# Visualización con LEDs del Funcionamiento de una Red TSCH/RPL

**Autores:** Diego Fraga, Giuliano Turpía, Ignacio Valettute

**Emails:** diego.fraga@fing.edu.uy, giulianoturpia@fing.edu.uy, ignacio.valettute@fing.edu.uy

**Tutor**: Leonardo Steinfeld

**Email:** leo@fing.edu.uy

Instituto de Ingeniería Eléctrica - Facultad de Ingeniería - UDELAR

## Resumen

Se realizaron una serie de programas para cargar en placas de tipo *Remote* con el objetivo de observar el funcionamiento de una red TSCH/RPL interactivamente. Este tipo de redes funcionan con tiempos muy cortos durante sus varias etapas la vida, con lo cual se dificulta observar la operación en tiempo real. Enlenteciendo la red sin destruir su funcionalidad, es posible observar mediante un conjunto de LEDs en dichas placas la formación, modificación y el entablado de comunicación luego de establecida la red. Los distintos programas compilados para el proyecto permiten observar las características deseadas de la red, permitiendo mostrar redes de sensores inalámbricos de manera didáctica en instancias educativas.

# Índice

# 1. Introducción

La idea principal del proyecto es visualizar de manera interactiva, mediante LEDs, una red de sensores inalámbricos que utiliza TSCH/RPL. Se busca observar el funcionamiento de la red en las distintas etapas de vida de la misma (creación de la red, intercambio de mensajes, direccionamiento, entre otros). Los resultados podrían ser usados en una instancia de IdM (Ingeniería de Muestra), en una instancia en el IIE, o mismo en futuras iteraciones del curso de RSI, buscando mostrar redes de alrededor de 10 nodos.

## 1.1. Conceptos clave

### 1.1.1. TSCH: Time-Slotted Channel Hopping

TSCH es un protocolo de comunicaciones comúnmente implementado en redes de sensores inalámbricos, a nivel de capa de acceso al medio (MAC). Como su nombre lo indica, se basa en dos principios fundamentales: el uso de *timeslots (TS)* y saltos de canal (o *channel hopping (CH)*). La motivación principal tras este protocolo recae en la necesidad de sincronizar las acciones de comunicación entre dispositivos de una misma red de manera de evitar interferencias y colisiones, así como mitigar el impacto negativo que pueda presentar un canal débil. A continuación se presentan algunas definiciones clave para la implementación del protocolo.

**Time Division Multiple Access (TDMA).** TDMA es uno de los principios clave en los que se basa el protocolo TSCH, y consiste en la división del tiempo en ***timeslots***. Un *timeslot* es un período de tiempo de un largo predeterminado[1], en la que un dispositivo puede ejecutar una acción de comunicación, ya sea enviar o recibir datos, o "dormir" (apaga la radio, ahorrando energía).

**Channel Hopping.** En lugar de permanecer en un solo canal para la comunicación, TSCH utiliza el salto de canal. Los dispositivos cambian entre un conjunto predefinido de canales dentro de cada *timeslot*. Este mecanismo de salto ayuda a superar la interferencia y evita el impacto de las perturbaciones continuas en un canal específico.

**Slotframes.** Un *slotframe* es una secuencia o un conjunto de *timeslots* que se repiten periódicamente. Define la estructura del cronograma utilizado por los dispositivos en la red.

**SlotOffset.** *SlotOffset* hace referencia a la posición de un *timeslot* dentro de un *slotframe*. Define cuándo ocurre un *timeslot* específico dentro de la estructura repetitiva del *slotframe*. Los dispositivos sincronizan su operación al conocer su posición (su *SlotOffset*) dentro del *slotframe* para participar en la comunicación en los momentos correctos.

**ChannelOffset.** El *ChannelOffset* determina qué canal utiliza un dispositivo dentro de un *timeslot* dado. Este parámetro ayuda en la sincronización del salto de canal entre los dispositivos que se comunican.

**Absolute Slot Number (ASN).** ASN representa la posición absoluta de un intervalo de tiempo dentro de todo el cronograma de la red. Es una referencia global de tiempo utilizada para sincronizar diferentes dispositivos dentro de la red. ASN aumenta continuamente a medida que avanza el tiempo y proporciona una forma para que los dispositivos sincronicen sus cronogramas, especialmente al unirse por primera vez o volver a unirse a la red después de estar inactivos.

[1] Un largo comúnmente utilizado es el de 15 ms, suficiente para enviar un marco MAC y recibir un ACK.

**TSCH Cell.** Una celda TSCH está dada por un *SlotOffset* y un *ChannelOffset*. Un dispositivo de la red puede realizar una de tres acciones en cada celda: transmitir, recibir o dormir.

**Enhanced Beacons (EB).** Los *Enhanced Beacons* son paquetes especiales de datos transmitidos periódicamente. Contienen información crucial para la sincronización y configuración de la red, como ser la estructura de los *timeslots* y la disposición de canales, permitiendo a los dispositivos ajustar sus relojes internos y coordinar sus actividades. Los EB se envían en intervalos específicos designados como slots de beacon, asegurando que los dispositivos reciban actualizaciones de sincronización y configuración sin interferir con los slots regulares de comunicación.

### 1.1.2. RPL: Routing Protocol for Low-Power and Lossy Networks

RPL es un protocolo de enrutamiento diseñado para redes de sensores inalámbricos de bajo consumo (*low-power*) y alta pérdida de datos (*lossy*). Su principio básico de funcionamiento radica en establecer rutas eficientes y adaptables entre nodos, priorizando el ahorro de energía y la superación de condiciones de comunicación poco confiables. Utiliza métricas como el costo de ruta y la calidad del enlace para dirigir el tráfico de manera óptima en entornos de red con recursos limitados.

**Función objetivo (OF).** RPL utiliza métricas de enrutamiento, como el costo de ruta (generalmente basado en el número de saltos) y la calidad del enlace. Además, tiene en cuenta restricciones como la capacidad energética de los nodos y la fiabilidad de las conexiones para determinar las rutas más óptimas. La función objetivo es una combinación de estas métricas y limitaciones, que RPL utiliza para dirigir el tráfico de la red.

**Destination-Oriented Directed Acyclic Graph (DODAG).** El término DODAG se refiere a una estructura de grafo dirigido sin ciclos que representa la topología de la red de forma jerárquica y orientada al destino. En este grafo, un nodo raíz es el destino final hacia el cual se dirigen los datos, mientras que los demás nodos se organizan en niveles descendentes, estableciendo relaciones padre-hijo.

**Nodos padres y vecinos.** En el contexto de RPL, cada nodo puede tener padres y vecinos. Los padres son nodos hacia los cuales un nodo envía sus datos, mientras que los vecinos son nodos directamente alcanzables. La selección eficiente de padres y vecinos está basada en la función objetivo, que establece reglas para la misma.

**DODAG Information Solicitation (DIS).** El mensaje DIS se utiliza para solicitar información sobre la topología de la red. Un nodo que necesita unirse a la red o actualizar su información de enrutamiento envía un DIS. Los nodos que reciben este mensaje y tienen información del DODAG pueden responder con un DIO.

**DODAG Information Object (DIO).** Los mensajes DIO son enviados por los nodos que actúan como anclas o raíces del DODAG. Estos mensajes contienen información sobre la topología de la red, como la identificación del DODAG, la configuración del enrutamiento, métricas, direcciones IP y otros parámetros necesarios para construir y mantener el grafo de enrutamiento. Los nodos que reciben un DIO actualizan su información de enrutamiento según los detalles proporcionados en este mensaje.

**Destination Advertisement Object (DAO).** Los mensajes DAO se utilizan para anunciar la disponibilidad de nodos como destinos finales para el enrutamiento. Un nodo que desea ser alcanzable o anunciarse como destino envía un mensaje DAO, indicando su presencia y capacidad para recibir datos. Estos mensajes ayudan a construir y mantener las rutas desde otros nodos hacia el nodo anunciado.

## 1.2. Descripción del problema

En una red RPL que utiliza TSCH existen ciertas etapas durante el tiempo de vida, como el descubrimiento de la DODAG, el intercambio de mensajes controlado pasando un nodo raíz, la transmisión de mensajes dada por timeslots (TSCH), entre otros. Estos eventos si bien pueden perdurar en el tiempo (como descubrir la red en su totalidad) , descubrir la red o la transmisión/recepción de mensajes son eventos con tiempos asociados muy cortos, por ejemplo una red TSCH con *Orchestra* [1], o el recorrido de un mensaje UDP por el DODAG y como se anuncian los distintos mensajes en una red (EB, DIO, DAO).

Se busca entonces, contar con un sistema que permita visualizar la operación de una red de este estilo mediante la utilización de LEDs, a modo de facilitar el entendimiento de los conceptos asociados a las Redes de Sensores Inalámbricos.

## 1.3. Antecedentes

A continuación se mostrarán diferentes proyectos que serán consultados como guía para llevar a cabo el desarrollo del proyecto, así como la documentación de *Contiki-NG* tanto en formato web [2] como un paper oficial [3].

### 1.3.1. Ejemplos nativos de Contiki-NG

**Ejemplos base para TSCH-RPL**

1. **simple-node**: Aporta conceptos básicos sobre la configuración e implementación de los nodos configurados para funcionar en una red TSCH. A su vez implementa un *Makefile* más completo que puede ser útil para el proyecto.

2. **custom-schedule**: Este ejemplo es útil para entender la configuración de redes TSCH custom, lo cual da pie a configurar tanto los slots disponibles como el tiempo de slot para lograr la ralentización de las comunicaciones y así poder ser visualizadas con LEDs.

**Ejemplos generales**

1. **Hello-world**: Este ejemplo básico es el más utilizado por los tutoriales de Contiki-NG para explicar los conceptos, funcionamiento y configuraciones del sistema operativo. Por lo tanto, es de interés estudiar a fondo utilizar el código de este proyecto como fundamento para desarrollar el proyecto.

### 1.3.2. Otros proyectos

Por otro lado, se revisarán las implementaciones realizadas para los Laboratorios del curso de RSI durante el transcuro del semestre, en particular, la implementación de una red RPL y capa MAC *IEE 802.15.4* donde se implementa TSCH en sus versiones *Minimal* y *Orchestra*.

#### 1.3.3. Documentación de Contiki-NG

A lo largo del desarrollo del proyecto, se utilizará la documentación de *Contiki-NG*, así como sus tutoriales nativos [2].

# 2. Objetivos

Dada la descripción del problema, se parte el problema en objetivos generales y específicos, de tal modo que permita desglosar el trabajo en varias etapas tanto claras como bien definidas.

## 2.1. Objetivos Generales

El objetivo general del proyecto es generar una aproximación didáctica-demostrativa a las redes de comunicación inalámbricas, particularmente a las redes de sensores inalámbricos. Con esto en mente, se plantean los siguientes objetivos:

1. Mostrar el funcionamiento por detrás de la construcción de una red TSCH-RPL mediante LEDs.
2. Lograr el funcionamiento de una red TSCH-RPL en tiempo real o "enlentecida".
3. Realizar un set de demos o scripts tal de obtener modos de funcionamiento distintos de la red TSCH-RPL.

## 2.2. Objetivos Específicos

Dados los objetivos generales, los objetivos específicos se centran principalmente en completar los generales en trabajo más acotado para ejecutar los objetivos generales de manera más eficiente. Dicho esto, se obtienen los siguientes objetivos:

1. Lograr modificar a elección los tiempos de slot de una red TSCH-RPL de Contiki-NG.
2. Manipular los LEDs integrados en los Launchpad (CC1350 o CC2650) del curso para indicar radio ON/OFF y diferenciar entre TX/RX junto a los canales que se utilizan.
3. Desarrollar un frame de proyecto que envíe periódicamente mensajes para visualizar la construcción y funcionamiento de la red, y generar a partir del mismo versiones en las cuales la red TSCH tenga diferentes tiempos de slot, así como poder mostrar la radio on/off los tiempos TX/RX.
4. Lograr una correlación y progresividad entre los proyectos construidos para así facilitar el enfoque didáctico del proyecto.
5. Con los frames de proyecto, determinar un set de demos que permita combinar varias de estas opciones con el fin de observar la operación de la red.

# 3. Alcance

El alcance del proyecto radica al desarrollo del firmware necesario para visualizar el funcionamiento de una red TSCH-RPL. El firmware desarrollado tendrá la posibilidad de configurar su tiempo de slot frame, y de utilizar los LEDs para visualizar los tiempos ON/OFF de la radio, así como visualizar el momento TX/RX o los canales en los cuales se están comunicando.

Lo anterior se va a implementar para un nodo *servidor* y nodos *cliente*. Previo al cargar las placas se podrá seleccionar si en los nodos se muestra el estado ON/OFF de la radio, y en particular o el estado TX/RX o los canales por los cuales se encuentra la radio. Se abarca a su vez el desarrollo de una manera de mostrar una red "lenta" con tal de facilitar la visualización del funcionamiento de la red. Abarca desarrollar una manera de seleccionar la comunicación entre dos nodos de la red.

Las plataformas de interés para el proyecto son *cooja* (para la simulación), los launchpad *CC1350* y *CC2650* (los del curso) y los *remote* de tipo *Zoul* (serían utilizados para la red de muestra). Se mencionan pues Contiki-NG posee muchas plataformas, y desarrollar código particular para cada una de ellas puede irse fuera del alcance factible del proyecto.

Por otro lado, no se abordará la preparación de la actividad didáctica ni el desarrollo de la misma.

# 4. Desarrollo

Considerando los objetivos particulares de la Sección 2, las siguientes secciones presentan el proceso de desarrollo e investigación para poder cumplir con los objetivos.

## 4.1. TSCH Timings en Contiki-NG

Referido al segundo objetivo general, se debe de poder obtener una red TSCH-RPL modificada pues los tiempos asociados a los *timeslots*, que son las unidades básicas de comunicación de TSCH, poseen cierta estructura y tiempos minimos que deben de cumplirse con tal de asegurar que la capa MAC pueda comunicarse por el canal compartido. De manera ilustrativa y sin entrar en detalle, en la Figura 1 se muestra la estructura definida para una comunicación válida.

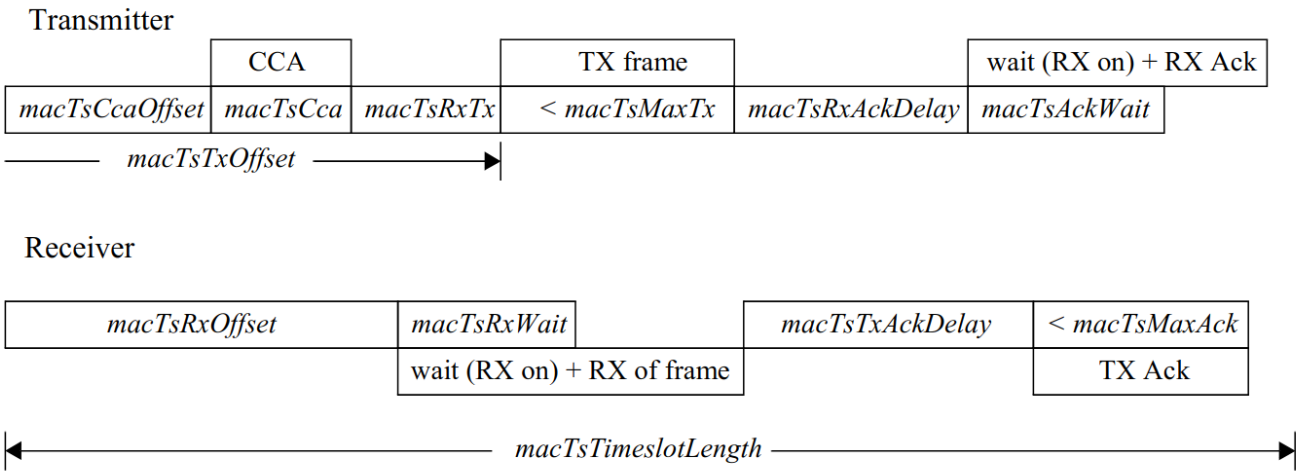


**Figura 1:** Diagrama de Transmisor/Receptor con Comunicación Exitosa - Extraído de la Figura 6-30 de *IEEE Std 802.15.4-2015*[4].

La idea principal es modificar los *timeslots*, dado que los *slotframes* son los que al final en la sub-capa MAC al utilizar TSCH, se encargan de la comunicación uno a uno entre nodos con este protocolo MAC. Como sugerencia de la propuesta, los tiempos asociados al esquema de la Figura 1 se encuentran en `os/net/mac/tsch/tsch-timeslot-timing.c` dentro del directorio de Contiki-NG. Los tiempos **por defecto** corresponden a un timeslot de 10 ms (donde los tiempos se definen en micro segundos) para guardar los tiempos inviduales en un struct de tipo `tsch_timeslot_timing_usec`.

Cada plataforma dentro de las disponibles en Contiki-NG define su configuración, en general en `contiki-conf.h`, utilizando ciertos archivos con configuraciones para la plataforma. En particular, en los archivos de la forma `cpu-def.h`, se encuentran los *timings* los cuales va a utilizar la plataforma para TSCH mediante un macro:

```
#define TSCH_CONF_DEFAULT_TIMESLOT_TIMING = ...
```

En caso de no definirse, se utilizan los tiempos por defecto. Si bien en este punto es posible probar incrementar los tiempos de la red por un factor de `STRETCH`, cabe destacar que el struct, definido en `os/net/mac/tsch/tsch-types.h` es de tipo `uint16_t`. A modo de ejemplo, el caso por defecto de 10 ms corresponde a 10000 $\mu$s y con un factor para incrementar todos los tiempos en 10, ya produce un overflow en el entero sin signo de 16 bits. Para solucionar este inconveniente se cambian todas las referencias en las cuales aparece `uint16_t` por `uint32_t` en variables con una convención de nombres similar a `tsch.timing`, como `tsch_default_timing_us`.

A modo de demostrar visualmente este efecto, se presenta a continuación en la Figura 2 y Figura 3 en una red TSCH/RPL en modo *RPL Lite* un antes y después de enlentezerla. El tiempo mostrado en las capturas es desde el inicio de la foto hasta el momento marcado en la misma.

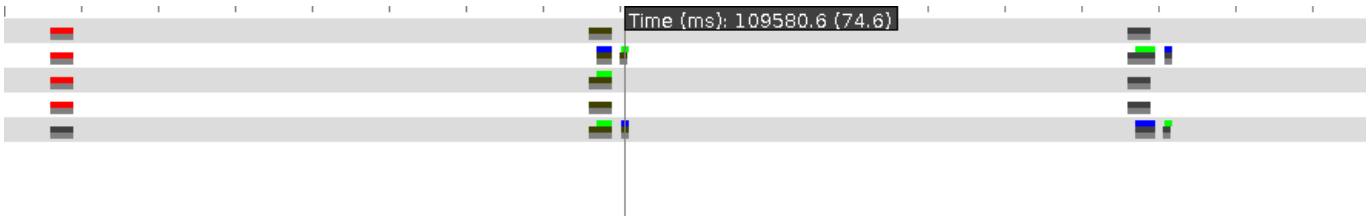


**Figura 2:** Red TSCH-RPL con tiempos de *timeslot* por defecto

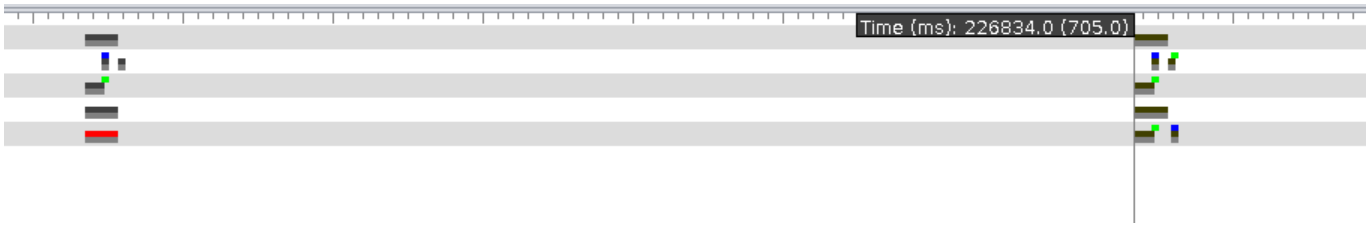


**Figura 3:** Red TSCH-RPL con tiempos de *timeslot* 10 veces más lentos

Al ser una simulación, la misma utiliza *cooja* como plataforma la cual utiliza los tiempos por defecto, los cual son incrementados proporcionalmente. En dicha red los *slotframe* son de 7 *timeslots* (por defecto) y no se envían mensajes, solo se observa la sincronización TSCH mediante EB y el formado de la red RPL. Dicho esto, la distancia entre slotframe es coherente que pase 70 ms a 700ms, en donde los tiempos de la Figura 1 son "estirados" y son mayores.

Una observación que surgen de las modificaciones, es la diferencia de establecimiento tanto de la coordinación TSCH como la red RPL. En la Figura 2 en simulación ambos demoraban alrededor de 3 minutos, mientras que Figura 3 alrededor de 6 u 8 minutos, en los cuales ciertos nodos desvinculaban de la red.

## 4.2. Radio - Detección de Estados y TX/RX

Para la detección de los estados de transmición y recepción (TX/RX), se implementó un *sniffer* de radio, para el cual se tuvo que modificar el sistema operativo. Para esto, se definieron eventos

que abstraen el encendido y apagado de la radio, además de eventos que abstraen el momento de transmisión o recepción de la radio. Lo anterior fue posible ya que las funciones del driver de la radio (`NETSTACK_RADIO`) están definidos por Contiki-NG como una estructura de punteros a funciones.

**Implementación del *sniffer*.** En primer lugar, se guarda en una variable la función que se quiere espiar (Ejemplo: si se define una variable `radio_on_function`, se le asigna `radio_on_function = NETSTACK_RADIO.radio_on()`). Por otro lado, se define una función que emita un evento correspondiente a lo que se esté espiando y luego ejecute como función al puntero donde se guarda la función original (que ejecuta el comando espiado). Por último, para poder realizar lo descrito anteriormente, se debe asignar la función original a la variable para guardarla y luego reemplazarla por el puntero a la función espía.

**Ejemplo.**

```
function_on = NETSTACK_RADIO.on;
NETSTACK_RADIO.on = on_handler;
```

Aquí, `function_on` es el puntero donde se guarda la función original y `on_handler` es aquella función que devuelve el resultado de la función original, pero produce un evento previo a llamarla.

Para lo anteriormente descrito, se necesita que la estructura de definición de driver de radio pueda ser alocable dinámicamente (es decir, que esté alojada en RAM) y, en consecuencia, que no esté declarada como `const` en su implementación. La implementación de Contiki-NG, define sus estructuras de driver de radio como constantes, por lo que fue necesario modificar el sistema operativo mediante un *patch* para que ya no estuvieran definidos de esa forma. Se considera que la utilización de un patch es la forma menos invasiva de modificar el código de Contiki-NG.

## 4.3. Visualización mediante LEDs

La visualización mediante LEDs fue implementada esencialmente con un proceso de Contiki-NG que actúa como máquina de estados. Se definieron 2 modos: uno para poder visualizar los momentos de transmisión/recepción (TX/RX) de los nodos, y otro modo para visualizar el canal por el cual se realizan las comunicaciones.

### 4.3.1. Modo TX/RX

En este modo, al prenderse o apagarse la radio se encienden los LEDs de todos los colores. El estado de transmisión se indica con el parpadeo del LED verde mientras que el estado de recepción se indica con el parpadeo del LED rojo. Por ejemplo, en caso de transmitir un mensaje, el nodo comienza con los LEDs apagados. En el momento de comienzo de su timeslot, todos los LEDs se deberían prender, y luego al comenzar a transmitir su mensaje pendiente, el LED rojo debería parpadear.

De manera ilustrativa, se adjunta la Figura 4 con tal de ver este comportamiento:

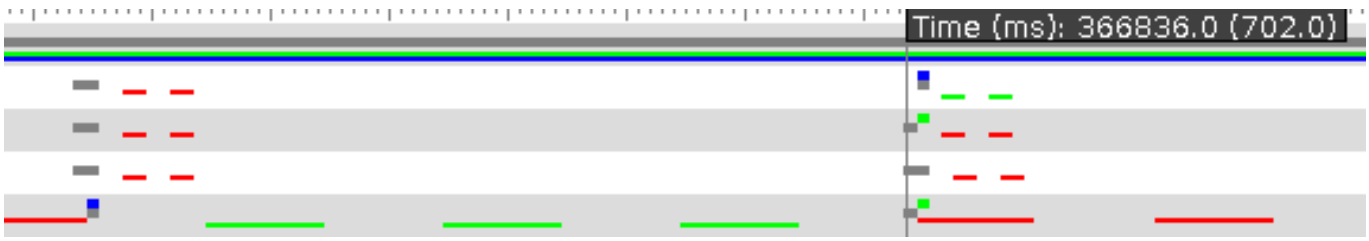


**Figura 4:** LEDs en modo TX/RX con tiempos de *timeslot* 10 veces más lentos.

Efecivamente, en la Figura 4 se observa como en el segundo timeslot, un nodo transmite, blinkeando su LED verde (actividad de radio azul) y luego se recibe por otros (actividad radio verde), blinkeando su LED rojo. Notar que si bien uno de los nodos no recibe, se prende la radio por eso el LED rojo prendido parpadeando. El tiempo mostrado en las capturas es desde el inicio de la foto hasta el momento marcado en la misma.

Destacar que el nodo *server* es el de la parte inferior de la Figura, presentando un tiempo de blinkeo mayor, por construcción. Destacar que para estos tiempos, los nodos *cliente* parpadean con una frecuencia de 20 ms la cual es apenas suficiente para poder observar cambios sin presentar efecto *dimmer* (apagado "tenue" y suave).

### 4.3.2. Modo Visualización de Canal

Para este método, se decidió que el estado de encendido-apagado de la radio se indica con el LED rojo, mientras que el canal de comunicación se indica con el LED verde del kit de desarrollo. La implementación tiene la limitación de poder representar los canales de una red TSCH de solamente 2 canales; sin embargo se consideró que didácticamente lo anterior es suficiente para visualizar el funcionamiento de la misma. Esta limitante es facil de modificar pues los modulos de *tsch* en Contiki-NG permiten redifinir la secuencia de saltos y la cantidad de canales maximos que debe de saltar (channel offset).

Nuevamente, se adjunta una captura en la Figura 5 para observar este comportamiento:

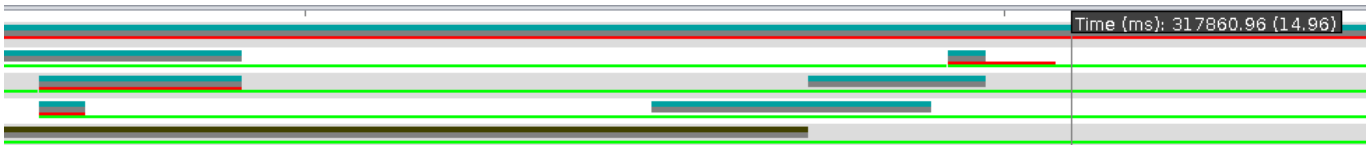


**Figura 5:** LEDs en modo Visualización de Canal con tiempos de *timeslot* 10 veces más lentos.

Destacar, como el caso anterior, que la red es la misma, y se observa dificultad para que todos los nodos se unan a la red, y se enlacen mediante TSCH. Destacar el tiempo en ms que permanece prendido el LED rojo, lo cual es comportamiento esperado, pero no deseado. El tiempo mostrado en las capturas es desde el inicio de la foto hasta el momento marcado en la misma.

## 4.4. Integración de los Módulos

Los módulos descritos en Subsección 4.2 y Subsección 4.3 fueron integrados en los archivos fuente `udp-client.c` y `udp-server.c`, que se basan en los archivos de los ejemplos nativos de Contiki-NG mencionados en la Subsubsección 1.3.1 y ejemplos de laboratorio del curso de RSI 2023. Todo el código fuente fue desarrollado en un repositorio de Git-Hub, el cual es público y puede accederse mediante este link[5].

**udp-client.c** Compuesto por 3 procesos de Contiki-NG, el firmware de los nodos cliente está pensado para que el usuario seleccione el nodo destino de cada cliente oprimiendo su botón, tantas veces como el *node ID* del nodo destino deseado. Además, hasta que no se oprima el botón, no se envían mensajes UDP, por lo que da tiempo al usuario para inicializar la red y observar su armado. Los tres procesos implementados son `udp_client_process` (Subsubsección 1.3.2, Laboratorio 4), `node_select_process` (Subsubsección 1.3.2, Laboratorio 1) y `radio_sniffer_pr`. Este último es producto de lo discutido en las secciones 4.2 y 4.3.

**udp-server.c** Es el firmware pensado para el nodo raíz de la red TSCH-RPL. Está implementado de manera similar a `udp-client.c`, pero sin el proceso de selección de nodos ni funcionalidad de botón.

Dado que por el flujo de trabajo de Git-Hub no es deseable tener carpetas con "builds" de archivos pues las mismas pueden no funcionar al ser utilizados en otro equipo se omiten. Esto se relaciona con el tercer objetivo general propuesto pues no se podria en este caso crear "demo" pre compilados.

Este requerimiento de trabajo da la necesidad de crear scripts que faciliten el armado de los "proyectos demos", para ser compilados en el momento. Con este fin se crean dos scripts, llamados `patch-setup.bash` y `visual-options.bash` para facilitar tanto el proceso de parcheo como de cargado de las placas *client* y *server*.

La herramienta de patch ayuda a modificar el codigo fuente de Contiki-NG, ayudando a su vez a propagar el factor de "estiramiento" del tiempo en cada plataforma (las mencionadas al principio del documento). Luego, la herramienta visual permite tener la utilidad de modificar el comportamiento visual, el desarrollado en la Subsección 4.3.

A su vez, los *Makefile* de los proyectos de los dos tipos de nodos desarrollados fueron mejorados con tal de permitir tener la flexibilidad a la hora de querer compilar los distintos parametros "grandes" de una red TSCH/RPL. Estos son, por ejemplo, el utilizar *Orchestra* como *scheduler* para TSCH o modificar el comportamiento de las tablas de routeo en RPL. La utilización de dichas herramientas se detalla en profundidad en el ***README.md*** del repositorio del proyecto en la branch *main*.

**Observación.** No se implementa la modificación de parámetros de la red en tiempo de ejecución mediante, por ejemplo, la utilización de CoAP. Esto se debe a que si bien puede resultar algo engorroso el tener que cargar el firmware en los nodos cada vez que se quiera modificar algún parámetro, la utilización de los scripts previamente mencionados agiliza considerablemente este proceso. Además, no se cuenta con un estimativo realista del trade-off entre el esfuerzo que conllevaría ahondar en una implementación de este tipo, contra el tiempo que ahorraría una vez logrado, de ser posible. Es altamente probable que la modificación de parámetros en tiempo de ejecución quede por fuera del alcance del proyecto debido a la complejidad de su implementación.

## 4.5. Pruebas

Dadas las herramientas desarrolladas en el proyecto, es mucho mas sencillo el análisis de las pruebas tanto para hardware como las simulaciones, pues en general es tedioso modificar código y cargar el firmware en los nodos. Esencialmente se busca ver que dadas las opciones disponibles, el sistema se comporta de manera acorde en cada una de ellas. Teniendo en mente las 2 plataformas de prueba distintas, siendo *cooja* (simulaciones) y los launchpad CC1350 disponibles (hardware), los casos que se busca observar son:

1. TSCH Minimal y RPL Lite
    *a*) Modo TX/RX
    *b*) Modo Visualización de Canal
2. Orchestra y RPL Classic
    *a*) Modo TX/RX
    *b*) Modo Visualización de Canal

Se considera que son los casos mas pertinentes en analizar dado que presentan los cambios visuales mas importantes al funcionamiento de los LEDs (sabiendo como se comportan). Destacar que las pruebas se van a centrar en factor de "estiramiento" de 10 pues como se observó en la Sección 4, tener un factor muy pequeño o muy grande puede afectar por un lado la visualización y por otro el armado de la red TSCH/RPL.

### 4.5.1. En Simulaciones

Mencionadas las pruebas anteriores, la topología de la red TSCH/RPL que se va a utilizar en ellas va a ser la presente en la Figura 6:

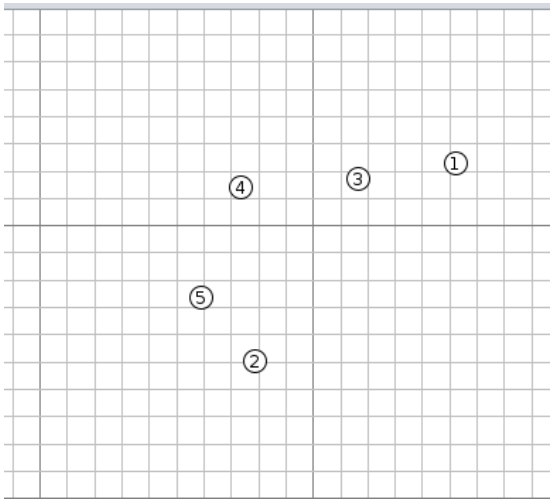


**Figura 6:** Topología de la red utilizada en las simulaciones.

Wireshark es útil a la hora de analizar la comunicación en redes, pero como en este caso hay cierto componente visual, se opta por utilizar la interfaz de *cooja* para facilitar el análisis del sistema. La cantidad de nodos simulados es menor (deberían de rondar los 10 nodos en total), realizar el análisis en una red de ese estilo se torna complejo. Se espera que los resultados sean igualmente representativos.

**TSCH Minimal y RPL Lite.** En esta prueba se obtuvieron los mismos resultados visuales que en las Figuras 4 y 5, produciéndose exactamente los mismos patrones en el modo TX/RX como en el modo çanal". Esencialmente sucede que el modo TX/RX funciona de manera perfecta, en el cual los tiempos de blinkeo son los esperados y no permiten un efecto "dimmer". En cambio el otro modo de visualización funciona de manera correcta pero en intervalos de tiempo extremadamente cortos lo cual hace perder el efecto de visualizar los canales como el encedido y apagado de la radio. Destacar que los tiempos de armado de la red fueron muy elevados, en simulación tardando unos 10 minutos.

**Orchestra y RPL Classic.** En el caso de *TSCH Orchestra*, el funcionamiento fue el esperado, armándose la red correctamente, en tiempos considerablemente menores al de *TSCH Minimal*. Se atribuye a que las tablas de ruteo junto al scheduler de *Orchestra* permiten solventar la falta de sincronización que puede ocurrir en *TSCH Minimal*. Nuevamente la funcionalidad de TX/RX se aprecia de manera excelente en la Figura 7 y con los mismos tiempos LEDs que en la Figura 4. Destacar que, como en el caso del otro test, la funcionalidad de visualización del canal funciona en tiempos muy cortos y no se puede apreciar, tal como se observa en la Figura 8.

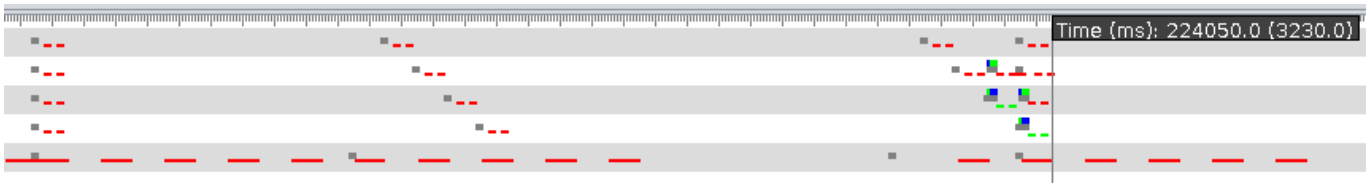


**Figura 7:** *Timeline* de la red *Orchestra* y *RPL CLassic* en modo de visualización TX/RX.

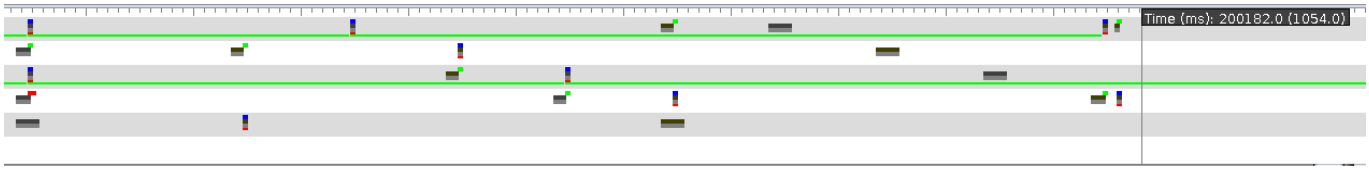


**Figura 8:** *Timeline* de la red *Orchestra* y *RPL CLassic* en modo de visualización de canales.

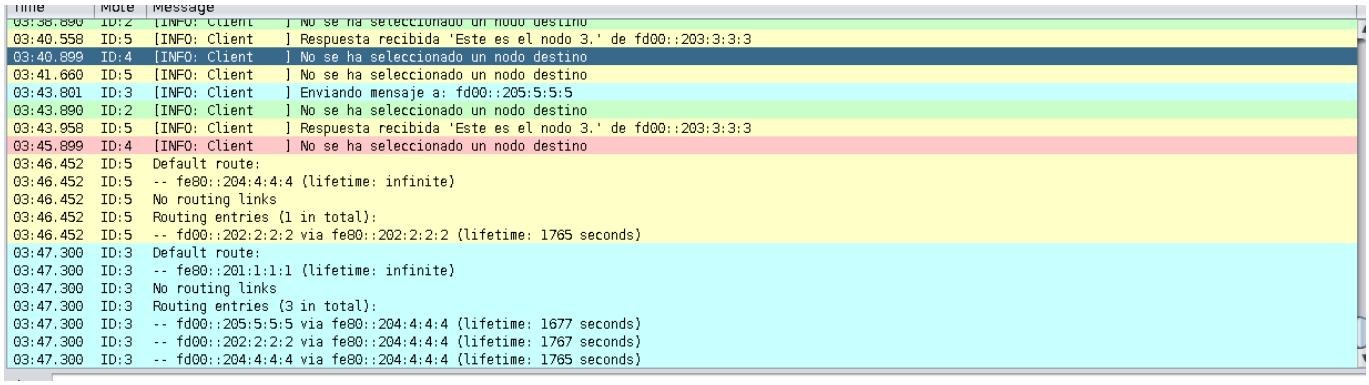


**Figura 9:** Salidas de los nodos de la red *Orchestra* y *RPL CLassic* al mismo tiempo que las Figuras 7 y 8.

### 4.5.2. En Hardware

Se realizaron las pruebas en hardware sobre dos placas *LaunchPad CC1350*, una cargada con `udp-client.c` y otra con `udp-server.c`. Tras varias pruebas, observamos que nunca termina de armarse correctamente la red TSCH. Las Figuras 10 y 11 muestran el estado de la red TSCH transcurridos 150 segundos y más de una hora respectivamente.


**Figura 10:** Estado de la red TSCH en client (derecha) y server (izquierda) transcurridos 150 segundos de la inicialización. Se observa cómo el nodo client no alcanza a comunicarse con el nodo server.

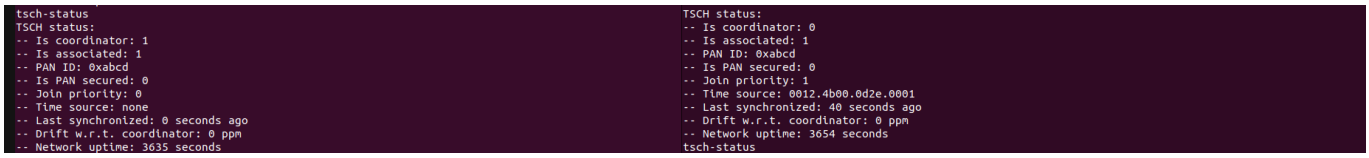


**Figura 11:** Estado de la red TSCH en client (derecha) y server (izquierda) luego de transcurrido una hora de la inicialización.

# 5. Conclusiones

Observando los objetivos generales mencionados en la Subsección 2.1, se puede concluir lo siguiente:

- Todos los objetivos fueron cumplidos a nivel de simulación;
- No se cumplió con los objetivos generales 1 y 2 a nivel de hardware.

Así como se detalla en la Subsubsección 4.5.1, se logró el funcionamiento de una red TSCH-RPL simulada hasta 15 veces más lenta que la velocidad por defecto. Se considera que este enlentecimiento, junto con la visualización de eventos de radio mediante LEDs es suficiente para mostrar las etapas de funcionamiento de una red TSCH-RPL.

Aún así, se considera también que hay lugar para mejoras, en lo que al factor máximo de enlentecimiento se refiere. Como la prioridad durante el desarrollo del proyecto fue la de lograr modificar exitosamente los parámetros de la capa de acceso al medio (por una cuestión de estabilidad de la red), no se ha podido profundizar en la modificación de aspectos relacionados al protocolo RPL, como puede ser, por ejemplo, la frecuencia de envío de mensajes DIO.

Por otra parte, a nivel de hardware no se pudo cumplir con los objetivos, dado que a la hora de realizar las pruebas nunca termina de establecerse la red entre los nodos. Una posible causa de esto es interferencia y/o colisiones de paquetes en el medio físico, que hacen que no se puedan intercambiar los mensajes necesarios para el correcto armado de la red. Otra posible causa es la falta de coordinación entre los nodos, ocasionada por la ausencia de un *Enhanced Beacon* dentro del *timeout* de la red. Como se modifican los *timeslots* para que sean más grandes, dependiendo del *SlotOffset* del EB, es posible que nunca llegue a mandarse antes de que expire la red. Si bien se ha incrementdo dicho *timeout*, puede que no sea incremento suficiente. Sin embargo, no es aconsejable continuar aumentando el *timeout*, pues puede agravar el *clock drift* (deriva) que ya presenta la red de por sí.

En cuanto a los objetivos específicos de la Subsección 2.2, el único que no se ha podido completar en su totalidad es el último, pues no se ha generado un conjunto de demos *plug & play*. Sin embargo, cabe destacar que el proyecto en su estado actual sienta las bases para una rápida elaboración de dicho set. Es decir, las piezas fundamentales, dígase el patch de Contiki-NG, los modos de visualización (tx/rx versus canal), el *Makefile*, los scripts mencionados en la Subsección 4.4 y, especialmente, el *README.md*, habilitan al usuario a generar un set de demos de manera rápida, sencilla y adaptada a sus requisitos.

## 5.1. Comparación con la planificación original

Cabe destacar que la realidad del desarrollo del proyecto no se correspondió con la planificación original (reproducida en el Apéndice A), por diversos motivos.

**Subestimación del tiempo de investigación requerido.** Dada la buena documentación que posee Contiki-NG, se asumió al principio del proyecto que una semana de investigación sería suficiente para averiguar los mecanismos adecuados para implementar lo que se pretendía. Sin embargo, resulta que la documentación no tiene el nivel de detalle necesario para nuestros propósitos, particularmente en lo que refiere a las modificaciones que se debían hacer al sistema operativo para poder enlentecer los tiempos de la red. Esto tuvo el efecto de incrementar las horas necesarias para determinar con precisión los archivos del código fuente de Contiki-NG que era necesrio modificar.

**Imprevistos y percances en el transcurso del tiempo planificado.** Hubo varios impedimentos durante el tiempo original de desarrollo, entre ellos, la enfermedad de algunos compañeros del equipo, así como entregas y parciales de otros cursos de la facultad, que en su conjunto llevaron a la suspensión de avances durante una semana entera.

# A. Planificación Original (Extraído de la especificación del proyecto)

Se estima que al proyecto se deben de dedicar 24 horas semanales (contando a todos los integrantes del equipo) al proyecto en el transcurso de las 4 semanas del mismo. La planificación se dará de la siguiente manera:

1. Semana 1 (23/10 - 29/10): En la primera semana se investigará cómo modificar los tiempos de slotframe de las redes TSCH basado en los códigos de ejemplos enumerados en la sección 1.3. Al final de esta semana se espera tener los conocimientos no desarrollados en el curso necesarios para poder llevar a cabo el proyecto junto a una primera prueba corta producto de la investigación.

2. Semana 2 (30/10 - 5/11): En esta semana se comenzará con el desarrollo del firmware. Al final de esta semana, se espera que el firmware desarrollado sea capaz de construir la red TSCH con los tiempos de slotframe configurados por usuarios. Por otro lado, también se espera que a esta altura del proyecto se puedan visualizar las comunicaciones a partir de los LEDs.

3. Semana 3 (6/11 - 12/11): En la penúltima semana, se concluirá el desarrollo de firmware. Se espera haber terminado la interfaz de usuario para modificar el display de los LEDs en los nodos cliente a partir de la shell del nodo border router. Otro avance que se planifica realizar esta semana es el de optimizar la potencia de las comunicaciones para poder formar correctamente una red mesh en espacios cerrados y con mucha cercanía.

4. Semana 4 (13/10 - 19/11): La última semana será reservada para la preparación de la entrega final, centrándose en la preparación de la documentación de como utilizar la red armada en el proyecto, detallando cada demo disponible junto a lo que se espera visualizar. Se espera avanzar con la entrega de la documentación final del proyecto.

El modo de operación que se utilizará para desarrollar firmware, se basará en paralelización y verificación de pares. Esto significa que cada integrante del grupo será responsable de desarrollar un módulo a la vez. Cuando un integrante considere su módulo como terminado, previo a integrarlo al proyecto, el mismo pasará a revisión de pares, donde el resto de integrantes realizarán sugerencias respecto al trabajo realizado. Una vez aprobado el desarrollo del módulo, el mismo se integrará al proyecto principal. Para esto, se utilizará *Git-Hub*, el cual permite el control de versiones, trabajo paralelo mediante branches y verificación mediante la herramienta de pull request.

# Bibliografía

[1] Orchestra in a Contiki-NG Context. (© Copyright 2018-2022, Contiki-NG maintainers and contributors. Revision bbf05fd4) Orchestra. [Online]. Available: https://docs.contiki-ng.org/en/develop/doc/programming/Orchestra.html

[2] Contiki-NG Documentation. (© Copyright 2018-2022, Contiki-NG maintainers and contributors. Revision bbf05fd4) Contiki-ng. [Online]. Available: https://docs.contiki-ng.org/en/develop/

[3] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The Contiki-NG open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, p. 101089, 2022. [Online]. Available: https://doi.org/10.1016/j.softx.2022.101089

[4] "Ieee standard for low-rate wireless networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, 2016.

[5] Giuliano Turpía, Diego Fraga, Ignacio Valettute. (© 2023 GitHub, Inc.) rsi-proyecto-grupo6. [Online]. Available: https://github.com/giulianoturpia1/rsi-proyecto-grupo6