etc/clock_config.sh

sudo killall -9 wpa_supplicant
sudo killall -9 hostapd
sudo rmmod nrc
sudo systemctl stop dhcpcd
sudo systemctl stop dnsmasq

echo "Inserting driver nrc.ko"
sudo insmod /home/pi/nrc_pkg/sw/driver/nrc.ko sudo insmod
    /home/pi/nrc_pkg/sw/driver/nrc.ko hifspeed=20000000 spi_bus_num=0
    spi_cs_num=0 spi_gpio_irq=5 spi_polling_interval=0 fw_name=uni_s1g.bin
    power_save=0 auto_ba=1 disable_cqm=1 listen_interval=1000 credit_ac_be=40
sleep 4

sudo ifconfig wlan0 up
if [ $? -ne 0 ]; then
   echo "Can't get wlan0 up";
   exit $?
fi

cli_app set txpwr 20
cli_app set gi short
cli_app set config 1 1 7
```

```
echo "starting wpa_supplicant on $country"
sudo wpa_supplicant -i wlan0 -c
    /home/pi/nrc_pkg/script/conf/$country/sta_halow_open.conf

sudo ip addr add 192.168.200.2/24 dev wlan0
```

G Configuració dels mòduls 802.11ah

G.1 AP (camp base)

Modifiquem la última línia del fitxer `startAP.sh` posant la adreça IP `192.168.150.1`. Al ser un punt d'accés, modifiquem el fitxer `~/nrc_pkg/script/conf/[país]/ap_halow_open.conf` i especifiquem el SSID que volem utilitzar. Ara executem `sudo crontab -e` i afegim `@reboot bash -lc`, així, `bootAPsetup.sh` s'executa sempre que s'inicia la RaspberryPi. Aquest és el fitxer que executarà el script d'inici especificant la banda freqüencial que es vol utilitzar. A més, configura les rutes estàtiques per arribar a la resta de xarxes.

```
#!/bin/bash
#bootAPsetup.sh
./startAP.sh US > /home/pi/startScript_log.txt 2>&1
ip r add 192.168.200.0/24 via 192.168.150.2 dev wlan0
ip r add 192.168.10.0/24 via 192.168.150.2 dev wlan0
```

Finalment, fem la configuració de "eth0" obrint el fitxer `/etc/interfaces` i afegint:

```
auto eth0
iface eth0 inet dhcp
```

G.2 Repetidor STA

Modifiquem la última línia del fitxer `startSTA.sh` posant la adreça IP `192.168.150.1`. També necessitem comentar la línia on es desactiva el DHCP i executar `sudo systemctl enable` per tal que el *daemon* DHCP s'iniciï automàticament quan s'engega la RaspberryPi. Al ser una STA, modifiquem el fitxer `~/nrc_pkg/script/conf/[país]/sta_halow_open.conf` i especifiquem el SSID del punt d'accés al que ens volem connectar. Ara executem `sudo crontab -e` i afegim `@reboot bash -lc`, així, `bootRELAYsetup.sh` s'executa sempre que s'inicia la RaspberryPi.

```
#!/bin/bash
#bootRELAYsetup.sh
./startSTA.sh US > /home/pi/startScript_log.txt 2>&1
sleep 2
sudo sysctl -p /etc/sysctl.conf
```

Al repetidor configurem "eth0" de manera que intentarà obtenir una adreça per DHCP i,

en cas que no ho aconsegueixi, s'assigna una adreça estàtica. Així aconseguim que en cas que ens volguem connectar amb el nostre ordinador a algun dels nodes, aquest proporcionarà una adreça a la interfície "eth0" i podrem solucionar problemes de connectivitat ràpidament. Per a fer-ho modifiquem `/etc/dhcpcd.conf`

```
profile static_eth0
static ip_address=192.168.10.2/24

interface eth0
fallback static_eth0
```

Finalment, afegim les rutes estàtiques al fitxer `/lib/dhcpcd/dhcpcd-hooks/40-route` (s'ha de crear en cas que no existeixi)

```
ip r add 10.42.0.0/24 via 192.168.150.1 metric 250 dev wlan0
ip r add 192.168.200.0/24 via 192.168.10.1 dev eth0
```

G.3 Repetidor AP

Modifiquem la última línia del fitxer `startAP.sh` posant la adreça IP `192.168.200.1`. També necessitem comentar la línia on es desactiva el DHCP i executar `sudo systemctl enable` per tal que el *daemon* DHCP s'iniciï automàticament quan s'engega la RaspberryPi. Al ser un AP, modifiquem el fitxer `~/nrc_pkg/script/conf/[país]/ap_halow_open.conf` i especifiquem el SSID que volem utilitzar. Ara executem `sudo crontab -e` i afegim `@reboot bash -lc`, així, `bootRELAYsetup.sh` s'executa sempre que s'inicia la RaspberryPi.

```
#!/bin/bash
#bootRELAYsetup.sh
cd /home/pi
./startAP.sh US > /home/pi/startScript_log.txt 2>&1
```

Al repetidor configurem "eth0" de manera que intentarà obtenir una adreça per DHCP i, en cas que no ho aconsegueixi, s'assigna una adreça estàtica. Així aconseguim que en cas que ens volguem connectar amb el nostre ordinador a algun dels nodes, aquest proporcionarà una adreça a la interfície "eth0" i podrem solucionar problemes de connectivitat ràpidament. Per a fer-ho modifiquem `/etc/dhcpcd.conf` i afegim:

```
profile static_eth0
static ip_address=192.168.10.1/24

interface eth0
fallback static_eth0
```

Finalment, afegim les rutes estàtiques al fitxer `/lib/dhcpcd/dhcpcd-hooks/40-route` (s'ha de crear en cas que no existeixi)

```
ip r add 10.42.0.0/24 via 192.168.10.2 metric 250 dev eth0
ip r add 192.168.150.0/24 via 192.168.10.2 dev eth0
```

G.4 STA (sensor)

Modifiquem la última línia del fitxer `startSTA.sh` posant la adreça IP `192.168.200.2`. Al ser una STA, modifiquem el fitxer `~/nrc_pkg/script/conf/[país]/sta_halow_open.conf` i especifiquem el SSID del punt d'accés al que ens volem connectar. Ara executem `sudo crontab -e` i afegim `@reboot bash -lc`, així, `bootSTAsetup.sh` s'executa sempre que s'inicia la RaspberryPi. Aquest és el fitxer que executarà el script d'inici especificant la banda freqüencial que es vol utilitzar. A més, configura les rutes estàtiques per arribar a la resta de xarxes.

```
#!/bin/bash
#bootSTAsetup.sh
cd /home/pi
./startSTA.sh US > startScript_log.txt 2>&1
sleep 2
sudo sysctl -p /etc/sysctl.conf
sudo ip r add default via 192.168.200.1
```

Per acabar fem la configuració de "eth0" obrint el fitxer `/etc/network/interfaces` i afegint:

```
auto eth0
iface eth0 inet dhcp
```

Finalment necessitem fer la configuració de l'ordinador del camp base. Simplement s'han de configurar una sèrie de rutes per arribar a totes les RaspberryPi o bé configurar una ruta per defecte a través del punt d'accés del camp base:
`ip r add default via [IP] dev [INTERFACE]`
A més, cal comptar amb un servidor DHCP a l'interfície eth0 per tal de proporcionar adreces IP a les RaspberryPi que es connectin. Si fem ús de Linux podem utilitzar `dhcpcd` i modificar el fitxer `/etc/dhcpcd/dhcpcd.conf`.

H Automatització de processos

En aquesta secció es descriuen alguns scripts que faciliten les proves de camp i eliminen la necessitat de fer tasques repetitives. Primerament tenim `pingsweep.sh` que ens permet detectar tots els usuaris connectats a la nostra xarxa. Es fa un escombrat de totes les adreces IP mitjançant la comana `ping`. Per a fer-ho ràpidament es fa ús de múltiples fils d'execució, de manera que s'executen tots els `ping` gairebé simultàneament.

```
#!/bin/bash
# pingsweep.sh
```

```
for i in {1..254};
do
   ping -c1 -W1 10.42.0.$i | grep "bytes from" | cut -d " " -f4 | cut -d ":"
       -f1 &
done
wait
```

La comunicació amb totes les RaspberryPi utilitzades es fa mitjançant SSH. Si no volem estar autentificant-nos constantment podem executar aquestes comanes per tal de generar un parell de claus ssh pública i privada i copiar la clau pública a la màquina que desitgem.

```
$ ssh-keygen -t rsa
$ ssh pi@192.168.10.2 mkdir -p .ssh
$ cat ~/.ssh/id_rsa.pub | ssh pi@192.168.10.2 'cat >>
    /home/pi/.ssh/authorized_keys'
```

El següent script, `modify_channel.sh`, està pensat per a ser usat en la solució final del repetidor. S'executa des de l'ordinador del camp base i s'encarrega de modificar els canals de totes les RaspberryPi indicant-los com a paràmetres. Així, s'elimina la necessitat de modificar-los manualment a les 4 RaspberryPi, feina que es pot fer una mica tediosa. Cal comentar que és necessari tenir la clau pública de l'ordinador des d'on s'executa l'script a totes les màquines que conformen la solució del repetidor, tal i com s'ha explicat a l'script anterior.

```
#!/bin/bash
# modify_channel.sh

#check number of arguments given
if [ "$#" -ne 2 ]; then
   echo -e "Usage:\n./modify_country.sh [country1] [country2]"
   echo -e "\tcountry1: 1st link country (STA <-> Relay AP)"
   echo -e "\tcountry2: 2nd link country (Relay STA <-> AP)"
   exit 1
fi

new_country=$1
new_country_second=$2

#check if countries are valid
countries=("AU" "EU" "US" "KR" "TW" "JP")
if [[ ! " ${countries[*]} " =~ " ${new_country} " ]] || [[ ! " ${countries[*]}
    " =~ " ${new_country_second} " ]]; then
   echo "Invalid country $new_country"
   exit 1
fi
```

```
echo "Config for STA"
old_country=$(ssh pi@10.42.0.60 'cat bootSTAsetup.sh | grep start.py | cut -d
    " " -f4')
echo -n "old_country=$(echo $old_country), new_country1="
ssh pi@10.42.0.60 "sed -i 's/$(echo $old_country)/$(echo $new_country)/'
    bootSTAsetup.sh; cat bootSTAsetup.sh | grep start.py | cut -d ' ' -f4"

echo "Config for Relay AP"
echo -n "old_country=$(echo $old_country), new_country1="
ssh pi@192.168.150.1 "sed -i 's/$(echo $old_country)/$(echo $new_country)/'
    bootRELAYsetup.sh; cat bootRELAYsetup.sh | grep start.py | cut -d ' ' -f4"

echo "Config for Relay STA"
old_country=$(ssh pi@192.168.200.3 'cat bootRELAYsetup.sh | grep start.py |
    cut -d " " -f4')
echo -n "old_country=$(echo $old_country), new_country_second="
ssh pi@192.168.200.3 "sed -i 's/$(echo $old_country)/$(echo
    $new_country_second)/' bootRELAYsetup.sh; cat bootRELAYsetup.sh | grep
    start.py | cut -d ' ' -f4"

echo "Config for AP"
echo -n "old_country=$(echo $old_country), new_country_second="
ssh pi@192.168.200.1 "sed -i 's/$(echo $old_country)/$(echo
    $new_country_second)/' bootAPsetup.sh; cat bootAPsetup.sh | grep start.py
    | cut -d ' ' -f4"
```

Finalment tenim `reboot_all.sh`. Tal i com indica el seu nom, s'encarrega de reiniciar totes les RaspberryPi de la solució final del repetidor.

```
#!/bin/bash

ssh pi@192.168.200.1 "sudo reboot now"
ssh pi@192.168.20.2 "sudo reboot now"
ssh pi@192.168.150.1 "sudo reboot now"
ssh pi@10.42.0.60 "sudo reboot now"
```