Universidad Central “Marta Abreu” de Las Villas
Facultad de Ingeniería Eléctrica
Departamento de Electrónica y Telecomunicaciones

Trabajo de diploma

Análisis de desempeño del protocolo MAC multicanal del estándar IEEE 802.15.4

Autor: Luis Hernández Feal
lhfeal@uclv.cu
Tutor: MSc. Carlos Manuel García Algora
Prof. Dpto. Electrónica y Telecomunicaciones, Facultad de Ingeniería Eléctrica, UCLV
cgalgora@uclv.edu.cu

Santa Clara
2017
"Año 59 de la Revolución"

Pensamiento

“Hello world!”

Brian Kernighan.

Resumen:

Con la aparición en los últimos tiempos de soporte de hardware para el trabajo a múltiples frecuencias, los protocolos MAC para las LLNs han experimentado una evolución hacia la operación multicanal, para aumentar la razón de datos y disminuir los efectos de la interferencia y el desvanecimiento multipunto. Uno de los protocolos MAC multicanal de estado del arte es el modo TSCH del estándar IEEE 802.15.4, que está basado en TDMA. TSCH no define cuál es la entidad funcional que programa la comunicación de los nodos en la red. Orchestra puede actuar como dicha entidad: es un programador con enfoque autónomo mediante el cual los nodos adaptan su horario explotando la información de la topología de RPL y un conjunto de reglas de programación, lo que resulta en patrones de actividad periódicos, con bloques de ranuras asignados a diferentes planos de tráfico tales como EBs TSCH, señalización RPL o datos de aplicación. En este trabajo se realiza una comparación entre las diferentes configuraciones que permite la combinación de TSCH y Orchestra: secuencias de salto de canal, longitud de los bloques de ranura y operación de Orchestra (basada en el modo almacenamiento o modo no-almacenamiento de RPL). Para el logro de este objetivo se usan implementaciones de TSCH y Orchestra en el sistema operativo ContikiOS, las cuales se corren en el simulador COOJA sobre el nodo sensor Z1. La comparación se basa en tres escenarios de simulación que permiten evaluar el desempeño cada configuración bajo diferentes cargas de tráfico. Los resultados de la simulación de estos escenarios demuestran que la mejor secuencia de salto de canal de TSCH es 16_16, la mejor longitud de bloques de ranuras es la que permita manejar justo el tráfico generado por la red y el modo no-almacenamiento de RPL es recomendable para aplicaciones de sensores dedicados a la recolección de información y el monitoreo de parámetros.

Índice

Introducción

El surgimiento de la Internet de las cosas (IoT, por sus siglas en inglés) y su desarrollo exponencial supone el empleo de nodos limitados en potencia, memoria y capacidad de procesamiento puesto que se emplean como dispositivos empotrados en los objetos cotidianos y que pueden actuar como sensores para la recolección y el monitoreo de parámetros y como equipos terminales de datos que responden de forma remota o autónoma en coordinación con otros objetos [2]. La interconexión de estos nodos se logra mediante Redes de Baja Potencia y con Pérdidas (LLNs, por sus siglas en inglés) en las que resulta primordial establecer el mejor equilibrio entre flexibilidad y fiabilidad [5].

Las LLNs encuentran gran diversidad de aplicaciones en la domótica, la administración de recursos (agua, energía, iluminación, ventilación…), la automatización de la actividad industrial y comercial, el monitoreo de variables ambientales y meteorológicas (empleadas en la agricultura, en el control de incendios forestales, la reducción de daños por sismos…), entre otras [1]. El desarrollo actual de la ciencia y la técnica apunta a un incremento exponencial de la demanda de soluciones basadas en LLNs.

El creciente interés por las LLNs se observa en el aumento de la cantidad de publicaciones y artículos en revistas y sitios de connotado prestigio como IEEE Communications Society. En IEEE Xplore Digital Library se recogen desde el 2000 hasta la actualidad 15238 trabajos de ellos 1378 artículos en revistas y 13860 presentaciones en congresos, siendo el 78% de dichos trabajos posterior a 2010 y el 34% posterior a 2014.

Las LLNs se caracterizan por estar formadas por nodos restringidos en potencia, memoria y capacidad de procesamiento; además los enlaces presentan inestabilidad, bajas tasas de transferencia y pérdidas. Normalmente estos nodos son dispositivos cuyo precio promedio resulta bajo y cuya fuente de alimentación generalmente son baterías, por lo que el ahorro energético es el tema primordial [1, 2, 5]. Teniendo en cuenta que el mayor costo energético lo provoca la interfaz de radio, se necesita optimizar la comunicación para conmutar dicha interfaz entre los estados de actividad e hibernación para lograr el mejor equilibrio entre costo energético y requerimientos de red. Atendiendo a esto, se necesita un modelo de protocolos de red que comprende tanto las capas del modelo TCP/IP como un grupo de capas de adaptación necesarias para adecuar estas a las limitaciones de las LLNs como son el protocolo de aplicación restringida (CoAP, por sus siglas en inglés) [11], el protocolo de enrutamiento IPv6

para LLNs (RPL) [8], el protocolo de compresión de formatos para datagramas IPv6 (6LowPAN) [10], e IEEE 802.15.4 [2].

El Protocolo de Aplicación Restringido (CoAP, por sus siglas en inglés) proporciona un modelo de interacción solicitud/respuesta entre los puntos finales de la aplicación, admite el descubrimiento integrado de servicios y recursos e incluye conceptos clave de Internet como URLs y tipos de medios de Internet. CoAP está diseñado para interactuar fácilmente con HTTP para la integración con la Web mientras cumple con requisitos especializados tales como soporte de multidifusión, gastos generales muy bajos y simplicidad para entornos restringidos.

El protocolo de enrutamiento IPv6 para las LLNs (RPL, por sus siglas en inglés) es el estándar de facto propuesto por IETF para LLNs y proporciona mecanismos para encaminar el tráfico punto a punto (entre dispositivos dentro de la LLN), punto a multipunto (desde un punto de control central a un subconjunto de dispositivos dentro de la LLN) y multipunto a punto (desde dispositivos dentro de la LLN hacia un punto de control central). Para ser útil en una gran variedad de aplicaciones de LLNs, RPL separa el procesamiento de paquetes y el reenvío de enrutamiento con el objetivo de optimizar el costo energético, la latencia y satisfacer las limitaciones de los sistemas. RPL crea un árbol de ruteo, cuya raíz es el punto de colección de información (conocido como sumidero) en forma de grafo acíclico orientado y dirigido (DODAG, por sus siglas en inglés). De forma general RPL organiza a los nodos en padres e hijos en dependencia de la topología: el sumidero es padre y los extremos son hijos, pero los intermedios son hijos de los más cercanos al sumidero y padres de los más alejados, como capas superpuestas. La comunicación se realiza padre a hijo o hijo a padre (ascendente o descendente). De esa forma se optimiza la comunicación desde cualquier punto hacia el núcleo de la red que es el nodo sumidero en la mayoría de las aplicaciones [8].

6LowPAN es una capa de adaptación entre la capa de red y la capa de acceso al medio y su función consiste en proporcionar mecanismos de encapsulación y compresión de cabeceras para permitir que los paquetes IPv6 viajen sobre tramas IEEE 802.15.4.

A su vez, IEEE 802.15.4 [2] es el estándar propuesto por la IETF para la capa de acceso al medio de las LLNs que define diferentes modos de operación orientados en dos líneas opuestas: las deterministas y las no deterministas. En las primeras la comunicación se planifica de antemano independiente del tráfico, en las segundas la comunicación es rd uns respuesta a la demanda de tráfico. En un principio se pretendía que las soluciones deterministas eran inadecuadas para las LLNs cuyas aplicaciones mayormente generan patrones de tráfico

aleatorios y las topologías crecen y cambian con dinamismo. Pero se ha demostrado que las soluciones no deterministas o asíncronas como ContikiMAC [9, 5] presentan tasas de pérdidas de hasta el 1% [5] y los costos de energía y latencia son bastantes similares a las soluciones deterministas o síncronas como las que se mencionan a continuación.

De las soluciones deterministas, una de las más relevantes es la de Salto de Canal por ranuras de Tiempo (TSCH, por sus siglas en inglés) que está basada en Acceso Múltiple por División de Tiempo (TDMA, por sus siglas en inglés). TSCH propone dividir el tiempo en ranuras, agrupadas en bloques de ranuras que se repiten periódicamente. La cantidad de ranuras determina el tamaño de los bloques. En una misma ranura varios nodos pueden trasmitir empleando diferentes canales. Para el correcto funcionamiento de TSCH los nodos deben sincronizarse y coordinar en qué ranura y canal de frecuencia le corresponde hibernar, escuchar o trasmitir a cada uno. Los algoritmos de planificación (en este documento se emplean los términos “programador” para referirse a los algoritmos de planificación, y “horario” para referirse al plan en sí) son precisamente los encargados de disponer el orden en el que cada nodo conmuta de estado y de canal para lograr comunicación exitosa y libre de colisiones. [9]

Dos de los principales programadores son Orchestra [5] y *Scheduling Function 0*. El reto fundamental de estos es crear horarios para TSCH que no afecten la flexibilidad de las LLNs y se adecúen a los requerimientos del tráfico no determinista exigido por la mayoría de las aplicaciones en IoT [5]. *Scheduling Function 0* [12, 13] propone una programación muy básica que apenas permite la comunicación de los nodos dentro de la red. Por otra parte, Orchestra ha demostrado tener un buen desempeño, incluso en comparación con otros protocolos del estado del arte [7].

Atendiendo a que la combinación de TSCH y Orchestra aún está en la fase de desarrollo y evaluación los autores de esta investigación proponen como **problema científico**: ¿Cuál es el desempeño de la combinación de TSCH y Orchesta ante diferentes cargas de tráfico, secuencias de canales longitudes de bloques de ranuras y operación de Orchestra (basada en modo almacenamiento o no-almacenamiento de RPL)?

Para dar solución al problema científico, se define como **objetivo general** evaluar el desempeño de TSCH y Orchestra empleando herramientas de simulación.

A fin de alcanzar el objetivo general se proponen los siguientes objetivos específicos:

1. Caracterizar el funcionamiento de la capa MAC de las LLNs.

2. Describir el funcionamiento de TSCH y Orhestra propuestos para la operación multicanal síncrona en las LLNs.
3. Diseñar escenarios de simulación para la medición del desempeño de TSCH y Orchestra.
4. Describir el desempeño de TSCH y Orchestra a partir de los datos obtenidos en los escenarios simulados.

Los objetivos específicos tienen como propósito dar respuesta a las siguientes **interrogantes científicas**:

1. ¿Cuáles son las características del funcionamiento de la capa MAC dentro del contexto de las LLNs?
2. ¿Qué elementos permiten describir el protocolo TSCH y Orchestra propuestos para operación multicanal síncrona en LLNs?
3. ¿Qué escenarios permiten medir el desempeño de TSCH y Orchestra en ambientes de simulación?
4. ¿Cuál es la configuración óptima de los protocolos TSCH y Orchestra según los datos obtenidos en los escenarios de simulación diseñados?

En la obtención de los resultados que validen esta investigación se emplea el sistema operativo ContikiOS y el simulador COOJA contenido en él para realizar las simulaciones. ContikiOS es un sistema operativo de código abierto orientado al trabajo en el IoT que provee poderosas herramientas para la construcción de complejos sistemas inalámbricos. TSCH y Orchestra se encuentran implementados en ContikiOS, específicamente para la plataforma Z1 de la compañía española Zolertia.

La novedad científica y el principal aporte de esta investigación es la evaluación del desempeño de TSCH y Orchestra atendiendo a sus diferentes parámetros configurables.

Organización del informe:

El informe de investigación está compuesto por una introducción, capitulario, conclusiones, recomendaciones para investigaciones futuras en el tema, referencias bibliográficas y anexos:

Capítulo 1: Se caracterizan las LLNs y se enuncian los retos y beneficios de la operación MAC multicanal. Se describen a fondo los protocolos TSCH y Orchestra.

Capítulo 2: Se fundamenta la selección de las tecnologías ContikiOS, COOJA y Z1. Se describen detalladamente los escenarios de simulación que se utilizan para comparar el desempeño de TSCH y Orchestra con sus diferentes configuraciones.

Capítulo 3: Se discuten los resultados de las simulaciones. Se realiza una comparación entre las distintas configuraciones de TSCH y Orchestra según los datos recolectados en las tareas de simulación.

Capítulo 1

1.1. LLNs: Generalidades y arquitectura

Una LLN consiste básicamente en un conjunto de dispositivos alimentados por baterías que se despliegan en posiciones desconocidas a priori y se comunican entre sí usando enlaces inalámbricos. Forman topologías de árbol, malla o estrella y se comunican entre sí usando tramas cortas y bajas razones de datos. En la figura 1.1 puede apreciarse la pila de protocolos habilitados para IPv6 en la arquitectura de las LLNs.

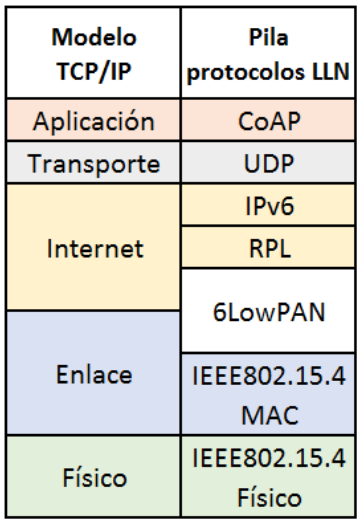


Figura 1.1: Pila de protocolos habilitados para IPv6 en la arquitectura de las LLNs en relación con el modelo TCP/IP.

El Protocolo de Aplicación Restringida (CoAP, por sus siglas en inglés) es un protocolo de transferencia web especializado para uso con nodos restringidos y redes limitadas como las LLNs. Está diseñado para aplicaciones de máquina a máquina como la redes inteligentes de distribución de energía y la automatización de edificios. CoAP proporciona un modelo de interacción solicitud/respuesta entre los puntos finales de la aplicación, admite el descubrimiento integrado de servicios y recursos e incluye conceptos claves de la Web, como identificadores de recursos uniforme (URIs, por sus siglas en inglés) y tipos de medios de Internet. CoAP está diseñado para interactuar fácilmente con HTTP en la integración con la Web mientras cumple con requisitos especializados tales como soporte de multidifusión, gastos generales muy bajos y simplicidad para entornos restringidos.

RPL es el protocolo de enrutamiento para las redes IPv6 de baja potencia estandarizadas por la IETF. Se trata de un protocolo de enrutamiento de vector distancia orientado que crea un

árbol de ruteo en forma de grafo acíclico orientado y dirigido (DODAG, por sus siglas en inglés). El DODAG está enraizado en el nodo de borde que conecta una LLN a Internet, conocido como sumidero o enrutador de borde. De forma general RPL organiza a los nodos en padres e hijos en dependencia de la topología: el sumidero es padre y los extremos son hijos, pero los intermedios son hijos de los más cercanos al sumidero y padres de los más alejados, como capas superpuestas. La comunicación se realiza padre a hijo o hijo a padre (ascendente o descendente). De esa forma se optimiza la comunicación desde cualquier punto hacia la raíz, que es el nodo sumidero en la mayoría de las aplicaciones. Cada nodo se encuentra a una distancia o rango de la raíz determinado por alguna función de coste. Un nodo envía un paquete hacia la raíz enviándolo a un nodo vecino con un rango más pequeño. El enrutamiento desde la raíz a uno de los nodos se realiza utilizando los enlaces inversos. El enrutamiento de un nodo a otro se realiza primero por enrutamiento ascendente hasta un ancestro común (si se emplea el modo almacenamiento de RPL; si se empleara el modo no-almacenamiento el paquete tendría que retroceder hasta la raíz) y luego descendente hasta el destino. La operación de TSCH y Orchestra se centra en el modo de almacenamiento de RPL: cada nodo mantiene una tabla de enrutamiento con respecto a sus hijos de enrutamiento.

En las LLNs el consumo de energía es una cuestión primordial. Los dispositivos desactivan su interfaz de radio la mayor parte del tiempo para ahorrar energía. Asimismo, el carácter inalámbrico de los enlaces significa que no son confiables. Debido a estas cuestiones no resulta fácil adaptar IPv6 en la parte superior de las LLNs. Ejemplo típico es que el tamaño mínimo de un paquete IPv6 es de 1280 Bytes por lo que resulta demasiado grande para caber en una trama IEEE.802.15.4 y la sobrecarga debido a la cabecera de 40 Bytes desperdicia el poco ancho de banda disponible en la capa física. Para solucionar estas cuestiones se define 6LoWPAN como capa de adaptación entre la capa de enlace de IEEE.802.15.4 y la capa de red. Las prestaciones fundamentales de 6LowPAN son que incluye información de clase de tráfico y compresión de etiquetas de flujo, comprime dirección de multidifusión de estado, comprime encabezado siguiente de IPv6, comprime cabeceras de extensión IPv6, comprime cabeceras UDP, comprime puertos UDP y comprime chequeo de paridad de UDP.

1.2. TSCH

TSCH es un modo de operación de IEEE.802.15.4 de 2015 definido para la capa MAC de LLNs y encaja bajo la pila de protocolos habilitados para IPv6 en la arquitectura de las LLNs (ver

figura 1.1). TSCH fue diseñado para permitir que las LLNs basadas en IEEE.802.15.4 soportasen una gran variedad de aplicaciones dentro de las que se tuvo en cuenta las industriales. La técnica MAC de TSCH emplea sincronización de tiempo (para lograr un funcionamiento de baja potencia) y salto de canal (para lograr alta fiabilidad) [9]. La sincronización impacta directamente en el consumo de energía y puede variar de milisegundos a microsegundos dependiendo de la solución. Gracias a su naturaleza de salto de canal TSCH puede enfrentar ambientes con difíciles condiciones de trabajo, como los ambientes industriales donde suele haber grandes entornos de despliegue con equipamiento metálico que causa desvanecimiento e interferencia multi-ruta haciendo inviable las soluciones basadas en un solo canal. IEEE.802.15.4 no define la entidad funcional que actúa de programador.

1.2.1. Ranuras de Tiempo y Bloques de Ranuras

El funcionamiento de una red TSCH se basa en el sincronismo. Para lograr este sincronismo el tiempo se divide en “ranuras” lo suficientemente largas para que un nodo envíe una trama MAC de tamaño máximo a otro y reciba un acuse de recibo (ACK) de éste que indique la recepción satisfactoria. El estándar no define la duración de una ranura pero normalmente con radios compatibles con IEEE.802.15.4 que operan en la banda de 2,4 GHz, una trama MAC máxima (127 bytes) tarda 4 ms en ser transmitida y un ACK tarda 1 ms. De esta forma, la duración típica de una ranura (10 ms), deja 5 ms libres para preparación de la interfaz de radio, procesamiento de la trama y operaciones de seguridad.

Las ranuras se agrupan en bloques de ranuras. Un bloque de ranuras se repite continuamente en el tiempo. TSCH no impone un tamaño del bloque de ranuras. Dependiendo de las necesidades de la aplicación, estos tamaños pueden variar de 10 a 1000 ranuras. Cuanto más corto sea el bloque de ranuras, más frecuentemente se repetirá una ranura, resultando en un mayor ancho de banda disponible, aunque también en un mayor consumo de energía.

1.2.2. Horario TSCH y Celdas Programadas

Es importante recordar que en este informe se maneja “horario” para referirse a la planificación en sí y “programador” a la entidad funcional que gestiona la planificación, así como al algoritmo de planificación. El horario indica a cada nodo qué hacer en cada ranura: transmitir, recibir o dormir. De esta forma, para cada ranura activa se planifica la acción a realizar, transmitir o recibir; un canal y la dirección del vecino con el cual comunicarse.

Una vez que un nodo obtiene su horario, lo ejecuta:

- Para cada ranura de transmisión: el nodo comprueba si hay algún paquete en el búfer saliente que coincida con el vecino indicado por el horario para esa ranura. Si no hay ninguno, el nodo mantiene su radio apagado durante la duración de la ranura. Si hay alguno, el nodo lo envía al vecino, en cuyo caso debe esperar el ACK tras la transmisión.
- Para cada ranura de recepción: el nodo escucha las posibles tramas entrantes. Si no se recibe ninguna después de un período de escucha, apaga su radio. Si se recibe una trama, dirigida al nodo en cuestión y pasa el control de seguridad, el nodo devuelve un ACK.

Cómo se crea, actualiza y mantiene el horario, y por qué entidad, está fuera del alcance del estándar IEEE 802.15.4e. En la sección 1.3 se aborda el tema. Suponiendo que el horario esté bien construido, si el nodo A está preparado para transmitir al nodo B en la ranura 5 y el canal 11, el nodo B se preparará para recibir desde el nodo A en la misma ranura y canal.

Un único elemento del horario caracterizado por una ranura y un canal, y reservado para que el nodo A transmita al nodo B (o para que el nodo B reciba desde el nodo A) dentro de un bloque de ranuras dado, se denomina “celda”.

Si hay una gran cantidad de datos que fluyen desde el nodo A al nodo B, el horario podría contener varias celdas de A a B, en diferentes ranuras. Múltiples celdas para el mismo vecino pueden ser equivalentes, es decir, la capa MAC envía la trama en la primera de estas celdas que aparezca después de que la trama se pone en la cola MAC. La unión de todas las celdas entre dos vecinos, A y B, se denomina "haz". Dado que el bloque de ranuras se repite en el tiempo (y la longitud del bloque de ranuras es típicamente constante), cada celda da una "cantidad" de ancho de banda a un vecino determinado. La modificación del número de celdas equivalentes en un haz modifica la cantidad de recursos asignados entre dos vecinos.

La figura 1.2 representa un horario genérico: nótese que una celda es un punto inequívoco en el sistema de coordenadas (ranura; canal). La figura 1.3 representa un ejemplo de horario estático para una red de 4 nodos.

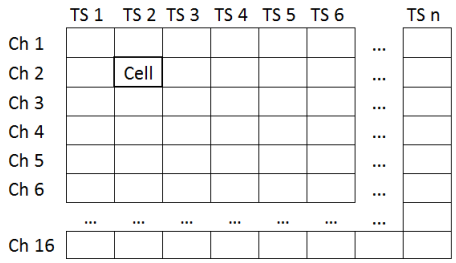


Figura 1.2: Representación del horario (“TS” se emplea para referirse a las ranuras y “Ch” para referirse al canal).

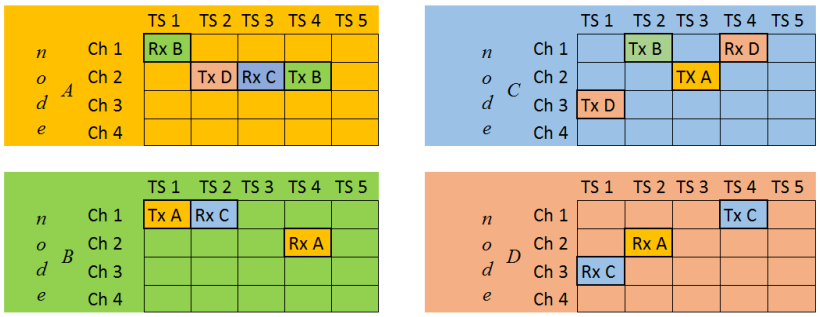


Figura 1.3: Ejemplo de horario estático para una red TSCH de 4 nodos (“TS” se emplea para referirse a las ranuras y “Ch” para referirse al canal).

Por defecto, cada celda de transmisión del horario está dedicada, es decir, reservada únicamente para que el nodo A transmita al nodo B. IEEE 802.15.4 también permite que una celda sea marcada como compartida. En una celda compartida, múltiples nodos pueden transmitir al mismo tiempo y en la misma frecuencia. Para evitar la contención, TSCH define un algoritmo de *backoff* para las celdas compartidas.

Una celda puede ser marcada como transmisión y recepción. En el primer caso, el nodo transmite si tiene un paquete apropiado en su búfer de salida, y en el segundo escucha. Marcar una celda como [transmitir, recibir, compartir] da como resultado un comportamiento de ALOHA (medio compartido) con ranura.

1.2.3. Número de Ranura Absoluta y Salto de Canales

TSCH define un contador de ranuras denominado Número Absoluto de Ranura (ASN, por sus siglas en inglés). Cuando se crea una nueva red, el ASN se inicializa a 0; a partir de entonces, se incrementa 1 en cada ranura. En detalle:

ASN = (k * S + t)

Donde k es el ciclo del bloque de ranuras (es decir, el número de repeticiones del bloque de ranuras desde que se inició la red), S la longitud del bloque de ranuras y t la ranura de apertura. Un nodo aprende la ASN actual cuando se une a la red. Dado que los nodos están sincronizados, todos conocen el valor actual de la ASN, en cualquier momento. La ASN está codificada como un número de 5 bytes: esto le permite incrementarse durante cientos de años (el valor exacto depende de la duración de una ranura) sin que se desborde o reinicie. El ASN

se utiliza para calcular la frecuencia para comunicarse y se puede utilizar para operaciones relacionadas con la seguridad.

Para cada celda, el horario especifica una ranura y un canal. El canal es traducido por ambos nodos en una frecuencia usando la siguiente función:

$$Frecuencia = F^*\{(ASN + canal) mod\ nFreq\}$$

La función F consiste en una tabla de búsqueda que contiene el conjunto de canales disponibles. El valor nFreq (el número de frecuencias disponibles) es el tamaño de esta tabla de consulta. Hay tantos valores de canales como frecuencias disponibles (por ejemplo, 16 cuando se utilizan todos los canales en radios que cumplen con IEEE 802.15.4 a 2,4 GHz). Dado que ambos nodos tienen el mismo canal escrito en su horario para esa celda, y el mismo contador ASN, calculan la misma frecuencia. Sin embargo, en el siguiente ciclo del bloque de ranuras, mientras que el desplazamiento de canal es el mismo, la ASN ha cambiado, dando como resultado el cálculo de una frecuencia diferente.

Esto resulta en "salto de canal": incluso con un horario estático, los pares de vecinos "saltan" entre las diferentes frecuencias en la comunicación. Una manera de asegurar que la comunicación ocurre en todas las frecuencias disponibles es establecer el número de ranuras de un bloque de ranuras en un valor primo. El salto de canal es una técnica conocida para combatir eficazmente el desvanecimiento multipunto y la interferencia externa.

1.2.4. Sincronización de tiempo

Debido a la naturaleza ranurada de la comunicación en una red TSCH, los nodos tienen que mantener la sincronización. Se supone que todos los nodos están equipados con osciladores para controlar el tiempo. Sin embargo, debido a que los osciladores en diferentes nodos fluctúan entre sí, los nodos vecinos necesitan resincronizarse periódicamente.

Cada nodo necesita sincronizar periódicamente su tiempo de red con otro nodo, y también proporciona su tiempo de red a sus vecinos. Corresponde al programador asignar un vecino como fuente de tiempo adecuado a cada nodo, es decir, indicar a cada nodo en el horario cuál es su vecino de origen temporal. Al establecer la fuente de tiempo vecino, es importante evitar bucles de sincronización, lo que podría dar lugar a la formación de grupos independientes de nodos sincronizados entre sí pero desincronizados con el resto de la red.

TSCH añade información de sincronismo en todas las tramas que se intercambian (tanto datos como tramas ACK). Esto significa que los nodos vecinos pueden resincronizarse entre sí cuando

intercambian tramas. Se definen dos métodos en IEEE 802.15.4 para permitir que un dispositivo se sincronice en una red TSCH. En ambos casos, el receptor calcula la diferencia de tiempo entre el tiempo esperado de llegada de la trama y su llegada real.

- Sincronización basada en el ACK: El receptor proporciona dicha información al nodo emisor en su ACK. En este caso, es el nodo emisor el que se sincroniza con el reloj del receptor.
- Sincronización basada en tramas: El receptor utiliza el delta calculado para ajustar su propio reloj. En este caso, es el nodo receptor que se sincroniza con el reloj del trasmisor.

Diferentes políticas de sincronización son posibles. Los nodos pueden mantener la sincronización exclusivamente intercambiando EBs. Los nodos también pueden mantenerse sincronizados enviando periódicamente paquetes válidos a un vecino de origen temporal y utilizar el ACK para volver a sincronizarse. Ambos métodos (o una combinación de los mismos) son políticas de sincronización válidas; cuál usar depende de los requisitos de la red.

1.2.5. El consumo de energía

Sólo hay 3 actividades que un nodo puede realizar durante una ranura: transmitir, recibir o dormir. Cada una de estas operaciones tiene algún costo energético asociado a ellas; el valor exacto depende del hardware utilizado. Dado el horario de un nodo, es fácil calcular el consumo de energía promedio esperado de ese nodo.

1.2.6. Horario

El horario define completamente la sincronización y la comunicación entre los nodos. Mediante la adición o eliminación de celdas entre vecinos, se puede adaptar un horario a las necesidades de la aplicación. Por ejemplo:

- Hacer el horario "escaso" para aplicaciones en las que los nodos necesitan consumir la menor cantidad de energía posible, al precio de un ancho de banda reducido.
- Hacer el horario "denso" para aplicaciones donde los nodos generan una gran cantidad de datos, al precio de un mayor consumo de energía.
- Agregar más celdas a lo largo de una ruta de salto múltiple sobre la cual fluyen muchas tramas.

1.2.7. Proceso de unión

Los nodos que ya forman parte de la red pueden enviar periódicamente tramas EB para anunciar la presencia de la red. Éstas contienen información sobre el tamaño del bloque de ranuras utilizado en la red, la ASN actual, información sobre los bloques de ranuras y las ranuras que el nodo que intenta unirse está escuchando y 1 byte de prioridad de unión. El campo de prioridad de unión proporciona información para tomar una mejor decisión de qué nodo debe unirse. Incluso si un nodo está configurado para enviar todas las tramas EB en el mismo canal, debido a la naturaleza de salto de canal de TSCH descrita, este canal se traduce en una frecuencia diferente en diferentes ciclos del bloque de ranuras. Como resultado, las tramas EB se envían a todas las frecuencias.

Un nodo que desee unirse a la red escucha los EBs. Como los EB se envían a todas las frecuencias, el nodo que intenta unirse puede escuchar en cualquier frecuencia hasta que escuche un EB. Qué frecuencia escucha por defecto al encender depende de la implementación. Una vez que ha recibido uno o más EBs, el nuevo nodo habilita el modo TSCH y utiliza el ASN y la otra información de temporización del EB para sincronizarse con la red. Usando el bloque de ranuras y la información de la celda de los EB, sabe cómo ponerse en contacto con otros nodos de la red. TSCH no define los pasos más allá de esta sincronización inicial.

1.2.8. Resumen de características destacadas de TSCH

- Comunicación sin colisiones: TSCH permite diseñar un horario que produzca una comunicación sin colisiones. Esto se hace construyendo el horario con celdas dedicadas de tal manera que, como máximo, un nodo se comunique con un vecino específico en cada celda (ranura; canal). Varios pares de nodos vecinos pueden intercambiar datos al mismo tiempo, pero en frecuencias diferentes.
- Salto de canales: Un horario se parece a una matriz de ancho "tamaño del bloque de ranuras", S, y de altura "número de frecuencias", nFreq. Para un programador, las celdas pueden ser consideradas "unidades" autónomas para programar. En particular, debido a la naturaleza de salto de canal de TSCH, el programador no debe preocuparse de que la comunicación de frecuencia real varíe, ya que cambia en cada ciclo del bloque de ranuras.

- Coste de la sincronización (continua): Cuando hay tráfico en la red, los nodos que se están comunicando se resincronizan implícitamente utilizando las tramas de datos que intercambian. En ausencia de tráfico de datos, se requiere que los nodos se sincronicen periódicamente con sus vecinos de fuente de tiempo para no derivarse en el tiempo. Si no han estado comunicándose durante algún tiempo (normalmente 30 s), los nodos pueden intercambiar una trama de datos vacía para resincronizarse. La frecuencia con la que estos mensajes deben transmitirse depende de la estabilidad de la fuente de tiempo y de la forma en que cada nodo comienza a escuchar los datos (el "tiempo de guarda"). Teóricamente, con un reloj de 10 ppm y un tiempo de guarda de 1 ms, este periodo puede ser de 100 s. Suponiendo que este intercambio hace que la radio del nodo esté encendida durante 5 ms, esto produce un ciclo de trabajo de radio necesario para mantener sincronizado 5 ms / 100 s = 0,005%. Aunque TSCH requiere que los nodos vuelvan a sincronizarse periódicamente, el costo de hacerlo es muy bajo.
- Estabilidad de la topología: La naturaleza de salto de canal de TSCH hace que los enlaces sean muy "estables". Los fenómenos inalámbricos, como el desvanecimiento multipunto y la interferencia externa, afectan de forma diferente a cada frecuencia en un enlace inalámbrico entre dos nodos. Si una transmisión desde el nodo A al nodo B falla, la retransmisión en una frecuencia diferente tiene una mayor probabilidad de éxito que la retransmisión en la misma frecuencia. De esta forma, incluso cuando algunas frecuencias están "comportándose mal", el salto de canal "suaviza" la contribución de cada frecuencia, dando como resultado enlaces más confiables y por lo tanto una topología más sólida.
- Múltiples bloques de ranuras concurrentes: El estándar TSCH permite que múltiples bloques de ranuras coexistan en el horario de un nodo. Es posible que, en una ranura, un nodo tenga múltiples actividades programadas (por ejemplo, transmitir al nodo B en el canal 2, recibir desde el nodo C en el canal 1). Para manejar esta situación, el estándar TSCH define las siguientes reglas de prioridad:
    a. Las transmisiones tienen prioridad sobre las recepciones.
    b. Los identificadores de bloque de ranuras inferior tienen prioridad sobre los identificadores de bloque de ranuras más altos.

  En el ejemplo anterior, el nodo A transmitirá al nodo B en el bloque de ranuras 2.

1.3. Programador

1.3.1. Enfoques del programador

En TSCH si dos nodos situados muy cerca tratan de enviar paquetes en la misma celda, entrará en conflicto y la transmisión de ambos fallará. Por lo tanto, para evitar la contención mientras se logra una alta utilización de los canales y la eficiencia energética, se debe usar un programador inteligente y prudente en la planificación de celdas TSCH. Los métodos actuales apuntan a tres enfoques: centralizado, descentralizado y autónomo.

1.3.2. Enfoque centralizado

En las redes TSCH que utilizan un programador centralizado, un nodo representativo programa el despertar y el dormir de cada nodo en la misma red. Dado que sólo un nodo programa toda la comunicación en la red, para ese nodo, se necesita una gran cantidad de información. Dependiendo del programador, se puede utilizar una gran cantidad de información para elaborar un horario optimizado. Sin embargo, dado que sólo un nodo específico puede ejecutar la programación, no hay respuesta rápida cuando la topología de red cambia repentinamente. Además, cada nodo tiene que entregar la información que se utiliza en el nodo central, lo que provoca la sobrecarga de comunicación utilizando energía aditiva y recursos de ancho de banda. Además, el enfoque centralizado requiere mucho tiempo para la inicialización.

Se clasifican los siguientes tipos de programación como enfoque centralizado. La red está dividida en varias regiones. En cada región, un nodo central ejecuta la programación para cada nodo situado en la misma región que ese nodo. En esta red, aunque múltiples nodos ejecuten la programación, no es totalmente descentralizada. Por el contrario, se puede ver como una combinación de múltiples programadores centralizados.

1.3.3. Enfoque descentralizado

En las redes TSCH que utilizan un programador descentralizado, cada nodo gestiona su propio horario. Cada nodo se comunica con sus vecinos para evitar conflictos causados por el uso simultáneo del mismo canal entre nodos interferentes. Para implementar esta red, debe decidirse un protocolo de negociación específico. En comparación con el enfoque centralizado, cada nodo que utiliza el programador descentralizado necesita sólo una pequeña cantidad de información recibida de sus vecinos cercanos. Como resultado, es robusto para un repentino cambio de red. Sin embargo, debido a la escasa información, el enfoque descentralizado carece de visión global de la red.

1.3.4. Enfoque autónomo

Aunque el enfoque descentralizado muestra una respuesta rápida para los repentinos cambios de la topología de la red, todavía necesita una comunicación aditiva para la negociación entre vecinos. En el enfoque autónomo, no se necesita negociación para programar el horario. (Un método representativo es Orchestra, que se aborda en detalle en la sección 1.3.) Sólo se utiliza la información recibida del protocolo de enrutamiento RPL. No se utiliza comunicación aditiva para negociar la entre los vecinos el horario. Cada nodo calcula de forma autónoma su horario basado en la estructura RPL. Utilizando NodeID como la dirección MAC del nodo padre RPL se ejecuta la programación basada en nodos. Por ejemplo, en el programador basado en receptor, los nodos receptores están en el horario de cada nodo. El nodo indicado escucha en la celda correspondiente y cualquier nodo que quiera enviar una trama a ese nodo despierta y contiende por enviar la trama. El programador basado en remitente se ejecuta de la misma manera.

Aunque el programador autónomo también carece de otra información disponible, los mayores beneficios de este son su ejecución sencilla y ligera sin ninguna negociación y su respuesta rápida a los repentinos cambios de red.

1.3.5. Consideraciones de diseño

En el diseño del programador, uno de los mayores desafíos es hacer el horario evitando conflictos entre los nodos que interfieren. Sin embargo, como el programador autónomo no utiliza ninguna negociación, carece de información. En esta condición, lo que el programador autónomo puede hacer es reducir la probabilidad de conflicto.

1.3.6. Programador basado en enlaces

Consideración 1: Un nodo no puede recibir tramas de dos nodos diferentes en la misma ranura. Por ejemplo, en una topología donde un nodo padre A tiene dos nodos hijos B y C respectivamente, si B y C envían tramas al nodo A en la misma celda, A no recibirá ninguna de las 2 tramas correctamente. En esta situación, el programador puede colocar el enlace A-B y el enlace A-C en el mismo canal o en diferentes canales. En el primer caso, si los dos nodos hijos envían paquetes simultáneamente al nodo padre, las dos transmisiones estarán en conflicto en el mismo canal y A no recibe ninguna de las dos. En el último caso, dependiendo de la selección de canal del nodo A, uno de los paquetes enviados desde el nodo B o el nodo C puede ser recibido satisfactoriamente.

1.3.7. Programador basado en contador de salto

Consideración 2: Un nodo no puede escuchar y enviar paquetes simultáneamente en la misma ranura. Por ejemplo, en la topología descrita en la sección 1.3.6, si el nodo B tuviese a su vez dos nodos hijos D y E respectivamente, y el enlace B-A y el D-B están programados en el mismo TS, el nodo D enviará al nodo B y el nodo B enviará al nodo A. Sin embargo, el nodo B no puede escuchar y enviar simultáneamente en la misma ranura.

1.3.8. Identificador de bloque de ranuras

Consideración 3: En un nodo específico, el horario de diferentes bloques de ranuras debe cambiarse para evitar conflictos consistentes. Si el horario para un bloque de ranuras persiste, los nodos en conflicto en una celda volverán a entrar en conflicto en el siguiente bloque de ranuras. Por ejemplo, en la topología descrita en 1.3.6, en el enlace ascendente, hay dos enlaces B-A y C-A. Si estos enlaces están programados en la misma celda y el nodo B y el nodo C envían tramas al nodo A, las dos transmisiones fallarán. Si este horario persiste, en el siguiente ciclo del bloque de ranuras se producirá la misma situación de nuevo.

1.3.9. En sentido ascendente: basado en enlace; en sentido descendente: basado en nodo.

Consideración 4: El programador de enlaces ascendentes y descendentes debe ser diferenciado. En las LLNs, la cantidad de tráfico ascendente es mucho mayor que la del tráfico descendente. Por otra parte, si la topología de árbol se utiliza para el protocolo de enrutamiento como se describe en 1.3.6 y 1.3.7, un nodo tiene múltiples nodos secundarios y sólo un nodo padre. En esta estructura, en el caso del enlace ascendente, cada nodo hijo tiene paquetes diferentes para enviar a su nodo padre. Por lo tanto, un nodo padre debe escuchar a sus múltiples hijos usando diferentes celdas. Sin embargo, en el caso de enlace descendente, el servidor puede solicitar o responder a cada nodo de la red o al nodo específico. Además, la cantidad de tramas descendentes es menor que la de tramas ascendentes y la celda para el flujo descendente puede no ser usada con alta probabilidad. Por lo tanto, en la topología descrita en 1.3.6 y 1.3.7, cuando el enlace B-D y el enlace B-E están en la misma celda, es más eficiente que si estos enlaces estuviesen en diferentes celdas. Haciendo esto, en el caso en que el servidor envía una trama a cada nodo de la red, el nodo B puede reenviar esta trama a todos sus nodos secundarios en una sola transmisión. En el otro caso en el que el nodo B envía una trama a uno de sus nodos secundarios, puede enviar la trama al nodo específico mientras varios nodos secundarios escuchan en la misma ranura.

1.3.10. Programador ascendente

Consideración 5: Un nodo no puede recibir o enviar paquetes desde su nodo padre y su nodo hijo simultáneamente. En otras palabras, para un nodo específico los enlaces ascendente y descendente no pueden ocurrir en la misma ranura. Por ejemplo, en la topología descrita en la figura 1.2, el nodo B está conectado tanto al nodo A como al nodo D. Si tanto el nodo A como el nodo D están programados para enviar tramas al nodo B en la misma ranura o el nodo B es programado para enviar tramas al nodo A y al nodo D en la misma ranura como resultado del cálculo de la celda usando un programador autónomo, al menos una transmisión fallará ya que el nodo B no puede recibir tramas diferentes de los diferentes nodos y enviar tramas diferentes a los diferentes nodos.

1.4. ORCHESTRA

En Orchestra, los nodos emplean horario simple y actualizan estos de forma automática e instantánea a medida que evoluciona la topología de enrutamiento. Un horario en Orchestra consiste en un conjunto de bloques de ranuras superpuestos dedicados cada uno a un plano de comunicación específico: MAC, enrutamiento y aplicación. Como resultado, Orchestra permite construir una columna de enrutamiento genérico, flexible y de bajo consumo usando RPL y aprovechando la robustez de TSCH. Estos horarios permiten reducir drásticamente la contención, o incluso eliminarla en ciertos casos. Con Orchestra los nodos mantienen su horario de forma local y autónoma, basado en sus vecinos y padres de RPL. Como resultado, Orchestra hace que TSCH sea tan flexible como las capas MAC asíncronas, y capaz de soportar tráfico de acceso aleatorio.

Un horario de Orchestra contiene diferentes bloques de ranuras de diferentes longitudes. Cada bloque de ranuras está dedicada a un tipo particular de tráfico. Los nodos seleccionan los bloques de ranuras usando reglas de programación que reducen drásticamente la contención, o en ciertos casos elimina la contención. Esto hace a Orchestra de especial interés para los escenarios IPv6 de baja potencia que en diferentes aplicaciones generan datos basados en eventos, sin ningún tipo de patrón de tráfico predefinido.

El horario de Orchestra contiene:

- Una celda dedicada de la difusión de cada nodo padre a sus nodos hijos para las balizas de TSCH, repitiéndose cada X ranuras.

- Una celda común para todos los nodos en la red para multidifusión + unidifusión, para señalización RPL (DIO, DIS, DAO), repitiéndose cada Y ranuras.
- Una celda unicast dedicada de cada nodo hijo a su nodo padre RPL, repitiéndose cada Z ranuras.
- N celdas dedicadas desde cada nodo padre a cada uno de sus nodos hijos, repitiéndose cada Z ranuras.

Orchestra utiliza longitudes de bloque de ranuras que son mutuamente primos, asegurando que las celdas se superponen de manera uniforme, sin efectos de sincronización no deseados. La clave es que selecciona la ranura y el canal de cada celda como una función del identificador del remitente o del receptor (dirección MAC o un ID de nodo de red único). Dependiendo de las reglas de programación que se le configuren, Orchestra puede alcanzar niveles muy bajos de contención, u operar libre de contención.

1.4.1. Diseño de Orchestra

En Orchestra, los nodos adaptan su horario mediante la explotación de la información de la topología de RPL, y siguiendo un conjunto de reglas de programación. Esto da como resultado patrones de actividad periódicos, con bloques de ranuras asignados a diferentes planos de tráfico tales como EBs, señalización RPL o datos de aplicación.

1.4.2. Inicialización

Cuando se enciende un nodo, este se une a la red TSCH escuchando hasta que recibe un EB, ya sea desde el coordinador de PAN o de otro nodo. Después de sincronizarse con ese EB, el nodo ejecuta Orchestra. Una configuración viable de Orchestra requiere celdas para enviar y recibir paquetes desde y hacia cualquier vecino. Esto emula un enlace permanente con todos los vecinos, permitiendo que los nodos RPL descubran a sus vecinos y construyan una topología.

1.4.3. Mapeo de topología TSCH-RPL

Orchestra utiliza constantemente al padre preferido RPL como su fuente de tiempo. A medida que evoluciona la topología RPL y se producen cambios de padre, los nodos actualizan su fuente de tiempo en consecuencia. Esto produce una estructura de sincronización sin bucles (un árbol en este caso), aprovechando el mecanismo para evitar bucles de RPL. Además, se utiliza el rango RPL como prioridad de combinación TSCH, como se define en la arquitectura

de las LLNs. Haciendo esto, también se aprovecha los mecanismos RPL para la convergencia y estabilidad del gradiente. En caso de que un nodo pierda la sincronización, también abandona la red RPL, asegurando una inicialización limpia después de la reincorporación.

1.4.4. Sincronización de tiempo TSCH.

La sincronización de tiempo de TSCH ocurre en cualquier paquete (o ACK) del vecino de origen de tiempo. En Orchestra, la sincronización de tiempo ocurre principalmente a través de la transmisión periódica de balizas TSCH y RPL. Esto es eficiente, ya que un único mensaje de difusión permite a todos los nodos hijos actualizar sus relojes. Cada vez que un nodo no se ha comunicado con su vecino de fuente de tiempo tras un tiempo determinado (por defecto se usa 12 s), envía un mensaje *keepalive* unicast. El paquete se reenvía hasta que se reconoce y la resincronización se realiza utilizando la información de temporización incorporada en el ACK mejorado IEEE 802.15.4.

1.4.5. Programador con direccionamiento de rutas

A lo largo de la vida útil de la red, Orchestra instala y actualiza los horarios utilizando la información de la capa de enrutamiento. RPL se ejecuta sin modificaciones, con celdas que se configuran automáticamente a medida que evoluciona la topología, garantizando la conectividad de la red y permitiendo que las capas superiores se ejecuten de forma transparente. Un ejemplo básico es donde Orchestra mantiene una celda dedicada para que los nodos padres se comuniquen con los nodos hijos.

1.4.6. Celdas de Orchestra

Se identifican cuatro tipos principales de celdas en Orchestra:

- Celdas comunes compartidas (CS): Las CS consisten en una celda compartida usada por todos los nodos en la red para recepción (Rx) y transmisión (Tx). La celda se instala en coordenadas fijas (ranuras; canales), dando como resultado un comportamiento similar al ALOHA ranurado. Esto emula un vínculo siempre activo, lo que permite a RPL descubrir vecinos y transmitir información de control. Debe tenerse en cuenta que TSCH utiliza un back-off exponencial para resolver la contención en las celdas compartidas, que se activa cada vez que una transmisión de unidifusión no se confirma con un ACK.
- Celdas Compartidas Basadas en Receptor (RBS, por sus siglas en inglés): Las RBS se asignan para la comunicación entre dos vecinos, en las coordenadas (ranura; canal)

derivadas de las propiedades del receptor. En cada nodo, una celda de Orchestra RBS tiene como resultado una celda Rx (coordenadas basadas en el nodo receptor), y una celda Tx por vecino (coordenadas basadas en el vecino). Para calcular las coordenadas de la celda, se puede usar un hash de la dirección MAC del nodo, modulo la longitud del bloque de ranuras o explotar identificadores de nodo únicos cuando están disponibles.

Un ejemplo típico es la comunicación de hijo a padre: los nodos padres escuchan cualquier tráfico en una celda y los hijos mantienen una celda de transmisión hacia sus padres. Cuando los nodos cambian de padre, ellos actualizan su celda de transmisión de forma autónoma. Debido a que varios nodos pueden instalar celdas hacia el mismo receptor, la contención puede surgir en dichas celdas.

- Celdas Compartidas Basadas en Remitentes (SBS, por sus siglas en inglés): Las SBS son similares a las RBS, excepto que las coordenadas de la celda se obtienen de las propiedades del nodo emisor en lugar del receptor. En cada nodo, una celda SBS resulta en una celda Rx por vecino (coordenadas basadas en el vecino) y una sola celda Tx (coordenadas basadas en el nodo remitente). Esto resulta en un mayor consumo de energía que el RBS (las celdas Tx no cuestan nada cuando no hay tráfico, mientras que las celdas Rx siempre requieren una activación), pero también puede ayudar a disminuir la contención evitando la asignación de celdas por receptor.

  Por ejemplo, para la comunicación de hijo a padre, los nodos tienen una celda Tx fija, y los padres mantienen una celda Rx para cada uno de sus hijos. Siempre que cambie de padre, el hijo no necesita actualizar su celda de Tx, pero el padre debe eliminar una celda de Rx y el nuevo padre instalar una nueva.
- Celdas Dedicadas al Emisor (SBD, por sus siglas en inglés). En un bloque de ranuras lo suficientemente larga como para acomodar las celdas de transmisión únicas a cada nodo, y suponiendo que se disponga de identificadores de nodo únicos, es posible la comunicación libre de contención. Orchestra logra comunicación libre de contención con SBD, que son similares a SBS excepto que utilizan celdas dedicadas en lugar de compartidas. Debe tenerse en cuenta que con las celdas dedicadas, los paquetes perdidos son reenviados sin back-off (utilizando la siguiente celda hacia el mismo vecino). Los identificadores de nodo únicos pueden ser codificados durante la implementación u obtenidos durante la ejecución desde un administrador de red.

1.4.7. Bloque de ranuras de Orchestra

Orchestra administra varios bloques de ranuras en cada nodo, cada uno de los cuales se asigna a un plano de tráfico particular, por ejemplo, balizamiento TSCH, el tráfico de enrutamiento, aplicación. Los bloques de ranuras consisten en un conjunto de celdas, con propiedades definidas por simples reglas de programación. Los bloques de ranuras se repiten en periodos que son mutuamente primos, asegurando que funcionan de forma independiente. En caso de que celdas de diferentes bloques de ranuras se superpongan, la celda en el bloque de ranuras de prioridad más alta tiene prioridad.

La longitud de un bloque de ranuras introduce una relación de compromiso en capacidad de tráfico, latencia de red y consumo de energía:

- Capacidad de tráfico: Los bloques de ranuras más cortas repiten sus celdas activas más a menudo, resultando en una mayor capacidad de tráfico. El enfoque de Orchestra es sobre-provisionar TSCH para soportar el tráfico no determinista y la longitud de los bloques de ranuras es la principal manera de controlar la cantidad de sobre-provisionamiento para un plano de tráfico dado.
- Latencia de red: La latencia por salto en un plano de tráfico dado es básicamente proporcional a la longitud de los bloques de ranuras para este plano particular de tráfico.
- Consumo de energía: Del mismo modo, cuanto más corto sea el bloque de ranuras, más frecuentemente los nodos tienen que despertarse para escuchar o transmitir, lo que resulta en una línea de base de energía más alta.

1.4.8. Reglas de programación

Orchestra mantiene sus horarios usando reglas de programación simples. Las reglas de programación son un conjunto de bloques de ranuras con una serie de propiedades específicas de Orchestra. Las propiedades de un bloque de ranuras Orchestra son:

- *Handle*: Un entero positivo único para la identificación y la prioridad. Cuanto menor es, mayor es la prioridad.
- Longitud: Número de ranuras que componen el bloque de ranuras. Debe ser mutuamente primo con todas las otras longitudes de bloques de ranuras en la red.
- Filtrar el tráfico: El plano de tráfico para el que está previsto el bloque de ranuras. Propiedades de los filtros de paquetes (por ejemplo, unicast, broadcast) y protocolos (por ejemplo, TSCH, RPL).

Los bloques de ranuras están formados por celdas de Orchestra, cada una asignada en 0, 1 o múltiples celdas TSCH dependiendo del estado de TSCH y RPL. Por ejemplo, una celda de Orchestra puede ser reservada para la comunicación con todas las fuentes de tiempo de TSCH, hijos de RPL o el padre preferido de RPL. Las propiedades de una celda son:

- Vecinos: El vecino o conjunto de vecinos de la celda de Orchestra se declara, como el padre RPL o todos los hijos RPL. Las ranuras TSCH resultantes se actualizan automáticamente cuando se producen cambios en el estado TSCH o RPL.
- Coordenadas: La ranura y el canal dentro del bloque de ranuras. Puede ser fijo o una variable como un ID de nodo o un hash de la dirección MAC vecina.
- Opciones: Opciones estándar de una celda TSCH. Incluye: Rx, Tx y compartida (S), la definición de para qué se puede utilizar la ranura, y si es compartida o dedicada.

Capítulo 2

2.1. Selección de la tecnología a emplear.

Para correr las simulaciones que fundamentan esta investigación, los autores decidieron emplear el sistema operativo ContikiOS, el simulador COOJA y el mote Z1 de la compañía española Zolertia atendiendo a las poderosas características de estos, las cuales son analizadas en detalle en las secciones 2.1.1, 2.1.2 y 2.1.3.

2.1.1. El Sistema Operativo Contiki

Para seleccionar el sistema operativo se tuvo en cuenta las siguientes alternativas: TinyOS, FreeRTOS, Contiki, QNX y RTEMS. Se selecciona Contiki atendiendo a que es el más desarrollado y lo respalda una enorme comunidad puesto que está desarrollado por personas tanto del ámbito industrial como del académico [16, 17]. Lo dirige Adam Dunkels, del Instituto Sueco de Ciencias de la Computación. Este coordina un equipo formado por miembros de SICS, SAP, Cisco, Atmel, NewAE y TU Munich.

Una instalación completa de Contiki incluye las siguientes características:

- Núcleo multitarea.
- Subscripción opcional por tarea a multihilo.
- Protohilos.
- Protocolo de red TCP/IP, incluyendo IPv6.
- Interfaz gráfica para el usuario y sistema de ventanas.
- Acceso mediante virtual network computing.
- Un navegador web (decía ser el más pequeño del mundo).
- Servidor de web personal.
- Cliente sencillo de Telnet.
- Protector de pantalla.

Se ha transferido a múltiples arquitecturas, incluyendo la AVR de Atmel. En la figura 2.2 se muestra el aspecto de este SO en un terminal VNC. Como se puede observar, el resultado es notable trabajando sobre un pequeño microcontrolador.

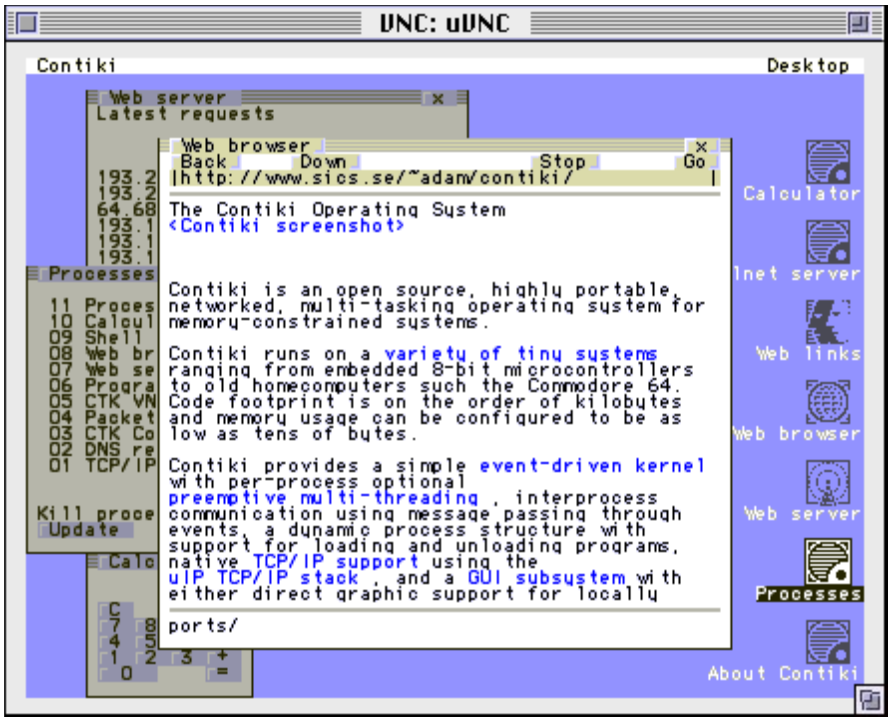


Figura 2.2: Aspecto de Contiki en un terminal VNC corriendo sobre un AVR.

Contiki es un sistema operativo diseñado para entornos con restricciones de memoria, como los nodos utilizados en LLNs. Está basado en un nucleo orientado a eventos, sus características incluyen cargas y descargas dinámicas de programas y servicios individuales, así como multihilo opcional preventivo. También admite una pila TCP/IP completa a través de la biblioteca uIP, así como la abstracción de programación mediante Protohilos. Contiki está implementado en el lenguaje C y ha sido diseñado para ser fácilmente exportable a nuevas plataformas de hardware. Ha sido exportado a más de 20 plataformas diferentes desde su lanzamiento en 2003. Los procesos pueden ser creados por el usuario para desarrollar tareas específicas. Estos se guardan en una lista encadenada: solo se necesita conocer la dirección del primero puesto que cada proceso conoce la dirección del siguiente. Cada proceso tiene un Protohilo asociado, el cual se ejecuta cuando el proceso es llamado. Los estados posibles de los procesos son: desactivado, activado y llamado. Un proceso está en estado llamado cuando se está corriendo su Protohilo. Está activado cuando puede ser llamado mediante eventos y está desactivado cuando ningún evento puede despertar al Protohilo.

En un sistema puramente basado en eventos, se implementa un proceso como manejador de eventos, permitiendo que se ejecuten distintos bloques de código, dependiendo de qué evento ocurre. Estos bloques siempre se ejecutan hasta su finalización una vez llamados. Dado que

un solo bloque de código nunca se interrumpirá, estos bloques se pueden diseñar para que todos puedan compartir la misma pila. En comparación, un modelo de múltiples hilos requiere menos memoria y sobrecarga computacional al tener varios procesos simultáneos. El kernel de Contiki contiene un manejador de eventos que envía los eventos a los procesos y periódicamente sondea los procesos que registraron una función de manejador de encuestas. Utiliza una única pila para todos los procesos, que es rebobinada entre cada invocación de un manejador de eventos.

A diferencia de la mayoría de los sistemas basados en eventos, Contiki soporta multihilo preventivo. Esto se realiza a través de una biblioteca que puede estar opcionalmente enlazada con programas que lo requieran. Permite a los nodos ejecutar aplicaciones que normalmente no funcionan bien en sistemas basados en eventos, como los cálculos criptográficos. Tal cálculo ocuparía de otro modo el sistema entero durante mucho tiempo.

Además de los hilos preventivos, Contiki también soporta Protohilos. Normalmente, cuando se escriben programas para sistemas basados en eventos, éstos deben escribirse como máquinas de estado explícitas con un código resultante que puede ser difícil de entender y mantener. Protohilos son una abstracción de programación utilizada en la parte superior de estos sistemas, con el propósito de simplificar las implementaciones de funcionalidad de alto nivel. Mediante el uso de Protohilos, los programas pueden realizar bloqueo condicional sin la sobrecarga de subprocesos regulares; los Protohilos son apilados y solo se requieren 2 bytes de RAM para cada uno. Un proceso regular de Contiki consiste en un solo Protohilo.

En una puesta en marcha regular de Contiki el sistema, entre otras cosas, inicializa algunos procesos. A continuación, llama repetidamente a la función del sistema *process_run ()*. Esta función llama a todos los manejadores de encuesta registrados y a continuación procesa un evento de la cola de eventos del sistema actual. Después de procesar el evento único, la función devuelve el número de eventos no procesados que todavía están en la cola. Si la cola de eventos se vacía, el sistema puede optar por hibernar para ahorrar energía. Si una alarma externa despierta al sistema más tarde, vuelve a realizar los bucles alrededor de la misma función *process_run ()* hasta que se manejan todos los nuevos eventos.

UIP (micro IP) es una pequeña implementación TCP / IP adecuada para nodos de sensores y otros dispositivos de recursos limitados. Está diseñado para tener sólo el mínimo absoluto de las características necesarias para una pila TCP / IP completa, y se centra en los protocolos TCP, ICMP e IP. UIP utiliza un solo proveedor global para mantener paquetes, lo

suficientemente grande como para contener sólo un paquete de tamaño máximo. Cuando un nuevo paquete de datos llega de la red, el controlador del dispositivo de red lo coloca en el proveedor global y llama a uIP a manejar los nuevos datos. Después de analizar los datos de paquetes entrantes, uIP notifica la aplicación deseada. Debido a la única fuente, esta aplicación debe actuar de inmediato para evitar que los datos sean sobrescritos por otro paquete entrante. En este punto, la aplicación también puede optar por enviar inmediatamente una respuesta utilizando el mismo proveedor global. De forma similar, cuando una aplicación desea enviar datos, pasa un puntero a los datos, así como la longitud de uIP, que escribe los encabezados y llama al controlador de dispositivo de red para enviar el paquete a la red.

2.1.2. El simulador COOJA

COOJA es un simulador de aplicaciones de Contiki OS, no un emulador de nodo. Cuando un usuario crea un nuevo tipo de nodo, el sistema Contiki resultante se compila para la plataforma de simulación, en el entorno regular de Contiki. Mientras que los controladores de una plataforma de hardware operan en el hardware, los controladores de la plataforma de simulación operan en la parte Java del simulador. Pero como la plataforma de simulación soporta las mismas intefaces que cualquier plataforma de hardware, no hay diferencia entre ellos desde el punto de vista de una aplicación. El mismo código de aplicación se compila y ejecuta en ambas plataformas. Así que los periféricos de hardware de una plataforma no son emulados, sino que son reemplazados por otros dispositivos simulados. Una alternativa sería emular todo el hardware de un nodo sensor, lo que podría dar resultados más precisos, pero también limitaría el simulador a una o dos plataformas de hardware soportadas. Mediante el uso del primer método, diferentes plataformas de hardware pueden ser soportadas simplemente conectando todos los controladores deseados.

En resumen, una simulación COOJA consiste en un número de nodos que se están simulando. Cada nodo está conectado a un tipo de nodo. Cuando se ejecuta la simulación, todos los nodos actúan a su vez. Cuando todos los nodos han actuado, el tiempo de simulación se actualiza y se repite el proceso.

En detalle, cada nodo tiene su propia memoria y un número de interfaces. La memoria consiste en uno o varios segmentos de memoria, cada uno con una dirección de inicio y datos. Los segmentos de memoria deben incluir todas las partes necesarias y de interés de un sistema operativo Contiki simulado.

Las interfaces actúan sobre la memoria y simulan dispositivos de nodo como un reloj o un transmisor de radio. Por ejemplo, cuando el tiempo cambia, una interfaz de reloj debe actualizar alguna variable de tiempo específico. Y esa variable reside en la memoria de ese nodo.

El tipo de nodo es el puente entre el nodo que se explicó anteriormente y una ejecución de código c del nodo específico con Contiki OS cargado. Este es el punto donde se inicializa el sistema operativo Contiki simulado y se crea la memoria inicial. Todos los nodos del mismo tipo están relacionados con el mismo sistema operativo Contiki cargado. El tipo de nodo también recibe el nombre de la variable de la asignación de dirección. Esto implica que, si la interfaz de reloj quiere cambiar el tiempo de variable de núcleo, se pregunta al tipo de nodo en qué dirección está esa variable. Cuando un nodo actúa, el tipo de nodo es responsable de vincular el nodo correspondiente a su sistema operativo Contiki.

Hay un funcionamiento Contiki OS para cada tipo de nodo existente. Esto significa que todos los nodos del mismo tipo comparten el mismo sistema operativo Contiki. Se compone de código de Contiki compilado como una librería compartida hacia una plataforma de simulación, y la comunicación entre la parte de Java y el núcleo es a través de *Java Native Interface*. La diferencia entre un proceso de Contiki OS y un proceso de aplicación de Contiki es que la primera es todo el sistema operativo Contiki compilado y ejecutado, mientras que la última es un proceso existente sólo dentro del entorno Contiki.[1] Hay sólo unas pocas funciones nativas que conectan el núcleo al simulador, de los cuales los tres más importantes son una función *memory set ()*, una función *memoria get ()* y una función *tick* cuando un nodo actúa.

El simulador está basado en Java puro mientras que el núcleo está implementado en el lenguaje C (como el propio Contiki OS). Como regla general, el simulador es responsable de todo lo externo a un nodo específico, cómo funciona el medio de radio circundante o el tiempo de simulación actual. El núcleo es responsable de los trabajos internos del nodo; que ejecuta código Contiki real, permitiendo que el mismo código de aplicación simulada sea utilizado sin alteraciones en los nodos físicos reales.

[1] Cuando se hace referencia al proceso de Contiki OS el término "núcleo" será utilizado, y cuando se hace referencia al resto del simulador el término "simulador" será el utilizado. Obviamente, el núcleo es también una parte del simulador, pero para simplificar el texto se hace esta distinción.

2.1.3. Plataforma Z1

Z1 de la compañía española Zolertia es un módulo de LLNs que sirve como una plataforma de desarrollo de propósito general para desarrolladores, investigadores, entusiastas y aficionados de WSN.

Este módulo se desarrolló con dos objetivos:

- Máxima compatibilidad con los exitosos motores de la familia Tmote-X mientras mejora el rendimiento en varios aspectos, y
- Máxima flexibilidad y capacidad de expansión con respecto a cualquier combinación de fuentes de alimentación, sensores y conectores.

Viene con el apoyo de algunos de los sistemas operativos open source actualmente empleados por la comunidad WSN, como TinyOS y Contiki. Las pilas de red compatibles incluyen 6LowPAN (por medio de BLIP en TinyOS), SimpliciTI de Texas Instruments, Z-Stack (hasta Zigbee 2006), y Freaklabs de código abierto de Zigbee y Freaklabs de Chibi.

El Z1 es un módulo inalámbrico de baja potencia que cumple con los protocolos IEEE 802.15.4 y Zigbee destinado a ayudar a los desarrolladores de LLNs a probar y desplegar sus propias aplicaciones y prototipos con el mejor equilibrio entre el tiempo de desarrollo y la flexibilidad del hardware. Su arquitectura principal se basa en la familia de microcontroladores y transmisores de radio MSP430 + CC2420 de Texas Instruments, lo que la hace compatible con motes basadas en esta misma arquitectura. Sin embargo, la MCU presente en Z1 es la MSP430F2xxx en lugar de la MSP430F1xxx, como es habitual entre otras plataformas, como TelosB de Crossbow's, Tmote de Moteiv's, y similares [15]. Este hecho supone diferencias sutiles debido a cambios internos entre los dispositivos F1xxx y F2xxx, pero no se espera que estas diferencias surjan en el nivel de aplicación firmware si se emplea un sistema operativo soportado al desarrollarse.

2.2. Escenarios

Atendiendo a la carga computacional que supone la simulación en COOJA se emplea la simulación automática, o sea, prescindiendo de la interfaz gráfica que a pesar de ser bastante intuitiva no aporta datos relevantes. En cada simulación se realiza de 25 a 50 iteraciones para asegurar la validez estadística de los resultados. En cuanto a la topología, se emplea una distribución seudo-aleatoria de 22 nodos (un sumidero y 21 nodos sensores) generada por el propio COOJA, la cual puede apreciarse en la tabla 2.1 en correspondencia con el sistema de

coordenadas cartesianas, así como en la figura 2.1. Todos los escenarios se corrieron con las cargas de tráfico {8 4 2 1 0.5 0.25 0.125} paquetes por minuto, lo que se traduce en un paquete cada {7.5, 15, 30, 60, 120, 240, 480} segundos.

Tabla 2.1: Distribución exacta de los nodos en el escenario según el sistema de coordenadas cartesianas. Es importante destacar que el nodo 1 corresponde al sumidero y el resto a los nodos sensores.

Id nodo	**x (metros)**	**y (metros)**
1	46.91201961487572	41.73811830288886
2	65.50409002427536	103.78614926871103
3	40.40998286571654	84.79302344635137
4	30.80168370046492	59.39523541288516
5	7.331083671174838	83.4039317249113
6	47.54017353939433	97.25224324084007
7	59.489632349700045	62.02566364848339
8	11.987307746017606	130.21270551793566
9	29.321581660183696	134.8310334644794
10	-11.007431344030138	101.08734740186985
11	72.16214791345901	40.85610949482632
12	12.76793706445929	26.79630095231974
13	27.30616229415061	27.342010896341947
14	47.09774258087775	113.5023728561163
15	20.232169092663923	105.40890889068986
16	2.180522397366772	28.63691567171493
17	70.35279916174879	94.27089925604041
18	72.5452876592082	-26.04216703750994
19	69.26337838201054	-6.00491470631642
20	74.98263662451411	15.137472592738423
21	94.22519987555619	2.0327297850837596
22	44.28614234525038	1.637141826094492

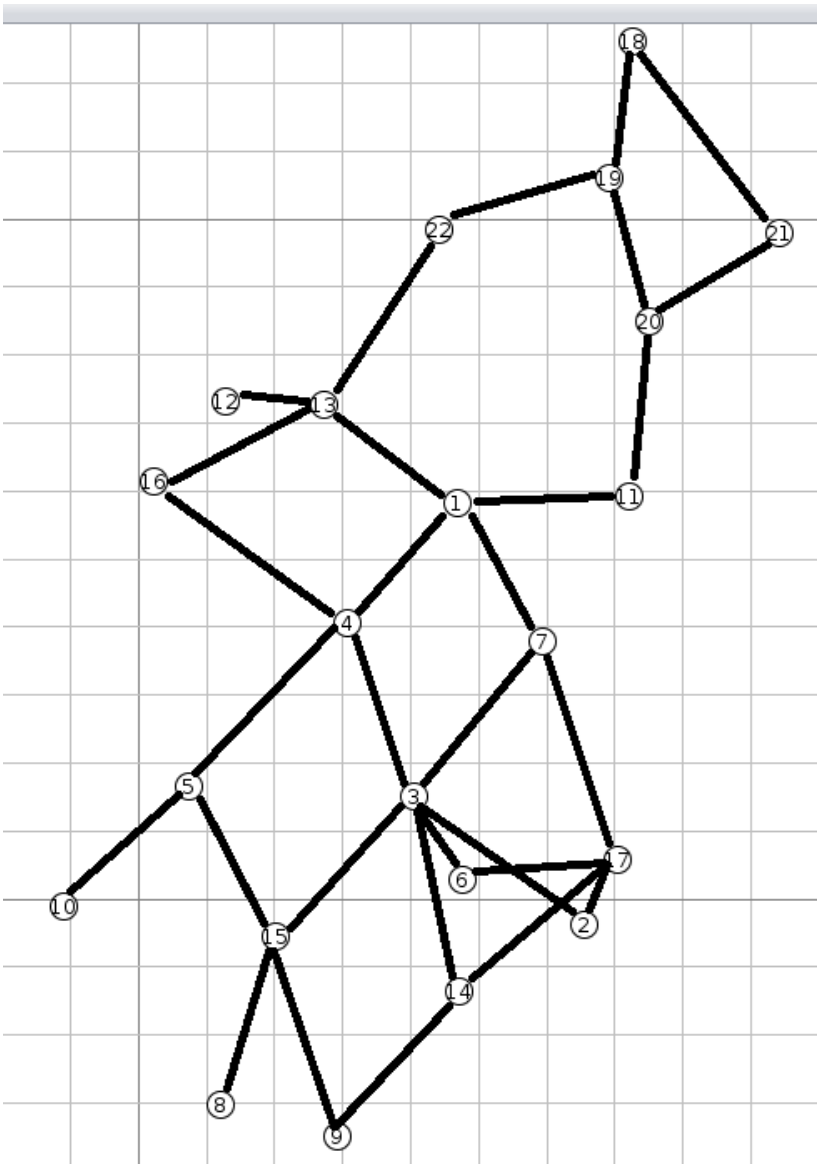


Figura 2.1: Topología a emplear para todos los escenarios.

2.2.1. Escenario 1: "Tiempo requerido para unirse a la red."

En este escenario se modifica el código fuente de la implementación de TSCH en ContikiOS para que los nodos impriman un mensaje cuando se asocian a la red TSCH. De esta forma se puede registrar el momento exacto en que cada nodo se une a la red para así discernir sobre la secuencia de salto de canales óptima para el proceso de incorporación de los nodos a la red TSCH. Es importante destacar que TSCH distingue dos tipos de secuencias de salto de canal: una para la unión de los nodos a la red y la otra para la propia operación de estos nodos después de pertenecer a la red. Asimismo, también debe destacarse que la implementación de TSCH

en ContikiOS predefine cinco secuencias de salto de canal como puede apreciarse en la tabla 2.2. Nótese que se emplea con especial énfasis los canales 15, 20, 25 y 26 porque reducen la interferencia con las redes IEEE802.11.

Tabla 2.2: Secuencias de salto de canal predefinidas para TSCH en ContikiOS.

Canales empleados	Longitud secuencia	Nombre Secuencia	Secuencia
1	1	TSCH_HOPPING_SEQUENCE_1_1	{ 20 }
2	2	TSCH_HOPPING_SEQUENCE_2_2	{ 20, 25 }
4	4	TSCH_HOPPING_SEQUENCE_4_4	{ 15, 25, 26, 20 }
4	16	TSCH_HOPPING_SEQUENCE_4_16	{ 20, 26, 25, 26, 15, 15, 25, 20, 26, 15, 26, 25, 20, 15, 20, 25 }
16	16	TSCH_HOPPING_SEQUENCE_16_16	{ 16, 17, 23, 18, 26, 15, 25, 22, 19, 11, 12, 13, 24, 14, 20, 21 }

Este escenario se corre un total de 50 iteraciones empleando todas las posibles combinatorias de las cinco secuencias de salto de canal antes mencionadas: funcionando como secuencias para la unión a la red TSCH; y como secuencias para la operación de los nodos ya incorporados a la red TSCH.

2.2.2. Escenario 2: "Longitud de los bloques de ranuras."

En este escenario se varía las longitudes de los bloques de ranuras. La implementación de Orchestra en ContikiOS predefine la construcción de tres bloques de ranuras: uno para tramas EBs, otro para celdas compartidas y el otro para el tráfico de unidifusión a nivel de aplicación. Para lograr la construcción de estos bloques basta declarar el periodo del bloque de ranuras del mismo tipo (el cual se maneja en cantidad de ranuras). Por defecto se define periodo del bloque para EBs igual a 397 ranuras, el del bloque de celdas compartidas para RPL igual a 31 ranuras y periodo del bloque para tráfico de unidifusión igual a 17 ranuras. Nótese que los valores deben ser mutuamente primos para que las ranuras no se solapen a medida que aumente el ASN.

En la implementación de este escenario los autores emplean el periodo predefinido, uno que es el doble y otro que es la mitad, o sea, {397, 31, 17}, {197, 17, 7} y {787, 61, 31} respectivamente, atendiendo a que deben ser valores primos.

De esta forma, atendiendo a las diferentes cargas de tráfico definidas para todos los escenarios, se obtiene una medida de la relación costo beneficio de los horarios densos o escasos en cuanto al manejo de la congestión y el consumo de energía. Para ello se mide PDR (del inglés *packet delivery ratio*), demora y energía.

2.2.3. Escenario 3: "RPL en modo almacenamiento contra RPL en modo no-almacenamiento."

La implementación de Orchestra en ContikiOS se centra, por defecto, en el modo almacenamiento de RPL pero es fácilmente configurable para trabajar en modo no-almacenamiento. Recuérdese que en modo almacenamiento cada nodo mantiene una tabla de enrutamiento con respecto a sus hijos DODAG, en cambio, en el modo no-almacenamiento la raíz del DODAG es la que mantiene la tabla de enrutamiento con respecto a todos los nodos de la red. En el primer caso, si un nodo hoja le envía un paquete a otro nodo hoja, dicho paquete solo tiene que retroceder hasta el ancestro común más próximo. En el segundo caso, el paquete siempre tiene que retroceder hasta la raíz del DODAG. En el primer caso el procesamiento de cada paquete recibido en cada nodo es más complejo y cuando un nodo cambia a su padre RPL se generan varios mensajes DAO para que cada uno de sus ancestros actualice sus tablas de enrutamiento. En el segundo caso el procesamiento de los mensajes recibidos en cada nodo es más simple y con un solo mensaje DAO se le notifica a la raíz DODAG cuando un nodo cambia de padre.

En este escenario se comprueba el efecto de emplear RPL en modo almacenamiento o en modo no-almacenamiento atendiendo a los resultados de PDR, demora y energía.

Capítulo 3

3.1. Discusión de los resultados obtenidos en el escenario 1: "Tiempo requerido para unirse a la red."

La simulación del escenario 1 permite obtener el momento exacto de incorporación a la red TSCH de cada nodo, empleando las cinco secuencias de salto de canal predefinidas en TSCH (ver sección 2.2.1). La tabla con los resultados preprocesados se encuentra en el Anexo A (puesto que la tabla con los resultados puros contiene 26250 filas lo que la hace difícil de incluir en el informe). A su vez la figura 3.1 muestra el resumen de los resultados. Nótese que para obtener el tiempo que necesita cada nodo para unirse a la red no basta el momento en el que este se unió, sino que hay que tener en cuenta la topología. Por ejemplo, según la topología mostrada en la figura 2.1, el nodo 8 estará en condiciones de unirse a partir de que el nodo 15 se haya unido y comience a enviar EBs; el nodo 8 tendrá que aguardar a que se una a la red el nodo 5 o el nodo 3 y que estos comiencen a enviar EBs. De esta forma el tiempo que necesita el nodo 15 para unirse es igual a su momento de unión menos el momento de unión de 8 y el tiempo que necesita 8 es igual a su momento de unión menos el mínimo entre los momentos de 5 y 3.

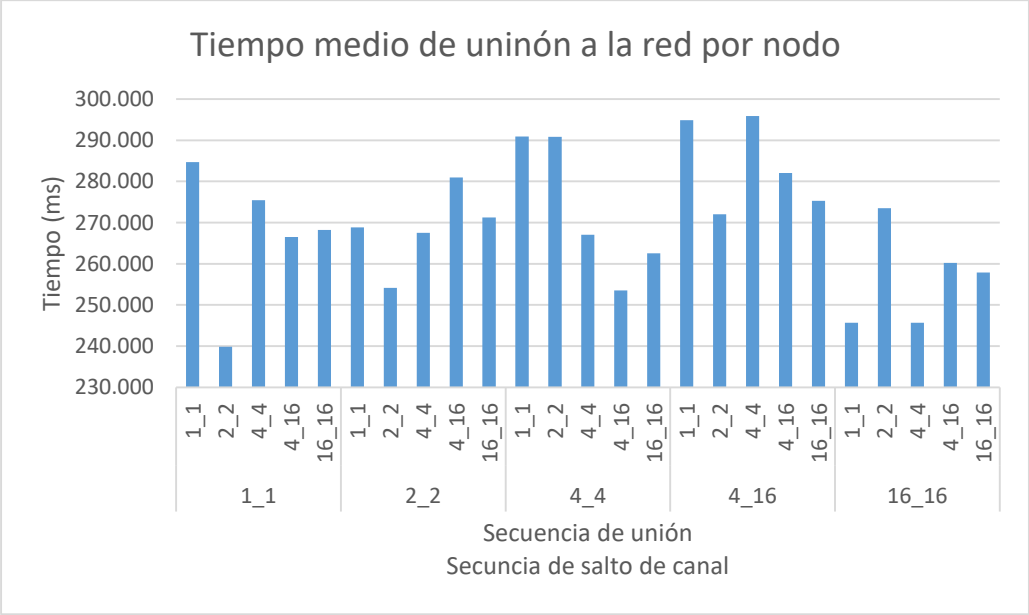


Figura 3.1: Tiempo de asociación a la red TSCH por nodo, empleando las cinco secuencias predefinidas en TSCH tanto de secuencias para el salto de canal como de secuencias para la unión.

En un primer momento se esperaba que el mejor tiempo de unión a la red lo lograra la secuencia de salto 1_1 en combinación con la secuencia de exploración 1_1 puesto que los nodos siempre iban a estar escuchando en el mismo canal donde se trasmitiría el EB: siguiendo esta lógica la red estaría conformada completamente en el instante del EB cuyo número se correspondiera con la distancia en saltos del nodo más alejado hasta la raíz. Asimismo, el tiempo de unión de los nodos a la red para las demás secuencias estaría determinado por un modelo probabilístico que relacionara los canales en los que se trasmitían EBs con los que se exploraba en busca de EBs para obtener la probabilidad de que los nodos escucharan los EBs.

Sin embargo, como puede apreciarse en la figura 3.1, la combinación 1_1 con 1_1 alcanza uno de los mayores tiempos de asociación por nodo demostrando que el tiempo de unión de los nodos a la red TSCH no se rige de la forma esperada. Esto se debe a que (como puede observarse en la sección 5.7. de [2]) los EB requieren de una confirmación por parte del nodo que intenta unirse a la red. Al estar operando en un único canal la probabilidad de que los nodos escuchen el EB es de 1, pero eso también incrementa la probabilidad de que la confirmación del EB colisione o sea postergada para el próximo EB (según el *back-off* que impone CSMA-CA) cuando hay más de un nodo escuchando. De esta forma queda planteado como recomendación realizar un estudio analítico de este tema para obtener el sistema real por el que se rige el tiempo de asociación de los nodos a la red TSCH empleando diferentes secuencias de canales.

Así que de los resultados de este escenario importante resaltar que los tiempos de asociación más bajos los obtuvo la secuencia de salto de canal 16_16, lo que sumado a los beneficios en cuanto a ancho de banda y resistencia al ruido y al desvanecimiento multi-ruta que significa el empleo de esta secuencia, la convierten en la secuencia de salto de canal idónea para la operación de TSCH. De igual forma, las mejores secuencias para la unión son 1_1 o 4_4.

Como conclusión parcial, la mejor secuencia de salto de canal para la operación de TSCH es 16_16 y las mejores secuencias para la unión son 1_1 y 4_4.

3.2. Discusión de los resultados obtenidos en el escenario 2: "Longitud de los bloques de ranuras."

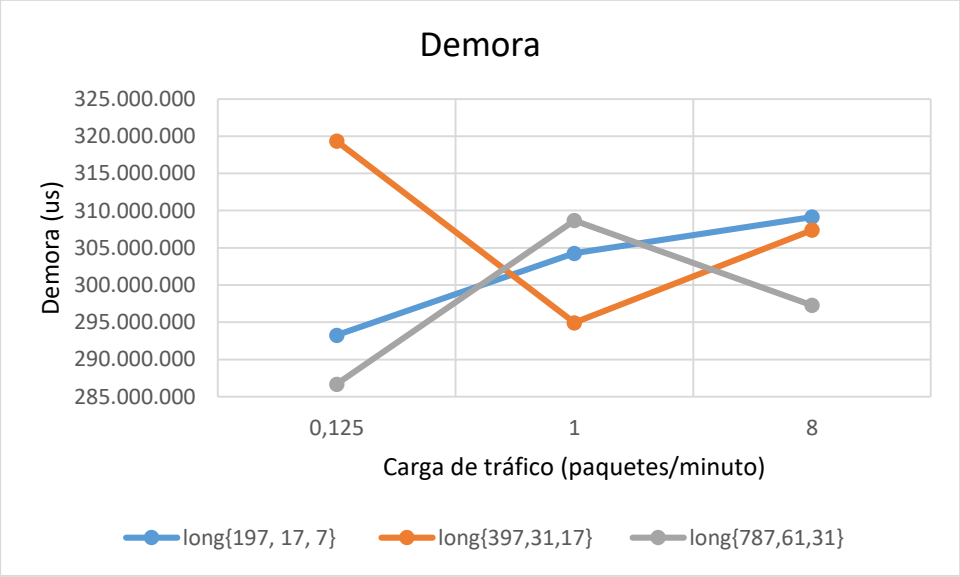
La simulación del escenario 2 permite obtener el comportamiento de TSCH y Orchestra para diferentes longitudes de bloques de ranuras (longitud 1{197, 17, 7}, longitud 2{397, 31, 17} y

longitud 3{787, 61, 31}; ver sección 2.2.2) y diferentes cargas de tráfico en cuanto a demora de los paquetes en la red, razón de entrega exitosa de paquetes (PDR, por sus siglas en inglés) y energía consumida por cada nodo. La tabla 3.2 muestra el resumen de los resultados obtenidos y las figuras 3.2. A), B), C) y D) representan dichos resultados de forma gráfica. El anexo B muestra los resultados puros obtenidos durante la simulación del escenario 2.

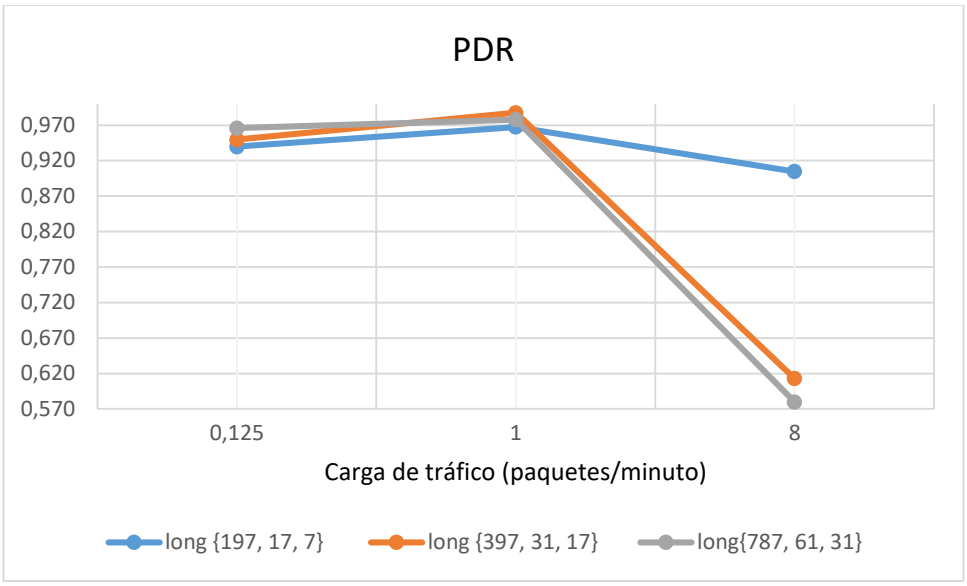
Tabla 3.2: Resultados del escenario 2 en cuanto a demora, PDR y energía consumida.

carga (pkt/min)	**0,125**	**1**	**8**
Demora {197, 17, 7}	293.258.085	304.263.378	309.149.573
Demora {397, 31, 17}	319.333.555	294.926.687	307.376.820
Demora {787, 61, 31}	286.684.189	308.682.941	297.272.369
PDR {197, 17, 7}	0,940	0,967	0,905
PDR {397, 31, 17}	0,950	0,987	0,613
PDR {787, 61, 31}	0,966	0,978	0,580
Energía {197, 17, 7}	2,864	2,958	3,127
Energía {397, 31, 17}	2,966	2,957	2,983
Energía {787, 61, 31}	3,154	2,947	2,885

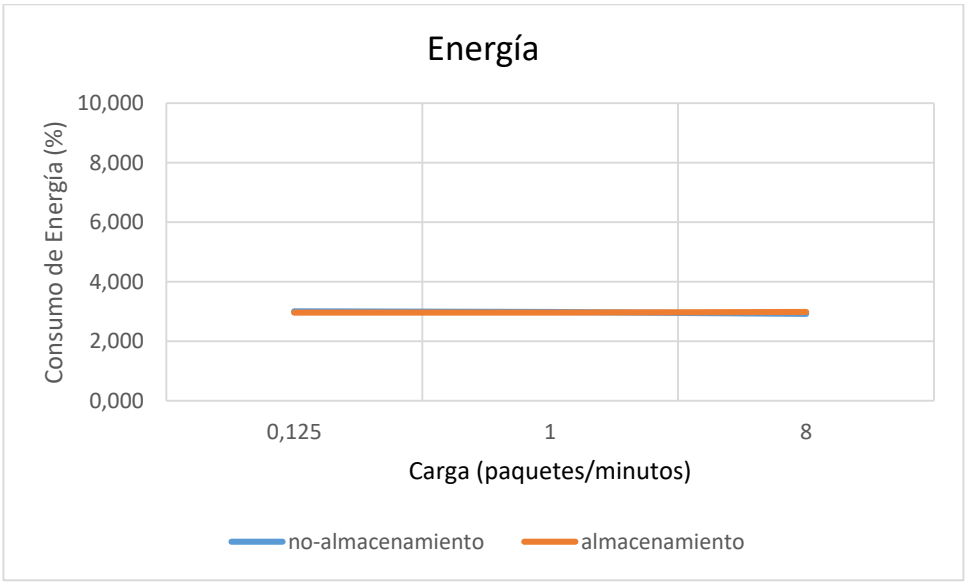
A)



B)


PDR
0,970
0,920
0,870
0,820
0,770
0,720
0,670
0,620
0,570
0,125
1
8
Carga de tráfico (paquetes/minuto)
long {197, 17, 7}
long {397, 31, 17}
long{787, 61, 31}

C)


Energía
Consumo de Energía (%)
10,000
8,000
6,000
4,000
2,000
0,000
0,125
1
8
Carga (paquetes/minutos)
no-almacenamiento
almacenamiento

D)

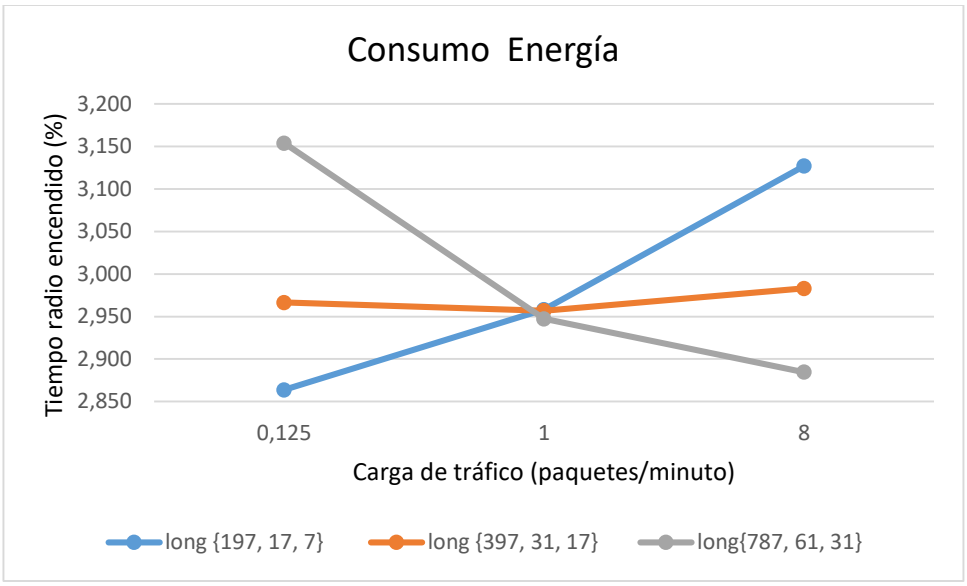


Figura 3.2: Resultados del escenario 2. **A):** Demora. **B):** PDR. **C):** Energía consumida (rango de 0% a 10%). **D):** Energía consumida (rango de 2,85% a 3,2%).

Como puede apreciarse en la figura 3.2 A), para cargas de tráfico bajas las longitudes 1 y 3 presentan una demora menor, espaciadas entre ellas apenas 7 ms, mientras que la longitud 2 alcanza 33 ms más de demora. Para cargas de tráfico medias, la diferencia en la demora es de apenas 14 ms logrando la longitud 2 un pico inferior y la 3 un pico superior. Para una carga de tráfico alta la diferencia entre las demoras es de solo 12 ms, mostrando las longitudes 1 y 2 un comportamiento ascendente y la longitud 3 un comportamiento descendente.

Esto se debe a que la longitud 1 está muy sobredimensionada en cuanto a ancho de banda con respecto a las otras longitudes y tiene sus EBs más cercanos en el tiempo por lo que puede mantener el sincronismo a pesar del poco tráfico y su comportamiento es ir aumentando la demora a medida que aumenta el tráfico. En el caso de la longitud 2, con un ancho de banda medio y sus EBs más distanciados, cuando la carga de tráfico es inferior a su umbral óptimo de trabajo (con umbral óptimo de trabajo nos referimos, según los experimentos realizados, a la carga de tráfico justa que puede manejar la longitud de bloques de ranura), la deriva del sincronismo hace que aumente la demora. A medida que aumenta el tráfico en la red y se aproxima a su umbral óptimo, la demora disminuye porque logra el mejor equilibrio entre paquetes en la red y sincronismo. Y cuando la carga aumenta por encima de lo que puede manejar comienza a incrementarse también las colas aumentando así la demora y, en última instancia, comenzaría a descartar paquetes cuando los buffers amenazaran con desbordarse.

En el caso de la longitud 3 logra su umbral óptimo de trabajo cuando la carga de tráfico en la red es pequeña. A medida que aumenta la carga de tráfico aumenta la demora porque la cantidad de paquetes en la red es mayor que la manejable. De esta forma llega al pico en el que ya son tantos los paquetes en la red que tiene que comenzar a descartarlos disminuyendo así la demora nuevamente, pero en detrimento del PDR. Se arriba en este experimento como conclusión parcial, a que el mejor comportamiento en cuanto a demora de las diferentes longitudes de bloques de ranura es cuando se aproximan a su umbral óptimo de trabajo donde logran equilibrar los paquetes manejados y el sincronismo.

En la figura 3.2 B), puede apreciarse que para cargas bajas y medias las tres longitudes logran resultados de PDR bastante similares porque aún enfrentan volúmenes de paquetes en la red manejables. Pero para una alta carga de tráfico solo la longitud 1 logra mantener el PDR por encima de 0,9 mientras que las longitudes 2 y 3 disminuyen su PDR hasta casi 0,6 debido a que la carga de tráfico en la red es mayor que la que pueden manejar y descartan paquetes. Como conclusión parcial, solo una longitud de bloques de ranura pequeña puede manejar grandes volúmenes de tráfico manteniendo el PDR cercano a la unidad mientras que para cargas de tráfico pequeñas o medianas no es significativa, en cuanto a PDR, la longitud de bloques de ranuras.

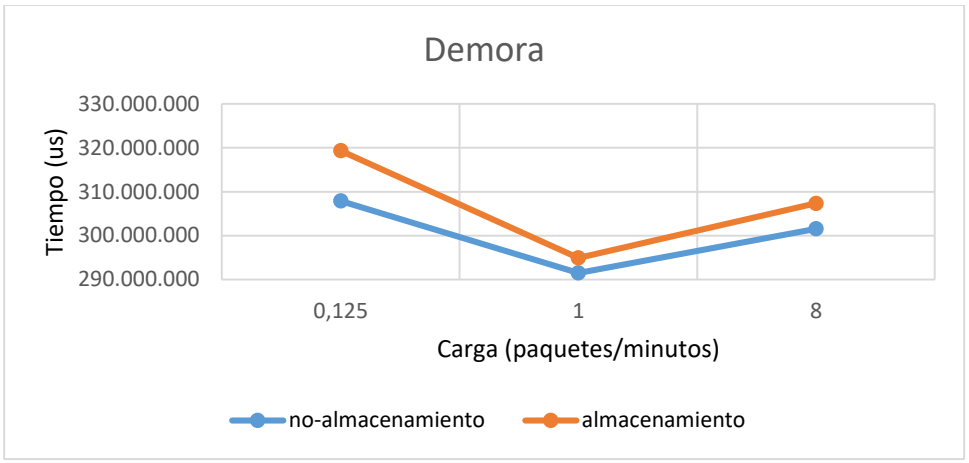
En la figura 3.2 C), puede apreciarse que el consumo de energía de los nodos es bastante similar con independencia de la carga de tráfico o la longitud de bloques de ranuras: apenas varía un 0,3 %. No obstante, como puede apreciarse en la figura 3.3 D), cabe destacar que en un primer momento el mayor consumo de energía lo presenta la longitud 3, después la 2 y por último la 1: esto se debe a que las tres longitudes están manejando la misma cantidad de paquetes por lo que para manejar el tráfico las interfaces de radio tienen que gastar aproximadamente la misma cantidad de energía. Pero a medida que los EBs están más espaciados en el tiempo demora más formar la red TSCH lo que implica que en la fase de formación de la red, las longitudes de bloques de ranura más grandes tienen que consumir más energía a través de sus interfaces de radio. De esta forma, a medida que más pequeño sea el tráfico en la red, más significativo será el aporte de la fase inicial de formación al total de energía consumida por la red. Asimismo, a medida que va aumentando el tráfico, se va haciendo cada vez menos significativo el aporte de la fase inicial de formación de la red al total de energía consumida.

3.3. Discusión de los resultados obtenidos en el escenario 3: "RPL en modo almacenamiento contra RPL en modo no-almacenamiento."

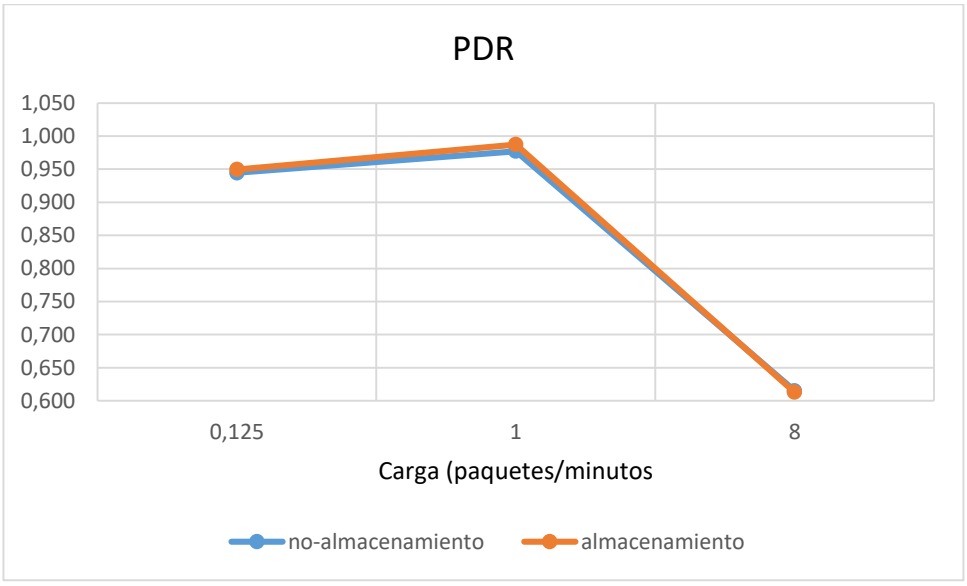
La simulación del escenario 3 permite comparar cuantitativamente el comportamiento de Orchestra empleando modo almacenamiento o modo no-almacenamiento de RPL (ver sección 2.2.3). De antemano señalar que este escenario está sujeto a varias recomendaciones puesto que se diseñó una red cuyos nodos solo enviaba paquetes al sumidero (como suelen ser la mayoría de las aplicaciones de sensores inalámbricos las cuales básicamente se dedican a recopilar información). Para tener una medida más amplia del comportamiento de Orchestra empleando uno u otro modo y así poder extender este informe a una mayor diversidad de aplicaciones de las LLNs es necesario implementar un escenario donde los nodos intercambien paquetes entre ellos y la topología varíe de forma aproximada a como varía en redes reales. Los resultados puros de la simulación del escenario 3 pueden apreciarse en el anexo C y su resumen se encuentra en la tabla 3.3; asimismo la figura 3.3 A), B) y C) muestra dichos resultados de forma gráfica.

Tabla 3.3: Resumen de los resultados del escenario 3 en cuanto a demora, PDR y energía consumida.

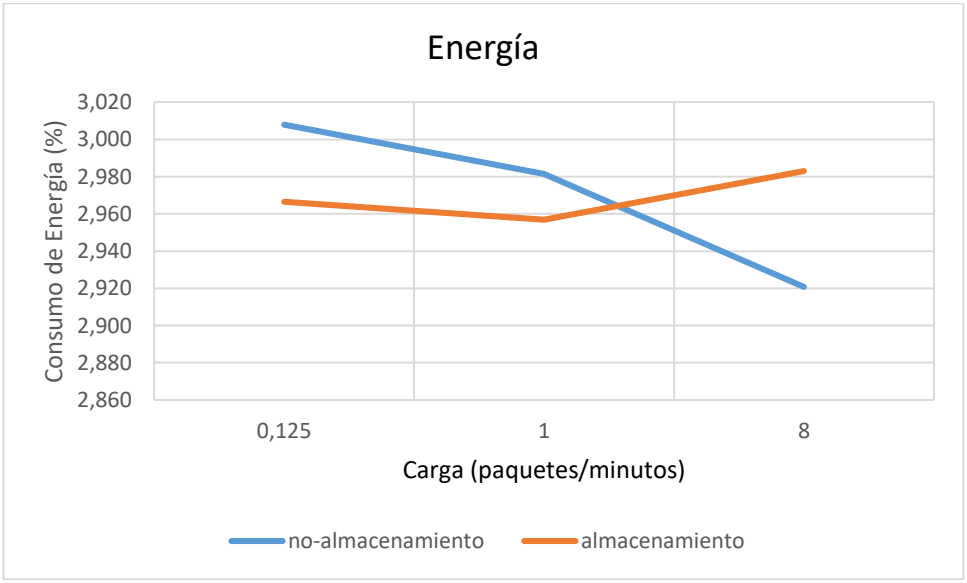
carga (pkt/min)	**0,125**	**1**	**8**
Demora no-almacenamiento	307.914.282	291.515.344	301.577.665
Demora almacenamiento	319.333.555	294.926.687	307.376.820
PDR no-almacenmiento	0,945	0,977	0,615
PDR almacenmiento	0,950	0,987	0,613
Energía no-almacenamiento	3,008	2,981	2,921
Energía almacenamiento	2,966	2,957	2,983



A)



B)



C)

Figura 3.3: Resultados del escenario 3. **A):** Demora. **B):** PDR. **C):** Consumo de energía.

Como puede verse en la figura 3.3 A), la demora apenas varía 12 ms como máximo entre Orchestra empleando modo almacenamiento o modo no-almacenamiento de RPL siendo la demora del modo no-almacenamiento siempre inferior a la demora del modo almacenamiento sin importar la carga de tráfico a la que se enfrentan. Esto se debe a que en modo almacenamiento el procesamiento de las colas es más lento debido a que cada nodo debe revisar su tabla de rutas mientras que en modo no-almacenamiento se favorece el tráfico hacia el núcleo de la red. Estos resultados se obtienen para una red cuyos nodos más alejados están a apenas cuatro saltos de la raíz, por lo que resulta lógico que en modo almacenamiento la demora aumenta a medida que aumente la complejidad de la red. Como conclusión parcial, para aplicaciones de sensores inalámbricos dedicadas a la recopilación de información es más eficiente en cuanto a demora el modo no-almacenamiento de Orchestra.

Como puede verse en las figuras 3.3 B) y C), el comportamiento de Orchestra empleando modo almacenamiento o modo no almacenamiento de RPL es similar en cuanto a PDR y consumo de energía. Como conclusión parcial, en aplicaciones de sensores inalámbricos dedicados a la recolección de información el PDR o el consumo de energía no son parámetros significativos para elegir entre el modo almacenamiento y no-almacenamiento de RPL.

Conclusiones

A partir del desarrollo de esta investigación se concluye que:

- La capa MAC de las LLNs define diferentes modos de operación orientados en dos líneas opuestas: las deterministas y las no deterministas. De las soluciones deterministas, una de las más relevantes es TSCH que está basada en Acceso Múltiple por División de Tiempo. TSCH es un modo de operación de IEEE.802.15.4 de 2015 definido para la capa MAC de LLNs y encaja bajo la pila de protocolos habilitados para IPv6 en la arquitectura de las LLNs. La técnica MAC de TSCH emplea sincronización de tiempo y salto de canal. TSCH puede enfrentar ambientes con difíciles condiciones de trabajo, como los ambientes industriales donde suele haber grandes entornos de despliegue con equipamiento metálico que causa desvanecimiento e interferencia multi-ruta haciendo inviable las soluciones basadas en un solo canal.
- IEEE.802.15.4 no define la entidad funcional que actúa de programador en la red TSCH. Orchestra y *Scheduling Function 0* son dos de las entidades funcionales que pueden realizar esta función. Con Orchestra los nodos mantienen su horario de forma local y autónoma, basado en sus vecinos y padres de RPL. Como resultado, Orchestra hace que TSCH sea tan flexible como las capas MAC asíncronas, y capaz de soportar tráfico de acceso aleatorio. Un horario de Orchestra contiene diferentes bloques de ranuras de diferentes longitudes. Cada bloque de ranuras está dedicada a un tipo particular de tráfico. Los nodos seleccionan los bloques de ranuras usando reglas de programación que reducen drásticamente la contención, o en ciertos casos elimina la contención.
- Los tres escenarios de simulación diseñados permiten examinar el desempeño de TSCH y Orchestra ante diferentes cargas de tráfico, longitudes de bloques de ranura y modos de operación de Orchestra (centrado en el modo almacenamiento o modo no-almacenamiento de RPL), en cuanto a tiempo de asociación de cada nodo a la red TSCH, demora, PDR y consumo de enrgía.
- Los resultados de las simulaciones demuestran que:
  - La mejor secuencia de salto de canal para la operación de TSCH es 16_16 y las mejores secuencias para la unión son 1_1 y 4_4.
  - El mejor comportamiento en cuanto a demora de las diferentes longitudes de bloques de ranura es cuando se aproximan a su umbral óptimo de trabajo (carga de tráfico muy

cercana al ancho de banda del sistema) donde logran equilibrar los paquetes manejados y el sincronismo.

- Solo una longitud de bloques de ranura pequeña (del orden de {397, 31, 17}) puede manejar grandes volúmenes de tráfico (del orden de 8 paquetes por minuto) manteniendo el PDR cercano a la unidad. Para cargas de tráfico pequeñas o medianas no es significativa, en cuanto a PDR, la longitud de bloques de ranuras.
- El consumo de energía no varía de forma significativa al variar la longitud de bloques de ranuras.
- Para aplicaciones de sensores inalámbricos dedicadas a la recopilación de información es más eficiente en cuanto a demora el modo no-almacenamiento de RPL.
- En aplicaciones de sensores inalámbricos dedicados a la recolección de información el PDR o el consumo de energía no son parámetros significativos para elegir entre el modo almacenamiento o no-almacenamiento de RPL.

Recomendaciones

Los autores de esta investigación proponen las siguientes recomendaciones para investigaciones futuras basadas en TSCH y Orchestra:

- Realizar un estudio analítico de los resultados obtenidos en el escenario uno para determinar el sistema por el que se rige el tiempo de asociación de los nodos a la red TSCH empleando diferentes secuencias de canales.
- Perfeccionar el escenario tres para que la topología varíe y los nodos intercambien paquetes entre ellos y con la raíz de forma similar a como suele ocurrir en las LLNs. De esta forma se puede extender la comparación entre el modo almacenamiento y no-almacenamiento de RPL más allá de las aplicaciones de sensores dedicadas a la recolección de información y parámetros.

Bibliografía

[1] P. Thubert and J. Hui, “Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks,” Internet Requests for Comments, RFC Editor, RFC 6282, Oct 2015.

[2] IEEE, IEEE 802.15.4 - IEEE Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (LRWPANs), 2011.

[3] P. Thubert, “An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4,” Internet Engineering Task Force, Internet-Draft draft-ietf6tisch-architecture-10, Jun. 2016, work in Progress.

[4] Q. Wang and X. Vilajosana, “6top Protocol (6P),” Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-6top-protocol-02, Jul. 2016, work in Progress.

[5] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, “Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH,” in Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems. New York, New York, USA: ACM Press, 2015, pp. 337–350.

[6] L. Tang, Y. Sun, O. Gurewitz, and D. B. Johnson, “EM-MAC: A Dynamic Multichannel Energy-Efficient MAC Protocol for Wireless Sensor Networks,” in Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing - MobiHoc ’11. New York, New York, USA: ACM Press, May 2011, p. 23.

[7] C. M. G. Algora, E. P. Lopez, V. A. Reguera, A. Nowe, and K. Steenhaut, “Poster: Comparative study of EM-MAC and TSCH/orchestra for IoT,” in 2016 IEEE Symposium on Communications and Vehicular Technologies (SCVT), 2016, pp. 1–6.

[8] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,” Internet Requests for Comments, RFC Editor, RFC 6550, Mar 2012.

[9] P. Thubert, “An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4,” Internet Engineering Task Force, Internet-Draft draft-ietf6tisch-architecture-10, Jun. 2016, work in Progress.

[10] “T. Chang, T. Watteyne, K. Pister, and Q. Wang. Adaptive synchronization in multi-hop tsch networks. Comput. Netw., 76(C):165{176, Jan. 2015.

[11] M. Kovatsch, S. Duquennoy, and A. Dunkels, “A Low-Power CoAP for Contiki,” in 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, 2011, pp. 855–860.

[12] Kushalnagar, N ; Montenegro, G ; Schumacher, C: IPv6 over low-power Wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals. En: RFC4919, August (2007), p. 1{12

[13] M. Palattella, L. Grieco, D. Dujovne, and N. Accettura, “6TiSCH 6top Scheduling Function Zero (SF0).” [Online]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-6top-sf0-04.

[14] S. Anamalamudi, C. Perkins, S. V. R. Anand, M. Zhang, and A. Sangi, “Scheduling Function One (SF1) for hop-by-hop Scheduling in 6tisch Networks.” [Online]. Available: https://tools.ietf.org/html/draft-satish-6tisch-6top-sf1-00.

[15] “Z1 | Zolertia.” [Online]. Available: http://zolertia.io/z1

[16] Contiki OS. Contiki Hardware. http://www.contiki-os.org/hardware.html.

[17] J. M. Gómez Cama, “Sistemas operativos empotrados,” Universitat Oberta de Catalunya, 2016.

Anexos:

Anexo A: Resultados del escenario 1

Secuencia de salto	Secuencia de unión	Nodo	Tiempo puro	time/nodo
1_1	1_1	2	662.140	193.684
		3	345.808	180.763
		4	104.246	104.246
		5	389.703	285.456
		6	656.101	187.646
		7	165.045	165.045
		8	956.175	336.234
		9	763.120	129.539
		10	803.790	414.087
		11	165.049	165.049
		12	1.047.940	421.933
		13	247.468	247.468
		14	633.581	165.126
		15	619.941	230.238
		16	1.037.104	411.097
		17	468.455	303.410
		18	1.679.424	545.942
		19	870.273	144.953
		20	725.320	560.272
		21	1.133.482	408.162
		22	626.007	378.539
	2_2	2	644.280	146.143
		3	320.732	149.792
		4	102.973	102.973
		5	360.990	258.017
		6	591.671	93.534
		7	170.940	170.940
		8	960.875	361.356
		9	771.390	148.649
		10	761.029	400.039
		11	170.950	170.950
		12	786.769	268.104
		13	162.770	162.770
		14	622.740	124.603
		15	599.519	238.529
		16	760.451	241.785
		17	498.137	327.198
		18	1.303.804	303.910
		19	750.959	182.095

	20	568.864	397.914
	21	999.894	431.031
	22	518.665	355.896
4_4	2	614.621	209.373
	3	320.829	151.002
	4	109.034	109.034
	5	378.986	269.952
	6	603.784	198.537
	7	169.827	169.827
	8	1.020.578	372.035
	9	827.462	178.919
	10	738.210	359.224
	11	169.854	169.854
	12	975.268	469.592
	13	167.558	167.558
	14	583.293	178.045
	15	648.543	269.557
	16	907.171	401.495
	17	405.248	235.421
	18	1.576.909	561.429
	19	815.460	129.653
	20	685.808	515.954
	21	1.015.480	329.672
	22	505.677	338.119
4_16	2	592.628	141.277
	3	318.077	136.690
	4	103.875	103.875
	5	352.355	248.480
	6	596.294	144.943
	7	181.388	181.388
	8	921.267	333.970
	9	763.930	176.633
	10	660.390	308.035
	11	195.746	195.746
	12	1.001.092	545.984
	13	160.100	160.100
	14	573.669	122.318
	15	587.297	234.942
	16	880.567	425.459
	17	451.351	269.963
	18	1.551.502	641.318
	19	754.264	216.574
	20	537.690	341.944

		21	910.184	372.494
		22	455.109	295.009
	16_16	2	626.820	178.333
		3	323.538	146.233
		4	111.736	111.736
		5	369.776	258.041
		6	646.022	197.535
		7	177.305	177.305
		8	883.818	270.798
		9	777.717	161.625
		10	723.216	353.439
		11	176.171	176.171
		12	1.003.024	556.712
		13	165.517	165.517
		14	616.092	167.605
		15	613.020	243.244
		16	942.079	495.767
		17	448.487	271.182
		18	1.454.436	514.931
		19	737.122	142.506
		20	594.616	418.445
		21	939.505	344.889
		22	446.312	280.795
2_2	1_1	2	665.621	222.627
		3	332.321	164.117
		4	103.830	103.830
		5	378.464	274.635
		6	651.283	208.290
		7	168.204	168.204
		8	1.058.244	418.462
		9	750.798	111.016
		10	745.020	366.556
		11	172.997	172.997
		12	805.510	364.704
		13	174.321	174.321
		14	608.387	165.394
		15	639.782	261.318
		16	812.572	371.767
		17	442.994	274.790
		18	1.582.444	618.660
		19	680.302	147.253
		20	533.049	360.052
		21	963.784	430.735

	22	440.805	266.484
2_2	2	621.389	169.823
	3	328.905	162.878
	4	105.221	105.221
	5	404.916	299.695
	6	617.830	166.264
	7	166.027	166.027
	8	921.310	346.996
	9	770.051	173.767
	10	811.957	407.040
	11	176.793	176.793
	12	804.513	284.802
	13	198.350	198.350
	14	596.284	144.718
	15	574.314	169.398
	16	908.118	388.406
	17	451.566	285.539
	18	1.394.280	383.799
	19	798.527	153.320
	20	645.207	468.414
	21	1.010.481	365.274
	22	519.712	321.361
4_4	2	633.510	168.732
	3	344.628	164.148
	4	100.595	100.595
	5	349.005	248.411
	6	635.871	171.092
	7	180.480	180.480
	8	985.939	407.833
	9	758.275	133.035
	10	667.904	318.898
	11	168.567	168.567
	12	983.501	456.437
	13	154.381	154.381
	14	625.240	160.461
	15	578.106	229.101
	16	934.490	407.426
	17	464.779	284.299
	18	1.443.304	489.902
	19	782.001	216.675
	20	565.326	396.759
	21	953.401	388.076
	22	527.064	372.683

	4_16	2	592.679	145.812
		3	345.918	173.861
		4	104.093	104.093
		5	444.370	340.277
		6	599.915	153.048
		7	172.058	172.058
		8	939.294	333.868
		9	807.051	192.975
		10	667.792	223.423
		11	172.091	172.091
		12	1.196.814	611.496
		13	166.232	166.232
		14	614.075	167.208
		15	605.426	161.056
		16	1.176.440	591.122
		17	446.867	274.809
		18	1.464.688	394.528
		19	812.878	204.740
		20	608.138	436.047
		21	1.070.160	462.022
		22	585.319	419.086
	16_16	2	642.100	167.652
		3	342.274	176.005
		4	110.237	110.237
		5	400.393	290.157
		6	631.527	157.079
		7	166.268	166.268
		8	932.353	330.382
		9	746.464	127.009
		10	720.458	320.064
		11	173.472	173.472
		12	1.104.026	510.145
		13	229.801	229.801
		14	619.456	145.008
		15	601.971	201.577
		16	1.034.764	440.883
		17	474.448	308.179
		18	1.444.593	345.627
		19	885.793	207.010
		20	678.783	505.310
		21	1.098.967	420.184
		22	593.881	364.081
4_4	1_1	2	667.325	245.144

	3	360.363	184.393
	4	98.467	98.467
	5	343.280	244.812
	6	669.573	247.392
	7	175.970	175.970
	8	971.925	300.554
	9	861.080	189.709
	10	709.826	366.547
	11	165.225	165.225
	12	992.124	531.625
	13	151.072	151.072
	14	646.979	224.798
	15	671.371	311.008
	16	852.503	392.004
	17	422.181	246.211
	18	1.701.058	687.790
	19	729.762	187.972
	20	541.790	376.565
	21	1.013.268	471.478
	22	460.499	309.427
2_2	2	629858,8	182.859
	3	330246	153.338
	4	104172,6	104.173
	5	389681,4	285.509
	6	610835,2	163.835
	7	176908,4	176.908
	8	965574,4	366.124
	9	745178,2	128.537
	10	701189,4	311.508
	11	174532,2	174.532
	12	1126558,6	623.383
	13	190176,6	190.177
	14	616641	169.641
	15	599450,8	209.769
	16	1047770	544.594
	17	447000,2	270.092
	18	1754447,4	795.104
	19	759326,4	159.318
	20	600008,4	425.476
	21	959343,6	359.335
	22	503175,6	312.999
4_4	2	616956,6	191.418
	3	361416,2	192.758

	4	101892,4	101.892
	5	346695,4	244.803
	6	603640,8	178.102
	7	168658	168.658
	8	962159,6	358.996
	9	859741,8	234.491
	10	788356,4	441.661
	11	171043,8	171.044
	12	847725	312.319
	13	158101,4	158.101
	14	625251	199.712
	15	603163,4	241.747
	16	718818,8	183.413
	17	425538,8	256.881
	18	1576920,6	585.256
	19	785670	189.286
	20	596383,8	425.340
	21	991664,6	395.281
	22	535406	377.305
4_16	2	627446	158.122
	3	349072,4	165.270
	4	111066,6	111.067
	5	398871,2	287.805
	6	620151,6	150.827
	7	183802	183.802
	8	1082227,6	453.171
	9	765168,8	136.112
	10	705671	306.800
	11	177847,4	177.847
	12	737640	73.743
	13	188738,8	188.739
	14	610803,8	141.479
	15	629056,6	230.185
	16	773302	109.405
	17	469324,4	285.522
	18	1696977,6	635.361
	19	899753,6	169.949
	20	729804,8	551.957
	21	1061616,6	331.812
	22	663897	475.158
16_16	2	664106,8	200.939
	3	361998,2	193.972
	4	103648,4	103.648

		5	396218,8	292.570
		6	617464,4	154.297
		7	168026,2	168.026
		8	940116	328.113
		9	820951,4	195.153
		10	854489,2	458.270
		11	182373,8	182.374
		12	818471,2	300.275
		13	136013,4	136.013
		14	625798,2	162.630
		15	612003,2	215.784
		16	765876,8	247.681
		17	463167,8	295.142
		18	1542918	563.794
		19	712312,8	135.446
		20	576867,2	394.493
		21	979123,8	402.257
		22	518195,8	382.182
4_16	1_1	2	685257	241.469
		3	331826,2	149.738
		4	109341,4	109.341
		5	411419,8	302.078
		6	696027,6	252.239
		7	182087,8	182.088
		8	987446,2	326.680
		9	811174,4	143.642
		10	687316,4	275.897
		11	182090,2	182.090
		12	1152019	618.626
		13	233439,8	233.440
		14	667532,4	223.744
		15	660766	249.346
		16	1077966,6	544.574
		17	443788,2	261.700
		18	1622525,6	565.007
		19	857521,8	154.537
		20	702985,2	520.895
		21	1057518,4	354.533
		22	533393	299.953
	2_2	2	622566,4	187.737
		3	331318	156.954
		4	108782,6	108.783
		5	394127,2	285.345

	6	669310	234.480
	7	174364,2	174.364
	8	994047,6	351.780
	9	804315,4	162.048
	10	834646,8	440.520
	11	177966,6	177.967
	12	824758,2	350.395
	13	185227,2	185.227
	14	618989	184.159
	15	642267,2	248.140
	16	762663,8	288.301
	17	434829,6	260.465
	18	1565933,4	555.468
	19	806845,8	237.989
	20	568856,4	390.890
	21	1010465,8	441.609
	22	474363,2	289.136
4_4	2	687069,2	260.633
	3	332466,4	156.975
	4	106344,8	106.345
	5	376229,8	269.885
	6	671494,4	245.059
	7	175491,6	175.492
	8	999940,8	401.899
	9	845074,6	216.668
	10	671115,8	294.886
	11	171932,8	171.933
	12	1122799,4	521.007
	13	169686,8	169.687
	14	628406,2	201.970
	15	598041,6	221.812
	16	1082111,6	480.319
	17	426435,8	250.944
	18	1725623,6	532.829
	19	868824,6	81.649
	20	787175,6	615.243
	21	1192794,4	405.619
	22	601792,6	432.106
4_16	2	741499,6	203.180
	3	362030	190.323
	4	110903,4	110.903
	5	453647,2	342.744
	6	727286,8	188.967

		7	171706,8	171.707
		8	983126	328.106
		9	905700,2	180.994
		10	743743	290.096
		11	193214	193.214
		12	917506	432.750
		13	137234	137.234
		14	724706,6	186.387
		15	655020,2	201.373
		16	982946,2	498.190
		17	538319,6	366.613
		18	1603812,4	693.756
		19	715960,4	142.529
		20	573431,6	380.218
		21	910056,2	336.625
		22	484756,2	347.522
	16_16	2	703029,8	259.784
		3	339736	162.859
		4	107719,6	107.720
		5	403875,6	296.156
		6	702800,2	259.554
		7	176877,4	176.877
		8	1006123,8	376.794
		9	809520,8	163.787
		10	756095,8	352.220
		11	163749,8	163.750
		12	892872,6	380.249
		13	182970,2	182.970
		14	645734,2	202.488
		15	629329,8	225.454
		16	837902,2	325.279
		17	443246,2	266.369
		18	1521956,4	562.626
		19	784352,6	190.406
		20	593947	430.197
		21	959330,8	365.384
		22	512623,4	329.653
16_16	1_1	2	567.753	145.783
		3	324.400	158.139
		4	100.688	100.688
		5	381.319	280.632
		6	574.974	153.003
		7	166261,4	166.261

	8	916819,2	351.791
	9	727390	138.048
	10	695409,8	314.091
	11	180599,2	180.599
	12	921580,2	393.254
	13	147331,6	147.332
	14	589341,8	167.371
	15	565028	183.709
	16	941745,2	413.419
	17	421970,8	255.709
	18	1208532,6	179.947
	19	736760,4	201.087
	20	535673,8	355.075
	21	1028585,2	492.911
	22	528326	380.994
2_2	2	689.340	242.783
	3	343.012	173.638
	4	108.571	108.571
	5	380.781	272.210
	6	652.545	205.988
	7	169374,8	169.375
	8	1066525	420.826
	9	797256,8	142.344
	10	731733,4	350.952
	11	171758,8	171.759
	12	1080850	445.811
	13	159957	159.957
	14	654912,6	208.355
	15	645698,6	264.918
	16	1083151	448.112
	17	446557,4	277.183
	18	1436972,4	337.316
	19	891307,2	118.780
	20	772527,2	600.768
	21	1099656,8	327.130
	22	635038,8	475.082
4_4	2	588.874	148.494
	3	310.743	133.143
	4	107.248	107.248
	5	391.399	284.150
	6	548139,4	107.760
	7	177600,4	177.600
	8	974620,2	362.488

	9	742266,8	130.134
	10	676787,8	285.389
	11	185984,8	185.985
	12	805386,4	358.851
	13	141965,2	141.965
	14	580369,2	139.990
	15	612132,6	220.734
	16	799277,4	352.742
	17	440379,6	262.779
	18	1459486	535.208
	19	708767,2	181.969
	20	526798,2	340.813
	21	924278,4	397.480
	22	446535,4	304.570
4_16	2	634.708	155.535
	3	356.624	175.989
	4	106.678	106.678
	5	415.947	309.269
	6	631349,8	152.176
	7	180635,6	180.636
	8	1043193,4	389.810
	9	814448,8	161.065
	10	726386,4	310.439
	11	180611,2	180.611
	12	982460,2	398.134
	13	179532,8	179.533
	14	621526,4	142.353
	15	653383,4	237.436
	16	991854	407.528
	17	479173,6	298.538
	18	1312282,2	269.346
	19	832194,2	142.448
	20	689746,2	509.135
	21	1042936,6	353.190
	22	584326,2	404.793
16_16	2	601.792	155.804
	3	337.676	155.766
	4	109.167	109.167
	5	389.801	280.634
	6	607799,4	161.811
	7	181910,4	181.910
	8	1064740,2	451.916
	9	769244,2	156.420

10	712184,6	322.383
11	173584,8	173.585
12	882409,2	342.091
13	130718	130.718
14	556262,6	110.274
15	612824,6	223.023
16	945461,4	405.143
17	445988,2	264.078
18	1386307,4	429.112
19	808461,8	167.685
20	640777,2	467.192
21	957195,6	316.418
22	540318,6	409.601

Anexo B: Resultados del escenario 2

Long. SF	Carga	Demora	Energía	PDR
1	7.5	289.252.880	3,475	0,908
1	7.5	330.886.873	3,434	0,899
1	7.5	307.983.264	2,717	0,937
1	7.5	264.029.960	3,066	0,895
1	7.5	309.723.718	3,247	0,891
1	7.5	346.173.790	3,231	0,918
1	7.5	344.413.300	3,034	0,911
1	7.5	296.902.715	2,609	0,895
1	7.5	349.241.333	2,817	0,926
1	7.5	295.350.024	2,826	0,94
1	7.5	288.372.131	2,624	0,902
1	7.5	361.605.231	2,828	0,933
1	7.5	343.819.852	3,001	0,906
1	7.5	291.967.503	3,067	0,899
1	7.5	347.642.387	2,990	0,947
1	7.5	133.529.187	2,944	0,634
1	7.5	337.995.944	3,313	0,916
1	7.5	265.523.658	4,720	0,915
1	7.5	275.506.132	2,727	0,939
1	7.5	284.664.765	3,712	0,899
1	7.5	330.795.408	2,623	0,915
1	7.5	322.422.474	4,385	0,917
1	7.5	324.842.621	2,962	0,93
1	7.5	330.581.125	2,903	0,925
1	7.5	355.513.041	2,920	0,92
1	60	356.438.517	2,799	0,996
1	60	281.697.182	3,141	0,951
1	60	330.581.125	2,903	0,978
1	60	245.229.340	3,021	0,973
1	60	321.906.516	2,964	0,966
1	60	276.119.405	2,948	0,967
1	60	297.335.260	2,907	0,951
1	60	359.041.008	3,045	0,974
1	60	307.715.002	2,905	0,973
1	60	266.721.124	2,611	0,948
1	60	366.013.247	2,594	0,977
1	60	259.941.159	2,854	0,944
1	60	231.222.961	2,455	0,949

1	60	299.938.964	3,028	0,949
1	60	328.524.470	2,797	0,978
1	60	274.579.261	3,770	0,985
1	60	275.031.269	2,551	0,955
1	60	336.007.379	2,828	0,985
1	60	334.108.703	2,912	0,973
1	60	286.172.533	3,276	0,984
1	60	277.041.051	2,858	0,956
1	60	313.033.938	3,397	0,955
1	60	342.700.885	2,550	0,981
1	60	326.760.600	3,300	0,968
1	60	312.723.550	3,539	0,97
1	480	245.345.461	2,890	0,935
1	480	340.652.403	2,990	0,963
1	480	262.967.651	3,023	0,938
1	480	299.480.304	2,654	0,932
1	480	347.972.455	2,827	0,939
1	480	272.603.884	2,796	0,933
1	480	258.893.931	3,033	0,937
1	480	293.521.218	2,952	0,907
1	480	376.142.548	2,809	0,957
1	480	300.723.522	2,724	0,931
1	480	299.245.294	2,858	0,976
1	480	260.585.204	2,634	0,921
1	480	264.119.254	2,675	0,943
1	480	302.048.998	3,097	0,963
1	480	355.459.599	2,751	0,949
1	480	231.914.495	3,067	0,934
1	480	253.802.341	2,969	0,941
1	480	279.591.812	2,969	0,9
1	480	280.042.055	2,552	0,934
1	480	266.089.635	3,131	0,961
1	480	287.607.540	2,750	0,934
1	480	319.802.029	2,796	0,964
1	480	295.309.444	2,747	0,93
1	480	347.933.935	2,759	0,944
1	480	289.597.112	3,143	0,925
2	7.5	291.421.556	3,064	0,585
2	7.5	300.127.882	2,952	0,575
2	7.5	259.378.024	2,537	0,642
2	7.5	296.370.284	2,635	0,627

2	7.5	257.849.650	3,004	0,644
2	7.5	379.921.844	2,967	0,593
2	7.5	303.889.539	2,659	0,587
2	7.5	278.296.686	3,055	0,629
2	7.5	329.260.892	2,636	0,604
2	7.5	255.727.935	3,694	0,631
2	7.5	420.033.769	2,734	0,593
2	7.5	341.270.969	3,445	0,622
2	7.5	415.682.780	2,694	0,587
2	7.5	311.702.371	3,513	0,616
2	7.5	298.444.989	3,073	0,598
2	7.5	304.374.251	2,789	0,6
2	7.5	251.726.761	2,688	0,654
2	7.5	297.616.923	3,440	0,645
2	7.5	336.503.318	3,723	0,603
2	7.5	308.604.066	2,967	0,612
2	7.5	302.584.011	2,649	0,591
2	7.5	291.905.319	2,673	0,629
2	7.5	243.174.956	3,032	0,604
2	7.5	337.156.430	2,966	0,614
2	7.5	271.395.304	2,986	0,638
2	60	301.052.929	2,564	0,996
2	60	315.251.670	2,764	0,992
2	60	348.493.107	2,708	0,986
2	60	338.786.108	3,491	0,985
2	60	345.372.452	3,569	0,99
2	60	277.787.378	3,022	0,972
2	60	310.899.589	2,556	0,988
2	60	290.811.010	3,131	0,99
2	60	293.562.321	3,191	0,979
2	60	292.612.767	2,937	0,991
2	60	272.590.635	2,664	0,976
2	60	211.059.212	3,723	0,966
2	60	238.309.672	3,014	0,982
2	60	324.170.263	2,980	0,995
2	60	314.135.739	3,083	0,989

2	60	295.355.195	2,834	0,997
2	60	348.027.453	2,620	0,998
2	60	266.517.096	2,709	0,995
2	60	246.287.293	2,576	0,982
2	60	308.742.154	2,508	0,982
2	60	304.536.464	2,805	0,993
2	60	269.155.221	2,728	0,997
2	60	336.589.293	2,559	0,99
2	60	284.664.765	3,712	0,985
2	60	238.397.395	3,473	0,991
2	480	276.503.054	2,985	0,926
2	480	337.536.349	2,598	0,965
2	480	319.374.036	2,710	0,964
2	480	332.649.581	2,842	0,94
2	480	336.432.275	2,686	0,966
2	480	362.471.407	3,087	0,964
2	480	244.894.865	2,818	0,926
2	480	297.519.649	3,494	0,944
2	480	328.917.081	2,692	0,944
2	480	305.813.456	3,140	0,974
2	480	321.630.964	3,006	0,975
2	480	342.170.763	2,787	0,942
2	480	346.298.897	2,767	0,959
2	480	434.775.768	2,700	0,968
2	480	241.542.796	2,601	0,927
2	480	292.005.523	3,285	0,932
2	480	340.251.353	3,289	0,944
2	480	346.197.431	2,745	0,94
2	480	349.191.301	3,179	0,937
2	480	344.474.985	2,880	0,966
2	480	286.991.886	2,911	0,94
2	480	313.590.511	3,610	0,969
2	480	275.468.943	3,222	0,927
2	480	314.033.799	3,141	0,943
2	480	292.602.190	2,986	0,959
3	7.5	303.074.957	2,762	0,581

3	7.5	268.820.723	3,006	0,603
3	7.5	282.914.186	2,670	0,579
3	7.5	280.183.418	2,747	0,589
3	7.5	311.564.938	2,834	0,572
3	7.5	289.946.462	3,016	0,579
3	7.5	261.205.831	3,012	0,567
3	7.5	318.297.061	3,831	0,574
3	7.5	315.860.911	3,052	0,58
3	7.5	325.548.802	2,806	0,601
3	7.5	363.246.074	2,572	0,584
3	7.5	319.444.515	2,714	0,568
3	7.5	286.750.303	3,020	0,559
3	7.5	332.887.705	2,626	0,59
3	7.5	343.721.760	3,275	0,586
3	7.5	352.833.141	3,113	0,591
3	7.5	218.239.185	2,780	0,571
3	7.5	279.007.166	2,949	0,599
3	7.5	387.365.534	2,877	0,596
3	7.5	251.566.125	2,922	0,556
3	7.5	308.552.143	2,802	0,578
3	7.5	291.110.138	2,639	0,577
3	7.5	279.359.542	2,653	0,567
3	7.5	226.794.511	2,592	0,562
3	7.5	233.514.106	2,847	0,583
3	60	258.607.793	2,548	0,991
3	60	286.555.615	3,054	0,993
3	60	283.702.999	3,084	0,966
3	60	363.745.278	2,825	0,983
3	60	328.587.351	2,513	0,985
3	60	293.341.421	3,117	0,957
3	60	363.722.708	2,753	0,984
3	60	342.188.111	2,868	0,977
3	60	334.813.309	2,732	0,977
3	60	279.028.770	2,983	0,896
3	60	203.540.430	2,846	0,964
3	60	372.782.939	3,412	0,992

3	60	315.383.697	3,308	0,983
3	60	347.722.264	3,069	0,996
3	60	199.591.839	2,909	0,992
3	60	335.582.319	2,616	0,981
3	60	331.469.141	2,822	0,956
3	60	308.670.571	2,798	0,967
3	60	296.204.305	3,544	0,981
3	60	258.363.630	3,079	0,999
3	60	350.656.440	2,855	0,985
3	60	308.955.411	2,966	0,971
3	60	319.668.627	3,246	0,996
3	60	357.630.948	3,008	0,998
3	60	276.557.613	2,727	0,973
3	480	329.760.424	4,174	0,989
3	480	251.778.718	3,200	0,952
3	480	307.602.871	2,970	0,954
3	480	221.349.774	3,089	0,908
3	480	301.840.602	3,128	0,963
3	480	292.551.158	3,041	0,96
3	480	298.904.523	2,674	0,949
3	480	247.654.312	2,713	0,933
3	480	272.843.544	2,496	0,966
3	480	319.033.816	4,283	0,954
3	480	267.094.055	4,107	0,968
3	480	383.689.361	2,586	0,988
3	480	330.803.679	2,694	0,967
3	480	275.031.269	2,551	0,955
3	480	234.423.839	3,302	0,948
3	480	280.456.891	2,926	0,955
3	480	295.202.158	2,575	0,99
3	480	252.874.202	2,794	0,988
3	480	326.513.161	4,167	0,965
3	480	273.030.558	3,942	0,981
3	480	348.692.825	3,143	0,986
3	480	238.330.602	2,540	0,988
3	480	285.389.184	3,406	0,997

3	480	305.712.359	2,967	0,957
3	480	226.540.838	3,377	0,986

Anexo C: Resultados del escenario 3

Long. SF	Carga	Demora	Energía	PDR
NOSTORING	7.5	342.298.449	2,890	0,6
NOSTORING	7.5	319.747.994	2,949	0,608
NOSTORING	7.5	313.909.165	2,492	0,62
NOSTORING	7.5	262.111.481	2,589	0,599
NOSTORING	7.5	306.152.783	2,686	0,62
NOSTORING	7.5	267.223.951	3,185	0,63
NOSTORING	7.5	311.770.302	2,844	0,571
NOSTORING	7.5	347.553.991	3,483	0,628
NOSTORING	7.5	288.002.530	2,645	0,594
NOSTORING	7.5	339.450.658	2,913	0,594
NOSTORING	7.5	305.897.026	2,995	0,621
NOSTORING	7.5	367.660.872	2,797	0,603
NOSTORING	7.5	245.440.914	2,812	0,633
NOSTORING	7.5	302.298.010	2,851	0,614
NOSTORING	7.5	250.953.280	2,866	0,628
NOSTORING	7.5	432.842.791	2,893	0,577
NOSTORING	7.5	269.454.061	2,791	0,624
NOSTORING	7.5	308.969.948	3,090	0,615
NOSTORING	7.5	269.890.626	2,796	0,645
NOSTORING	7.5	259.624.337	2,830	0,652
NOSTORING	7.5	307.438.368	2,922	0,619
NOSTORING	7.5	297.447.366	3,147	0,592
NOSTORING	7.5	310.778.781	2,908	0,636
NOSTORING	7.5	288.706.733	2,655	0,628
NOSTORING	7.5	223.817.209	3,988	0,626
NOSTORING	60	302.168.519	2,698	0,988
NOSTORING	60	285.934.231	3,234	0,98
NOSTORING	60	283.780.092	2,810	0,996
NOSTORING	60	197.095.932	3,272	0,86
NOSTORING	60	282.845.530	2,757	0,986
NOSTORING	60	349.189.491	2,771	0,991
NOSTORING	60	287.829.886	2,873	0,983
NOSTORING	60	228.547.093	3,106	0,983
NOSTORING	60	262.217.555	2,928	0,972
NOSTORING	60	334.213.651	2,814	0,996
NOSTORING	60	275.521.335	2,711	0,985
NOSTORING	60	312.113.048	3,005	0,99
NOSTORING	60	322.203.808	2,576	0,972
NOSTORING	60	363.712.779	2,765	0,995

NOSTORING	60	213.380.373	2,960	0,852
NOSTORING	60	367.340.245	2,815	0,996
NOSTORING	60	322.281.831	3,161	0,993
NOSTORING	60	199.591.839	2,909	0,997
NOSTORING	60	224.234.008	3,443	0,979
NOSTORING	60	306.497.186	3,518	0,992
NOSTORING	60	340.883.459	2,809	0,982
NOSTORING	60	239.689.666	2,823	0,99
NOSTORING	60	329.240.381	4,103	0,997
NOSTORING	60	331.457.590	3,009	0,99
NOSTORING	60	325.914.084	2,666	0,987
NOSTORING	480	361.018.841	3,076	0,967
NOSTORING	480	312.900.647	2,537	0,925
NOSTORING	480	295.171.962	2,889	0,956
NOSTORING	480	357.393.004	2,692	0,955
NOSTORING	480	240.073.461	2,670	0,920
NOSTORING	480	345.779.014	3,338	0,974
NOSTORING	480	416.217.315	3,914	0,980
NOSTORING	480	269.442.883	2,894	0,939
NOSTORING	480	347.936.689	2,665	0,966
NOSTORING	480	341.692.054	2,820	0,968
NOSTORING	480	200.746.594	4,803	0,960
NOSTORING	480	276.755.719	3,274	0,929
NOSTORING	480	327.692.983	3,050	0,925
NOSTORING	480	370.011.563	2,839	0,892
NOSTORING	480	266.493.706	3,100	0,934
NOSTORING	480	253.819.419	2,813	0,922
NOSTORING	480	309.391.650	2,890	0,949
NOSTORING	480	314.282.497	2,662	0,976
NOSTORING	480	313.138.704	2,686	0,936
NOSTORING	480	311.944.412	3,156	0,946
NOSTORING	480	241.641.332	3,029	0,906
NOSTORING	480	332.622.749	2,740	0,952
NOSTORING	480	284.557.697	3,009	0,938
NOSTORING	480	305.882.775	2,606	0,948
NOSTORING	480	301.249.383	3,048	0,951