Implementació d'una Xarxa de Sensors per a la Monitorització de Paràmetres Ferroviaris.

Autor: Francesc Guitart Bravo

Director: Jordi Casademont i Serra

27 de juliol del 2009

Universitat Politècnica de Catalunya - Escola Tècnica i Superior d'Enginyeria de Telecomunicacions.

Índex de contingut

Índex de figures

Índex de taules

Prefaci

Aquest projecte neix al juny de 2008, gràcies a que en Jordi Casademont em va proposar col·laborar en el desenvolupament d'una xarxa de sensors en uns trens.

Durant un any he pogut estar treballant desenvolupant un projecte per una de les empreses més importants del món en el sector ferroviari, de ben segur que totes les experiències viscudes enriquiran el meu futur professional en bon grau.

Des d'aquestes línies m'agradaria agrair l'oportunitat a en Jordi Casademont i l'ajuda a tots els companys, en especial a en Jordi Vilaseca i la Soledad Méndez, així com la confiança dipositada en mi.

En especial també agrair el suport als pares, l'Enric i la Bea, sense ells res d'això hagués estat possible.

1. Introducció

La història ferroviària catalana comença el dia 8 del mes d'Octubre de l'any 1848. En aquest primer trajecte, un gran tren metàl·lic viatjà des de Barcelona a Mataró per inaugurar la primera via fèrria de Catalunya i de tot l'estat Espanyol. De ben segur, els 400 passatgers d'aquest primer viatge gaudiren, confortablement distribuïts en 10 cotxes, mentre elucubraven sobre l'impacte socioeconòmic i cultural que s'aveïnava.

Però poc podien imaginar l'evolució d'aquell esdeveniment: 112 anys després es començarien a fer proves per a posar en circulació un tren que superava els 200 km/h amb una locomotora TALGO. Uns anys després, al 1992 s'inaugurava la primera línia d'alta velocitat, amb un ample de via 1435mm i alhora es van començar a veure les primeres unitats de la sèrie 100 de RENFE, un tren que podia arribar als 300 km/h i que estrenava l'alta velocitat a España amb un recorregut entre Madrid i Sevilla.

Avui en dia, un tren d'alta velocitat es pot considerar un dels productes més exclusius de l'alta enginyeria que arriba al ciutadà. Un sol tren pot arribar a atreure grans aportacions des de gairebé qualsevol àmbit de l'enginyeria, ja sigui industrial, aeronàutica, telecomunicacions, informàtica, etc.

De cara a l'usuari, ha ajudat a reduir distàncies i ha esdevingut una gran infraestructura per a unir territoris a un cost raonable i oferint un confort i seguretat inigualable en altres mitjans de transport de masses, convertint-se així en una bona alternativa a l'avió i en concret al Pont Aeri.

1.1. Motivació del projecte

CBM (Condition Based Maintenance) és un procés de manteniment d'equipament en el qual es monitoritza la temperatura i les vibracions de certs components per facilitar la detecció prèvia del mal funcionament o fallida [**1**]. L'equipament pot ser monitoritzat mitjançant mètodes sofisticats de sensorització o es pot fer mitjançant la percepció humana. El CBM automàtic proporciona una monitorització constant que és recollida per algun processador de dades, i que dóna una alarma quan certs paràmetres a analitzar superen els valors estipulats. Aquest tipus de sistema també es poden anomenar: *Condition Monitoring* (CM)*, Predictive Maintenance* (PdM) *i Condition Based Maintenance* (CBM). Se sol utilitzar en mecanismes que tenen un funcionament continu, i on les pèrdues per la fallida d'algun component són molt altes. La monitorització contínua permet un manteniment proactiu.

Les rodes dels vehicles que circulen sobre rails estan sotmeses a grans esforços. La conseqüència són els indicis de desgast que poden ser, entre altres coses, la formació d'ovalacions.

Fallides d'aquest tipus, així com àrees planes, són la causa de que es produeixin forces molt fortes, d'intervals de temps molt reduïts, a la transmissió de força de les rodes motrius al rail, que normalment és homogènia. Les conseqüències són:

- Un esforç molt gran del vehicle i de la via fèrria, i en conseqüència, costos de manteniment elevat.
- Una carga gran, en l'interior, de soroll i vibracions.
- Reducció del confort en el viatge.
- Avaries consecutives en el vehicle i la via fèrria.
- Descarrilaments a causa de danys a les rodes.

A més a més, tot el sobreescalfament dels rodaments en els vehicles que circulen pel ferrocarril és un problema comú. Una sola

roda que tingui una temperatura per sobre del normal pot fer que tot el sistema falli, i fins i tot fer que el tren descarrili.

Per tant, es fa necessari un sistema de monitorització que pot prevenir les fallides i facilitar la monitorització per al manteniment d'un tren. El problema és que el cost de sensoritzar totes les rodes pot arribar a ser elevat i complicat pel que fa a la instal·lació del cablatge. A banda de tenir algunes zones inaccessibles en el tren, la recollida de dades, molts cops, no es pot fer freqüentment per part de l'empresa que fa el manteniment. El problema s'agreuja quan els trens ja estan construïts, i per tant el cablejat és impossible.

És evident, a més, que la monitorització d'un tren té un inconvenient intrínsec; el tren es mou (a grans velocitats), i la recollida d'informació només podria fer-se quan s'atura. Molts cops, es necessari saber les condicions en les que es troba el tren abans d'arribar a les estacions de destí, ja que així s'aconsegueix un manteniment més acurat, per tant s'haurien de recol·lectar les dades mentre el vehicle està en servei.

Sembla doncs que es fa inevitable l'ús d'una tecnologia inalàmbrica adient per la connexió d'un terminal mòbil, a un servidor on bolcar les dades.

El present projecte, se centra en aportar una solució per a la sensorització de certs element propis dels trens d'alta velocitat, la finalitat de la qual es la construcció d'un sistema de CBM i una posterior connexió amb un gateway d'enllaç amb un servidor remot en terra ferma.

1.2. Objectius del projecte

Aquest projecte forma part d'un sistema més gran, que aportarà una solució real per a la monitorització d'algunes parts en un tren en funcionament (Figura 1).

Es vol desenvolupar un sistema de monitorització de certes peces del tren, de manera que es puguin obtenir valors de temperatura y vibració. Per aconseguir-ho, una xarxa de sensors ha d'obtenir les els paràmetres necessaris de les peces. Aquests sensors, enviaran les dades a un punt central de recol·lecció anomenat *gateway.*

El gateway és l'encarregat d'obtenir dades i tractar-les, així com d'oferir una interfície a l'usuari per tal de configurar paràmetres a la xarxa de sensors. El gateway, proveït de connexió mòbil sense fils, enviarà les dades a un servidor central juntament amb la posició geogràfica obtinguda d'un GPS.

Finalment, el servidor de fitxers oferirà les dades del tren, junt amb el posicionament per tal de poder estudiar les dades de la monitorització.

El resultat final de cara a l'usuari serà una aplicació web, on es situarà el tren en un mapa. Mitjançant la interfície web es podran consultar els valors monitoritzats per la xarxa de sensors, tant actuals com històrics. A més a més, es podrà accedir a l'element gateway per canviar els valors de configuració.

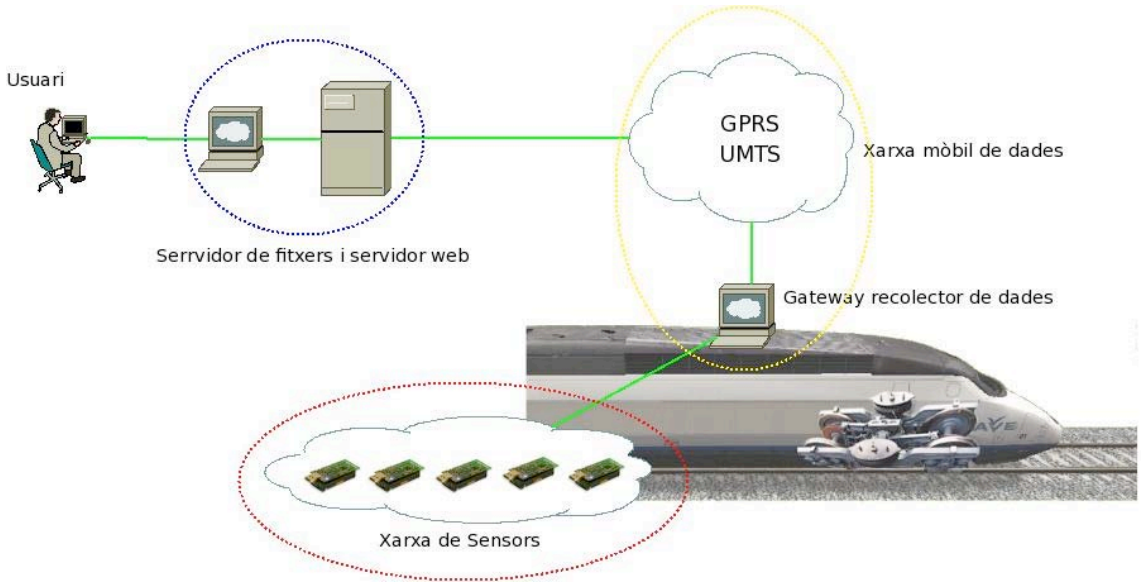


Figura 1 – Diagrama del projecte general

Com podem veure a la Figura 1, el projecte es divideix en tres grans parts. A continuació enumerarem els objectius específics de cada part:

- **Xarxa de sensors:**
    1. Obtenir dades per a la monitorització d'elements del tren.
    2. Enviament de dades fins a un punt de recol·lecció.
    3. Configuració dels paràmetres d'obtenció de les dades.
- **Gateway recol·lector de dades:**
    1. Obtenció i tractament de dades de la xarxa de sensors.
    2. Enviament de la informació via xarxa mòbil de dades.
    3. Configuració de la xarxa de sensors i paràmetres de la xarxa mòbil via web.
    4. Posicionament geogràfic del tren.
- **Servidor de fitxers i servidor web:**
    1. Emmagatzemament de les dades provinents del Gateway.
    2. Visualització de les dades obtingudes per la xarxa de sensors i la posició geogràfica via web.

En aquesta part fem front a la diagnosi, avaluació, disseny i implementació de la xarxa de sensors que actuarà com a

recol·lectora i transportadora de les dades a mesurar en les parts a monitoritzar. El present document s'organitza tal i com segueix:

Una primera part, introduirà el lector dins les tecnologies implicades en el projecte, fent una breu descripció i una anàlisi del perquè de la seva utilització enfront a altres solucions possibles.

Un cop vistes les motivacions i els objectius, es presenta un estudi de viabilitat fet durant la implementació del projecte on es veuen els tests de compatibilitat electromagnètica i proves de cobertura per justificar la topologia de xarxa que es construirà.

En les següents seccions s'aborda el desenvolupament del projecte, on es remarquen les característiques de les aplicacions que s'han codificat i on s'explica l'estructura que tenen i les seves funcionalitats.

Finalment, s'exposen unes proves amb els respectius resultats per justificar certes configuracions. El document acaba resumint algunes conclusions derivades de l'elaboració de la memòria i s'esmenten algunes línies de treball futur.

2. Tecnologies implicades

Les xarxes de sensors o WSN (*Wireless Sensor Networks*) són agrupacions de sensors sense fils que controlen i en registren condicions físiques o ambientals molt diverses, i basen la gestió de les dades en una política col·laborativa, tant per a la distribució com per a la cobertura del territori.

Aquest tipus de xarxes han sofert un gran impuls en els últims anys gràcies als grans avenços en els sistemes MEMS (*micro-electro-mechanical systems*), tecnologies de comunicacions, i electrònica digital. Aquests avenços han fet que es puguin produir uns sensors inalàmbrics de dimensions reduïdes, amb una capacitat de computació prou bona per fer certes tasques, i tot això ho han aconseguit a baix cost.

Per tant, una xarxa de sensors es pot construir amb un nombre de nodes (o sensors) molt elevat sense repercutir massa en el cost. Aquests nodes poden distribuir-se per tota l'àrea del fenomen a sensoritzar sense haver de pensar en una arquitectura específica, i gràcies a la seva capacitat de computació, poden utilitzar mètodes cooperatius per a entregar les dades des d'un punt fins a un altre.

2.1. Tipus de sensors

Un node sensor (altrament anomenat *mote*[1]) és l'element principal en una Xarxa de Sensors Inalàmbrica (WSN, *Wireless Sensor Network*) que està dotat de capacitats computadores, de comunicació amb altres nodes i obtenció de dades.

[1] El projecte Smartdust pretenia fer una mena de "pols intel·ligent", d'aquí la paraula *mote* (unitat atómica formadora de la pols).

La història del desenvolupament d'aquests sensors es remunta al 1998. L'objectiu del projecte Smartdust era crear un dispositiu sensor d'un centímetre cúbic. Malgrat que aquest projecte va finalitzar poc després d'iniciar-se, va donar lloc a molts altres projectes similars. La tecnologia ha evolucionat cap a dispositius amb petites prestacions i poc consum.

En les següents subseccions descriurem els principals sensors utilitzats durant el projecte.

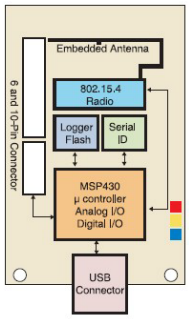
2.1.1. Telosb

Els sensors TelosB estan desenvolupats per l'empresa Crossbow[2], especialment el model TPR2400, és una plataforma de codi obert dissenyat per a la experimentació per a la comunitat de desenvolupadors [**2**]. Aquest model conté tots els elements essencials per als estudis de laboratori incloent:

- Connector USB per a la programació.
- Ràdio IEEE 802.15.4 amb antena integrada.
- Alternativament en el mode TPR2420 una unitat sensora.

[2] La companyia Crossbow Technology ha estat a la avantguarda de la tecnologia de sensors des de fa més d'una dècada. Ha venut centenars de milers de sensors intel·ligents a tot el món. Avui dia és el líder del sector de les xarxes de sensors inalàmbriques.

(a)



(b)

Figura 2 - (a) Sensor TelosB (Model TPR24200); (b) Arquitectura TelosB

Les propietats de la unitat TPR24200 són:

- Transceptor radiofreqüència compatible amb 802.15.4/ZigBee.
- Compatible amb les bandes de freqüència de 2.4 a 2.4835 GHz.
- Transmissió a 250kbps.
- Antena integrada.
- Microcontrolador MSP430 a 8MHz amb 10kB de memòria RAM.
- Prestacions de baix consum.
- Memòria flash externa de 1MB.
- Programació i obtenció de dades a través d'un connector USB.
- Compatibilitat amb TinyOS 1.1.10 o superior.

S'alimenta tant per dues piles tipus AA com per la interfície USB. També proporciona interfícies de connexió amb altres dispositius, té dos connectors d'expansió per a poder controlar sensors analògics, perifèrics digitals i displays LCD.

2.1.2. Micaz

El sensor MICAz, desenvolupat per l'empresa Crossbow, és un sensor compatible amb l'IEEE 802.15.4 utilitzat generalment per a la implementació de xarxes de sensors de baixa potència[**3**]. És un model que pertany a la família de sensors MICA i que fou tret al mercat aportant una millora de prestacions respecte als models precedents. Aquestes prestacions són:

- Transceptor radiofreqüència compatible amb IEEE 802.15.4/ZigBee
- Freqüències de treball: de 2.4 a 2.4835 GHz (banda compatible ISM)
- Taxa de transmissió: 250 kbps.
- Compatible amb TinyOS 1.1.7 i versions superiors.
- Interfície Plug and Play amb altres plataformes sensors Crossbow.

Figura 3 - Sensor MICAz

2.2. Descripció IEEE 802.15.4

La necessitat de mobilitat per part dels usuaris ha forçat l'evolució de les tecnologies de comunicació, cap al camp de les tecnologies inalàmbriques. Exemples com els estàndards GSM, IS-95, demostren com l'usuari pot fer ús de les comunicacions

sense deixar de moure's ni haver de suportar els cables que el lliguen a l'escomesa d'accés a la xarxa.

A mitjans dels anys 80, va ser el torn de les dades. L'aparició de l'estàndard 802.11 va donar lloc a les primeres WLANs (Wireless Local Area Network).

A principis de la dècada en la que estem, les comunicacions inalàmbriques donaren un pas més. L'aparició de les WPANs (Wireless Personal Area Networks) donava un altre sentit a la mobilitat d'equipament de xarxa de baix cost, baix consum, mida reduïda i petit abast, ja que s'obria la porta de comunicar petits dispositius en rangs de distància petits. Les WPAN foren concebudes com a petites xarxes utilitzades per a la comunicació de dispositius als voltants d'una persona. Aquest esforços es van dur a terme per mitjà del grup de treball 802.15 de l'IEEE, que va classificar les WPAN en tres grups diferenciats segons taxa de transmissió, consum de bateria i qualitat de servei (QoS):

- 802.15.3 (High Data Rate WPAN): destinat per aplicacions multimèdia que requereixen QoS.
- 802.15.1/Bluetooth (Medium Data Rate WPAN): està pensat per a dispositius com PDAs, telèfons mòbils, i que requereixen qualitat de servei per a mantenir comunicacions de veu.
- 802.15.4 (Low Rate WPAN): està destinat per a l'ús d'aplicacions industrials, mèdiques i residencials amb un consum molt baix.

Tot sovint se sol parlar (i es tendeix a confondre) de la tecnologia ZigBee quan parlem de xarxes de sensors. Ens equivocaríem si igualéssim ZigBee a 802.15.4, ja que ZigBee és una tecnologia sorgida de la ZiggBee Alliance, i és el nom

comercial d'una tecnologia encarregada dels protocols de comunicació per a aplicacions d'automàtica i control remot.

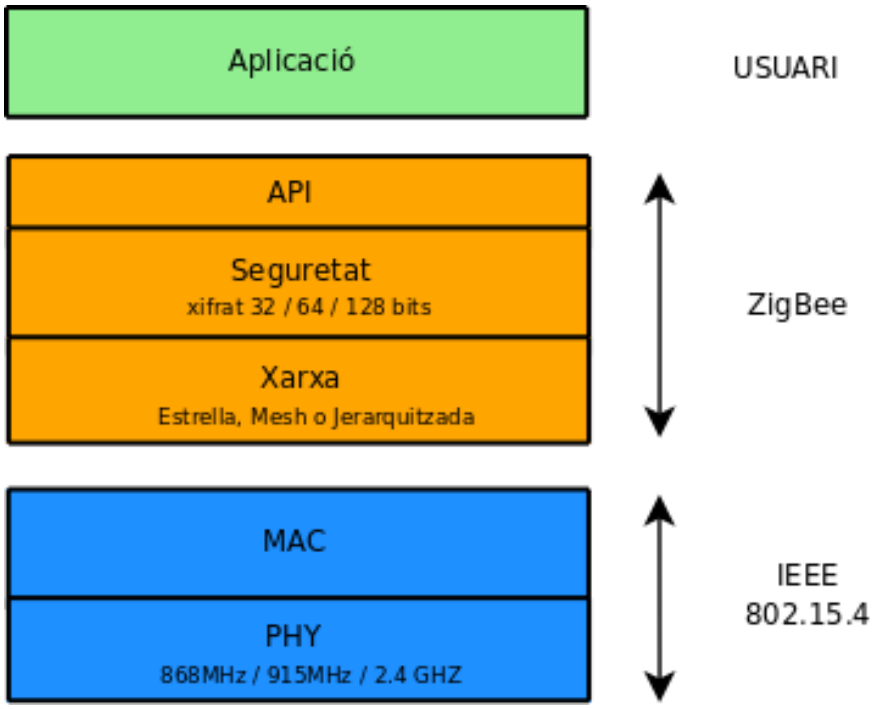


Figura 4 – Pila dels protocols ZigBee/IEEE 802.15.4

En el desenvolupament d'aquest projecte es descarta l'ús d'aquesta tecnologia, i es fa una aposta per als estàndards oberts i de lliure distribució. Per tant només es farà ús de la capa física (PHY) i la capa d'accés al medi (MAC) que proporciona l'estàndard IEEE 802.15.4. Tot seguit es farà una breu introducció a aquestes dues capes ja que ens situaran al punt de partida del projecte.

L'estàndard original 802.15.4 publicat en l'any 2003, va adoptar una capa física de banda ampla i una tècnica DSSS (Direct Sequence Spread Spectrum). L'estàndard original va proporcionar especificacions per operar en tres bandes de freqüència diferents:

- 868 MHz (disponible a Europa).
- 915 MHz (disponible als Estats Units d'Amèrica).
- 2400 MHz (disponible a tot el món).

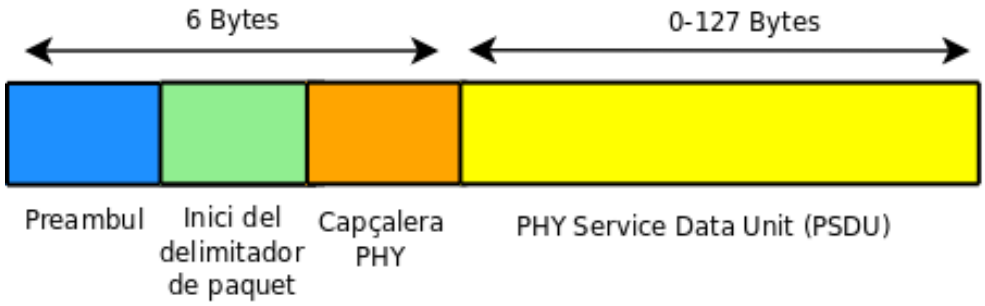


Figura 5 – Trama 802.15.4

Per tal de permetre la coexistència de diferents xarxes en el mateix espai, es va adoptar una tècnica FDM (Frequency Division Multiplexing), dividint cada banda en diferents canals; l'estratègia d'accés en cada xarxa o canal es basa en TDMA (Time Division Multiplexing Access). A continuació en la figura es mostra l'esquema de canalització[**4**]:

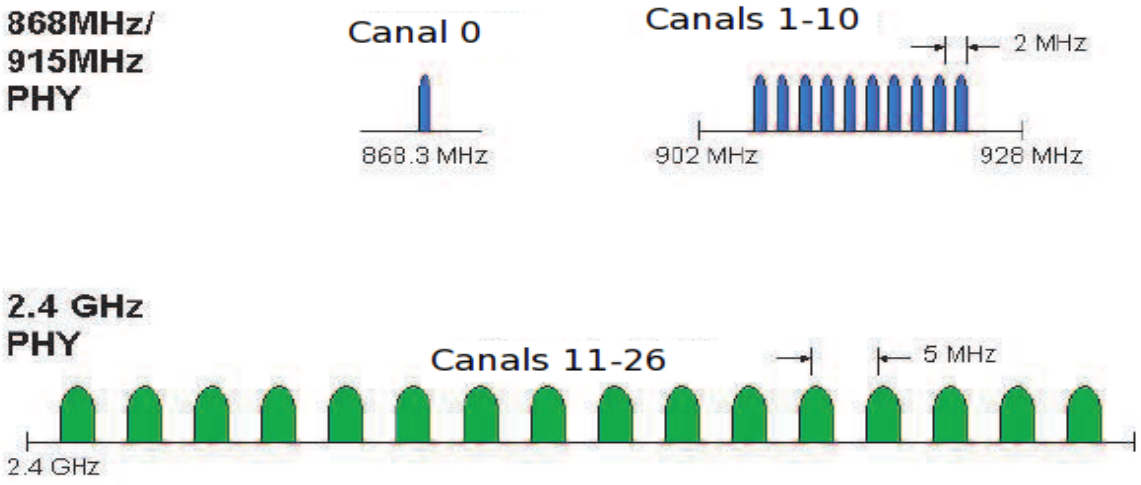


Figura 6 – Distribució de canals del protocol 802.15.4

- 1 canal (canal 0) definit en la banda dels 868 MHz.
- 10 canals (canals del 1 al 10) allotjats en la banda del 915 MHz , amb 2 MHz d'espai entre canals.
- 16 canals (canals del 11 al 26) definits en la banda dels 2.4 GHz, amb 5Mhz d'espai entre canals.

Els canals definits en les primeres dues bandes van ser pensats per a l'ús de taxes de transmissió molt baixes, amb

transmissions d'usuari de 20 kbps i 40 kbps per a les bandes de 868 MHz i 915 MHz respectivament. Per altra banda la part dels 2.4 GHz permet taxes de 250kbps per canal, gràcies al major ample de banda disposat per cada canal.

L'estàndard IEEE defineix dos tipus de dispositius:

- **Dispositiu amb funcionalitat complerta FFD (*Full Function Device*):**
  - Pot funcionar en qualsevol topologia.
  - Pot actuar com a coordinador de la xarxa.
  - Pot comunicar-se amb qualsevol altre dispositiu.
- **Dispositiu amb funcionalitats reduïdes RFD (Reduced Function Device):**
  - Limitat a formar part d'una topologia d'estrella (referència a la part més endavant on hi haurà el tema de topologies)
  - No pot actuar com a coordinador.
  - Només es comunica amb el coordinador de la xarxa.
  - Duu a terme aplicacions molt simples.

  Per a l'accés al medi es fa servir un mecanisme de CSMA-CA (Carrier Sense Multiple Acces with Collision Avoidance) que pot ser ranurat o no.

  La capa MAC de l'IEEE 802.15.4 defineix quatre estructures de trama:

  - **Beacon:** utilitzat per el coordinador per transmetre trames balisa.
  - **Data frame:** utilitzat per transmetre informació.
  - **Acknowledgment frame:** utilitzat per a la confirmació de dades.

- **MAC command frame:** utilitzat per controlar les transmissions entre entitats a nivell MAC.

Canal	Freqüència central (MHz)	Ample de banda (MHz)
0	499.2	499.2
1	3494.4	499.2
2	3993.6	499.2
3	4492.8	499.2
4	3993.6	1331.2
5	6489.6	499.2
6	6988.8	499.2
7	6489.6	1081.6
8	7488.0	499.2
9	7987.2	499.2
10	8486.4	499.2
11	7987.3	499.2
12	8985.6	499.2
13	9484.8	499.2
14	9984.0	499.2
15	9484.8	1355

Taula 1 - Distribució de canals en l'espectre electromagnètic.

2.3. Xarxes Multisalt

Els protocols d'encaminament que funcionen en un entorn distribuït i la missió dels quals es transportar la informació fins a un altre punt allunyat (fora de la cobertura del destí) mitjançant els altres veïns, s'anomenen protocols d'encaminament multisalt.

La idea bàsica del multisalt és augmentar la cobertura de cada node mentre es baixa la potència d'emissió. Pot semblar un contrasentit però, paradigmàticament, una xarxa formada per nodes amb una cobertura real que no supera les desenes de centímetres poden arribar a comunicar-se amb nodes centenars de metres més enllà.

Les primitives bàsiques de les xarxes de sensors són [**5**]:

- **Encaminament:** Com ja hem parlat, és l'aspecte bàsic de les xarxes de sensors. Tracta el problema de com fer arribar les dades d'un punt fins a un altre. Més endavant veurem diferents protocols que ens proporcionen aquesta connectivitat, però s'ha de veure que l'encaminament depenent en tot moment del tràfic que es mou per la xarxa i l'elecció d'un bon protocol recau en un compromís entre diferents paràmetres com: mobilitat, *thorughput*, escalabilitat, etc.
- **Obtenció de les dades:** Usualment es pot veure una xarxa d'aquest tipus com una base de dades, ja que quasi bé tota la interacció que farem amb ella tindrà a veure amb la recol·lecta de dades que varien en el temps. Tanmateix, molts cops es poden aconseguir entorns més complicats que simples peticions, i

aprofitant la capacitat computadora dels nodes atorgar certa intel·ligència a la xarxa.

- **Descobriment de la xarxa:** És un servei molt important ja que tot sovint els nodes es doten amb capacitats autoconfiguradores que els administradors de la xarxa no poden resoldre. Per exemple, en la col·locació d'un node en una posició aleatòria, el node ha descobrir coses com: següent salt, node destí, etc.

Dins de les xarxes multisalt també podem veure diferents aproximacions segons les propietats centrals en les que ens fixem, aquestes poden ser: nodes, dades i posició.

- **Node-cèntric:** És l'aproximació més tradicional, la que s'ha traduït des d'Internet on tots els serveis estan orientats a una adreça i l'encaminament es fa en base aquesta adreça.
- **Data-cèntric:** Aquesta aproximació va ser pensada per a respondre la primitiva "Aconseguir les dades que compleixen certa condició". A diferència d'una estructura basada en nodes, aquí el que realment importa és la informació, per tant s'hauran d'identificar els nodes els quals compleixin les condicions i obtenir-ne les dades.
- **Position-cèntric:** Com hem vist anteriorment pot ser que els nodes siguin el que menys ens importa i vulguem respondre altres tipus de primitives. Aquest és el cas de l'obtenció de dades segons posició, és a dir, obtenir les dades d'una zona geogràfica, per exemple la humitat d'un hort. En aquest sentit ens interessarà encaminar geogràficament [**6**] la petició cap una certa regió i que tots els nodes dins aquesta posició ens retornin la informació.

També podem classificar les xarxes multisalt depenent de la seva topologia: poden variar des d'una distribució totalment aleatòria, fins arribar a xarxes molt més estructurades.

- **Topologia d'estrella d'un sol salt (*single hop*):** És la més simple que podem obtenir. Simplement cada node es comunica directament i amb un salt amb un altre node anomenat *sink,* que serà el node destí de tots els nodes de la xarxa. És una xarxa pensada per a petites aplicacions ja que la seva escalabilitat és mínima i els nodes més allunyats del sink tenen una cobertura molt baixa.
- **Multisalt *mesh* i *grid:*** Per a aplicacions més grans i amb més prestacions, tot sovint es requereix de xarxes estructurades aleatòriament o en forma de graella utilitzant el multisalt.
- **Multisalt jerarquitzat:** També es donen tot sovint arquitectures molt més elaborades que requereixen que els nodes adoptin diferents maneres d'actuar, per tant parlem de xarxes heterogènies on fins i tot es poden introduir diferents tipus de sensors que realitzen diferents tasques. Es tracta de sensors que formen diferents xarxes, cadascuna amb un "cap" o *cluster-head* que tot sovint seran nodes amb més potència o més capacitat energètica. Aquestes sub-xarxes enviaran tota la informació al seu cluster, i els clusters mitjançant una xarxa troncal es comunicaran amb un *gateway* o *sink.*

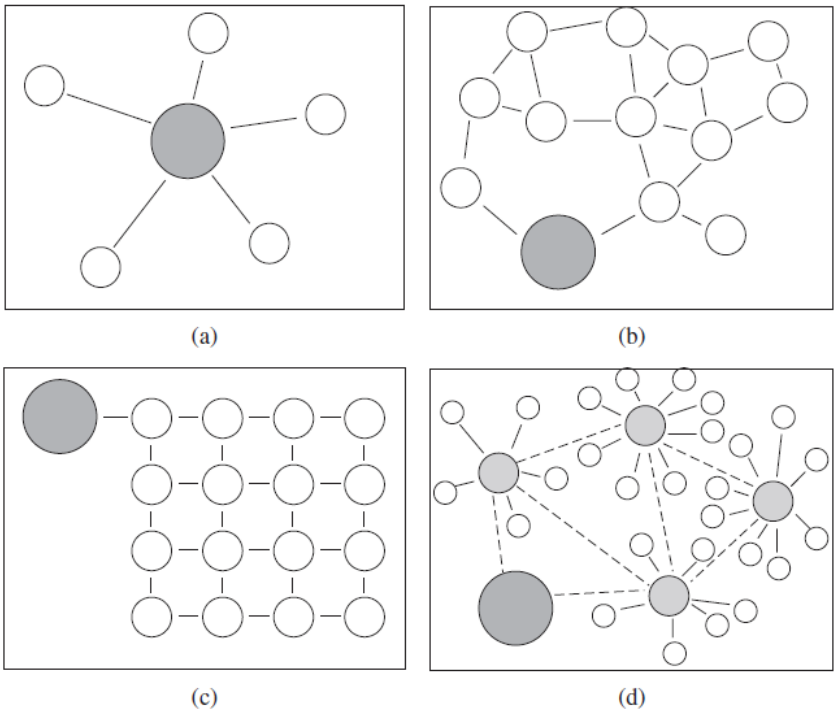


Figura 7 - Topologies: (a) En forma d'estrella d'un sol salt. (b)Mesh. (c) Grid o graella. (d) Jerarquitzada.

2.4. Protocols d'encaminament

2.4.1. AODV

AODV (Adhoc On Demand Distance Vector) és un protocol d'encaminament dissenyat per a xarxes adhoc mòbils[7]. AODV està pensat tant per l'encaminament de dades en mode unicast i multicast. És un algorisme que construeix les rutes sols quan són requerides per l'origen de les dades. Manté aquestes rutes mentre aquestes són necessitades per els nodes origen, quan no ho són s'esborren.

AODV consta d'un mecanisme d'intercanvi de missatges *"route request / route reply"* . Quan un origen necessita una ruta fins a un destí per el qual encara no té cap ruta, aquest envia un missatge de petició de ruta *"Route Request RREQ"* a *broadcast* a tota la xarxa. Els nodes que reben aquest paquet actualitzen la seva informació per al node origen i estableixen apuntadors en la taula de rutes cap al node

origen. A més, el node desa l'adreça, el número de seqüència, ID de boradcast. El RREQ també conte el número de seqüència més recent per al destí que el node origen té. Un node que rep un RREQ pot enviar de tornada un *Route Reply (RREP)* indicant el camí cap a l'origen, tant si és el node destí o si en la seva taula de rutes hi ha una entrada amb el número de seqüència igual o més gran del que conté el missatge RREQ. En aquest cas, envia un missatge RREP de tornada a l'origen. Altrament reenvia un altre cop a tots els nodes el missatge RREQ. El node descartarà els RREQ que ja ha tractat.

Com el RREP es propaga enrere cap a l'origen, els nodes actualitzaran els punters cap a la destinació. Un cop l'origen rep el RREP, es poden començar a enviar paquets a la destinació. Si l'origen rep un RREP que conté un número de seqüència més gran o conté el mateix número de seqüència amb un *hopcount*, pot actualitzar la informació d'encaminament per aquesta destinació i utilitzar una millor ruta.

Sempre que la ruta estigui activa, es continuarà mantenint. Una ruta es considera activa sempre que hi hagi paquets d'informació viatjant des de l'origen fins al destí per aquest camí. Un cop l'origen para d'enviar paquets, hi haurà uns temporitzadors per als enllaços, que un cop esgotats eliminaran l'entrada a la taula de rutes. Si un enllaç es trenca, es propaga un missatge d'error (*Route Error RERR),* a l'origen per informar que l'enllaç està trencat i que per tant, no es pot arribar al destí. Després de rebre el RERR, si l'origen encara vol conservar la ruta, pot reiniciar el procés de descobriment de ruta.

2.4.2. NST-AODV

Not So Tiny-AODV [**8**] és una modificació del protocol d'encaminament sota demanda AODV que intenta proporcionar certs avantatges addicionals a TinyAODV[3]:

1. S'activa un mecanisme de LLN (Link Layer Notification) per a descobrir el trencament d'enllaços. S'assumeix que el protocol pot operar en xarxes amb topologies dinàmiques.

2. El paquet que activa el descobriment de ruta no es descarta, sinó que utilitza la ruta activada recentment per arribar al destí.

3. Després d'una fallida en l'enviament a nivell d'enllaç, es fan dos intents més per tal d'intentar arribar al següent salt abans de donar l'enllaç per perdut.

4. Si un node considera que l'enllaç s'ha trencat després de les tres retransmissions de la capa d'enllaç, aquest paquet es desa en un *buffer* per a ser enviat un cop es descobreixi una nova ruta. Això tant pot ser quan el node és l'originador o quan l'està reenviant el paquet i està regenerant la ruta.

5. S'afegeixen dues cues FIFO. Una per desar els paquets que rep un node durant el procés de descobriment de ruta i una altra utilitzada com a *buffer* de sortida.

2.5. TinyOS

TinyOS és un sistema operatiu que ens permet l'utilització de tots el components d'un sensor mitjançant interfícies programables. Donades les mínimes prestacions dels sensors,

[3] TinyAODV és una implementació minimalista del protocol AODV per a dispositius que funcionen amb el sistema operatiu TinyOS.

es requereix un sistema operatiu que per una banda ens faciliti l'execució de programes i per l'altra ens asseguri un bon ús dels recursos oferts [**9**].

TinyOS està desenvolupat per la Universitat de Berekley, i en una primera aproximació podríem dir que a més es tracta d'un gran conjunt de programes que estan desenvolupats per diverses comunitats d'investigadors. Aquest fet es dóna gràcies a què es tracta d'un projecte Open Source, i també per aquest motiu s'ha fet un paral·lelisme i s'ha arribat a anomenar-lo "el Linux dels sensors". Potser també el fet que estigui desenvolupat en un meta-llenguatge derivat del C ha ajudat a aquesta comparació.

NesC (Network Embedded Systems C) és un dialecte del llenguatge de programació C optimització per a les limitacions de memòria de les xarxes de sensors. NesC és un llenguatge orientat a components i està especialment dissenyat per a programar aplicacions sobre xarxes de sensors, en particular en el sistema operatiu TinyOS. Ens ocuparem de donar una visió molt més ampla de NesC més endavant.

Un programa en NesC està estructurat mitjançant components. L'usuari crea un propi component ajudat a la vegada per altres components ja creats. Dos components podran comunicar-se entre ells mitjançant una interfície, la qual definirà una sèrie de mètodes (*commands* i *events*) els quals haurien de ser implementats en cada components. Així doncs, un mètode podrà sol·licitar l'execució d'un altre component; per altra banda, per enviar una notificació s'utilitzarà un event.

La metodologia de disseny en programació per el treball sobre aquest tipus de sistemes encastats es coneguda com a programació orientada a components, i de la qual surten

aplicacions totalment modulars on el realment important són les entrades i sortides de les aplicacions. Durant aquesta secció repetirem tot sovint el terme component, i poc a poc anirem tenint una visió més clara del que és. Per ara, només direm que hi poden haver dos tipus de components: mòduls i configuracions.

Tot i que es poden utilitzar diferents plataformes, el més habitual és treballar amb TinyOS a través d'un sistema operatiu Linux o Windows XP. És important mencionar que el projecte TinyOS es va iniciar en sistemes BSD i que per tant, per treballar des de WindowsXP es requereix un simulador de plataformes Unix (com pot ser Cygwin). La programació de dispositius es durà a terme a través de diferents ports, depenent del mòdul amb el que es conti. Aquests ports poden ser serial, USB o Ethernet.

El disseny de TinyOS està basat en respondre a les característiques i necessitats de les xarxes de sensors, això implica que el sistema operatiu haurà de treballar amb poca memòria rapida, baix consum d'energia, operacions de concurrència intensiva, sense descuidar el fet que s'ha de facilitar el desenvolupament fiable d'aplicacions.

Podem diferenciar dos nivells de planificació en el nucli del sistema operatiu:

- **Esdeveniments:** Ideats com a processos petits, poden interrompre les tasques que s'estan executant.
- **Tasques:** Aquestes estan pensades per a dur el pes de la major quantitat del processament i no són gens crítiques amb el temps. Les tasques s'executen en la seva totalitat, però la sol·licitud d'iniciar una tasca i el final d'aquesta, són funcions separades. Aquesta

característica és típica de la programació orientada a objectes.

Aquesta aproximació ens permet assolir un alt rendiment en aplicacions de concurrència intensiva. A més com es pot entreveure, l'aprofitament de la CPU és molt eficient i ens provoca una execució més ràpida i un menor consum.

Com hem dit anteriorment, el sistema TinyOS està programat amb NesC i com veurem més endavant aquestes dues entitats es veuen estretament relacionades en tot moment. NesC suporta un model de programació que integra d'utilització de comunicacions, i capacitat per afrontar les concurrències que provoquen les tasques i els esdeveniments. També optimitza el codi en les compilacions del programa a la vegada que també facilita el codi que s'ha de desenvolupar per a aplicacions d'aquest estil.

TinyOS aporta un model de programació característic dels sistemes *embeddeds* o encastats. A continuació descriurem les idees que presenta aquest model:

- **Arquitectura basada en components:** La base de TinyOS és un conjunt de components, que són el punt de partida per a la construcció de qualsevol aplicació. Connectar aquests components utilitzant un conjunt d'especificacions detallades, és el que finalment definirà una aplicació.
- **Concurrència en tasques i esdeveniments:** Anteriorment hem comentat la concurrència entre diferents tasques i esdeveniments. Els components entreguen les tasques al planificador, el qual retorna el control immediatament al programa principal, i un cop el planificador hagi vist que l'execució de la tasca ha estat

duta a terme, envia un senyal. És important tenir en compte que les tasques es duen a terme en la seva totalitat i que no tenen cap mena de prioritat davant altres tasques o esdeveniments.

- **Les operacions de llarga durada han de ser dividides en dos estats:** la sol·licitud de l'operació i l'execució de la mateixa.

Ara tenim una mica més clar com funciona TinyOS i quin ús fa dels components. L'usuari té accés als components mitjançant interfícies. Un component proveeix i utilitza interfícies.

3. Estudi tècnic previ

3.1. Estudi de Viabilitat

Les línies fèrries són un dels entorns més estrictes per a les comunicacions electromagnètiques que podem trobar. Definim la Compatibilitat Electromagnètica (EMC) com la capacitat d'un sistema per treballar en un cert entorn electromagnètic [**10**].

L'entorn ferroviari està ple de fonts d'interferències electromagnètiques causades per l'equipament utilitzat per al servei dels trens. Podem diferenciar entre l'entorn intern i l'extern. El nostre objectiu en aquesta secció del treball és identificar les fonts que poden afectar als nostres sensors en l'entorn intern.

A banda de la compatibilitat electromagnètica, trobem alguns problemes més en la instal·lació dels sensors. La natura del nostre projecte és afegir un equipament extra a un sistema que ja estava construït i que es troba en producció, per tant haurem d'adaptar-nos a les característiques que se'ns imposen. Podem veure que trobarem algunes restriccions en la instal·lació del nostre *hardware* i haurem de solucionar problemes com l'obtenció d'energia o la fixació dels sensors al tren.

3.2. Fonts d'interferència

- **Motors:**

Hi ha diferents tipus de motors que, depenen del tipus de tren. Els trens impulsat per màquines diesel no estan propulsats per estacions elèctriques, malgrat això, s'incorporen alguns components elèctrics per nodrir al tren d'energia.

Per altra banda tindrem més fonts d'interferències en els trens elèctrics, és a dir, els que estan propulsats per motors elèctrics.

Aquests trens obtenen l'energia d'estacions elèctriques remotes que distribuiran aquesta electricitat per cables al llarg de la via.

La manera més comú de proveir un tren d'electricitat és mitjançant pantògrafs elevats a unes corrents de 25kV de corrent alterna a 50 Hz o 15 kV de corrent alterna a 16.6 Hz.

També podem trobar abastiments de corrent continua. Poden treballar a 600, 750 i 3000 V.

Podem resumir els sistemes d'abastament d'energia en la taula a continuació:

Sistema	Area d'aplicació
600 Vdc	En ciutats per electrificar tramvies.
750 Vdc	En ciutats per electrificar tramvies
1500 Vdc	Trens de rutes llargues i trens suburbans.
3000 Vdc	Trens de rutes llargues i trens suburbans.
25/15 KVac	Trens d'alta velocitat.

Taula 2 - Sistemes i les seves àrees d'aplicació.

- **Abastiment d'energia:**

Una altra font d'emissions és d'interfície entre el pantògraf i el contacte amb el cable de potència i col·lector de corrent. Els resultants són interferències de radiofreqüència que es poden estendre fins a la regió VHF (*Very High Frequency*).

- **Comunicacions i senyalització:**

Tot i que hem de tenir en compte les comunicacions inherents a la industria del ferrocarril (tals com l'ERMTS), no ens endinsarem en les interferències causades per aquestes comunicacions per que presumiblement no causaran problemes en la part dels sensors. Malgrat això, pot ser que sí s'hagin de tenir en compte aquestes fonts d'interferències en la unitat *gateway*.

- **Interferències als serveis ràdio:**

Com s'ha descrit anteriorment, podem trobar un bon grapat de fonts d'interferència que poden afectar a les comunicacions ràdio. A la Figura 8 hi ha una descripció acurada de les interferències que poden afectar a les nostres comunicacions ràdio. Està basat en els estàndards de compatibilitat electromagnètic EN50121:2000 - IEC 62236:2003 [**10**].

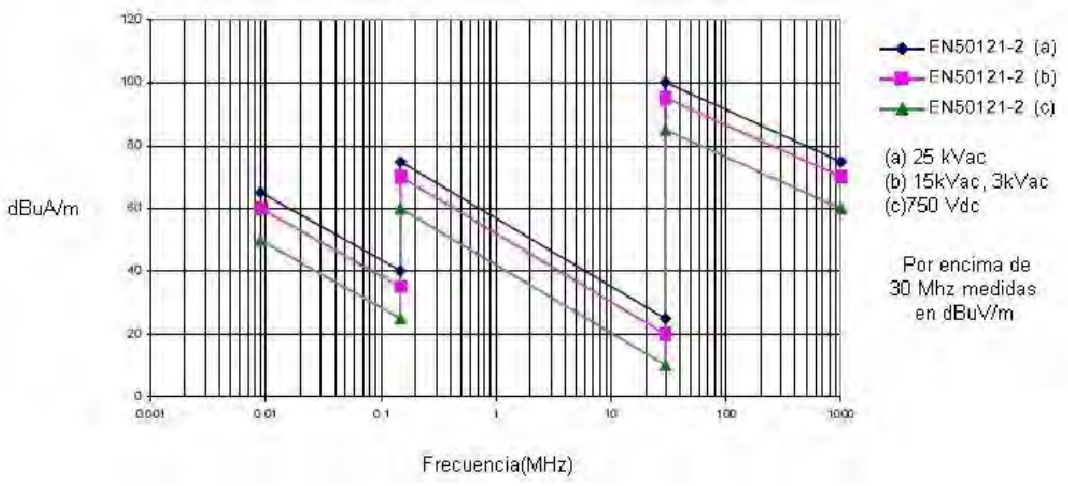


Figura 8 - Nivells d'interferència en l'entorn ferroviari.

Com podem veure, aquests nivells emesos en les àrees d'influència de l'entorn del ferrocarril afecten als principals serveis ràdio. Es mostra en la Taula 3 un resum de les bandes de freqüència dels principals serveis ràdio, característiques dels nivells de senyal, principal àrea d'utilització i nivell d'interferència.

Rang de freqüència	Serveis utilitzats definits per la ITU	Exemple d'usuaris	Mínim nivell de senyal/ Ample de banda ràdio	Ús proper a entorns ferroviaris	Interferències produïdes en el servei per entorns ferroviaris
598 MHz – 854 MHz	Difusió (TV)	Televiso digital Terrestre (TDT)	60dB(μVm^{-1})/4dB/8MHz	Mitjà	Baix
854 MHz – 960 MHz	Fix Mòbil	Telefonia mòbil digital (GSM, GSM-R) TETRA	38dB(μVm^{-1})/9dB/200Hz	Alt	Baix

960 MHz – 1.35 GHz	Radionavegació aeronàutica Radiolocalització	Equipament per a la medició de distàncies GPS militar	63dB(μVm^{-1})	Baix Baix	Baix Baix
1.71 GHz – 2.7 GHz	Fix/Mòbil Difusió per satèl·lit	Telèfons mòbils digitals (GSM) UMTS	38dB(μVm^{-1})/9dB/200Hz 43dB(μVm^{-1})/5dB/3.84MHz	Alt	Baix

Taula 3 - Principals serveis ràdio.

3.3. Informe primeres proves a Sant Andreu.

En la present secció es presentaran les proves dutes a terme als tallers de l'empresa ALSTOM a Sant Andreu (Barcelona) el dia 14/11/2008.

Els objectius dels tests són bàsicament dos:

- **Estudi de les interferències electromagnètiques generades per la xarxa de sensors:** Per tal d'assegurar que la nova xarxa no interferirà amb els sistemes de comunicacions existents en el tren, es mesurarà l'espectre ràdio generat per la xarxa de sensors amb l'ajuda d'un analitzador d'espectre. La conclusió principal serà que la potència generada per la xarxa de sensors és molt baixa i que es manté dins dels marges establerts per l'estàndard IEEE 802.15.4 a la banda ISM dels 2.4 GHz.
- **Estudi de la cobertura ràdio de la xarxa de sensors:** El desenvolupament del projecte requerirà la caracterització del canal en el qual la xarxa de sensors treballarà. Per tant, es duran a terme una sèrie de proves per tal d'assegurar algunes zones de cobertura i per tal de construir una xarxa capaç de mesurar certs punts i alhora encaminar la informació cap a el gateway col·lector d'informació. La conclusió principal serà que sota del tren s'obtenen distàncies de 10 metres. Aquestes distàncies seran suficients per al desenvolupament de l'aplicació malgrat que

aquesta requereixi d'encaminament multisalt per un increment de la cobertura.

3.3.1. Col·locació dels sensors

Els sensors es van col·locar en la posició més realista d'acord amb el posicionament que tindran en l'aplicació final, en la Figura 10 es veuen les situacions.

- Node 6: En un armari dins del tren.
- Node 7: Sota del tren, a la vora de la reductora.
- Node 2: Encapsulat amb el sistema que protegirà els sensors en l'aplicació final.
- Nodes 11 i 4: A sobre del tren (simulant la posició del gateway).
- Resta de nodes: distribuïts al llarg del tren sobre els *bogies*[4]i (Figura 10) els eixos de rodes.

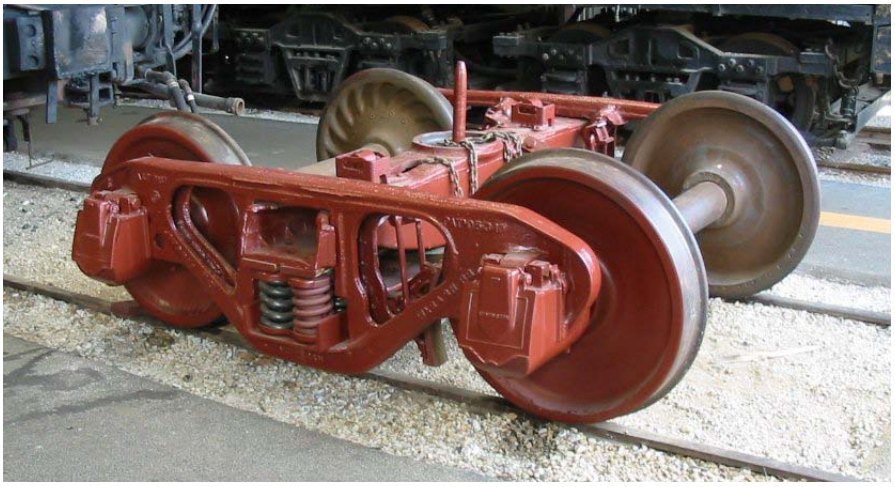


Figura 9 - Bogie exhibit l'Illinois Railway Museum.

[4] Un bogie és un conjunt de dos o tres parells de rodes (segons model) muntades a sobre dels respectius eixos pròxims, paral·lels i solidaris entre ells, que s'utilitzen en ambdós extrems dels vehicles de gran longitud destinats a circular sobre rails.

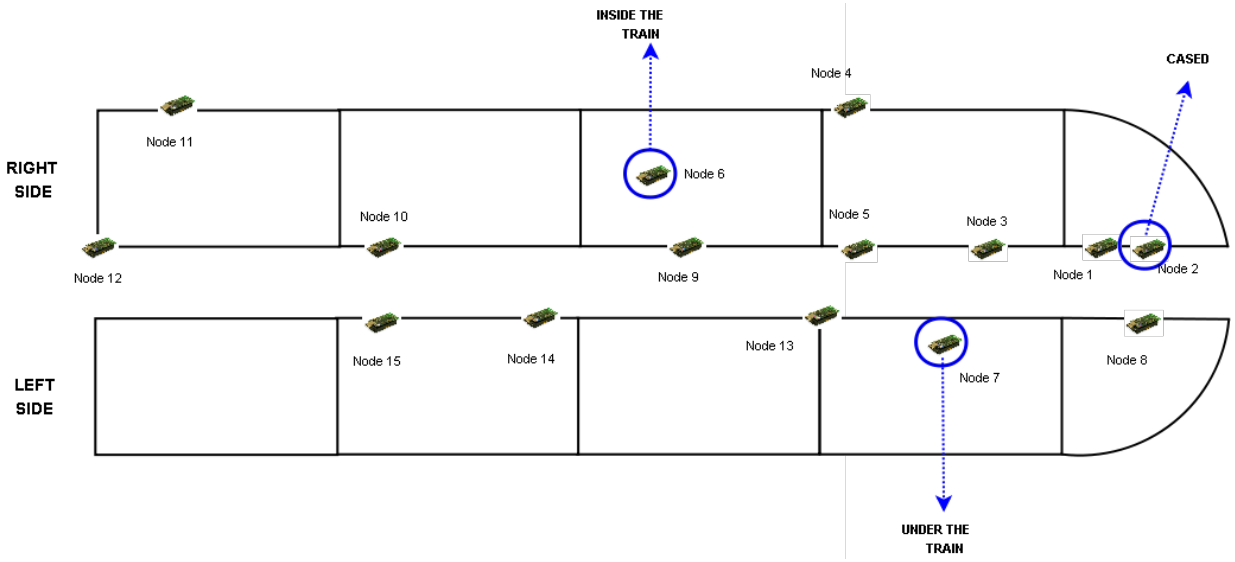


Figura 10 - Distribució dels sensors al llarg del tren.

3.3.2. Estudi de les interferències electromagnètiques produïdes per la xarxa de sensors.

Es va utilitzar un sol sensor amb 802.15.4 per a la primera prova duta a terme per a mesurar les possibles interferències introduïdes per els sensors. L'objectiu era mesurar i certificar que la potència de transmissió dels sensors a utilitzar complien les especificacions de l'estàndard de l'IEEE.

A la Figura 11 es poden veure tres zones ombrejades que indiquen els canals adjacents especificats per l'eina de mesura i la zona central mostra el canal mesurat. El canal de referència està centrat a la freqüència de referència (2.48 GHz), amb 1MHz d'ample de banda. Els canals superior i inferior també estan en un ample de banda de 1MHz i desplaçats 4 MHz. Els resultats mostren que la interferència produïda és -35dB i -41dB respecte el canal central.

L'estàndard també afegeix una màscara de densitat espectral de potència absoluta.

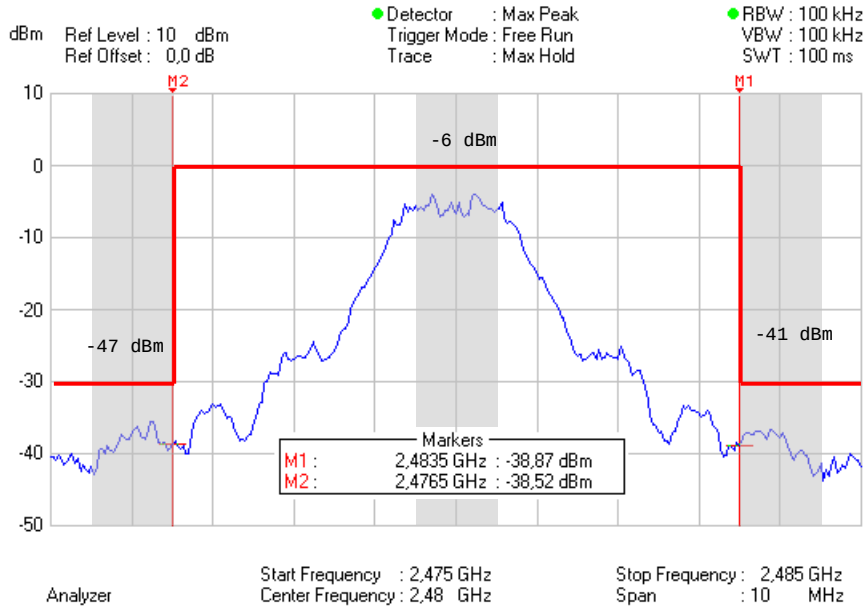


Figura 11 - Màscara de potència absoluta transmesa

Les línies vermelles visibles a la figura Figura 11 són els límits que determinen el criteris d'acceptació. Estan posats a -30dBm amb un offset de ±3.5 MHz des de la freqüència central. Es pot veure que els dispositius passen el test absolut per -5 dB a les freqüències baixes i per -8 dB a les freqüències altes.

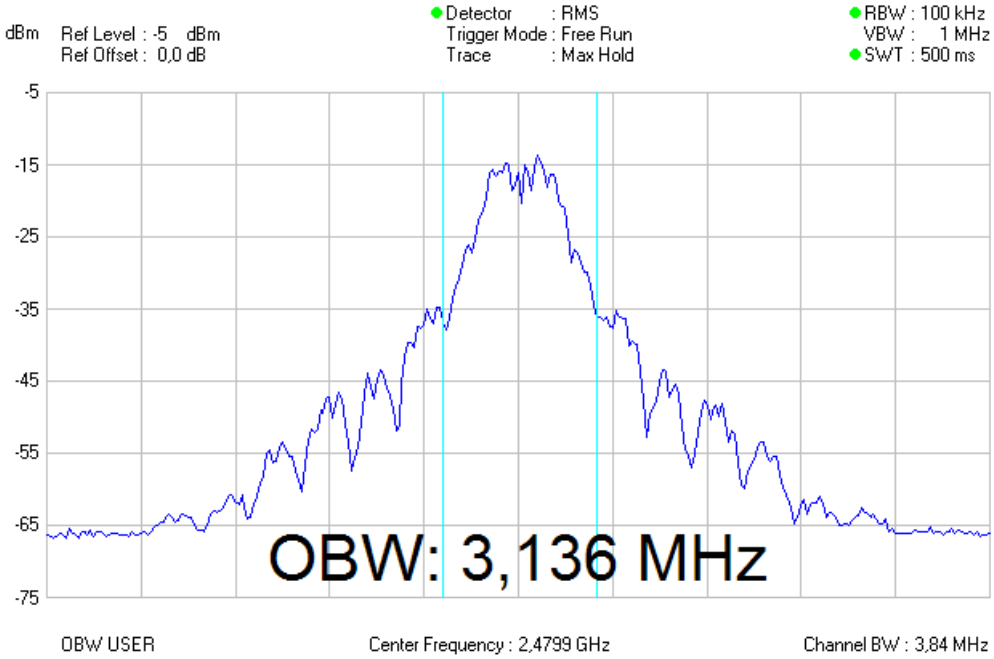


Figura 12 - Sensors transmetent al canal 11

Per tal de mesurar les interferències causades per les xarxes de sensors en un escenari pràctic, en aquest cas es va connectar una antena omnidireccional a l'entrada de l'analitzador. Es va col·locar l'analitzador d'espectre fora del tren, a la vora (< 1m) del Node 4.

En les següents figures (Figura 13, Figura 14, Figura 15) es mostren els resultats de les tres diferents mesures, on els canals utilitzats pels sensors són canal 11, canal 18 i canal 26 respectivament. Cal remarcar que les mesures corresponen a 7 minuts de *max-hold trace;* això vol dir que l'analitzador dibuixa els valors de la màxima potència observats durant 7 minuts. D'altra manera, degut a la baixa utilització (*u*) del canal, l'analitzador no hauria pogut detectar suficient energia per dibuixar a la pantalla.

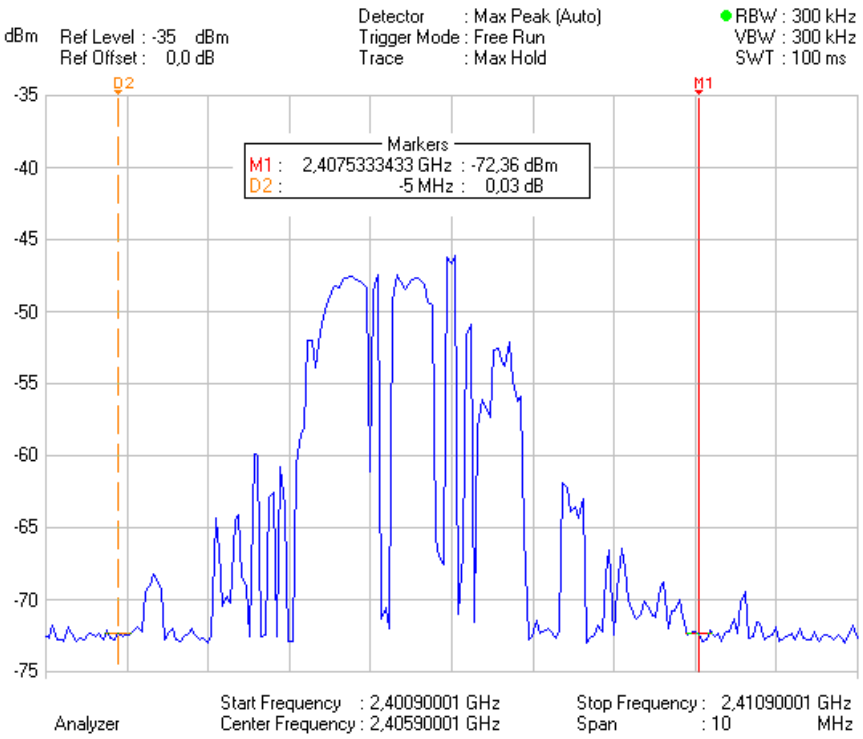


Figura 13 - Sensors transmetent al canal 11.

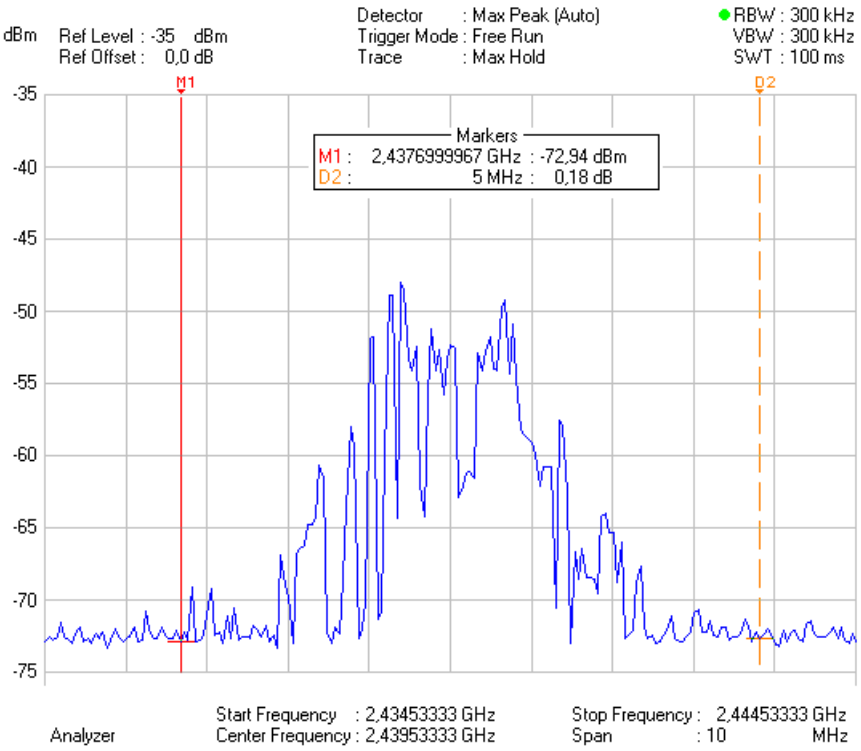


Figura 14 - Sensors transmetent al canal 18.

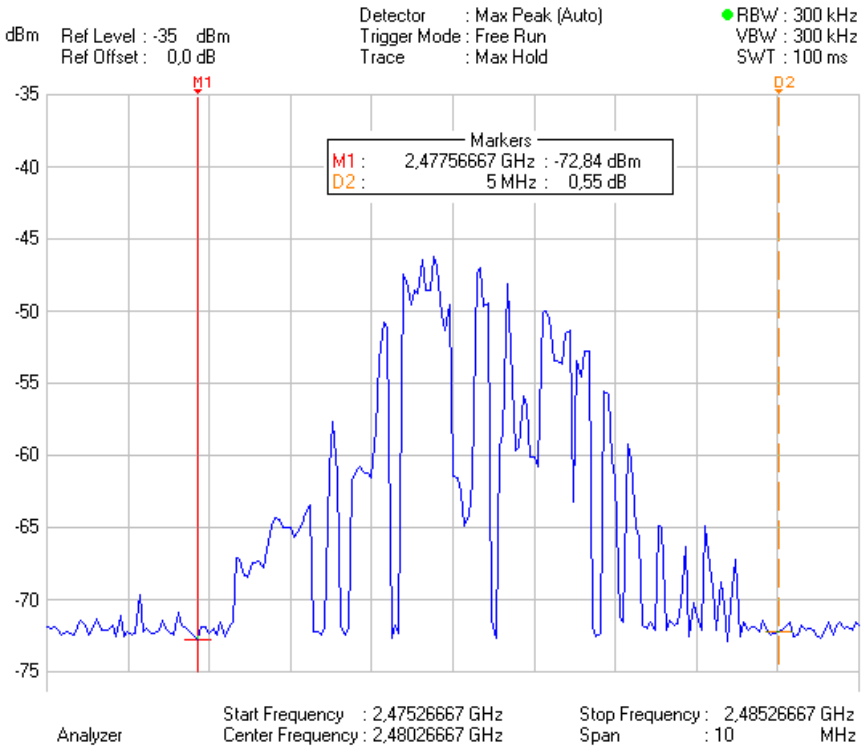


Figura 15 - Sensors transmetent al canal 26.

Tots els nodes al voltant contribueixen a la potència màxima rebuda, que es mostrada en les figures anteriors. En tots tres casos s'observa que no hi ha productes espectrals fora de la banda dels 5MHz que excedeixin el nivell de soroll.

Finalment, les mesures en el canal 11 es van repetir amb els sistemes del tren en funcionament. No es van observar diferències en aquest últim cas, tal i com es pot veure en la Figura 16.

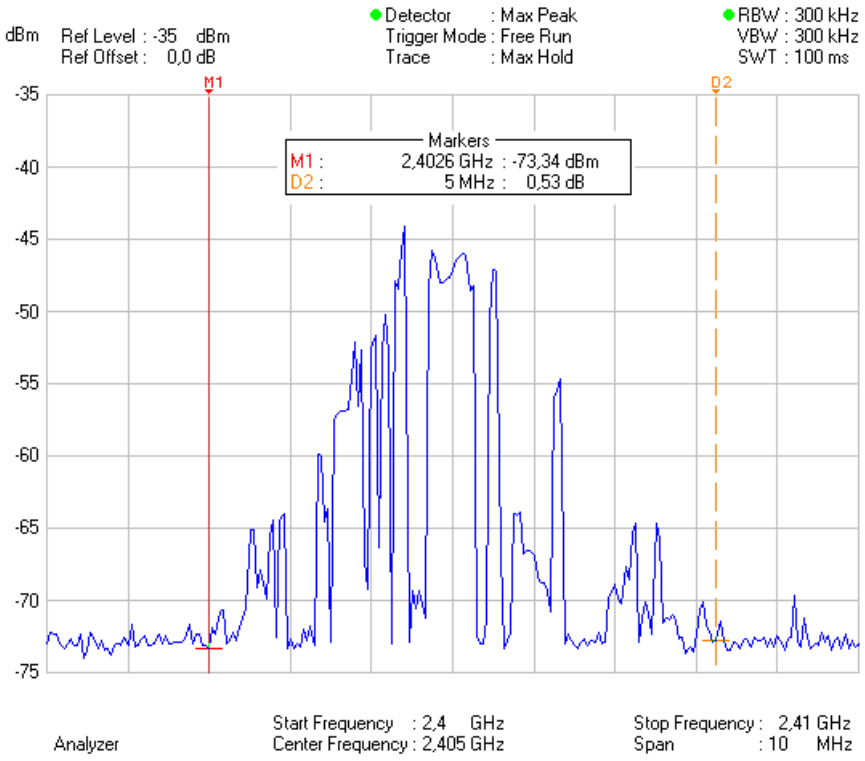


Figura 16 - Sensors transmetent al canal 11 mentre els sistemes del tren estaven en funcionament.

Les mesures aportades en aquesta secció certifiquen que la densitat de potència espectral dels dispositius IEEE 802.15.4, que hem posat sota estudi, compleixen amb les especificacions de l'estàndard. En conseqüència, la xarxa de sensors no pot veure's com una font d'Interferències amenaçadora pels altres sistemes de comunicacions que utilitzen canals fora de la banda dels 5 MHz

Aquest dispositius poden interferir amb altres sistemes que utilitzin la mateixa banda ISM. Tanmateix, qualsevol sistema que utilitza una banda ISM hauria d'implementar contramesures contra interferències de banda estreta, per tal de promoure la coexistència entre diferents tecnologies. Els valors mesurats ens condueixen a la conclusió que aquestes interferències són menys perjudicials que les interferències produïdes per altres tecnologies comuns com poden ser Wi-Fi o Bluetooth, ja que 802.15.4 ocupa menys amplada de banda que utilitza un nivell de potència menor per a la transmissió. A més a més, la utilització del canal per aquesta aplicació de prova és molt baixa, això vol dir que les transmissions des de diferents sensors estan molt separades en el temps. Aquesta situació redueix dràsticament els possibles efectes en les interferències.

3.3.3. Estudi de la cobertura de la xarxa de sensors

En aquesta secció cobrirem els aspectes de cobertura de la xarxa, on introduirem les proves dutes a terme per veure si els sensors col·locats en diverses parts sota del tren poden arribar (i amb quines condicions) al col·lector situat a dins o a sobre del tren. Aquests resultats no pretenen res més que demostrar que la comunicació dels sensors és viable en les posicions que s'indicaran. Per tant, donada la senzillesa de les proves, les conclusions que s'extrauran només ens serviran per intuir les zones amb més cobertura i on poder situar els sensors per una posterior implementació. La orientació dels sensors, canvis menors en la posició, instal·lació d'antenes omnidireccionals, i altres canvis podrien alterar, donar uns resultats diferents als que es mostren, però en essència el nostre objectiu era certificar la comunicació i detectar zones on la cobertura és més favorable.

Aquesta prova va ser duta a terme amb 15 sensors (TelosB). Cada sensor estava programat per enviar de forma *broadcast* una sèrie de paquets per tal de què fossin rebuts per els sensors veïns. De la recepció d'aquests paquets, cada node fa una sèrie de càlculs per tal de computar estadístiques en base a les dades rebudes. Els paràmetres de la prova són:

- Sensor origen.
- Sensor destí.
- Canal.
- Mida paquet.
- Potència.
- Paquets rebuts.
- CRC positius.
- Força del senyal.
- LQI (Link Quality Indicator)

Per tal de no esgotar la bateria, els sensors estan la majoria del temps en estat adormit (Asleep) en el qual apaguen la ràdio. Cada minut passen a un estat actiu (Awake) en el qual poden enviar i rebre informació. Només estan en estat Awake durant 2 segons, temps durant el qual estan esperant per rebre algun missatge de configuració que indiqui l'inici del test. Si durant els 2 segons no reben cap missatge, tornaran a l'estat Asleep.

Figura 17 - Diagrama d'estats i transicions.

El missatge de configuració conté els següents paràmetres:

- Identificació del sensor.
- Canal a utilitzar

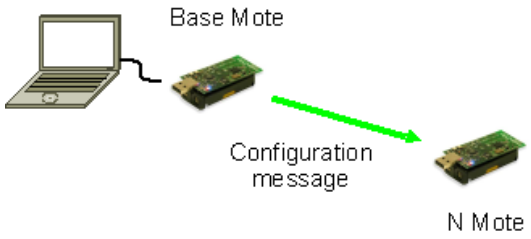


Figura 18 - Configuració d'un sensor

En el cas que un sensor rebi un paquet de configuració durant l'estat Awake, aquest inicia la transmissió de missatges. Aquest missatges, com hem dit abans, s'utilitzaran per a la generació d'estadístiques. Aquests seran de diferent mida i a diferent potència, per tant tindrem 9 tipus de missatges.

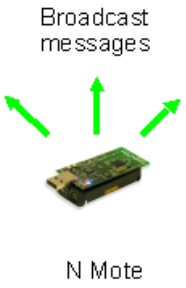


Figura 19 - Un sensor fent broadcasting d'informació.

Durant cada període d'enviament de dades, cada sensor envia 45 paquets. Com hem dit hi ha 9 tipus de paquet i dels quals se n'envien 5 de cada.

Les mides són:

- 18 Bytes
- 63 Bytes
- 108 Bytes

Les potències:

- 0 dBm
- -5 dBm
- -15 dBm

Aquest procediment va ser dut a terme 4 cops amb les següents variacions:

- Canal 11 (2405 MHz), motors parats.
- Canal 18 (2440 MHz), motors parats.
- Canal 26 (2480 MHz), motors parats.
- Canal 11 (2405 MHz), motors encesos. Aquesta variació va ser realitzada per tal de veure si els voltatges que afecten a l'entorn poden afectar en la transmissió de dades per la xarxa o al funcionament en general dels sensors.

Les dades i estadístiques extretes de les proves dutes a terme i després d'analitzar els resultats i generar estadístiques dels paquets rebuts per els sensors estan en els Apèndixs A i B,on hi ha un recull de tots els resultats en forma de gràfiques i taules. Tot seguit farem una descripció dels resultats i farem un anàlisi en forma de conclusions:

Tots els nodes tenen connectivitat directa amb altres elements de la xarxa, però no amb tots. Les posicions més crítiques són: el node del sostre (Node 4) i el node dins del tren (Node 6 en la zona de l'armari). El node 11 també està col·locat a sobre del tren, però separat dels altres per tenir una visió global de la cobertura. El node 4 (Figura 20) té una cobertura prou bona com per a donar la posició com a adient. El node 6 només veu als nodes pròxims a ell.

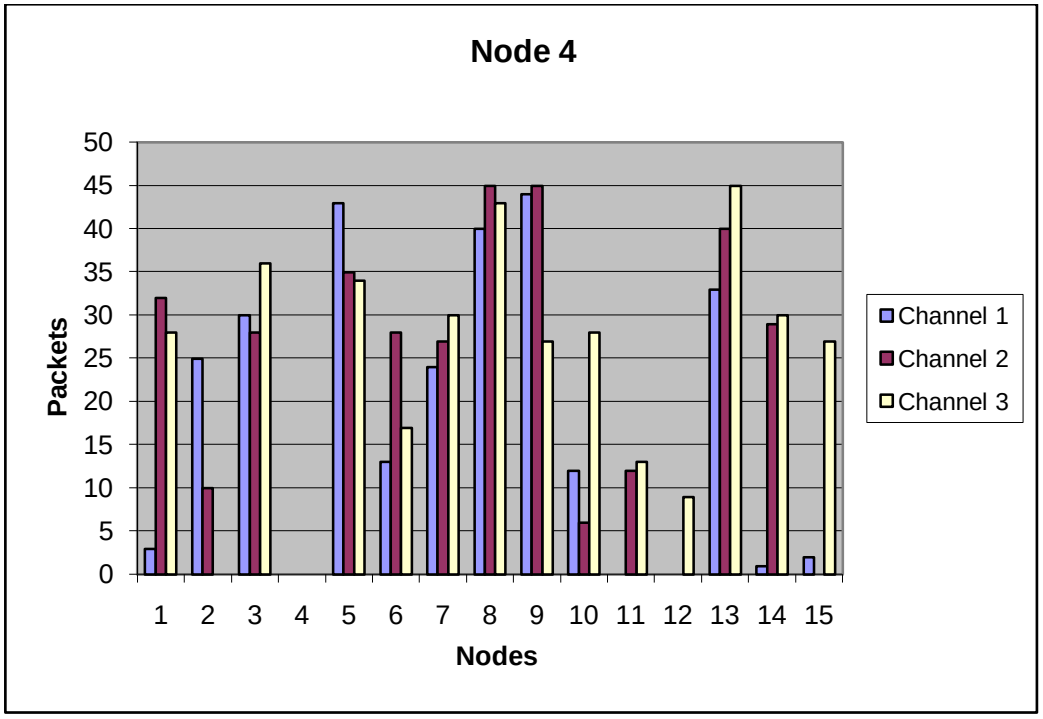


Figura 20 - Paquets rebuts pel node 4.

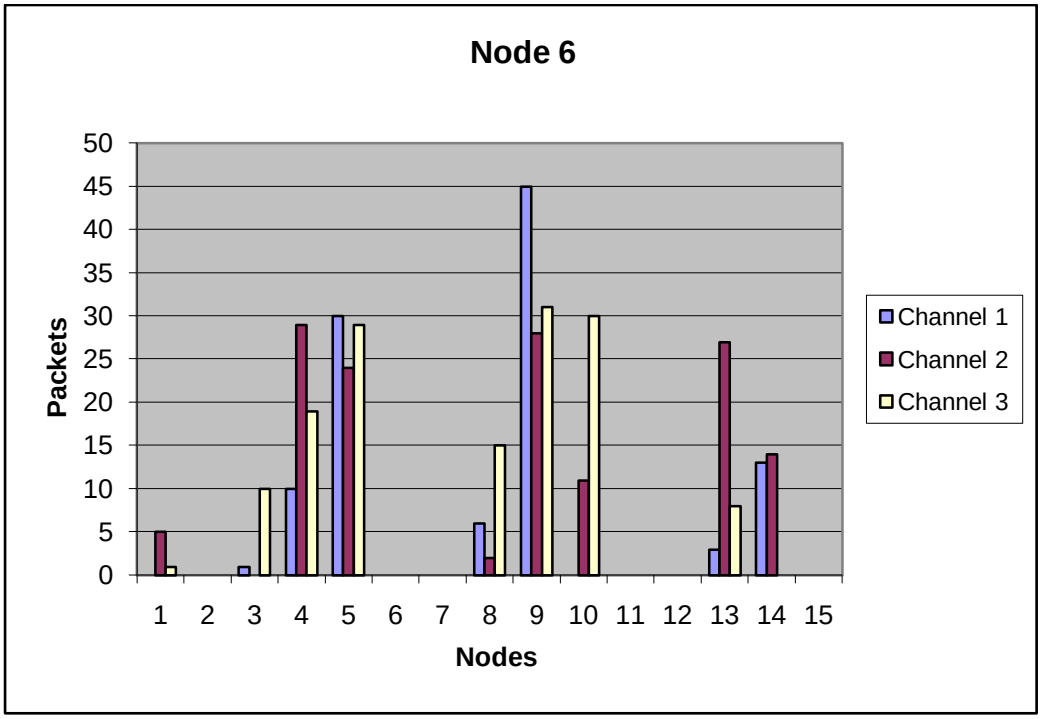


Figura 21 - Paquets rebuts pel node 6.

El canal utilitzat no té una influència significant. Els resultats entre Canals obtinguts després d'efectuar les proves utilitzant diferents freqüències mostren alguns canvis però carents de tendència. Com hem dit a l'inici de la secció, les diferències entre canals poden ser degudes a diferents motius que queden fora d'estudi ja que l'objectiu és certificar que existeix comunicació.

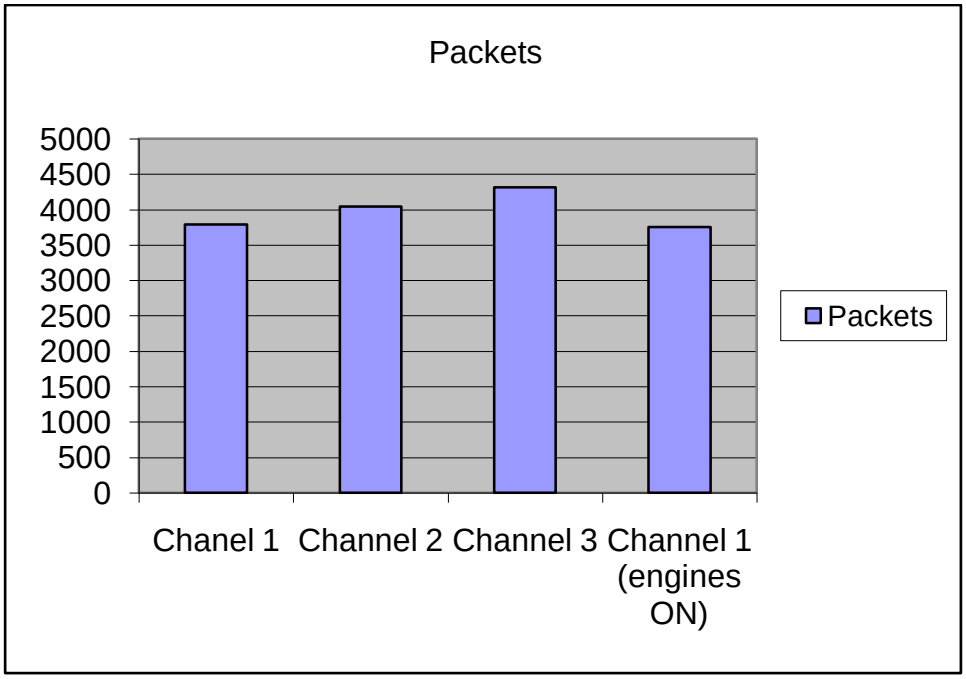


Figura 22 - Comparació del totals dels paquets rebuts en cada canal.

Podem veure en la Figura 23, la baixa influència dels motors sobre el funcionament dels sensors. Podem veure que no hi ha pràcticament cap diferència entre els paquets rebuts amb el tren, amb motors encesos i el tren amb els motors apagats.

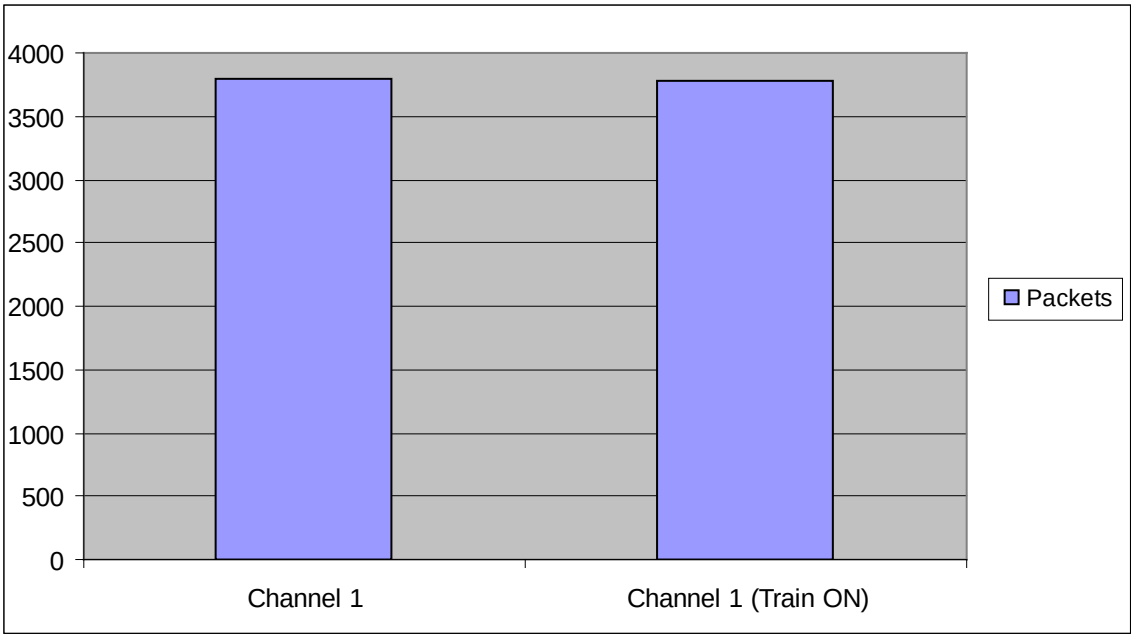


Figura 23 - Comparació del total de paquets rebuts amb el motors encesos i apagats.

L'elecció de l'encapsulament dels sensors té una influència baixa. Vàrem col·locar els sensors 1 i 2 molt a prop per veure la influència de les caixes sobre la transmissió i recepció de dades. Es mostra en la Figura 24 i Figura 25 que hi ha una petita pèrdua de cobertura per culpa de la caixa, i que si volem comunicació entre nodes, hauran d'estar prop una de l'altra.

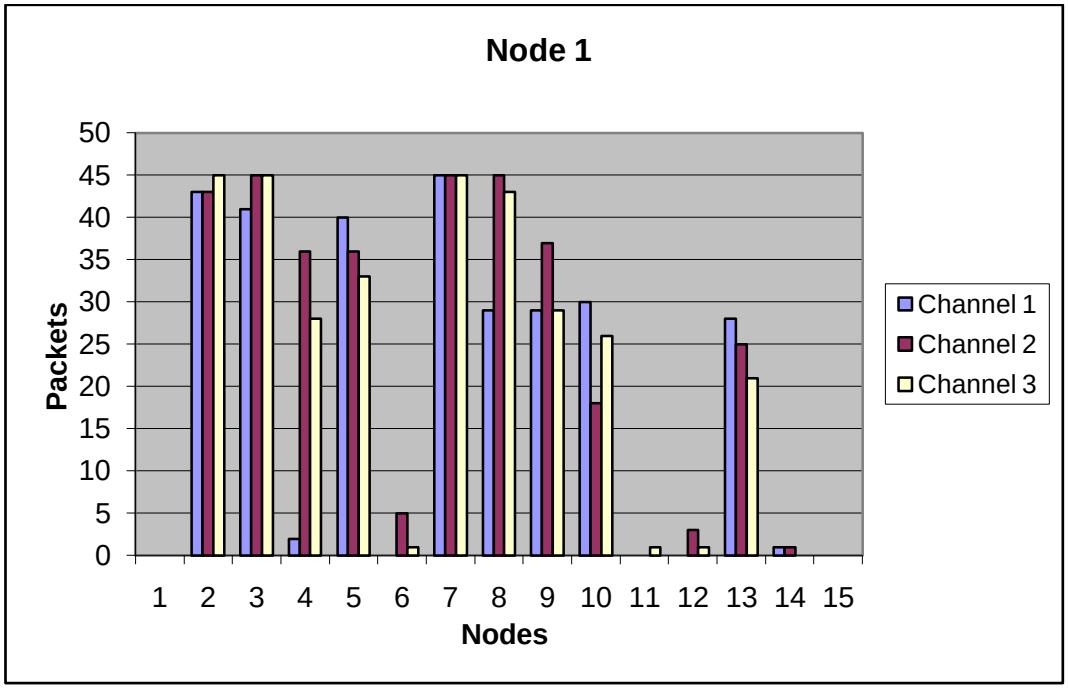


Figura 24 - Paquets rebuts pel node 1 (sense caixa).

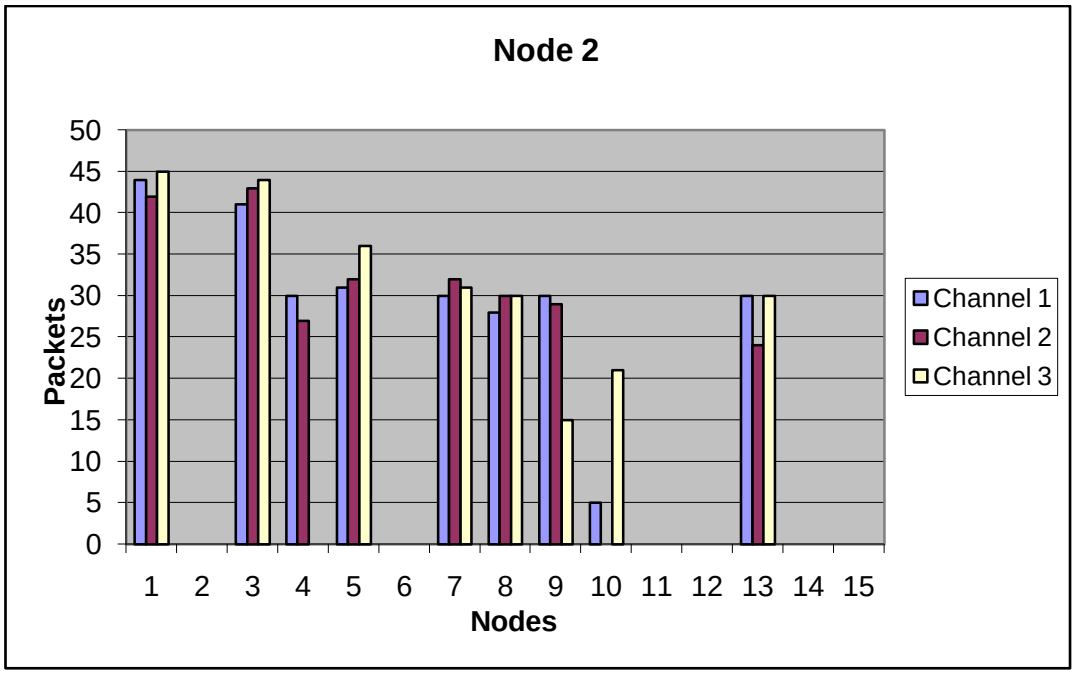


Figura 25 - Paquets rebuts pel node 2 (amb caixa).

La mida de paquet té una influència crítica (Figura 26). Els paquets de mida 108 sofreixen un PER (*Packet Error Rate*) 3 vegades superior al dels paquets de 63 Bytes. Tanmateix en la implementació final, es va decidir utilitzar la mida màxima de paquet, ja que per a transmissions llargues és millor corregir errors que patir sobrencapsulament.

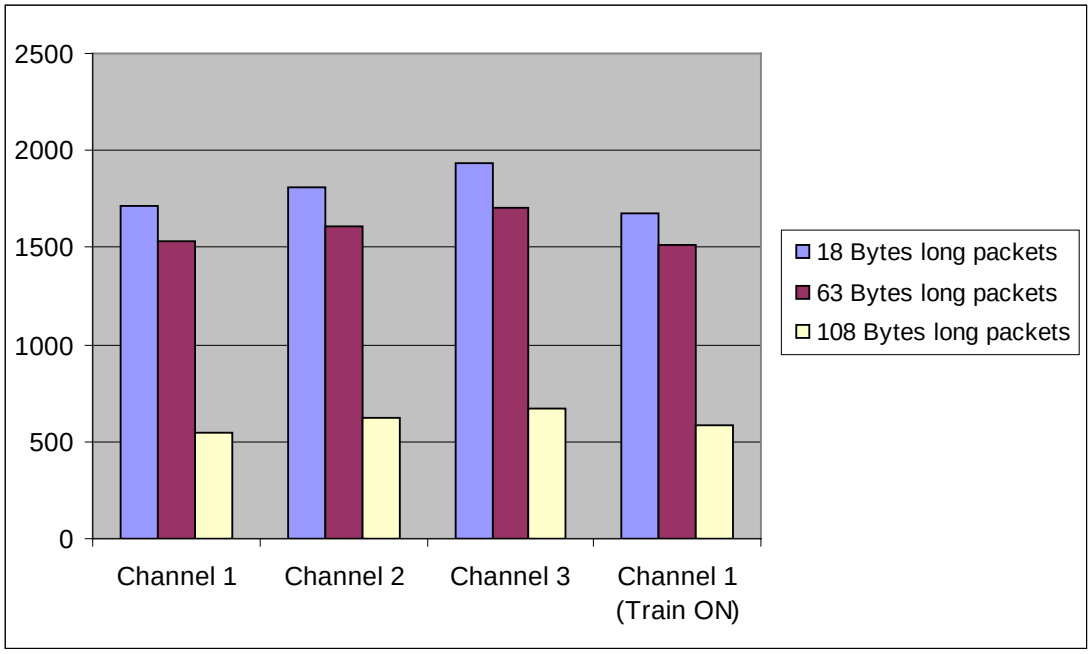


Figura 26 - Comparació de PER respecte a la mida de paquet.

4. Desenvolupament

4.1. Topologia de la xarxa

Fins ara hem pogut veure que l'entorn ferroviari presenta molt inconvenients per al desenvolupament d'una xarxa de sensors. Malgrat aquests inconvenients, l'estudi de viabilitat ens permet seguir avançant, ja que no ens mostra obstacles massa grans en termes de compatibilitat electromagnètica.

Tanmateix, l'informe sí que ens restringeix una mica l'ús dels sensors, ja que com hem pogut comprovar, la cobertura no és total, és a dir, hi ha sensors que hauran d'estar allunyats més de deu metres del gateway, i que no poden fer arribar la informació d'una manera directa.

Per tant, com s'ha apuntat en la secció introductòria, haurem d'augmentar la cobertura dels sensors sense que això repercuteixi d'una manera directa a la vida de la bateria que alimenta els sensors, és a dir, es descarta l'opció d'augmentar la potència de transmissió.

En una primera aproximació podríem pensar en fer ús d'alguna tècnica d'encaminament multisalt per transportar informació, però cal desestimar la idea ja que necessitem fer un disseny d'una eina que tingui un manteniment mínim, i el fet que un sensor hagi d'encaminar tot sovint informació per una ruta fins al destí, decrementa en bona mesura l'energia dels sensors.

Per tant, de cara als dispositius col·locats en les rodes, necessitem que estalviïn la màxima energia possible, i que per tant, només hagin de consumir bateria per sensar, enviar i rebre dades, i procediments de configuració. Sempre que no estiguin en cap

d'aquests estats hem de procurar apagar la ràdio i programar un estat d'estalvi d'energia.

Un cop arribats a aquesta situació, hem de solucionar una nova restricció: els sensors no poden participar d'una manera activa en una xarxa multisalt, a pesar que han de fer arribar les seves dades a un punt allunyat fora del seu radi de cobertura. Sembla doncs, que haurem de fer ús d'un altre tipus de sensors, que a nivell de hardware siguin iguals que els de les rodes, però que duguin a terme tasques diferents i col·locats de manera que cada relay estigui a l'abast d'un subconjunt de sensors de tal manera que la suma de relays garanteixin la cobertura de tots els sensors.

El fet de tenir els sensors col·locats a les rodes ens permet poder buscar una nova ubicació per aquests nous tipus de sensors, on puguin tenir un manteniment molt més l'abast, o fins i tot una font d'alimentació.

Aquests nous sensors faran de "*relay*" o retransmissor de la informació entre els sensors i el gateway, ja que estaran permanentment encesos per tal de poder escoltar tant les dades que hagin de viatjar des dels sensors cap al gateway o a l'inrevés.

Sembla doncs que, un cop la informació arriba al relay, podem trobar moltes maneres de transportar la informació cap al gateway, hem superat l'escull. Però un cop es van barallar les opcions tecnològiques per a construir una xarxa troncal, es va veure que la més viable tornava a ser una comunicació inalàmbrica. Per tant, la construcció de la xarxa troncal serà també per mitjà de l'estàndard IEEE 802.15.4. Com es pot imaginar, el transport en aquesta xarxa troncal requerirà encaminament multisalt que afrontarem a la següent secció. La topologia resultant serà la mostrada en la Figura 27.

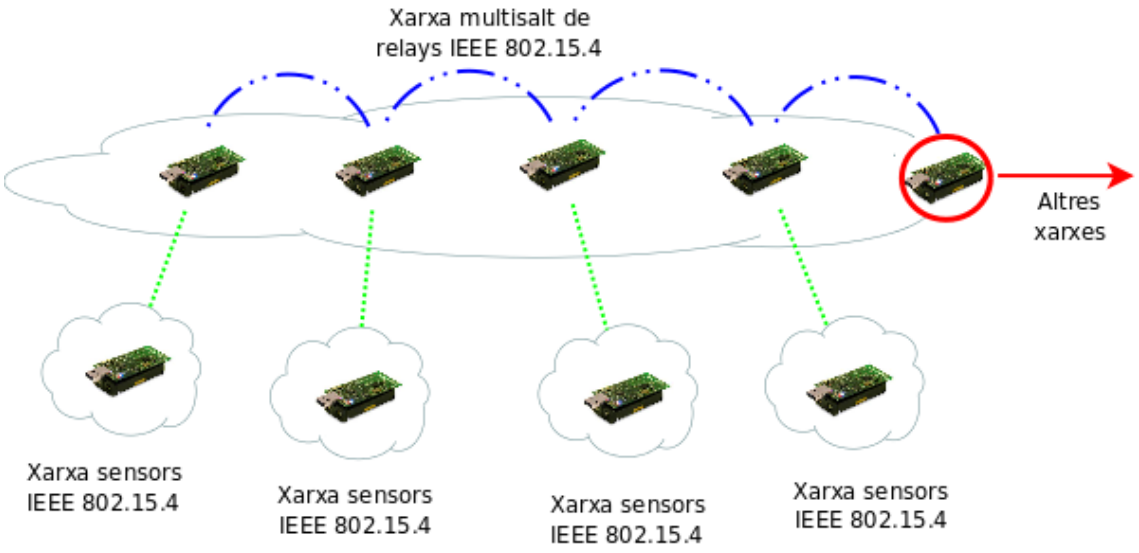


Figura 27 - Topologia de la xarxa

Ens queda un últim punt a tractar pel que fa a la topologia de la xarxa, i és l'element encerclat en vermell a la Figura 25. Es tracta de la interfície d'accés a la xarxa de l'element gateway. Com veurem més endavant es tracta d'un dispositiu especial ja que no serà ni un sensor ni un relay i durà a terme una sèrie de processos per a lliurar la informació l'ordinador recol·lector.

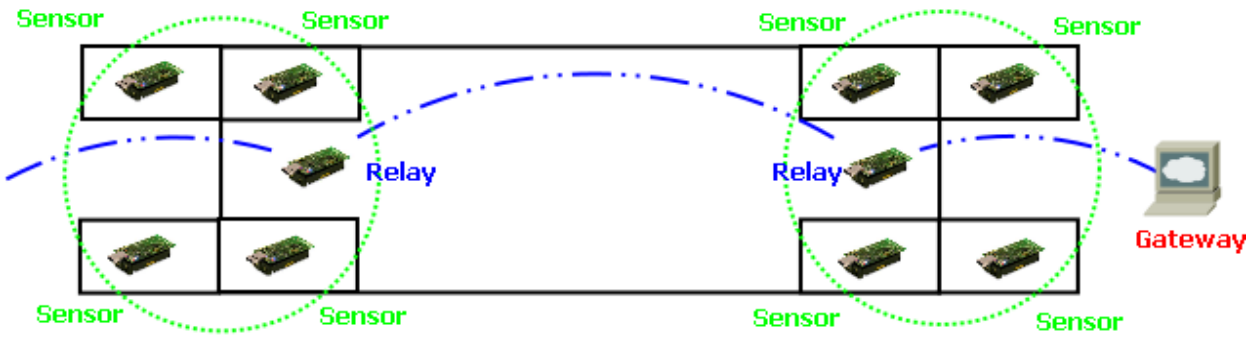


Figura 28 - Topologia de la xarxa en un cotxe[5] del tren.

[5] Val a dir que s'utilitza el terme cotxe enlloc del popular vagó, ja que durant el desenvolupament del projecte vam ser objecte de múltiples correccions pel fet que els vagons eren els cotxes que s'utilitzaven antigament per transportar la ramaderia.

Per acabar la secció farem un breu resum dels elements que conformen la nostra xarxa per tal de facilitar la comprensió i fer un colofó de la justificació dels diferents rols i jerarquies a la Taula 4.

Rol	Descripció	Propietats	Situació
Sensor	Element encarregat de la sensorització i enviament de dades al respectiu relay.	Ha de estalviar el màxim d'energia per tal d'evitar manteniment.	A l'exterior del tren, a la vora de la peça a sensar.
Relay	Element encarregat del transport de la informació sensors-gateway.	Té un manteniment més accessible i pot fer ús d'una font d'alimentació inesgotable	Dins del tren en panells o armaris
Gateway[6]	Element encarregat de l'obtenció de les dades per part de l'ordinador.	Accessible i programable fàcilment.	Dins del tren junt a l'ordinador encastat.

Taula 4 - Resum dels diferents rols del sensors.

4.2. Funcions específiques

En les següents seccions es descriuran en detall tant el codi implementat com les funcions que duran a terme els diferents tipus de sensors:

- **Sensor**:
  - Adquisició de les dades : Els sensors efectuaran operacions per tal d'obtenir dades de temperatura i

[6] Ens referim al sensor acoblat a l'ordinador *embedded* que també rep el nom de gateway en alguns moments. Per evitar confusions sempre que es refereixi a gateway serà el sensor, en altre cas s'especificarà degudament.

vibració i emmagatzemar-les per tal de què pugin ser enviades quan el gateway ho demani.

- Associació amb els relay: Cada sensor haurà de saber on enviar la informació de manera que si s'instal·len nous sensors o relays, no s'hagi de reprogramar res.
- Enviament de les dades: Les dades tindran diferents formats, i en especial les vibracions seran tractades com un fitxer de dades gran. S'haurà d'implementar algun mecanisme de finestra lliscant per tal que l'enviament sigui viable i eficient.
- Canvis de configuració dels procediments: El sensor enviarà cada cert temps dades de temperatura. Aquest temps ha de poder ser un paràmetre configurable així com la freqüència de mostreig de les vibracions, temps de mostreig i altres paràmetres.

- **Relay:**
  - Associació amb els sensors: El relay ha de ser capaç d'acceptar que els sensors que ho sol·licitin li puguin enviar informació i que el gateway sàpiga a quin relay ha d'enviar les configuracions per tal que arrivin al destí.
  - Encaminament de les dades: L'objectiu principal del relay és encaminar en ambdós sentits les dades que s'envien sensors i gateway. Aquest encaminament haurà de ser mitjançant un protocol multisalt.
- **Gateway:**
  - Obtenció de les dades dels sensors i enviament cap al PC: La funció bàsica del gateway serà traspassar informació de la xarxa de sensors cap al PC i a l'inrevés. Aquest procediment es durà a terme a través del port USB que proveeixen els TelosB.

L'obtenció de les dades haurà de segur també el mecanisme de finestra lliscant.

4.3. Bloc xarxa troncal

4.3.1. Modificacions AODV/NST-AODV

Ja s'ha explicat en la secció d'introducció el funcionament del protocol AODV. Ara aprofitarem per justificar el seu ús que, com hem mostrat en la secció anterior, serà en els nodes Relay i Gateway.

Si pensem en la distribució dels sensors relay al llarg del tren, veurem que es tracta d'una distribució en línia, és a dir, tots els sensors estaran col·locats en la mateixa direcció d'una línia recta imaginària. Serà una arquitectura que la majoria del temps restarà fixa, on les sensors relay restaran fixes en una mateixa posició.

Podríem pensar en un protocol d'encaminament fix, dit d'altra manera, programar cada sensor per a què passi tota la informació a un veí o un altre segons el sentit de les dades. Tanmateix, com hem dit prèviament, la xarxa multisalt s'encarregarà de transportar la informació obtinguda pels sensors a través dels relay fins al gateway. Aquesta decisió ha estat presa per tal d'estalviar energia, i fer que el manteniment de tota la xarxa es minimitzi i es faci d'una manera senzilla.

El fet que el manteniment sigui senzill també ens requereix que la instal·lació o substitució d'un sensor pugui fer-se sense l'assistència d'un tècnic, és a dir, que un encarregat del manteniment simplement posi un nou sensor en un nou lloc o que la substitució sigui immediata (sense la necessitat de reprogramar cap element).

Hem d'aplicar doncs un protocol que pugui efectuar les operacions de multisalt alhora que també canvia lleugerament la seva topologia segons demanda. A banda, l'ús del protocol AODV també ens proporcionarà solucions contra canvis de topologia (si un node es mou, per exemple: canviem la situació del gateway), caigudes de nodes (si un node deixa de funcionar es pot utilitzar el següent salt, sempre i quant la cobertura ho permeti).

TinyOS proporciona implementacions per a l'ús de AODV, com pot ser TinyAODV. Però nosaltres hem fet ús d'una implementació de NST-AODV per a TinyOS desenvolupada per el Grup de Xarxes Inalàmbriques del Departament d'Enginyeria Telemàtica de la UPC. Aquesta implementació incorpora les modificacions esmentades en l'apartat introductori respecte a l'AODV i esdevé un protocol ideal per aplicacions com la nostra.

Malgrat això, com hem dit anteriorment, la nostra aplicació serà utilitzada en una arquitectura completament lineal en un entorn força estricte, per tant, després de diverses proves es va procedir a fer les següents modificacions:

- **Mètrica dels salts LQI**: AODV fa servir una mètrica per tal de seleccionar el següent veí, aquesta mètrica intenta aconseguir el camí més curt en la implementació original de AODV. El problema que ens trobem nosaltres és que si busca el camí més curt, tot sovint l'elecció implicarà que el salt és de baixa qualitat, i que existeixen camins més llargs amb millor qualitat. Per tant fem servir una nova mètrica, que és el LQI (Link Quality Indicatori) per a què la ruta establerta tingui la millor qualitat i per tant es produeixin poques pèrdues.

- **10 intents d'enviament abans de enviar RERR:** Tot sovint ens trobem amb enllaços entre relays que són deficients o que presenten algunes interferències momentànies. Al tenir una topologia lineal, si el següent salt no és bo, no té sentit donar l'enllaç per dolent i buscar una nova ruta, ja que si la immediata és dolenta, la resta també ho serà. En la implementació inicial, un relay intentava 2 cops l'enviament, en la nostra implementació, el relay intentarà 10 cops aquest enviament, si no és possible se seguirà el procediment habitual.

4.3.1.1. Descripció de les interfícies de la implementació NST-AODV

```
#include "AODV_Msg.h"
#include "AODV.h"

#include <Timer.h>

configuration AODV {
     provides {

          interface Receive[uint8_t app];
          interface Intercept[uint8_t app];
          interface SendMHopMsg[uint8_t app];

          interface SingleHopMsg;
          interface MultiHopMsg;
          interface AODVMsg;
          interface Packet as AODVPayload;
          interface Reset;

          interface Receive    as ReceiveMsg[uint8_t app];
          interface AMSend     as SendMsg[uint8_t app];
          interface Packetas SingleHopPacket;

     }
}
```

En el codi precedent veiem una mostra del codi corresponent a la configuració de tota la implementació NST-AODV. El que ens mostra aquesta configuració són les interfícies que se'ns proporcionen per tal d'emprar el codi.

Les interfícies marcades en groc ens proporcionaran les funcions necessàries per tal de poder enviar informació utilitzant el protocol de multisalt.

D'altra banda, aquesta implementació també ens proporciona unes interfícies per a fer un enviament de dades d'un sol salt o salt simple, que són les que estan remarcades en verd. Aquestes interfícies ens seran de gran ajuda per a la implementació dels nodes relay, ja que l'alternativa d'utilitzar diferents components per als dos tipus de transport, implica certa redundància de codi.

La resta d'interfícies ens aportaran procediments necessaris per a la implementació d'una aplicació sobre TinyOS.

Donarem un breu cop d'ull cada interfície per tal d'entendre els seus paràmetres i poder veure en un futur el seu funcionament dins de les aplicacions de nivell superior.

4.3.1.2. Interfícies multisalt:

- `interface Receive:`

```
#include <TinyError.h>
#include <message.h>

interface Receive {
  event message_t* receive(message_t* msg, void* payload,
uint8_t len);
}
```

Aquesta interfície està composada d'un sol event, que serà llençat cada cop que es rebi un missatge per la interfície ràdio i que a més a més formi part del protocol AODV que estem utilitzant.

És a dir, aquest event només ens pasarà per paràmetre aquells missatges `msg` que estiguin destinats a nosaltres mitjançant el protocol NST-AODV.

- `interface Intercept:`

```
#include <TinyError.h>
#include <message.h>

interface Intercept {
event bool forward(message_t* msg, void* payload, uint8_t len);
}
```

Senyalitza que un missatge ha estat rebut, i que ha d'esser reenviat a una altra destinació.

- `interface SendMHopMsg:`

```
#include <AM.h>

interface SendMHopMsg {
        command error_t sendTTL(uint16_t address, uint8_t
length, message_t* msg, uint8_t ttl);
        command error_t forwardTTL(message_t* msg);
        command void* getPayload(message_t* msg, uint8_t len);
        event void sendDone(message_t* msg, error_t success);
}
```

Veiem com aquesta interfície ens proporciona tres commands i un event. El principal command `sendTTL`, bàsicament, enviarà el missatge `msg` que es passa per paràmetre a l'adreça `address`. Un cop s'hagi cridat el command, aquest retornarà un `error_t`, depenent de l'estat de l'execució de la funció.

Un cop el sistema hagi dut a terme el command, ens retornarà el control de la funció mitjançant l'event sendDone, el qual també ens passarà el missatge enviat i l'error corresponent.

4.3.1.3. Interfícies salt simple:

- `interface Receive`

És la mateixa funció que la funció Receive de la interfície multisalt, però en aquest cas només rebrem missatges d'un sol salt.

- interface AMSend

```
#include <TinyError.h>
#include <message.h>
#include <AM.h>

interface AMSend {

command error_t send(am_addr_t addr, message_t* msg, uint8_t
len);
command error_t cancel(message_t* msg);
event void sendDone(message_t* msg, error_t error);
command uint8_t maxPayloadLength();
command void* getPayload(message_t* msg, uint8_t len);
}
```

Aquesta interfície ens proporciona quatre commands i un event. Bàsicament farem ús de send, que ens permetrà enviar el missatge msg a la adreça addr. Un cop executada la funció, ens retorna l'event sendDone, igual que en la interfície multisalt d'enviament.

4.3.2. Mecanisme de finestra lliscant

Com hem dit, podem tenir fitxers de vibració molt llargs. Tal i com hem vist, l'entorn de desenvolupament de l'aplicació és un entorn on es produiran moltes pèrdues, això ens fa pensar que serà necessari un mecanisme que ens permeti que l'enviament sigui escalat, amb retransmissió d'errors i que permeti una rebuda dels paquets ordenada.

El mecanisme de finestra lliscant, és una tècnica de control de flux que pertany a la capa d'enllaç del model OSI. Soluciona el problema de la pèrdua de paquets durant la transmissió de dades entre dues capes, per tant permet assegurar la rebuda de dades completa i ordenada a capes superiors.

En el mecanisme de finestra lliscant, el receptor envia confirmacions (ACK, acknowledgments) al transmissor per notificar de la rebuda o no rebuda de paquets. Hi ha dos esquemes de confirmacions:

- **ACK:** S'utilitza en enllaços amb molt soroll, on el receptor rep un ACK per cada trama rebuda (quan el receptor envia un ACK per una trama n, s'entén que totes les trames enviades abans de n han estat rebudes correctament).
- **NAK:** Es tracta de la confirmació negativa (No ACK), i s'utilitza en enllaços on la pèrdua d'informació no és tan freqüent. En aquest esquema el receptor nomes envia ACK per trames perdudes.

Amb un protocol *stop-and-wait*, el transmissor espera una confirmació abans d'enviar el següent paquet. Com a resultat, hi ha com a molt un paquet en el canal en un moment donat (que segurament sigui molt menys del que la capacitat del canal pot oferir).

La principal característica del mecanisme de finestra lliscant és que permet una comunicació controlada per a una millor utilització del canal. El transmissor pot enviar com a màxim N trames sense confirmació. Es diu que N és la mida de la finestra. El mecanisme controla en tot moment les trames que poden ser enviades o que estan pendents de confirmació.

Per a enviar les dades, el transmissor assigna un número a cada trama. En tot moment el transmissor té una llista dels números de seqüència que corresponen a trames que es poden enviar.

Com les trames que el transmissor té marcades com a enviables es poden perdre o rebre amb errors, han de ser guardades en memòria per a possibles retransmissions. Així doncs, el transmissor ha de mantenir un *buffer* en memòria de la mida de la finestra.

Com s'ha esmentat prèviament, s'han implementat dues estructures de dades per a la construcció del mecanisme de finestra lliscant:

- **CuaMeva:** Cua que permet l'accés a qualsevol posició.

```
interface CuaMeva<t> {
  command void printQueue(uint8_t in, uint8_t en);
  command void printall();
  command void clear();
  command bool empty();
  command uint8_t size();
  command uint8_t maxSize();
  command t dequeue(uint8_t idx);
  command t element(uint8_t idx);
  command error_t enqueue(uint8_t position, uint8_t seqnum, t
newVal);
  command bool isOrdered(uint8_t start, uint8_t end);
}
```

- **MapOfInts:** Llista d'enters que permet inserció seqüencial i accés selectiu.

```
interface MapOfInts{
  command bool empty();
  command uint8_t size();
  command uint8_t maxSize();
  command uint8_t dequeue(uint8_t idx);
  command error_t enqueue(uint8_t elem);
  command uint8_t element(uint8_t idx);
  command uint8_t position(uint8_t Val);
  command void clear();
}
```

Evidentment, aquest mecanisme només s'implementa als extrems, és a dir, en el sensor i en el gateway. El seu funcionament és el següent:

1. El sensor té una finestra W. Això vol dir que té W paquets marcats per enviar. Cada cop que el sensor rebi un ACK, podrà actualitzar la seva llista de paquets (W') més el número de paquets que confirmi l'ACK (serà la diferència entre l'últim paquet confirmat i la confirmació actual).

2. El gateway té un buffer de rebuda de W posicions i dos punters que marquen els números de seqüència a rebre. Sempre que rebi els paquets adequats, els ordenarà i enviarà una confirmació quan es rebi W/2 ordenats des del primer enviat. Cada cop que envia una confirmació, envia i esborra el contingut de la W/2 part de la finestra ordenada i actualitza la seva llista de rebuda.
3. El gateway té una llista de paquets perduts que anirà demanant periòdicament. Un cop rebut el paquet pendent es passa al buffer de rebuda i s'esborra de la llista.
4. Si el sensor esgota la finestra (W'=0) enviarà l'últim paquet de la llista de marcats per rebre una confirmació o una petició de retransmissió.
5. Si el gateway rep un paquet fora d'ordre que ja havia rebut, reenvia l'últim ACK.

4.3.3. Configuracions dels sensors

Com hem vist, l'accés a la xarxa es farà a través de l'element gateway i per mitjà de l'USB. En aquesta secció es descriuran les opcions de configuració de la xarxa.

Ja s'ha comentat anteriorment que degut a què l'usuari necessita obtenir informació de la xarxa, d'alguna manera el sensor ha de proporcionar aquesta informació i la manera més senzilla de fer-ho es connectant-se via cable al sensor.

TinyOS 2.0 introdueix la noció d'enviament de paquets pel serial, que permet una comunicació entre sensor i PC senzilla i independent en quant a plataformes. Per tant, cada cop que enviem pel serial, ho haurem de fer en forma de paquet (també simplifica la possibilitat que el PC actuï com a *bridge*).

Per tant hi haurà un event encarregat de la recepció dels paquets i aquest és:

```
event message_t* SerialReceiveConf.receive(message_t *msg, void
*payload, uint8_t len)
```

En aquest event es programarà la lògica de comportament en la recepció d'un missatge que enviarem des del PC per el port serial. Si mirem en la configuració, el mòdul enllaçat amb aquesta interfície correspon a un Active Message `AM_NODE_CONF_MSG` que és el codi d'aplicació dels missatges de configuració.

Per a la comunicació via serial, l'entorn de programació que ofereix TinyOS 2.x aporta un programa (de fet n'hi ha diferents fets amb diferents llenguatges de programació) fet amb C que permet l'enviament de paquets via serial:

```
$ serialsend
Usage: serialsend <device><rate><bytes> - send a raw packet to a
serial port
```

Com veiem el programa demana un dispositiu on enviar el paquet, una taxa de transmissió i el paquet en bytes. Ja hem dit que per a que el sensor rebi un missatge de configuració, hem d'enviar un paquet de configuració com el declarat a la llibreria Node.h: `node_conf_msg_t`. Per tant podem enviar el següent:

```
$ serialsend /dev/ttyUSB0 telosb 00 00 00 00 00 06 00 192 00 30 00
09 01 00
```

4.3.4. Agregació de sensors als relays

Per a un millor desenvolupament del prototip inicial de tot el sistema, es va fer servir un adreçament fix. Aquest adreçament

estava lligat a cada adreça de sensor, de tal manera que els sensors tenien adreces de dos dígits i els relays adreces d'un dígit majors de zero.

Com es veurà més endavant, cada sensor resolia l'adreça del relay al qual estava associat fent una divisió entera entre 10 de la seva pròpia adreça. Per exemple: tots els sensors majors de 29 i menors de 40 estan associats al relay número 3.

Tanmateix, aquesta implementació es tracta d'una mesura temporal per tal de fer més senzill el desenvolupament del sistema troncal. És evident que aquest procediment implica molts problemes com:

- Difícil instal·lació de nous sensors.
- Reemplaçament de relays.
- Infrautilització d'adreces.
- Baixa tolerància a fallides.

Existeix una implementació d'una agregació de sensors més eficaç, la qual es divideix en dues parts:

- Part sensor:
  1) El sensor envia un paquet DISCOVERY de relay. Aquest paquet serà enviat en iniciar el sensor o sempre que no es confirmi un paquet després d'haver estat enviat el paquet de confirmació.
  2) El sensor es queda escoltant els missatges de HELLO que envia en relay, quan el sensor rebi un paquet amb millor LQI més un índex (per evitar que el sensor canviï entre dos sensors que tinguin LQI similar) tindrà aquest relay com a següent salt.

- Part relay:

1) Quan el relay rebi un paquet DISCOVERY, aquest iniciarà el procés de missatges HELLO, que consisteix en l'enviament de un cert número de paquets per a indicar LQI al sensor.

4.4. Desenvolupament de les aplicacions: Sensors, Relays i Gateway

En aquesta secció explicarem el desenvolupament de les aplicacions en cada tipus de sensor i les tècniques dutes a terme. Ens fixarem en cada sensor segons el rol que duguin a terme, i per tant, dividirem aquesta part en tres subseccions on apareixerà el codi de cada sensor, i on justificarem el desenvolupament de les aplicacions.

Abans d'entrar en el desenvolupament de cada codi, farem un resum de l'arbre de fitxers (simplificat).

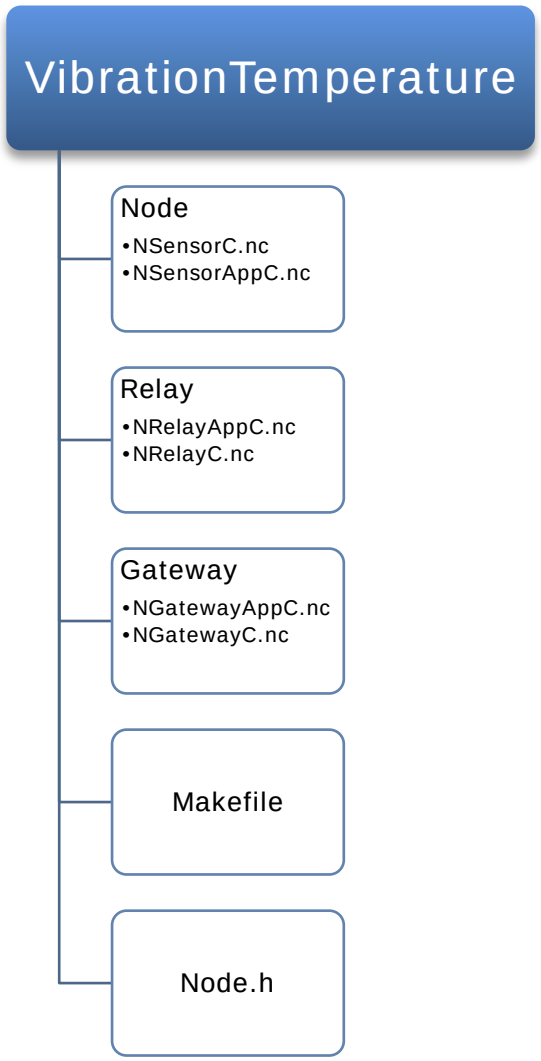


Figura 29 - Arbre de fitxers de l'aplicació Vibration Temperature

Com podem veure, cada codi de cada sensor esta format per dos fitxers: una configuració i un mòdul. En el Makefile hi haurà les comandes necessàries per a la generació i automatització de codi necessari per a la compilació de les aplicacions.

```
CFLAGS += -DCC2420_DEF_RFPOWER=2
CFLAGS += -DCC2420_DEF_CHANNEL=26
CFLAGS += -DTOSH_DATA_LENGTH=127
CFLAGS    +=    -I/opt/tinyos-2.x/tos/lib/aodv/interfaces    -
I/opt/tinyos-2.x/tos/lib/aodv/system
CFLAGS   +=   -I/opt/tinyos-2.x/apps/MapOfInts   -I/opt/tinyos-
2.x/apps/CuaMeva
CFLAGS   +=   -I/opt/tinyos-2.x/apps/MapOfInts   -I/opt/tinyos-
2.x/apps/Oscilloscope_Alstom
CFLAGS += -I$(TOSDIR)/lib/printf
BUILD_EXTRA_DEPS += PrintfMsg.class PrintfClient.class
ifdef GW
COMPONENT = NGatewayAppC
endif

ifdef NS
COMPONENT = NSensorAppC
endif

ifdef R
COMPONENT = NRelayAppC
endif

%.class: %.java
        javac $<

PrintfMsg.java: $(TOSDIR)/lib/printf/printf.h
        mig   java   -target=$(PLATFORM)   $(CFLAGS)   -java-
classname=PrintfMsg $(TOSDIR)/lib/printf/printf.h printf_msg -o
$@

CLEAN_EXTRA_DEPS=$(BUILD_EXTRA_DEPS)             PrintfMsg.java
PrintfMsg.class PrintfClient.class

include $(MAKERULES)
```

A l'encapçalament es defineix la potència de transmissió, el canal a utilitzar i la mida del paquet.

Tot seguit, s'inclouen camins a altres programes que s'empraran en les aplicacions de cada sensor.

En la part principal es defineixen unes macros per tal de poder compilar més fàcilment. Per a fer la compilació necessitem definir la configuració del codi a compilar.

Finalment es defineixen altres programes que s'utilitzaran i s'explicaran més endavant.

En el fitxer Node.h es defineixen les estructures dels paquets així com variables necessàries per al codi. El fitxer es mostra a continuació.

```
#ifndef NODE_H
#define NODE_H

typedef struct nx_node_data_msg_t{
  nx_uint16_t sens_src;
  nx_uint16_t relay_src;
  nx_uint16_t data;
} node_data_msg_t;

typedef struct nx_node_conf_msg_t{
  nx_uint16_t sens_dst;
  nx_uint16_t relay_dst;
  nx_uint8_t c_type;
  nx_uint8_t c_value;
} node_conf_msg_t;

typedef struct nx_node_vib_msg_t{
  nx_uint16_t sens_src;
  nx_uint16_t relay_src;
  nx_uint8_t seq_num;
  nx_uint8_t eix;
  nx_uint16_t data[48];
} node_vib_msg_t;

typedef struct nx_first_node_vib_msg_t{
  nx_uint16_t sens_src;
  nx_uint16_t relay_src;
  nx_uint8_t seq_num;
  nx_uint8_t data[2];
} first_node_vib_msg_t;

typedef struct pointer{
  uint8_t seq;
  bool received;
} pointer;

enum {
  // Timer values
  LONG_DATA_TIME=60*5,
```

```
  SHORT_DATA_TIME= 30,
  // AM values
  AM_GW_CONF_MSG=0xC1,
  AM_NODE_CONF_MSG=0xC0,
  AM_DATA_MSG=0xD0,
  AM_VIB_MSG=0xE0,
//Class Type Values
  CTYPE_SLEEP=0x00,
  CTYPE_ACK=0xE1,
  CTYPE_NACK=0xE2,
  CTYPE_MNACK=0xE3,
  CTYPE_STARTVIB=0x01,
  CTYPE_STARTVIB_NOW=0x02,
  CTYPE_CONF=0xB0,
  //provisional
  HELLO=0xff,
  DISCOVERY=0xfe,
  //Types of configuration
  CTYPE_TEMPTIME=0xB1,
  CTYPE_VIBLONG=0xB2,
  CTYPE_SAMPLFREQ=0xB3,
  CTYPE_STARTSENS=0xB4,
  //Sliding window constants
  CUA_SIZE=14,
  WINDOW=CUA_SIZE/2,
  MAX_VIBPACK=64,
  //Others
};
#endif
```

4.4.1. Sensor

Bàsicament, un sensor ha de dur a terme tres primitives, que són les següents:

- **Sensar:** El sensor llegirà per les interfícies software proporcionades per la lectura dels sensors corresponents.

  El TPA81 és un sensor tèrmic de 8 píxels que pot mesurar la temperatura d'un objecte a distància. Aquest sensor està firmat per una matriu de 8 sensors col·locats linealment, de forma que pot mesurar 8 punts adjacents simultàniament. A diferència dels sensors pir utilitzats en sistemes d'alarma i detectors per encendre llums, el sensor tèrmic no necessita que hi hagi moviment per detectar calor. El sensor es connecta per un bus I2C (Figura 30).

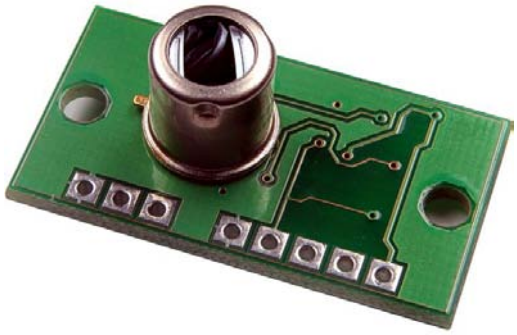


Figura 30 – Sensor TPA81

Pel que fa a les vibracions, s'han fet servir dos tipus de sensors de la companyia Freescale diferents: un per a la vibració en l'eix vertical (MMA2301) i un altre per a les vibracions dels eixos horitzontals (MMA1212).

Figura 31 – Sensor MMA2301

Els acceleròmetres Freescale estan formats per una acceleròmetre en un circuit integrat. El dispositiu consisteix en una cèl·lula sensora capacitiva (*g-cell*) i un condicionador de senyal CMOS en un ASIC contingut en un sol circuit integrat. L'element sensor està hermèticament tancat per evitar l'entrada d'aigua.

La g-cell és una estructura mecànica formada per materials semiconductors utilitzant processos semiconductors. Pot ser

modelada com dues plaques estacionaries amb una placa entre mig. La placa del centre es pot desviar de la seva posició original amb una acceleració del sistema. Quan la plata del centre es desvia, la distància entre aquesta i una de les altres dues disminueix mentre que amb l'altra augmenta. Un canvi en la distància aporta mesures d'acceleració.

- **Enviar:** El sensor haurà de saber a qui enviar les dades, i un cop fet això haurà de completar una seqüència d'enviament que pot variar segons el tipus de dades. Per a la temperatura, el procediment serà enviar cada cert temps la temperatura. Pel que fa a les vibracions se seguirà un procés de polling controlat per el gateway, en el que s'interrogarà als sensors: primer per a que capturin la informació i després per a que la enviïn.

- **Estalviar energia:** Sempre que no estigui fent res de les dues coses anteriors el sensor haurà de reposar en un estat d'estalvi d'energia. El procediment serà:
  - Enviar dades.
  - Romandre a l'espera de peticions:
    - Si es demana una petició acceptar-la.
    - Si no, apagar la ràdio i encendre el temporitzador per al pròxim enviament de dades.

Tot seguit pasarem a explicar les parts més importants del nostre codi, per tal de veure els detalls de la implementació.

```
#include "Node.h"

configuration NSensorAppC{
}
```

```
implementation{

  components MainC, LedsC, PrintfC;
  components NSensorC as App;
  components ActiveMessageC;
  components new AMReceiverC(AM_DATA_MSG) as AMReceiverData;
  components new AMSenderC(AM_DATA_MSG) as AMSenderData;
  components new AMReceiverC(AM_NODE_CONF_MSG) as
AMReceiverConf;
  components new AMSenderC(AM_NODE_CONF_MSG) as AMSenderConf;
  components new AMReceiverC(AM_VIB_MSG) as AMReceiverVib;
  components new AMSenderC(AM_VIB_MSG) as AMSenderVib;

  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as AwakeTimer;
  components new TimerMilliC() as SleepTimer;
  components new TimerMilliC() as SendTimer;
  components new TimerMilliC() as RepeatTimer;

  components OscilloscopeC;
  components HplTPA81LogicC as Sensor;
```

En aquesta primera part de la configuració, es declaren els components que després s'utilitzaran en el mòdul. Cal destacar una propietat d'aquesta aplicació, i que com veurem més endavant s'utilitza en tot el sistema:

Per a la comunicació entre sensors s'utilitza els sistema Active Message (ActiveMessageC, AMRecevierC, AMSenderC, etc). Aquest sistema ens aporta primer de tot uns "handler" per a d'utilització del camp AM del paquet 802.15.4, i en segon lloc, tota la llibreria de propòsit general per als Atmel 8535 [**11**].

Els handlers seran commands i events diferenciats segons identificador AM. Per tant, nosaltres utilitzarem diferents AM segons estem enviant o rebent informació de temperatura, vibració o configuració.

Una altra cosa que pot sorprendre és l'ús de temporitzadors (`components new TimerMilliC()`). Resulten molt pràctics per al

desenvolupament orientat a esdeveniments i en molts casos s'utilitzen com a substituts eficients dels bucles tradicionals.

```
App.Boot             ->MainC;
App.Leds         ->LedsC;

App.RadioControl      ->   ActiveMessageC;
App.PacketData       ->    AMSenderData;
App.SendTempMsg      ->    AMSenderData;
  App.ReceiveTempMsg  ->   AMReceiverData;

  App.PacketConf       ->   AMSenderConf;
  App.SendConf        ->    AMSenderConf;
  App.ReceiveConf      ->   AMReceiverConf;

  App.PacketVib       ->    AMSenderVib;
  App.SendVibMsg     ->     AMSenderVib;
  App.ReceiveVibMsg   ->    AMReceiverVib;

  App.Timer0          ->    Timer0;
  App.AwakeTimer->    AwakeTimer;
  App.SleepTimer      ->    SleepTimer;
  App.RepeatTimer     ->    RepeatTimer;
  App.SendTimer       ->    SendTimer;

  App.HplTPA81       ->Sensor;
  App.Dades           ->OscilloscopeC;
}
```

Aquesta segona part de la configuració enllaça els components amb les interfícies del mòdul per a la compilació del programa. A l'esquerra hi ha el nom de l'aplicació (reanomenat App per facilitat de programació) separat d'un punt per el nom de la interfície en l'aplicació.

A l'hora de sensar i enviar s'ha de tenir molt en compte un factor crucial, i és l'estructura de les dades que els components sensors entregaran al sensor i que hauran de ser enviades per la interfície ràdio. Les interfícies encarregades de aportar-nos la informació dels sensors seran:

```
interface Dades {
command error_t start_capture(uint8_t dimensio);
event void captureDone(uint8_t* mesures, uint16_t len);
command error_t setFreq(uint16_t freq);
```

```
command uint16_t getFreq();
}
```

Veiem que la interfície Dades (encarregada de retornar una captura de vibració) ens retorna un punter a una posició de memòria, això vol dir que podem tenir una estructura amb una longitud considerable, que evidentment haurem de segmentar per tal de poder enviar a paquets.

```
interface HplTPA81 {
  command error_t setRegConf( uint8_t val );
  async event void setRegConfDone( error_t error );
  command error_t setServPos( uint8_t);
  async event void setServPosDone( error_t error);
  command error_t getFirmware();
  async event void getFirmwareDone(error_t error, uint8_t val);
  command error_t measureTemperatureAmb();
  async event void measureTemperatureAmbDone( error_t error,
uint8_t val );
  command error_t measureTemperatureAmb();
  async event void measureTemperatureAmbDone( error_t error,
uint8_t val );
  command error_t measurePixel(uint8_t num);
  async event void measurePixelDone( error_t error, uint8_t
val);
  command error_t measureRawPixel();
  async event void measureRawPixelDone(error_t error, uint8_t
*buf);
}
```

La interfície HplTPA81 (corresponent a la sensorització de temperatura), retorna valor de 1 Byte que fàcilment encabirem en un paquet.

Les tasques i events programats en el sensor sensor són els següents:

```
   event void Boot.booted()
```

- Event retornat després d'iniciar-se el sensor

```
event void Timer0.fired()
```

- Temporitzador per a l'enviament de les dades de temperatura. Dins aquest temporitzador es crida a HplTPA81.

```
event void SleepTimer.fired()
```

- Temporitzador que atura els timers en funcionament i apaga la ràdio. Quan finalitza es torna a encendre la ràdio i engega de nou temporitzadors. Dins aquest timer es crida a RepeatTimer.

```
event void RepeatTimer.fired()
```

- Temporitzador encarregat de retransmetre l'últim paquet enviat sempre que s'esgota la finestra o bé expira el timer.

```
event void SendTimer.fired()
```

- Timer encarregat de fer els enviaments.

```
event void RadioControl.startDone(error_t err)
```

- Gestor de la encesa de la ràdio. Retorna SUCCESS si la encesa ha estat satisfactòria, altrament FAIL.

```
event void RadioControl.stopDone(error_t err)
```

- Gestor de l'apagada de la ràdio. Retorna SUCCESS si l'aturada ha estat satisfactòria, altrament FAIL.

```
task void repeat_vib()
```

- Es repeteix el paquet de vibració següent a l'últim confirmat.

```
task  void send_vib()
```

- Tasca que efectua l'enviament seqüencial del fitxer de vibració segmentat per paquets.

```
void send_single_vib(uint8_t seq_num)
```

- Funció que envia un únic paquet. Correspon a les retransmissions i requereix un número de seqüència per paràmetre.

```
async event void HplTPA81.measurePixelDone(error_t error, uint8_t val)
```

- Event retornat després d'una mesura de temperatura. Val és el valor de la temperatura. Retorna SUCCESS si la lectura ha estat correcta, altrament FAIL.

```
event void SendTempMsg.sendDone(message_t* msg, error_t error)
```

- Event retornat després de l'enviament d'un paquet de temperatura. Msg és un punter a l'inici de l'estructura de

paquet.Retorna SUCCESS si l'enviament ha estat correcte, altrament FAIL.

```
event void SendConf.sendDone(message_t* msg, error_t error)
```

- Event retornat després de l'enviament d'un paquet de configuració. Msg és un punter a l'inici de l'estructura de paquet. Retorna SUCCESS si l'enviament ha estat correcte, altrament FAIL.

```
event void SendVibMsg.sendDone(message_t* msg, error_t error)
```

- Event retornat després de l'enviament d'un paquet de vibració. Msg és un punter a l'inici de l'estructura de paquet. Retorna SUCCESS si l'enviament ha estat correcte, altrament FAIL.

```
event message_t* ReceiveVibMsg.receive(message_t* msg, void* payload, uint8_t len)
```

- Event de rebuda de missatge de vibració. Msg és un punter a l'estructura del missatge. Payload és un punter a la posició del contenidor de les dades. Len és la llargada del paquet.

```
event message_t* ReceiveConf.receive(message_t* msg, void* payload, uint8_t len)
```

- Event de rebuda de missatge de vibració. Msg és un punter a l'estructura del missatge. Payload és un punter a la posició del contenidor de les dades. Len és la llargada del paquet.

```
event message_t* ReceiveTempMsg.receive(message_t* msg, void* payload, uint8_t len)
```

- Event de rebuda de missatge de vibració. Msg és un punter a l'estructura del missatge. Payload és un punter a la posició del contenidor de les dades. Len és la llargada del paquet.

```
event void Dades.captureDone (uint8_t* dades, uint16_t len)
```

- Event de captura de dades de vibració. Dades és el punter a l'inici de l'estructura del fitxer. Len és la llargada del fitxer.

4.4.2. Relay

Les funcions d'un relay han de ser passar les dades de molts sensors a la xarxa multisalt, és a dir, ha de passar la informació que li passin els seus sensors al següent salt (que com hem dit

serà el que tindrà millor qualitat en la direcció cap al gateway). I en l'altre sentit, també haurà de transmetre ocasionalment algun paquet cap a un dels seus sensors, la resta els encaminarà segons el procediment habitual del protocol NST-AODV.

Per tant, el funcionament del relay és força simple: es tractarà d'enviar per una interfície el que es rebi per un event. A continuació fem una descripció de la configuració utilitzada:

```
#define PACKET_LINK

configuration NRelayAppC{
}
implementation{

components MainC, LedsC;
     components NRelayC as App;
     components AODV;
     components CC2420PacketC;
     components ActiveMessageC;

     App.Boot        ->   MainC;
     App.Leds        ->   LedsC;
     App.RadioControl     ->ActiveMessageC;
     App.CC2420Packet->   CC2420PacketC;
     App.AODVMsg          ->   AODV.AODVMsg;
     App.AODVPayload      ->   AODV.AODVPayload;
     App.MultiHopMsg      ->   AODV.MultiHopMsg;
     App.SingleHopMsg->   AODV.SingleHopMsg;
     App.SendMsg          ->   AODV.SendMsg;
     App.ReceiveMsg       ->   AODV.ReceiveMsg;
     App.Receive          ->   AODV.Receive;
     App.Intercept        ->   AODV.Intercept;
     App.SendMHopMsg      ->   AODV.SendMHopMsg;
}
```

Veiem que la configuració passa a ser molt més senzilla, i on només s'empren components de comunicació. També podem apreciar com es fa ús del component AODV, tant per al multisalt com per al salt simple.

Seguint amb la descripció de les aplicacions, farem una descripció de les tasques, esdeveniments i funcions programades en el mòdul del relay.

```
event void Boot.booted() Payload co
```

- Event retornat després d'iniciar-se el sensor

```
event void RadioControl.startDone(error_t error)
```

- Event retornat després de l'activació de la rèdio. Retorna SUCCESS si ha estat encesa satisfactòriament, altrament FAIL.

```
event void RadioControl.stopDone(error_t error)
```

- Event retornat després de l'apagada de la ràdio. Retorna SUCCESS si ha estat apagada satisfactòriament, altrament FAIL.

```
void forward_to_gw(node_data_msg_t* p, uint8_t id)
```

- Funció per a l'enviament cap al gateway de les dades de temperatura. P correspon al paquet de dades. Id és l'identificador d'aplicació AM.

```
void forward_conf_to_gw(node_conf_msg_t* p, uint8_t id)
```

- Funció per a l'enviament cap al gateway de les dades de configuració. P correspon al paquet de dades. Id és l'identificador d'aplicació AM.

```
void forward_vib_to_gw(node_vib_msg_t* p, uint8_t id)
```

- Funció per a l'enviament cap al gateway de les dades de vibració. P correspon al paquet de dades. Id és l'identificador d'aplicació AM.

```
void forward_to_son(node_conf_msg_t* p)
```

- Funció per a l'enviament cap al sensor de les dades de configuració. P correspon al paquet de dades.

```
event void SendMsg.sendDone[uint8_t id](message_t* msg, error_t
error)
```

- Event retornat després de la recepció de les dades de salt simple. Msg correspon a la posició del missatge. Error serà

SUCCESS si l'enviament ha estat correcte, FAIL altrament. Id correspon a l'identificador AM.

```
event void SendMHopMsg.sendDone[uint8_t id](message_t* msg, error_t error)
```

- Event retornat després de la recepció de les dades multisalt. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament. Id correspon a l'identificador AM.

```
event message_t* ReceiveMsg.receive[uint8_t id](message_t* msg, void* payload, uint8_t len)
```

- Event retornat després de la recepció d'un paquet de salt simple. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament. Payload apunta a la posició de memòria del contenidor de les dades. Len és la mida del paquet. Id correspon a l'identificador AM.

```
event message_t* Receive.receive[uint8_t id](message_t* msg, void* payload, uint8_t len)
```

- Event retornat després de la recepció d'un paquet de salt simple. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament. Payload apunta a la posició de memòria del contenidor de les dades. Len és la mida del paquet. Id correspon a l'identificador AM.

```
event bool Intercept.forward[uint8_t id](message_t* msg, void* payload, uint8_t len)
```

- Event de la recepció de dades multisalt, no destinades al sensor però que ha d'encaminar. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament. Payload apunta a la posició de memòria del contenidor de les dades. Len és la mida del paquet. Id correspon a l'identificador AM.

4.4.3. Gateway

L'element gateway de la nostra xarxa serà l'encarregat de connectar el PC incrustat amb la xarxa de sensors. Podríem dir que és la interfície de xarxa del PC que recollirà les dades.

TelosB proporciona connexió per a l'enviament de dades a través de l'USB, per tant no serà una tasca àrdua. Malgrat tot, hem de tenir en compte que la interfície de connexió USB és simplex, és a dir, només permet la connexió en un sentit en un mateix instant. Si el sensor vol enviar alguna cosa al PC mentre el PC es vol comunicar amb el sensor, hi haurà una col·lisió en l'USB i les dades es perdran.

En la majoria de casos, la probabilitat de col·lisió serà baixa, i en cas de col·lisió es podrien demanar de nou les dades de temperatura o configuració. Però mentre s'està transferint un fitxer de vibracions, una col·lisió provocaria que el paquet es perdés i que no es poguessin demanar retransmissions fins que no acabés el fitxer (ja que noves peticions provocarien més col·lisions). Les solucions podrien ser: establir un protocol d'accés al medi, sincronitzar els dos elements o traspassar la lògica de programa al sensor.

De totes les solucions, la més senzilla (tant d'implementació com de funcionament) és l'última. Això vol dir que el sensor s'encarregarà de passar per el serial el fitxer de vibració sense cap error i ordenat.

De moment, farem un cop d'ull al funcionament general de l'element gateway així com les tasques, funcions i elements que hauran de ser programats.

```
#include "Node.h"
#define PACKET_LINK

configuration NGatewayAppC {
}
```

```
implementation{

  components MainC, LedsC, new TimerMilliC();
  components NGatewayC as App;
  components AODV;
  components SerialActiveMessageC;
  components ActiveMessageC;
  components new CuaMevaC(message_t, CUA_SIZE);
  components new MapOfIntsC(CUA_SIZE);
  components new TimerMilliC() as PendingTimer;
  components new TimerMilliC() as SerialSendTimer;
  components CC2420PacketC;


  App.CC2420Packet    ->   CC2420PacketC;
  App.Boot      ->   MainC;
  App.Leds            ->   LedsC;

  App.CuaMeva           ->      CuaMevaC;
  App.PendingQueue      ->      MapOfIntsC;
App.PendingTimer      ->      PendingTimer;

  App.RadioControl    ->   ActiveMessageC;
  App.AODVPayload     ->   AODV.AODVPayload;
  App.Receive         ->   AODV.Receive;
  App.Intercept       ->   AODV.Intercept;
  App.SendMHopMsg     ->   AODV.SendMHopMsg;
  App.SubSend         ->   AODV.SendMsg;
  App.SubReceive->   AODV.ReceiveMsg;

  App.SerialSendVib     -
>SerialActiveMessageC.AMSend[AM_VIB_MSG];
  App.SerialReceiveData ->
SerialActiveMessageC.Receive[AM_DATA_MSG];
  App.SerialSendData  -
>SerialActiveMessageC.AMSend[AM_DATA_MSG];
  App.SerialReceiveVib  -
>SerialActiveMessageC.Receive[AM_VIB_MSG];
  App.SerialPacket      ->      SerialActiveMessageC;
  App.SerialSendConf  -
>SerialActiveMessageC.AMSend[AM_NODE_CONF_MSG];
  App.SerialReceiveConf -
>SerialActiveMessageC.Receive[AM_NODE_CONF_MSG];
  App.SerialControl   ->   SerialActiveMessageC;
App.SerialSendTimer   ->      SerialSendTimer;
}
```

Veiem que a banda del component `AODV`, també utilitzarem el component `SerialActiveMessageC`que s'encarregarà de aportar-nos les interfícies necessàries per a la comunicació via USB.

A banda, destacarem els components `CuaMeva` i `MapOfInts`que hem programat per a la gestió de les dades que haurem de passar per el serial.

Tot seguit farem una descripció de les tasques, funcions i events programats en el mòdul.

```
event void Boot.booted()
```

- Event retornat després d'iniciar-se el sensor

```
event void SerialControl.startDone(error_t error)
```

- Event retornat després de l'activació de la interfície de comunicació per l'USB. Retorna SUCCESS si ha estat encesa satisfactòriament, altrament FAIL.

```
event void SerialControl.stopDone(error_t error)
```

- Event retornat després de l'aturada de la interfície de comunicació per l'USB. Retorna SUCCESS si ha estat encesa satisfactòriament, altrament FAIL.

```
event void RadioControl.startDone(error_t error)
```

- Event retornat després de l'activació de la ràdio. Retorna SUCCESS si ha estat encesa satisfactòriament, altrament FAIL.

```
event void RadioControl.stopDone(error_t error)
```

- Event retornat després de l'aturada de la ràdio. Retorna SUCCESS si ha estat encesa satisfactòriament, altrament FAIL.

```
event void PendingTimer.fired()
```

- Event d'activació del temporitzador de paquets pendents per enviar.

```
event void SerialSendTimer.fired()
```

- Event d'activació del temporitzador d'enviament de dades per el serial.

```
event void SerialSendVib.sendDone(message_t *msg, error_t error)
```

- Event retornat després de l'enviament de les dades de vibració pel serial. Msg correspon a la posició del missatge.

Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament.

```
event void SerialSendConf.sendDone(message_t *msg, error_t error)
```

- Event retornat després de l'enviament de les dades de configuració pel serial. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament.

```
event void SerialSendData.sendDone(message_t *msg, error_t error)
```

- Event retornat després de l'enviament de les dades de temperatura per el serial. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament.

```
event message_t* SerialReceiveData.receive(message_t *msg, void *payload, uint8_t len)
```

- Event retornat després de la recepció de les dades de temperatura pel serial. Msg correspon a la posició del missatge. Payload correspon a la posició de memòria corresponent al contenidor de les dades. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament.

```
event message_t* SerialReceiveVib.receive(message_t *msg, void *payload, uint8_t len)
```

- Event retornat després de la recepció de les dades de vibració per el serial. Msg correspon a la posició del missatge. Payload correspon a la posició de memòria corresponent al contenidor de les dades. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament.

```
event message_t* SerialReceiveConf.receive(message_t *msg, void *payload, uint8_t len)
```

- Event retornat després de la recepció de les dades de configuració per el serial. Msg correspon a la posició del missatge. Payload correspon a la posició de memòria

corresponent al contenidor de les dades. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament.

```
event message_t* Receive.receive[uint8_t id](message_t* msg, void* payload, uint8_t len)
```

- Event retornat després de la recepció d'un paquet de salt simple. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament. Payload apunta a la posició de memòria del contenidor de les dades. Len és la mida del paquet. Id correspon a l'identificador AM.

```
event bool Intercept.forward[uint8_t id](message_t* msg, void* payload, uint8_t len)
```

- Event de la recepció de dades multisalt, no destinades al sensor però que ha d'encaminar. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament. Payload apunta a la posició de memòria del contenidor de les dades. Len és la mida del paquet. Id correspon a l'identificador AM.

```
event void SendMHopMsg.sendDone[uint8_t id](message_t* msg, error_t error)
```

- Event retornat després de la recepció de les dades multisalt. Msg correspon a la posició del missatge. Error serà SUCCESS si l'enviament ha estat correcte, FAIL altrament. Id correspon a l'identificador AM.

5. Proves i resultats

A continuació es descriuran els procediments seguits per quantificar els paràmetres de la transmissió de les dades utilitzant el procés de *polling* per a la petició de dades vibració, i el mecanisme de finestra lliscant per a la transmissió de les captures dels acceleròmetres.

Les proves s'han dut a terme en un laboratori, amb sensors TelosB sense sensors reals. L'única variació, serà que els sensors donaran l'estat de la bateria enlloc de temperatura i valors de soroll per a les vibracions. En tot moment s'ha intentat reproduir l'entorn que es trobarà en un tren, ajustant la potència de transmissió a les distàncies que es poden aconseguir en un laboratori.

5.1. Topologia de la xarxa de proves

La topologia de les proves serà el més similar possible a l'entorn real. Es procedirà a buscar una distància similar a la que hi haurà entre cotxes del tren i es transmetrà a màxima potència. Per als nodes sensors, es baixarà la potència i es col·locaran a la vora dels relays. Ens interessarà focalitzar les observacions entre relays, ja que en proves anteriors s'ha comprovat que els enllaços entre sensors i relays són força estables. Els resultats ens mostraran els següent paràmetres del funcionament de la xarxa:

- Temps de transmissió d'un sensor.
- Paquets perduts (ja sigui per esvaïments a nivell físic o pèrdues a nivell de transport).
- Temps total d'obtenció de les dades del procés de *polling*.
- Possibles *bugs* com a conseqüència del funcionament continu.

Les proves es duran a terme amb una variació de la finestra del protocol d'aplicació per a l'obtenció de les dades de vibració. Un

estudi posterior ens ajudarà a estimar la finestra que millor s'adeqüi al cas concret del tren.

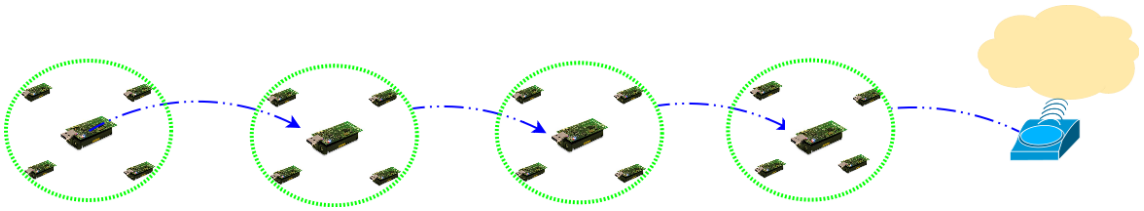


Figura 32 - Topologia de la xarxa amb 4 salts

5.2. Procediment de prova

El procediment que es durà a terme serà construir una maqueta de la xarxa que acabarà funcionant en el tren. Com hem dit, els relays es col·locaran en distàncies grans, similars a les que s'hauran de suportar al tren, i la potència de transmissió estarà ajustada al màxim. Pel que fa als nodes sensors, al considerar un enllaç estable en el tren, en aquest cas es reduirà la potència i la distància entre el relay i el sensor serà petita.

Cada sensor estarà en un estat de repòs i es despertarà cada 60 segons. En el moment de despertar-se enviarà un missatge comunicant la temperatura del sensor (en el cas d'aquesta prova transmetrà l'estat de la bateria). Durant 3 segons el sensor romandrà despert esperant un missatge de petició de vibració. En el cas que no es produeixi, tornarà a dormir. Un cop rebut el missatge de petició de vibració, el sensor enviarà una confirmació de la captura.

Per la part del gateway, aquest durà a terme un procediment de *polling*, el qual anirà fent peticions de la captura vibració a mesura que se li enviïn les temperatures. Un cop hagi obtingut totes les confirmacions de vibració començarà el procediment de petició de fitxers de vibració.

Un cop inicïi la transmissió del fitxers de vibració el gateway mantindrà uns temporitzadors per tal de mesurar el temps que es triga des de l'inici fins a la finalització que es desaran en un missatge de text. Pel que fa als sensors, un cop finalitzada la transmissió, enviaran un missatge anunciant la quantitat de paquets requerits per la transmissió.

5.3. Observacions

Finalment es van dur a terme tres tipus de proves diferents, amb procediments iguals, però canviant la mida de la finestra. Les mides han estat 6, 10 i 14 paquets.

La implementació de la maqueta es va fer utilitzant les adreces més altes més a prop del gateway. Per tant tenim que les adreces de 70 cap amunt són les més pròximes al gateway i les menors a 20 són les més allunyades.

Per a totes les proves, s'ha fet servir la mateixa mida de fitxer de vibració, 64 paquets de 96 bytes.

5.4. Resultats

En la Taula 5 veiem els paquets enviats per a la transmissió d'un fitxer de vibració sencer en mitja per cada sensor.

Sensor	Paquets enviats (finestra 6 paquets)	Paquets enviats (finestra 10 paquets)	Paquets enviats (finestra 14 paquets)
10	92	71	87
11	93	70	86
12	92	70	87
13	94	72	85
20		76	84
21	86	76	82
22	86	76	82
23	88	78	86
30	89	76	82
31	87	76	83
32	91	76	82
33	88	70	83
40	84	78	79

41	83	70	77
42	83	71	79
43	81	72	78
50	74	75	83
51	77	73	81
52	77	76	88
53	76	73	88
60	69	68	74
61	70	66	75
62	72	68	74
63	69	68	78
70	68	67	72
71	71	69	71
72	67	68	71
73	67	67	71
Mitja	**80**	**72**	**80**

Taula 5 - Taula de resultats totals

Els resultats ens mostren que la finestra de 10 paquets és la més eficient, i això es dóna per dos motius:

1. La finestra de 6 paquets utilitza més missatges de control per a les confirmacions i això fa que la transmissió en molts moments estigui infrautilitzada.
2. La finestra de 14 paquets envia massa informació entre confirmacions i això fa que s'hagin de demanar moltes més retransmissions i conseqüentment augmentarà el missatges de control.

En la taula també es pot apreciar com els nodes més pròxims al gateway requereixen menys paquets per completar la transmissió. Això és degut a què necessiten menys salts que els més allunyats, i per tant la probabilitat d'error en el paquet és menor.

Aquestes efectes també es veuen reflectits en el temps de transmissió dels fitxers de vibració. En la Taula 6 veiem com la mitja per fitxer de vibració és menor en la finestra de 10 paquets.

Finestra(paquets)	Temps (segons)
6	29,59
10	23,09

14	28,68

Taula 6 - Mitjana de temps d'enviament de fitxer de vibració

Finalment aquestes mesures es veuen reflectides també en el temps de polling, ja que a més temps per fitxer, més temps es trigarà a interrogar a tots els nodes (Taula 7).

Finestra (paquets)	Temps (minuts)
6	23,24
10	17,08
14	23,68

Taula 7 - Mitjana de temps de polling per finestra

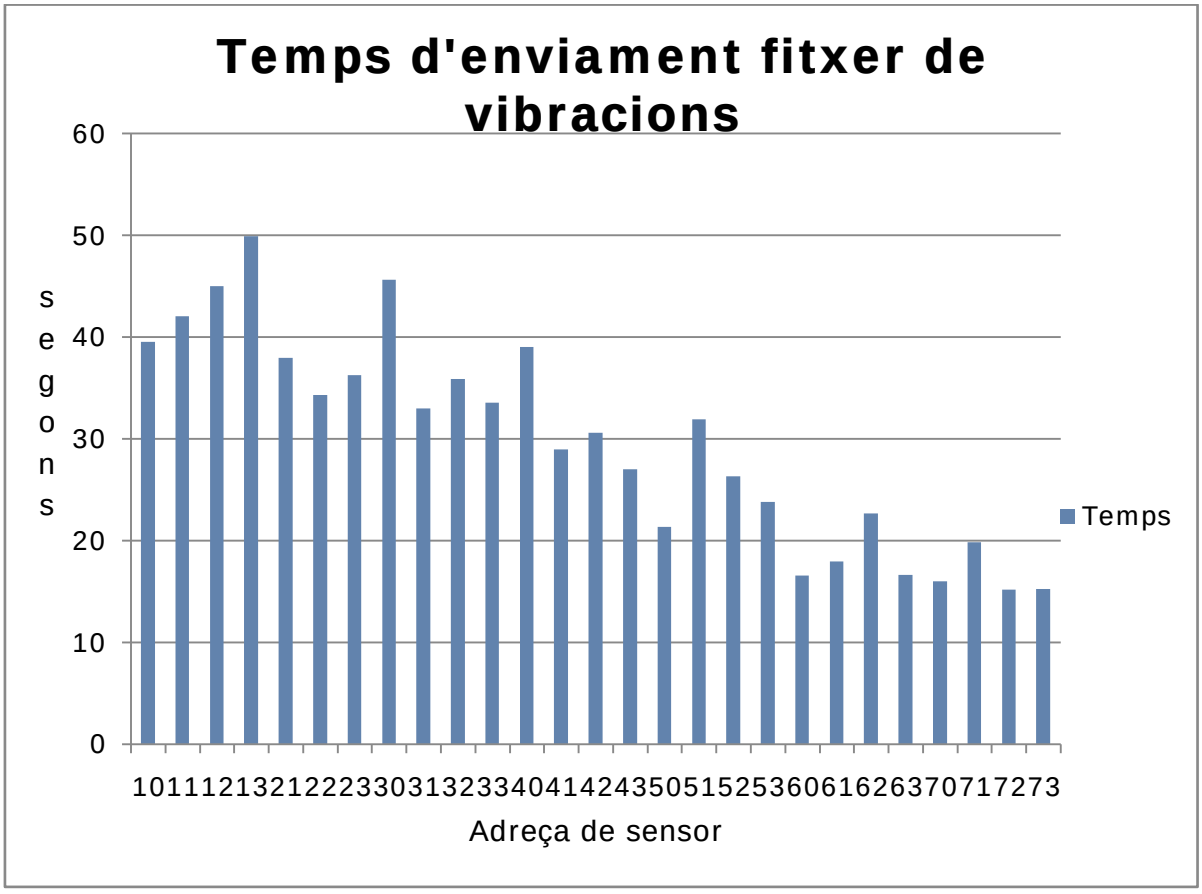


Figura 33 - Temps d'enviament fitxer de vibracions per sensor (finestra de 6 paquets)

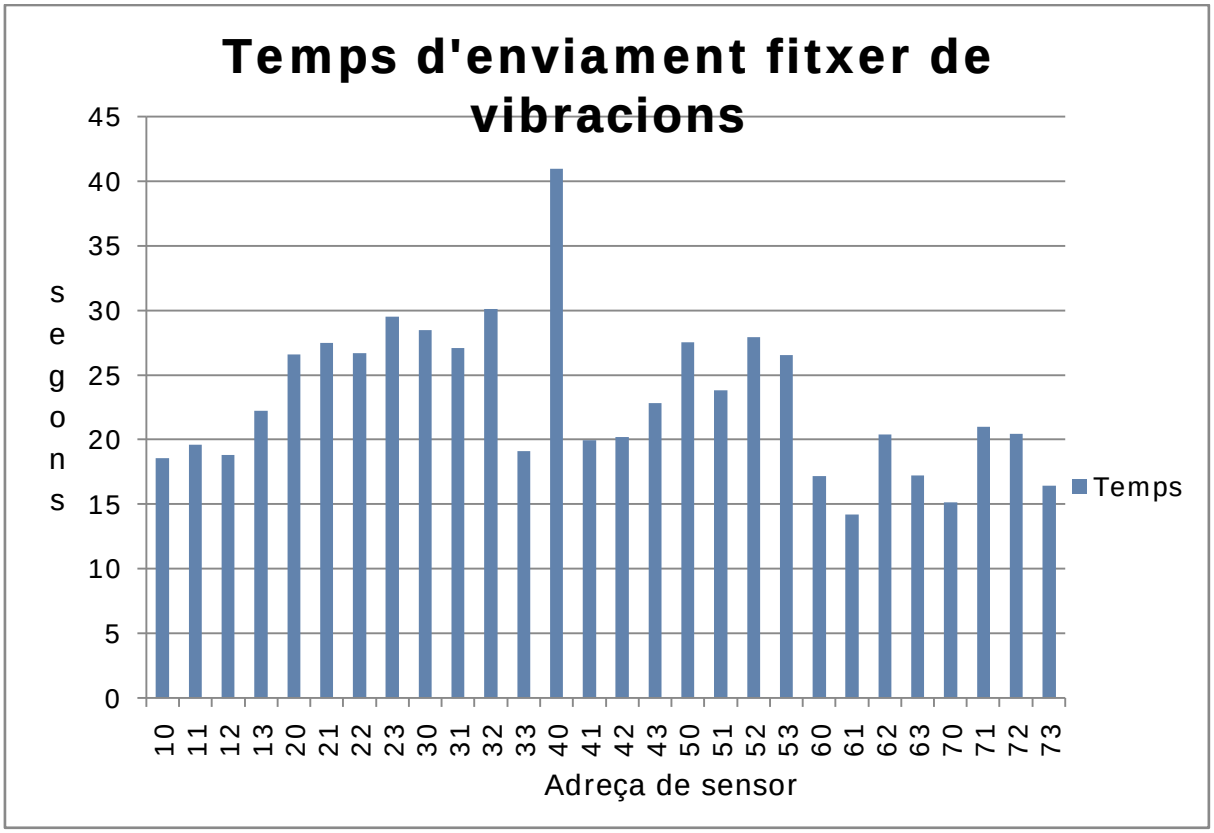


Figura 34 - Temps d'enviament fitxer de vibracions per sensor (finestra de 10 paquets)

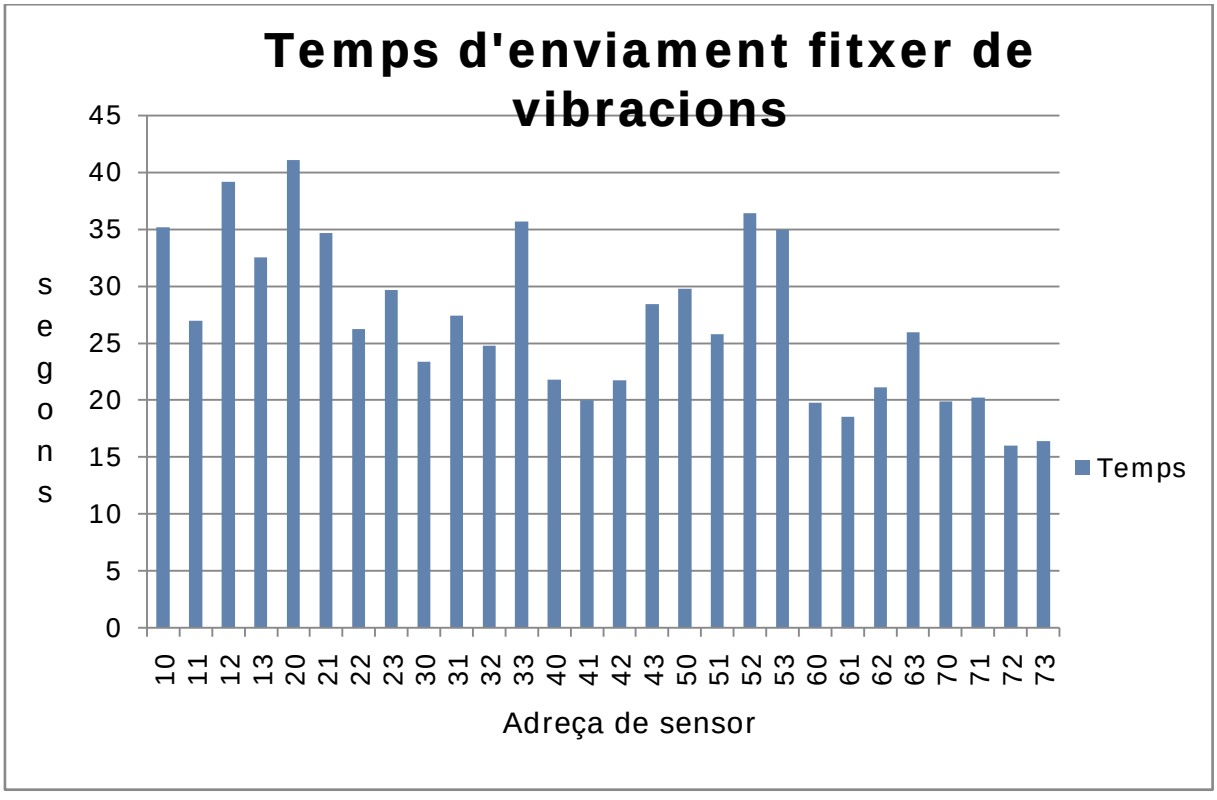


Figura 35 - Temps d'enviament fitxer de vibracions per sensor (finestra de 14 paquets).

6. Conclusions

Aquesta memòria documenta la implementació de la xarxa de sensors del projecte de monitorització d'un entorn ferroviari. S'han complert els objectius esmentats al principi del document i que resumirem en les següents línies:

Obtenir dades per a la monitorització d'elements del tren: Com hem vist, partíem de la premissa que era necessari obtenir certes dades de diferents parts del tren. Com la instal·lació de nous elements és realment difícil, es va procedir a pensar en una implementació amb sensors inalàmbrics. Finalment hem acabat desenvolupant uns elements sensors que monitoritzen la temperatura de manera periòdica i participen en un procés de sondeig per tal d'enviar fitxers de vibració.

Enviament de dades fins a un punt de recol·lecció: El manteniment dels sensors ha estat una restricció força gran, ja que s'han de minimitzar les accions de reparació, substitució o reprogramació dels sensors. Per aquest motiu s'ha construït una xarxa troncal pera què uns elements relay puguin enviar, mitjançant multisalt, les dades des de els sensors fins al gateway.

Per a què aquest enviament sigui fiable, s'ha implementat un mecanisme de finestra lliscant en els extrems de la xarxa: sensors i gateway. Aquest mecanisme permet que es recuperin paquets perduts i que la informació arribi ordenada al seu destí.

Configuració dels paràmetres d'obtenció de les dades: També s'han implementat mecanismes de configuració per a què des del gateway es puguin enviar diferents paràmetres que permetin configurar diferents procediments per tal de mesurar les dades de maneres diferents.

Per finalitzar aquesta secció enumerarem algunes conclusions per sintetitzar les idees presentades en el present document i també algunes apreciacions derivades de l'experiència de l'autor.

Versatilitat del protocol IEEE 802.15.4: Durant tot el document hem vist com el protocol 802.15.4 de l'IEEE és molt adequat per a la implementació que hem dut a terme. Si en un inici la idea era només transportar poca informació de temperatura i vibració, a mesura que evolucionàvem amb el projecte hem anat veient com mantenint l'ús del protocol podem solucionar tots els problemes que ens han anat sorgint. Finalment hem acabant construint una xarxa troncal amb aquest protocol i diferents mecanismes per a una transferència d'informació eficient.

TinyOS com a sistema operatiu, suite d'aplicacions i entorn de programació: El primer que sorprèn del sistema TinyOS es la seva comunitat, ja que malgrat que és un projecte de codi obert, manté un codi d'alta qualitat amb una documentació excel·lent que facilita la utilització a un forani en la temàtica.

Com a sistema operatiu esdevé totalment transparent arribant a convertir-se simplement en una suite d'aplicacions disponibles per a que l'usuari les utilitzi.

Com hem dit a l'inici del document, no es pot parlar de TinyOS sense parlar del llenguatge de programació NesC. Malgrat la dificultat inicial d'haver de programar en un entorn orientat a esdeveniments, finalment gràcies als tutorials i els ajuts publicats per la comunitat, es converteix en un hàbit.

El fet que el sistema en general i el llenguatge en particular ja estiguin pensats per aquests tipus d'entorns, fa que el haver de programar aquestes aplicacions tant específiques i destinades a les comunicacions i l'estalvi d'energia sigui una mica més senzill. Si

pensem en un llenguatge més habitual (com C), es podria fer molt farragós haver d'escriure events, sockets i estalviar energia.

Monitorització d'entorns mitjançant sensors: Tot i que no és l'objectiu d'aquest document, hem pogut veure com es poden instal·lar diferents tipus de sensors als sensors. En el cas que ens ocupa hem monitoritzat les vibracions i la temperatura de certs elements d'un tren, però ha quedat palès que la comunicació és efectiva en entorns desfavorables, i que mitjançant el desenvolupament d'aplicacions específiques per a cada entorn es poden instal·lar sensors en gairebé qualsevol lloc.

Xarxes multisalt com a solució a la manca de cobertura: Una de les mancances del protocol 802.15.4 és el seu poc abast, a més a més, hem vist que l'entorn electromagnètic que presenta el ferrocarril encara empitjora aquest aspecte. El multisalt esdevé crucial per a poder comunicar els dos extrems de la xarxa. De la mateixa manera també esdevé crucial el poder modificar cert paràmetres del protocol com les mètriques. Un altre èxit del codi obert.

Mecanisme de finestra lliscant: El fet d'haver d'enviar un fitxer amb una mida gran fa que s'hagi d'utilitzar un mecanisme per a que la transmissió sigui correcta i poder recuperar el fitxer en l'altre extrem. A més a més, nosaltres ens hem trobat amb l'inconvenient d'haver d'enviar a l'ordinador encarregat de capturar les dades, els paquets en ordre i sense errors.

És per això que hem dissenyat el mecanisme de finestra lliscant. En un inici sembla una tasca força complicada donada la limitació dels sensors i el fet que la memòria en que treballem sigui estàtica i no es pugi reservar i alliberar memòria en temps d'execució. Malgrat aquests inconvenients, hem ideat algunes estructures de dades que ens han facilitat força la tasca.

Experiència de treball amb l'empresa: Per últim però no menys important, l'experiència de treballar fent una aplicació que serà utilitzada en un entorn real per una empresa com ALSTOM canvia de manera radical el desenvolupament de tot el projecte. Primer de tot, és el fet d'haver d'adaptar-nos a les peticions del client, tant en mètodes de treball com les funcionalitats de l'aplicació. Per altra banda, el contacte amb el món real ens fa descobrir la necessitat de ser acurats i fer un projecte de qualitat. Per últim remarcar la experiència enriquidora que hem pogut viure tant per la vessant acadèmica com la professional.

7. Treball futur

Aquest treball documenta la part de la implementació de les aplicacions dels sensors d'un projecte molt més gran i més ambiciós. A més a més de tot el treball realitzat durant més d'un any, encara restarien moltes coses per dur a terme, en aquest apartat enumerarem les que el autor creu que es poden qualificar de línies futures:

Instal·lació en l'entorn real: La finalització del projecte va acabar amb les proves necessàries per a fer unes estimacions del temps en funcionament i les pèrdues que poden existir durant l'execució dels processos. Malgrat que es van fer en un entorn simulant el real i que durant tot el desenvolupament s'han dut a terme diferents proves en entorns reals, seria important veure com es comporten els sensors i sensors en la implementació final.

Agregació de sensors: Com s'ha apuntat en aquest document, donat que el desenvolupament que s'estava duent a terme, la agregació dels sensors als relay s'ha fet en tot moment de manera estàtica. Tot i que es una funcionalitat que no afecta massa al funcionament del sistema, seria bo poder comprovar que aquesta agregació aporta millores a la actual.

Mecanisme de control i alertes de manteniment: El codi està fet de manera que les exigències de configuració dels sensors per part de l'usuari puguin ser satisfetes sense haver d'aplicar grans canvis. Com ja s'ha explicat amb anterioritat, hi ha un entorn d'aplicació destinat a les aplicacions i on es poden introduir fàcilment canvis per adaptar el funcionament a les exigències de l'usuari.

Referències

[1] Perspective CMMS. (2007) Perspective CMMS. [Online]. http://www.pemms.co.uk/Condition_Monitoring.html

[2] Crossbow, "TelosB Datasheet," Crossbow, San Jose, Datasheet 2008.

[3] Crossbow, "MicaZ Datasheet," Crossbow, San Jose, Datasheet 2008.

[4] L. De Nardis and M. Di Benedetto, "Overview of the IEEE 802.15.4/4a standards for low data rate Wireless Personal Data Networks," *4th WORKSHOP ON POSITIONING, NAVIGATION AND COMMUNICATION*, 2007.

[5] B. Krishnamachari, *Networking Wireless Sensors*.: Cambridge University Press, 2005.

[6] D. Niculescu, "Communications Paradigms for Sensor Networks," *IEEE Communications Magazine*, pp. 116-122, March 2005.

[7] C. Perkins, "Ad hoc On-Demand Distance Vector Routing".

[8] C. Gomez, P. Salvatella, O. Alonso, and J. Paradells, "Adapting AODV for IEEE 802.15.4 Mesh Sensor Networks: Theoretical Discussion and Performance Evaluation in a Real Environment," *Proceedings of the 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)*.

[9] P. Levis, *TinyOS Programming*.

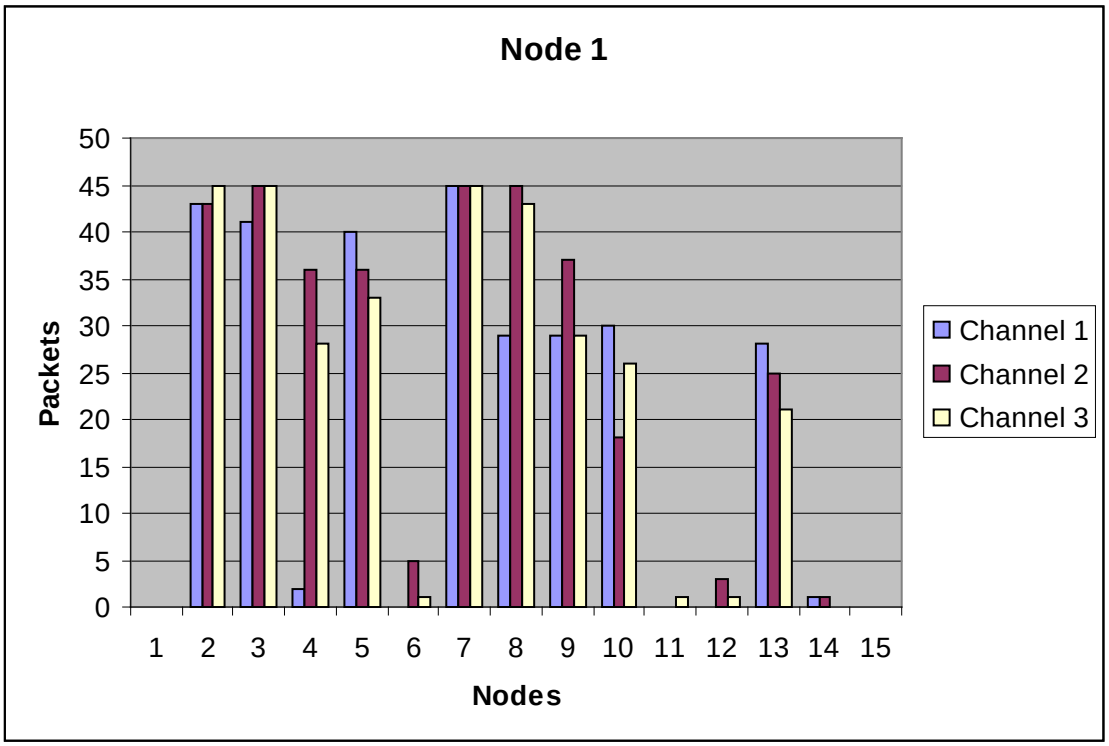
[10] Colegio Oficial de Ingenieros de Telecomunicación, *Elementos Técnicos para la Gestión de Frecuencias en Espacios Complejos: Entornos Ferroviarios*, J. F. Fernández, Ed.

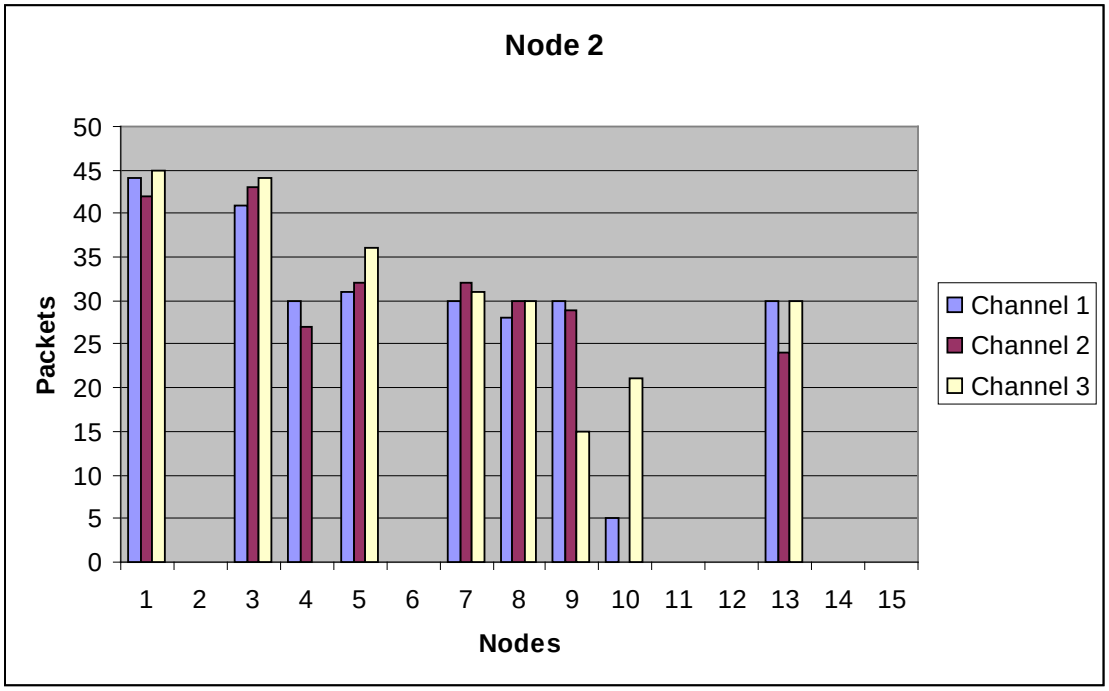
[11] P. Buonadonna, J. Hill, and D. Culler, *Active Message Communication for Tiny Networked Sensors.*, 2000.

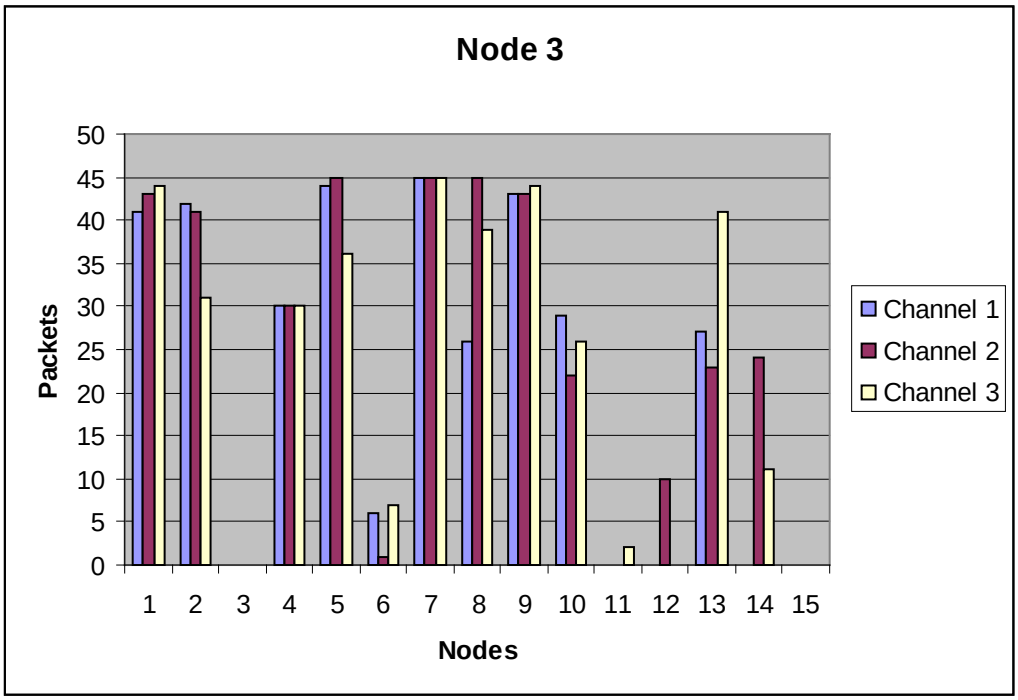
[12] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, 2002.

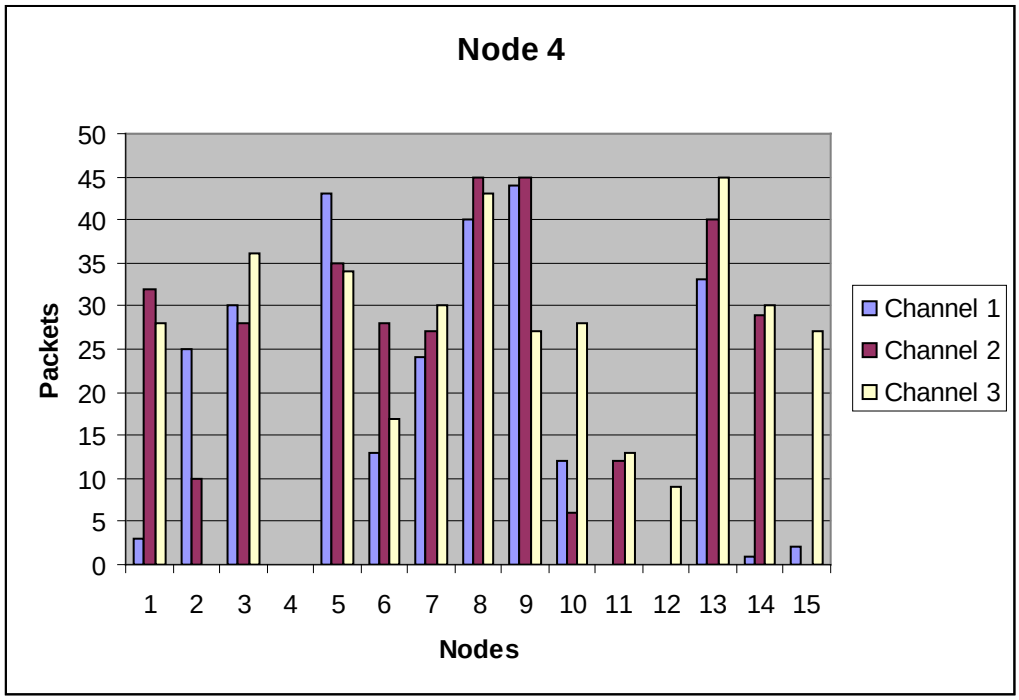
Apèndix A: Resultats complerts comparant diferents canals.

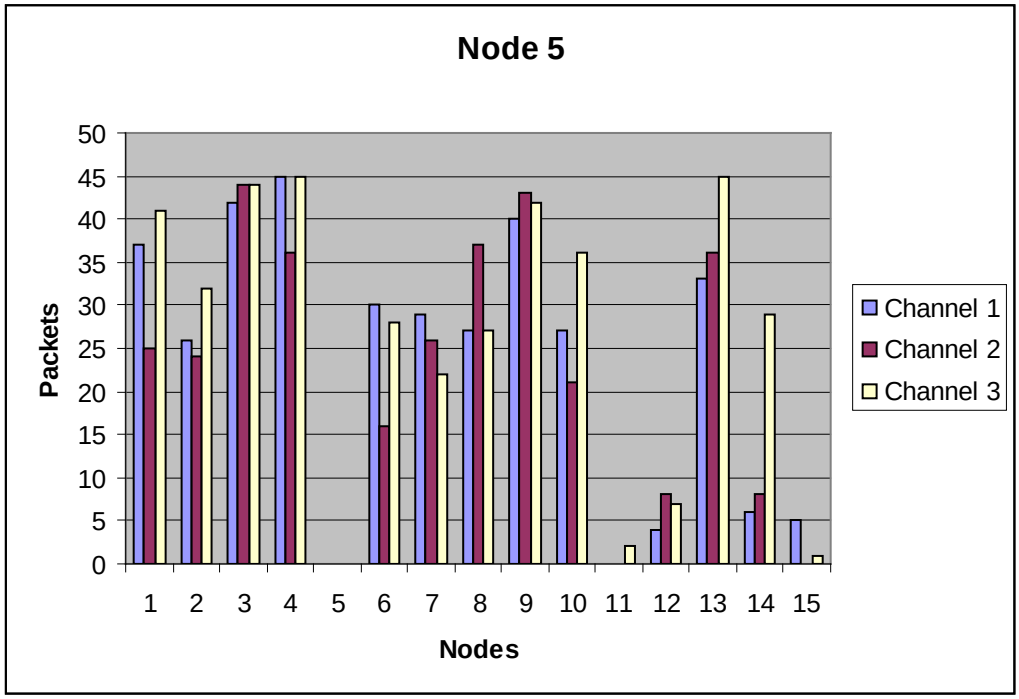
En aquest apèndix es mostren els gràfics dels paquets rebuts per els 15 nodes en les proves dutes a terme als Tallers d'ALSTOM a Sant Andreu. Els resultats mostren la cobertura que hi havia per la configuració que vam construir. El nombre màxim de paquets rebuts d'un altre node es 45.

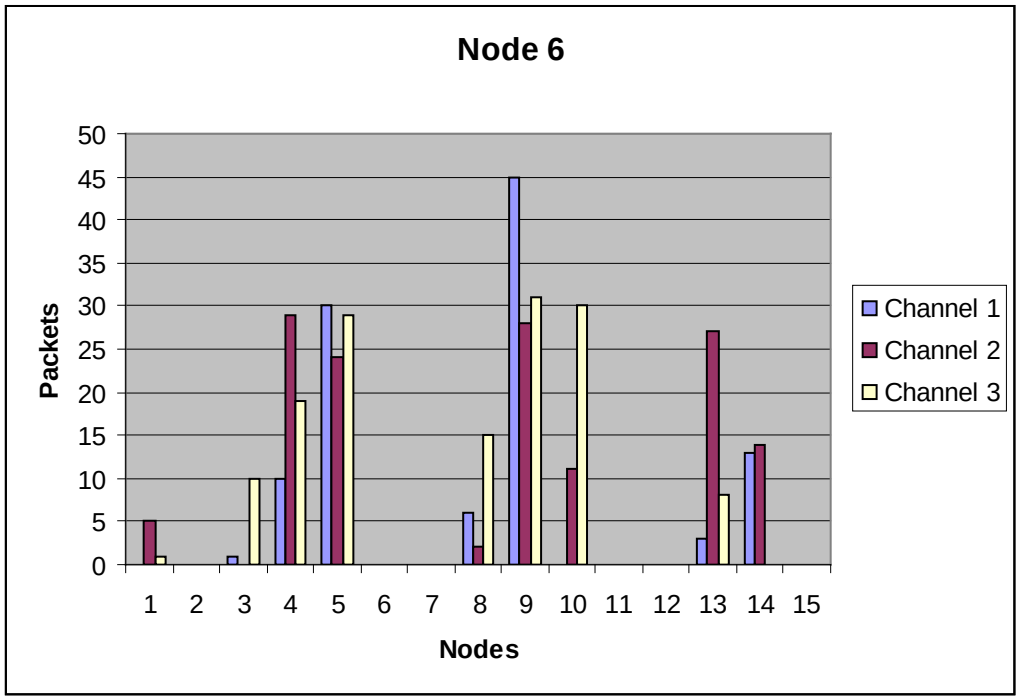


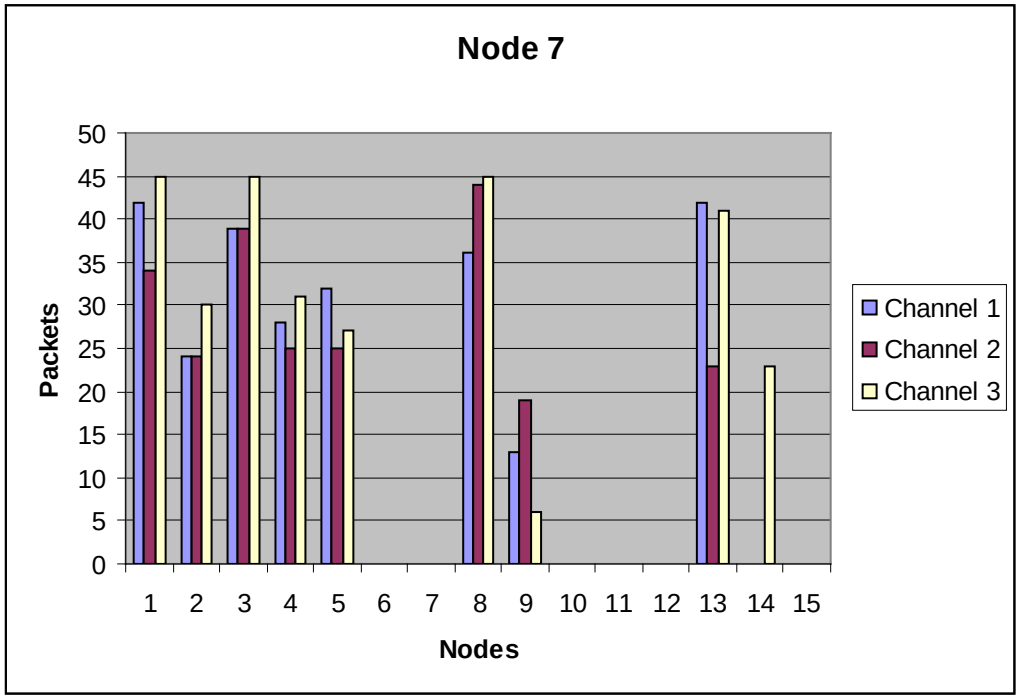

Node 2
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

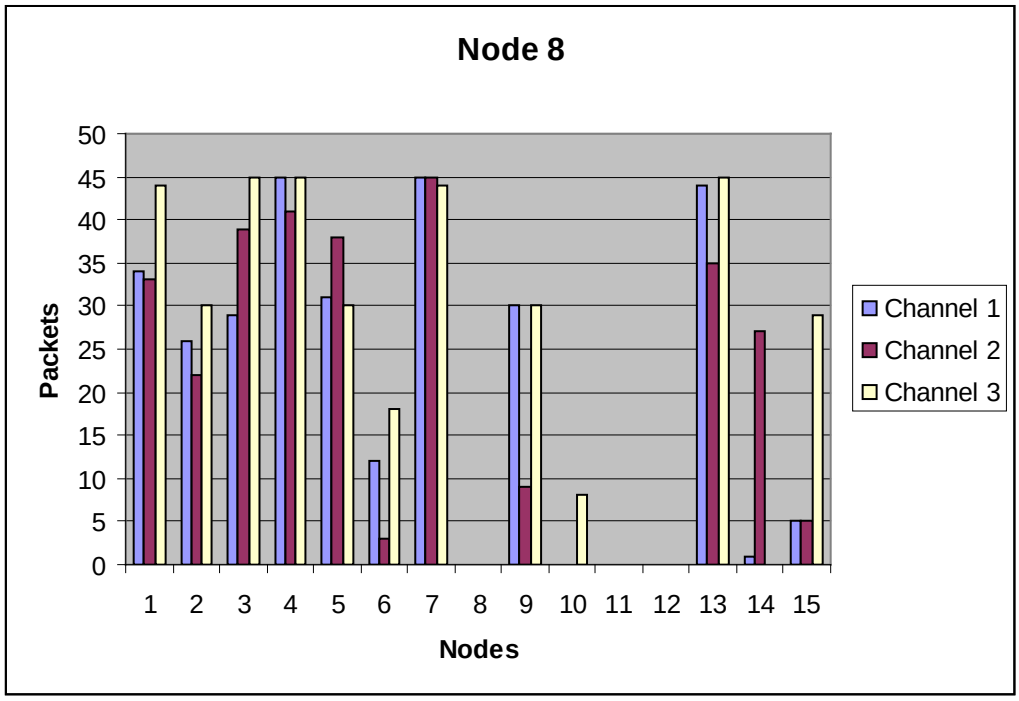

Node 3
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

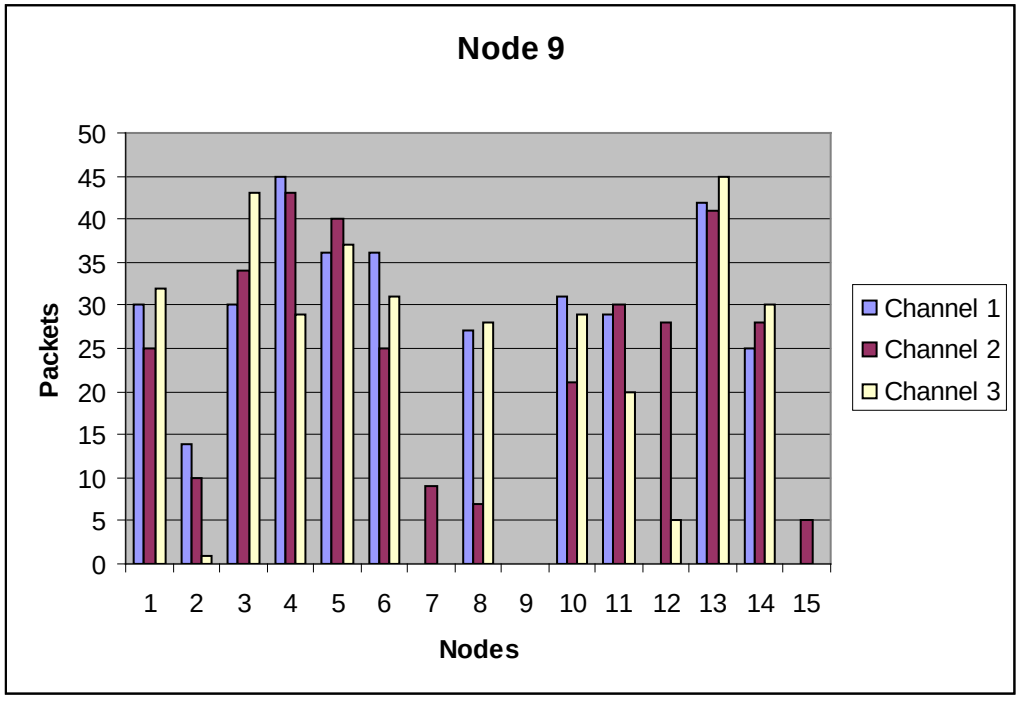

Node 4
Packets
Nodes
Channel 1
Channel 2
Channel 3

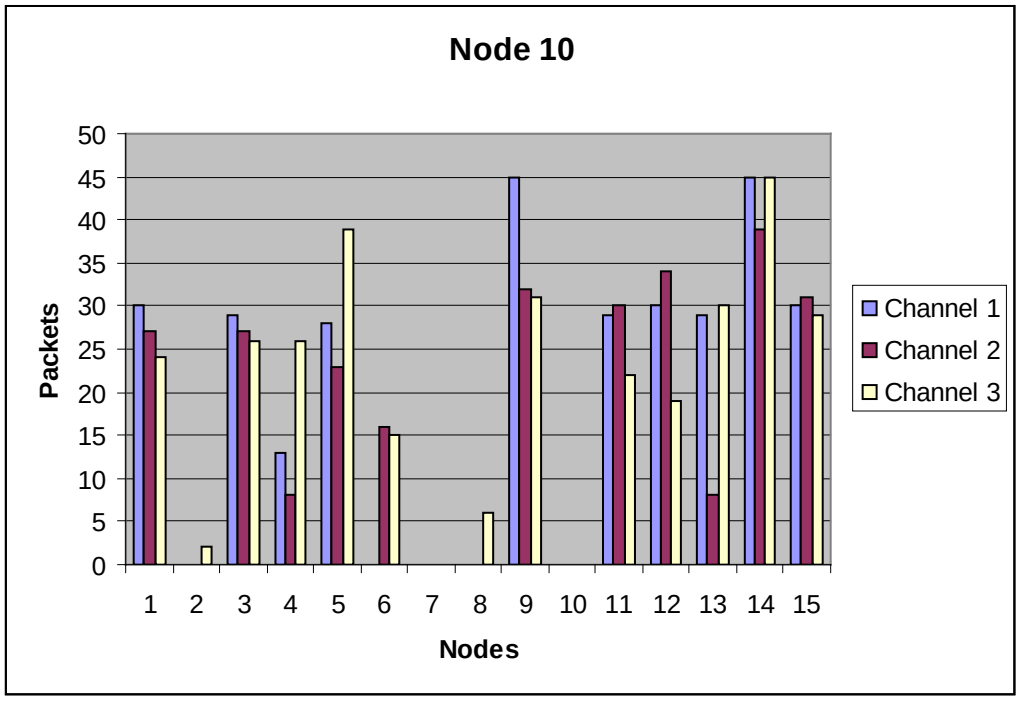

Node 5
Packets
Nodes
Channel 1
Channel 2
Channel 3

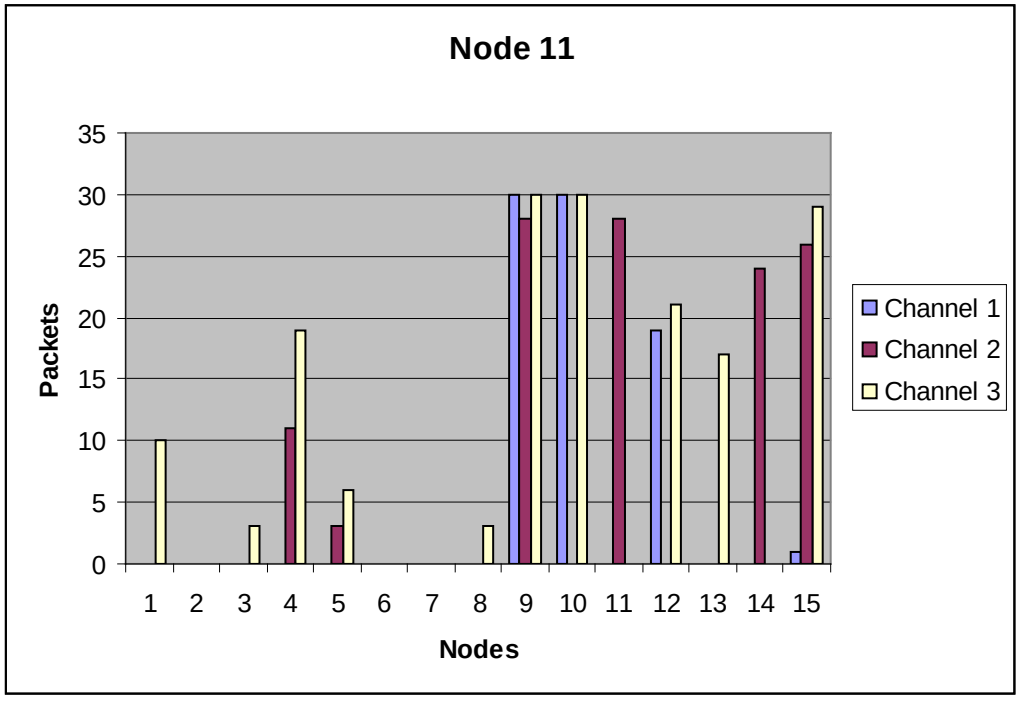

Node 6
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

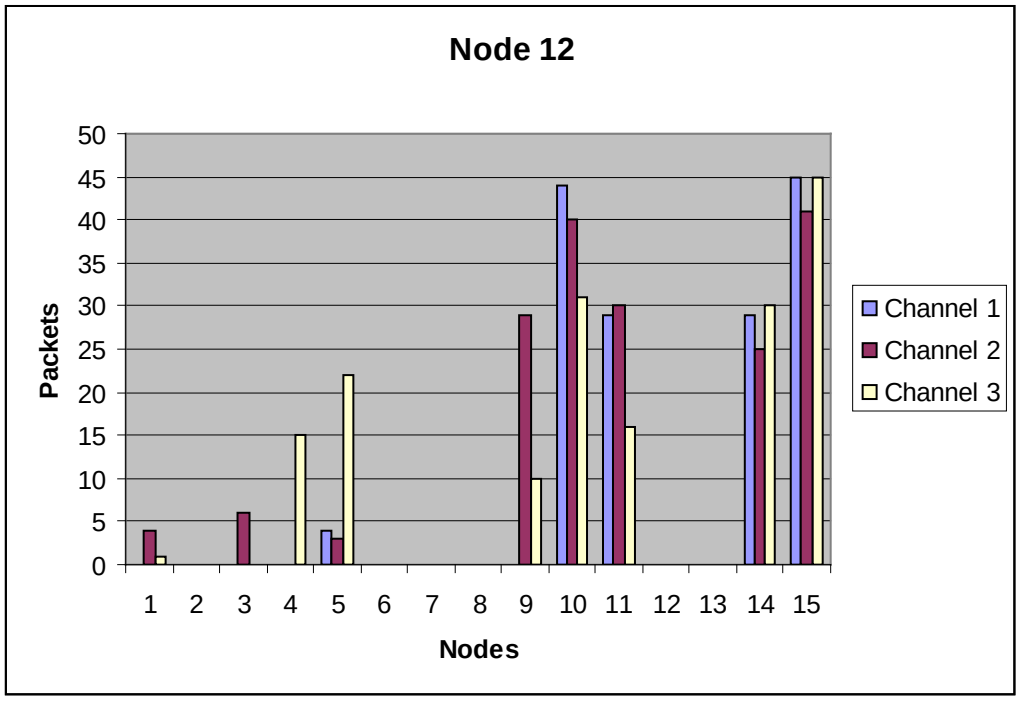

Node 7
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

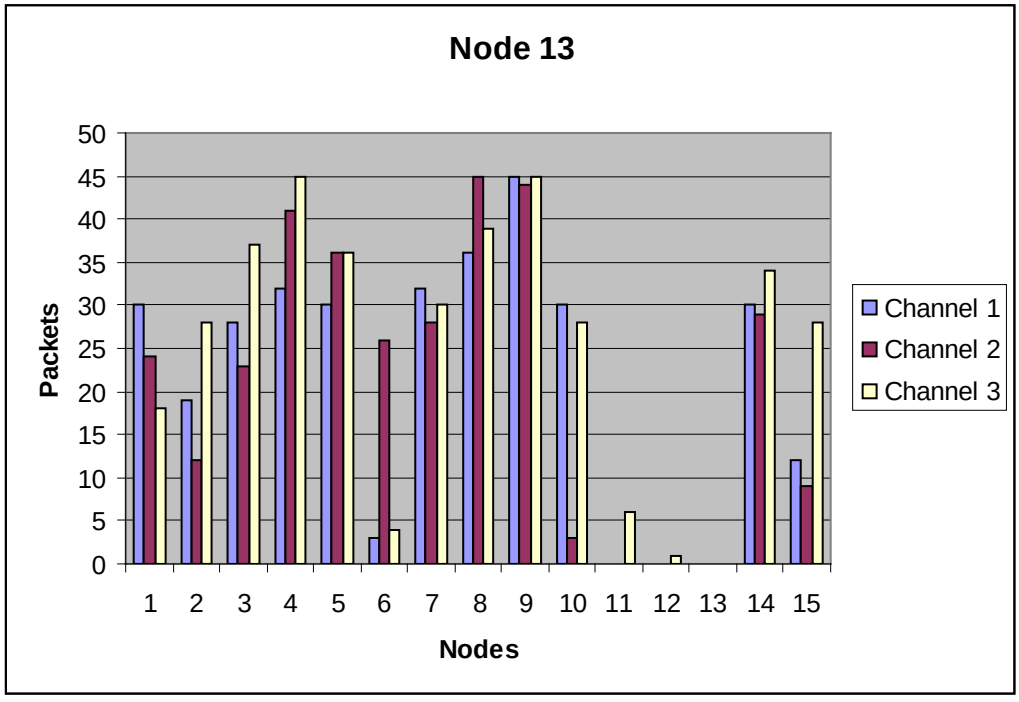

Node 8
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

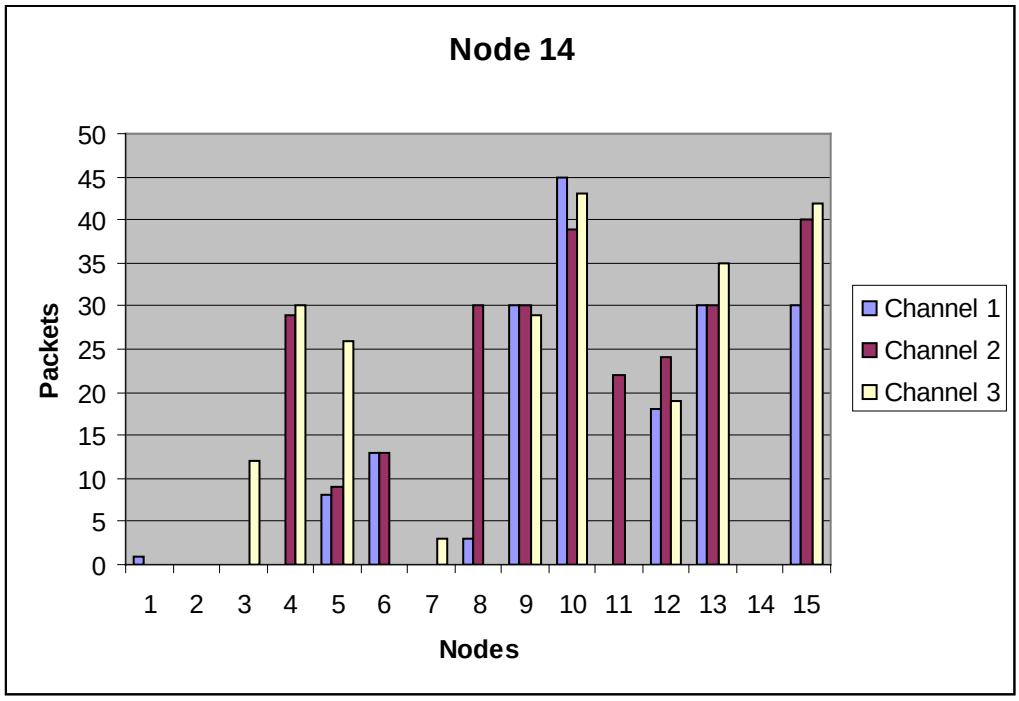

Node 9
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

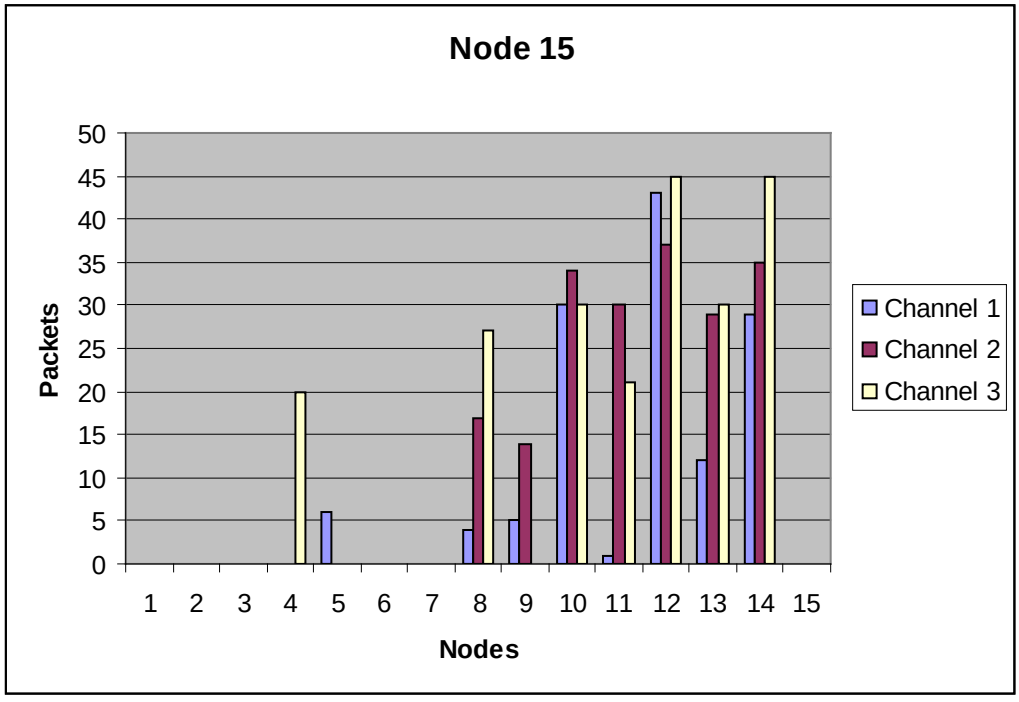

Node 10
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3


Node 11
Packets
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3


Node 12
Packets
Nodes
Channel 1
Channel 2
Channel 3

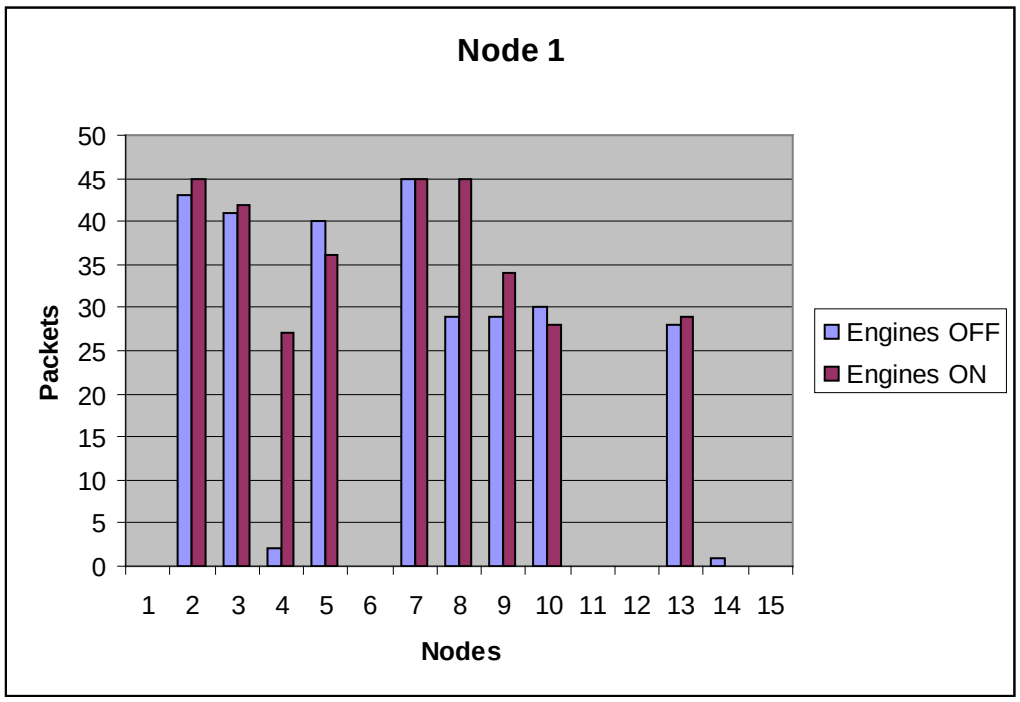

Node 13
Packets
Nodes
Channel 1
Channel 2
Channel 3

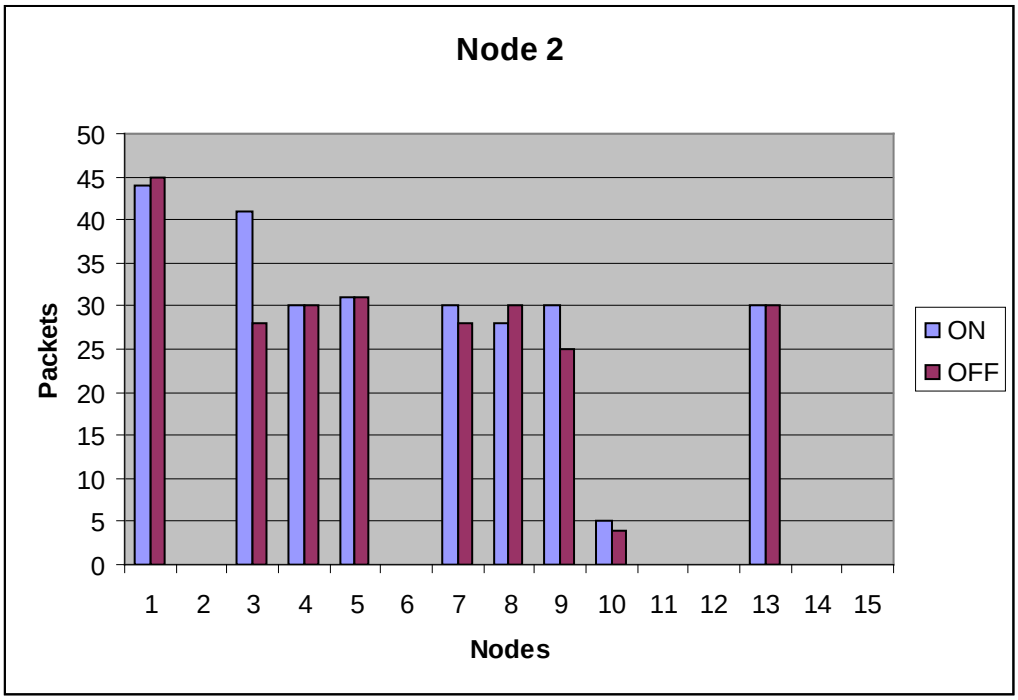

Node 14
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

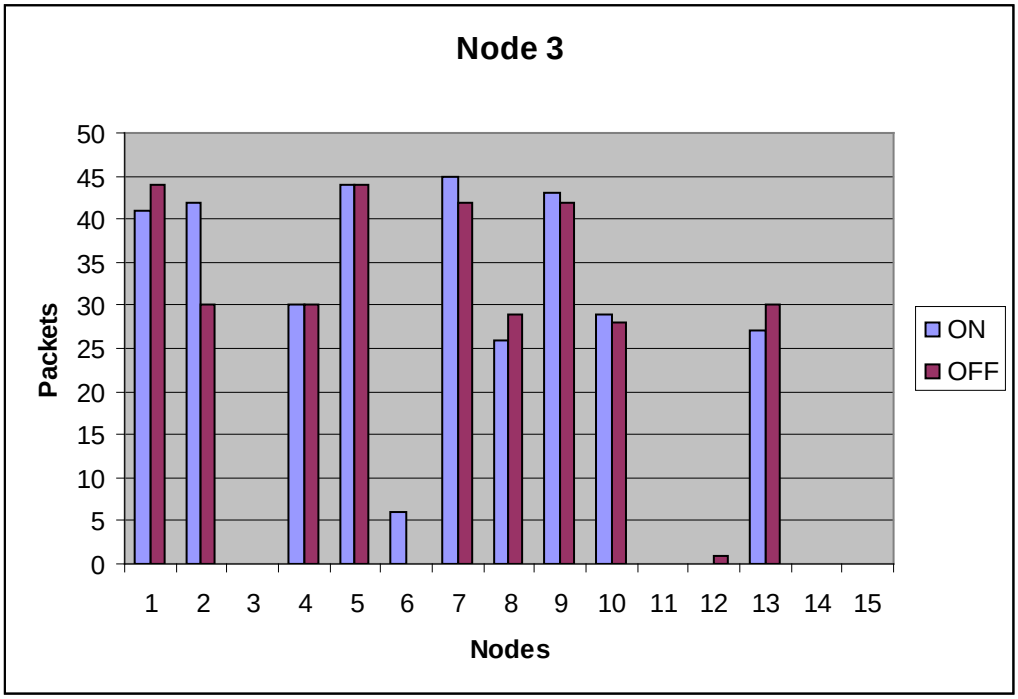

Node 15
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
Channel 1
Channel 2
Channel 3

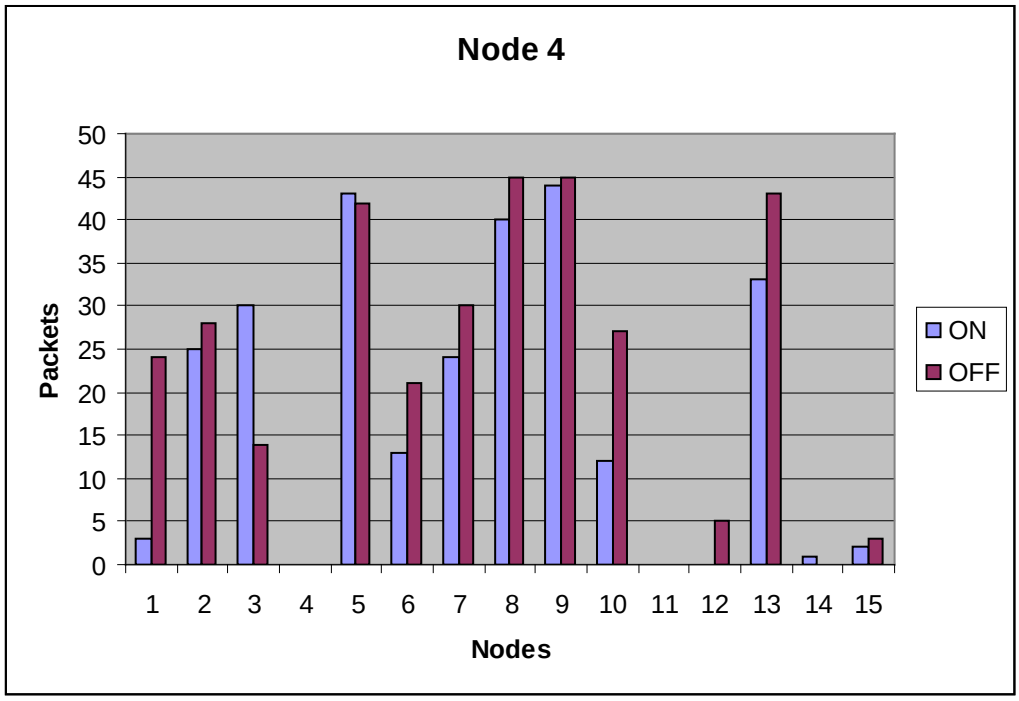
Apèndix B: Resultats complerts comparant la influència dels motors del tren.

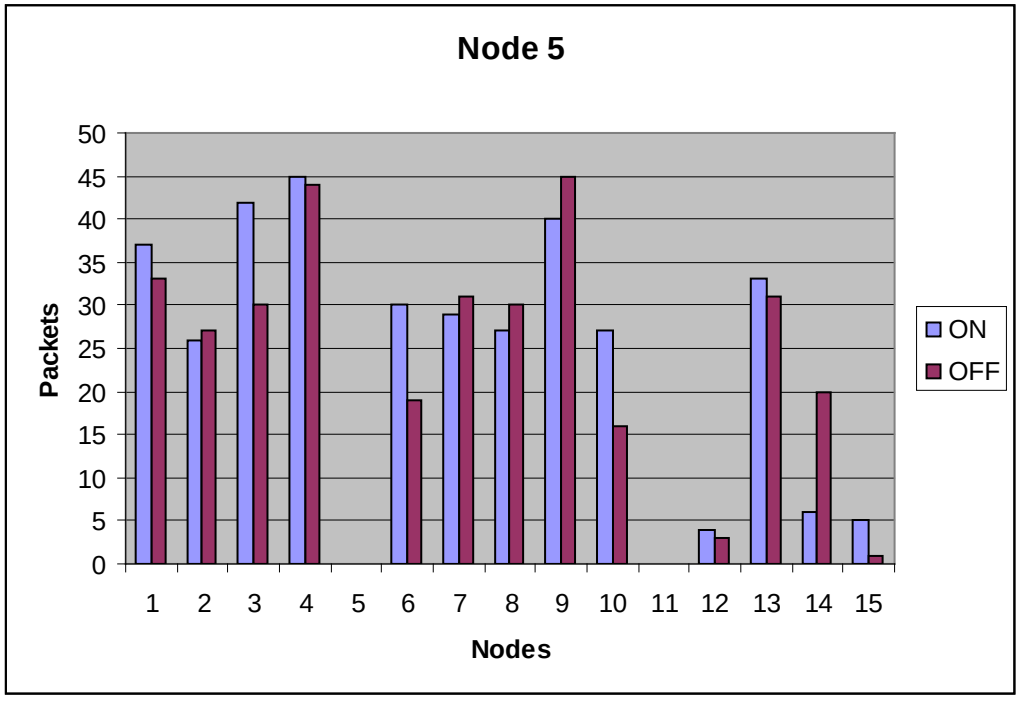
Aquest apèndix conté els gràfics dels paquets rebuts per els 15 nodes de diferents fonts utilitzant el canal 11 i comparant la influència dels motors encesos i apagats en les proves dutes a terme als Tallers d'ALSTOM a Sant Andreu. El nombre màxim de paquets rebuts d'un altre node són 45.

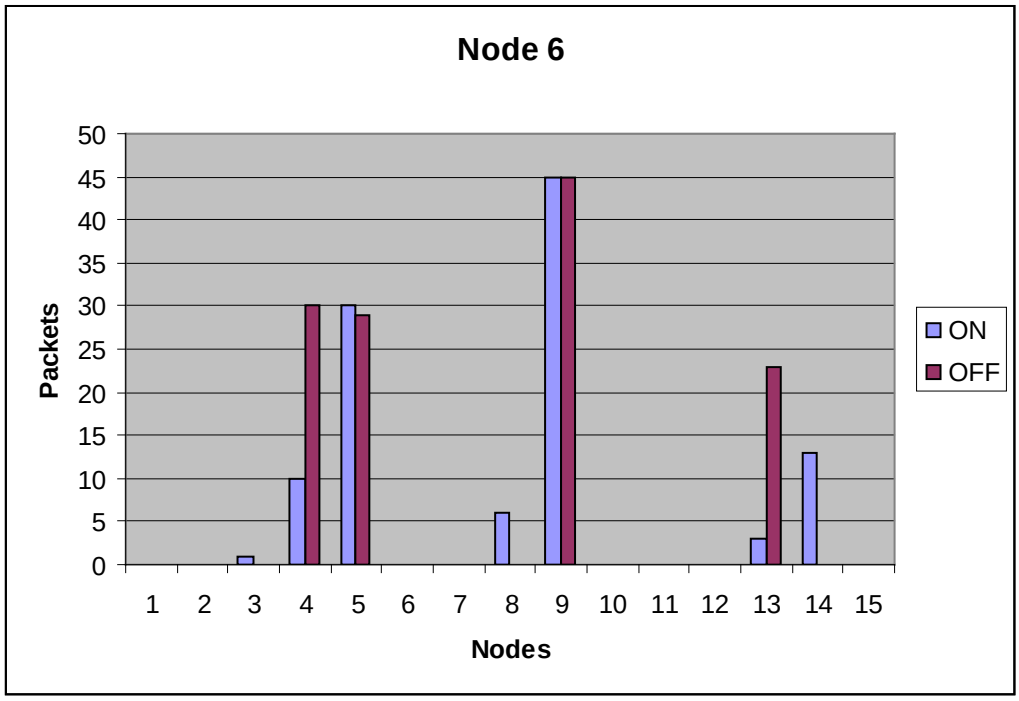


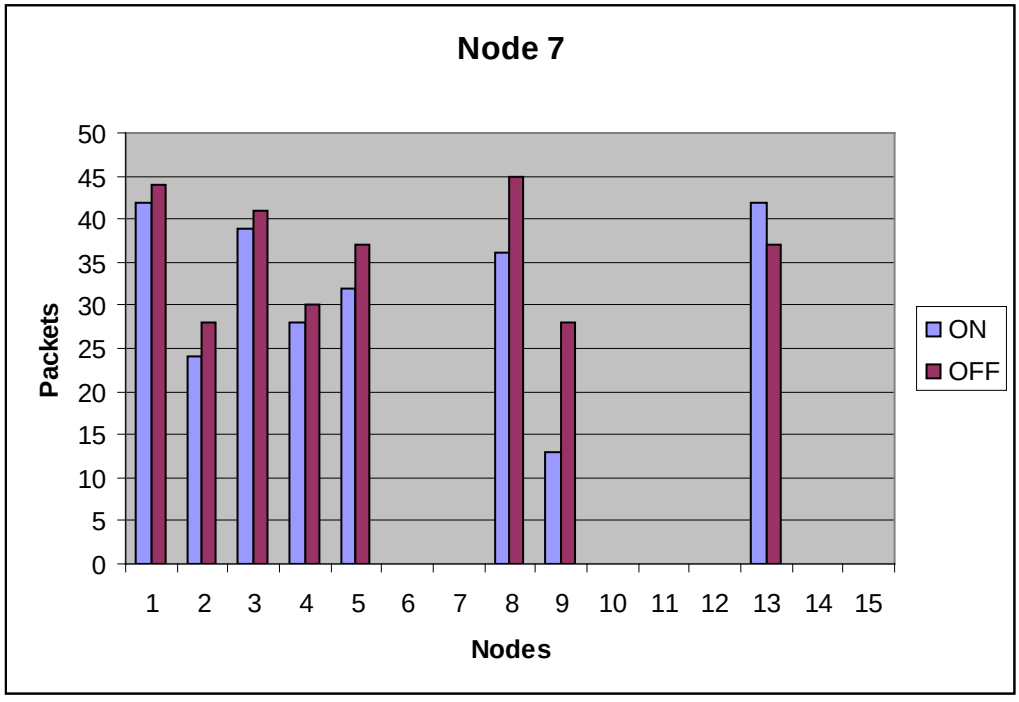

Node 2
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF

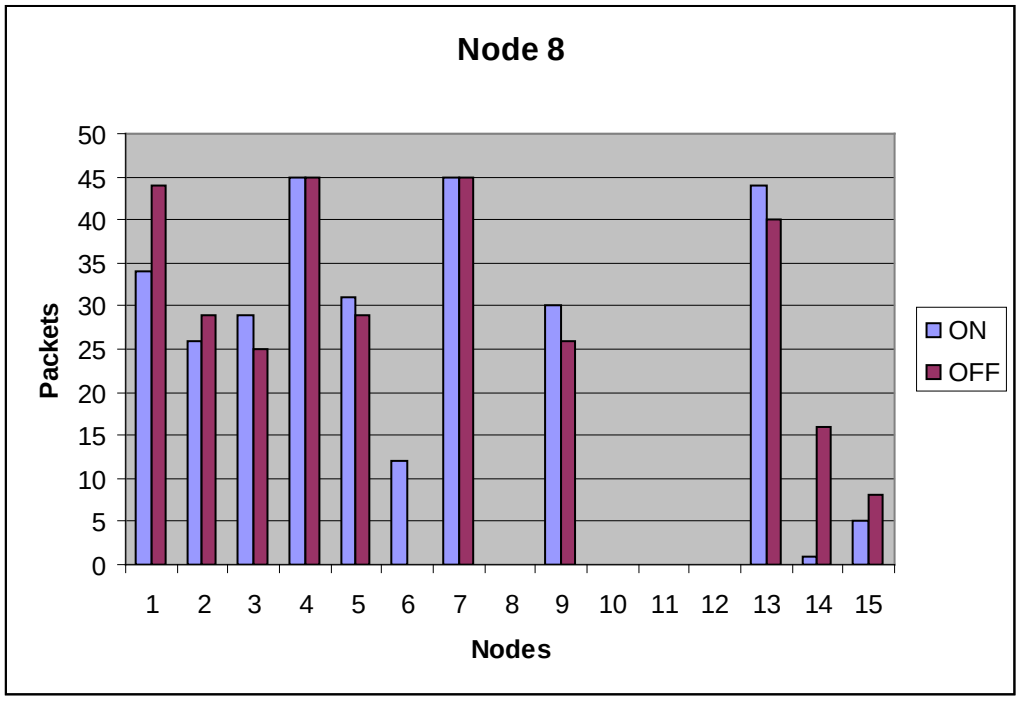

Node 3
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF

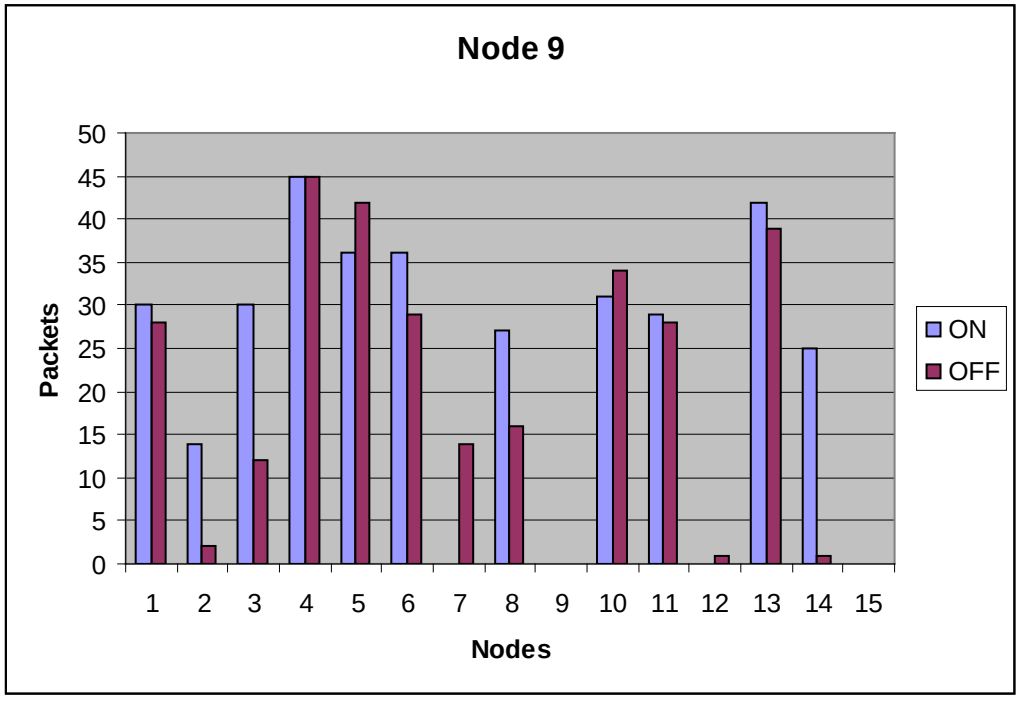

Node 4
Packets
Nodes
ON
OFF
0
5
10
15
20
25
30
35
40
45
50
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

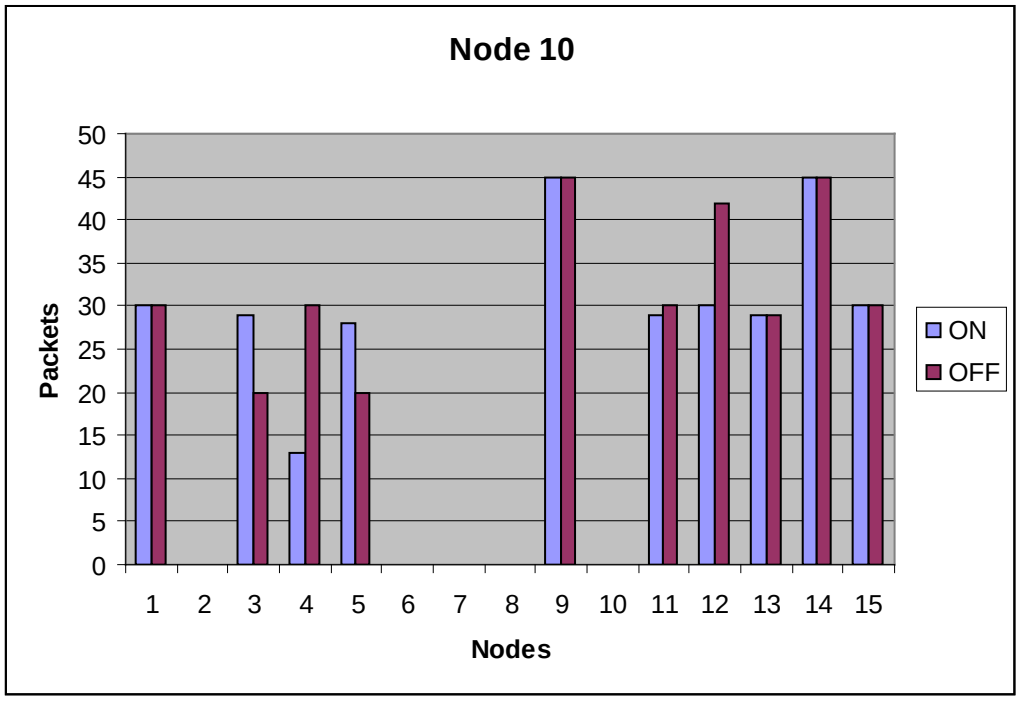

Node 5
Packets
Nodes
ON
OFF
0
5
10
15
20
25
30
35
40
45
50
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

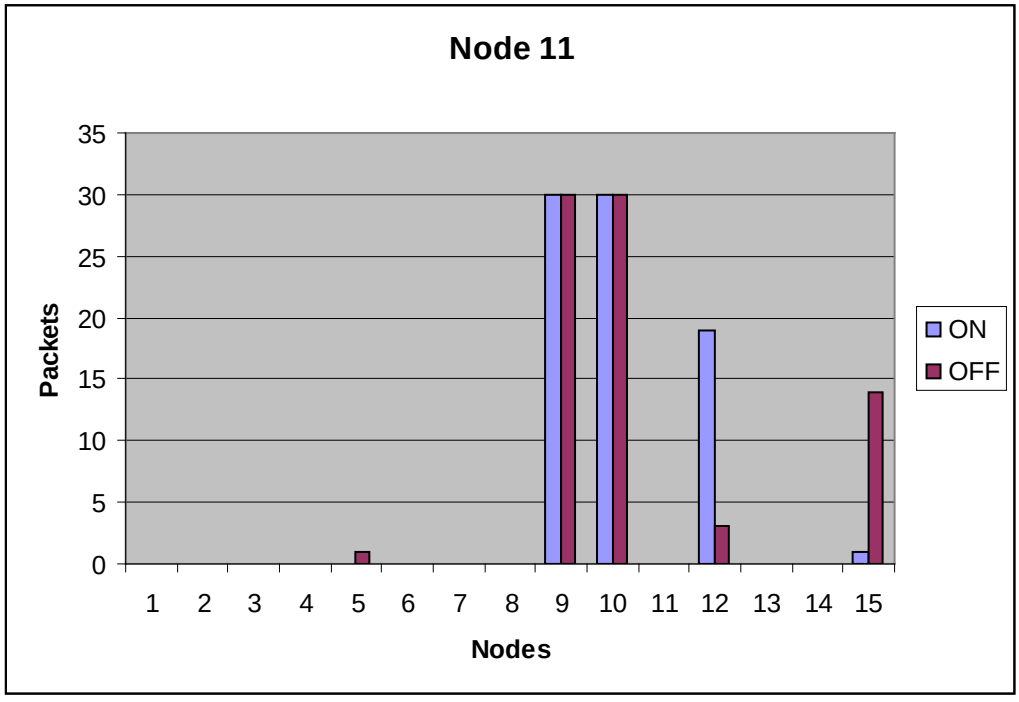

Node 6
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF

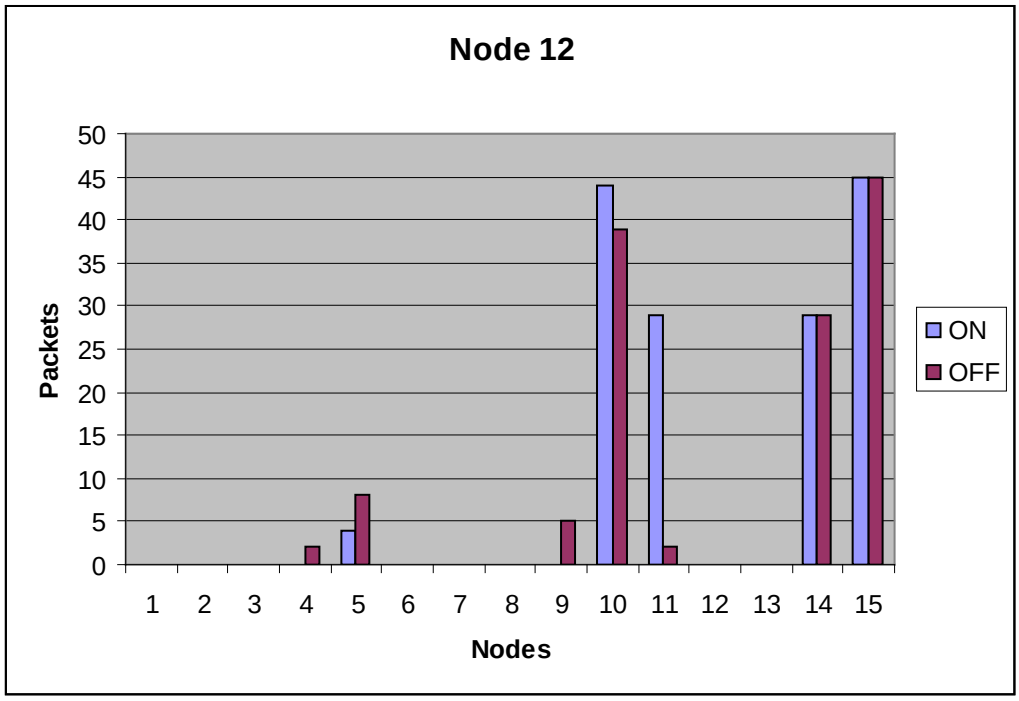

Node 7
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF

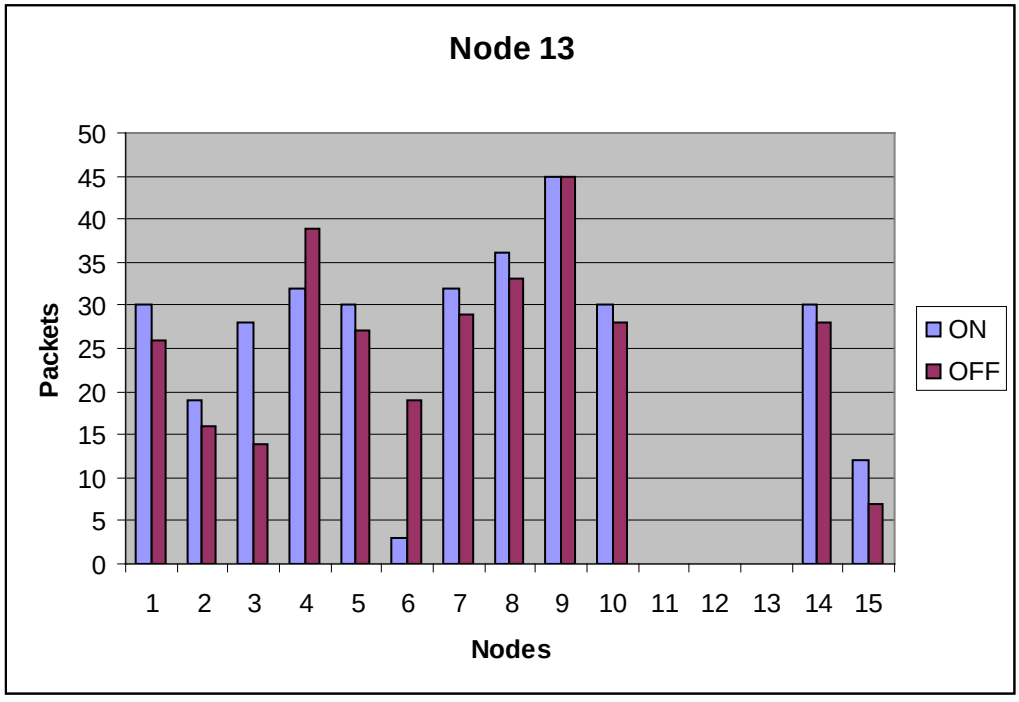

Node 8
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF

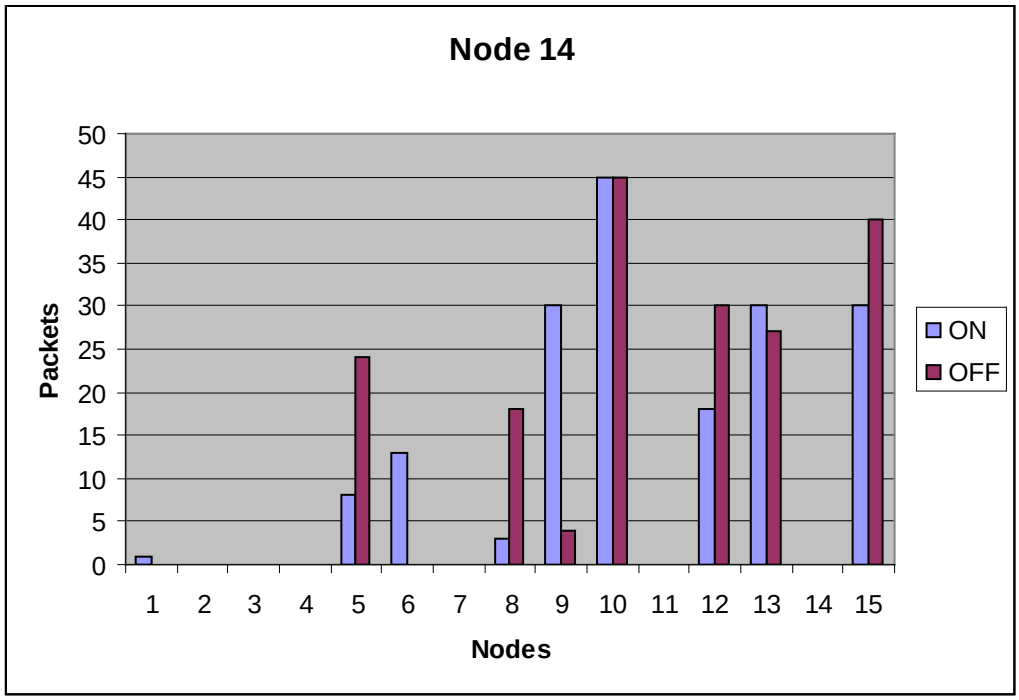

Node 9
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF

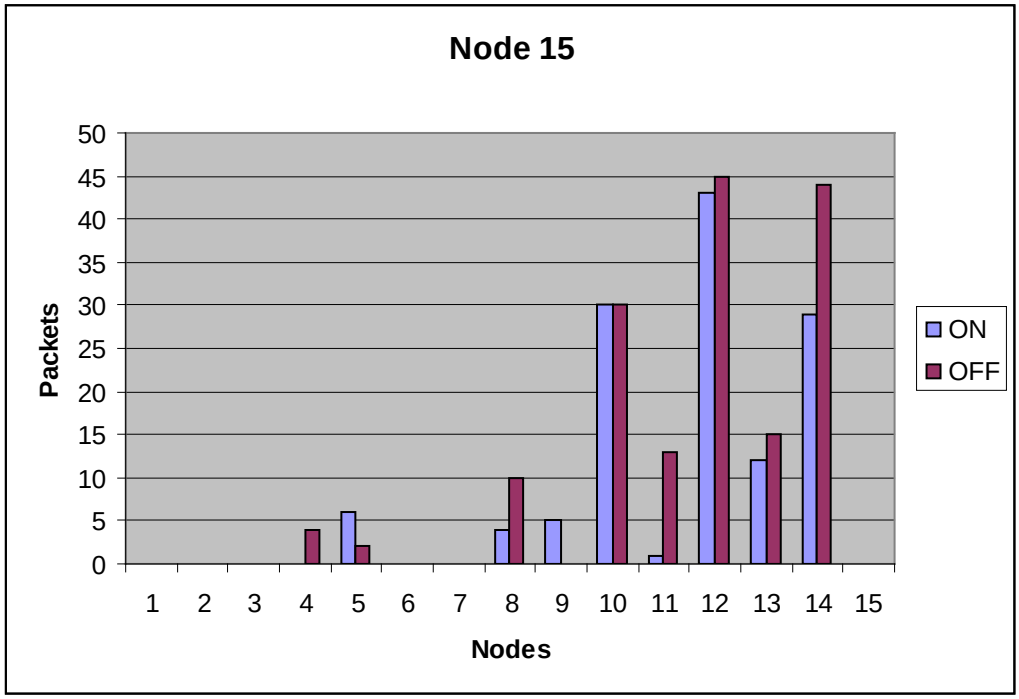

Node 10
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF


Node 11
Packets
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF


Node 12
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF


Node 13
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF


Node 14
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF


Node 15
Packets
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Nodes
ON
OFF