UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

INSTITUTO UNIVERSITARIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES

Tesis Doctoral

PLATAFORMA DE MONITORIZACIÓN HÍBRIDA PARA EVALUACIÓN DE REDES INALÁMBRICAS DE SENSORES

Autor:

Marlon Renné Navia Mendoza

Tutor:

José Carlos Campelo Rivadulla

Directores:

Alberto Miguel Bonastre Pina

José Carlos Campelo Rivadulla

Valencia, mayo de 2018

Para Eliana, Elián y Ezequiel, esto es por y para ustedes.

Agradecimiento

Quiero agradecer a Dios por permitirme ser lo que soy y darme todo lo que me ha dado en la vida. Gracias a Alberto y José Carlos por guiarme en el desarrollo de esta Tesis, y corregirme (para bien) cuando me he equivocado; a Ricardo y a Raúl por darme una mano cuando la necesitaba en la realización de los experimentos; a Juanjo, Rafa y Juanvi por las ideas que aportaban cuando tocó escribir artículos o preparar experimentos; a la SENESCYT por financiar mis estudios de posgrado; y de manera muy especial a mi familia por apoyarme todo este tiempo, este logro también es de ellos.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

RESUMEN

Desde hace unos años las Redes Inalámbricas de Sensores (WSN por sus siglas en inglés) han demostrado ser fundamentales en la implementación de paradigmas como el denominado Internet de las Cosas (IoT por sus siglas en inglés) o las ciudades inteligentes (*Smart Cities*). Sin embargo, por su forma de funcionamiento y entornos donde operan, este tipo de redes son susceptibles a errores o problemas durante su operación, debido principalmente a la influencia de factores de entorno. Además, al tratarse de sistemas distribuidos donde múltiples nodos (en algunos casos heterogéneos) colaboran en las tareas de sensorización, procesado de información y transmisión de los datos obtenidos; resulta extremadamente complejo evaluar su funcionamiento. La monitorización de las WSN, tanto durante su desarrollo, como en el despliegue o a lo largo de su vida útil, es la mejor forma con la que se puede observar cómo trabajan, ya sea con fines de depuración, verificación de su correcta instalación, o control de su operación.

Los monitores o plataformas de monitorización suelen clasificarse según su enfoque, bien *activo* o bien *pasivo*. Los monitores activos requieren algún nivel de modificación en el sistema monitorizado y permiten adquirir información más precisa de la operación de la WSN, pero pueden interferir con su funcionamiento. Por otro lado, los monitores pasivos no requieren modificación del sistema observado por lo que prácticamente no causan interferencia o intrusión en la misma, pero la información obtenida puede no ser suficiente. Existen también los bancos de pruebas (*testbeds*), que pueden funcionar tanto de forma pasiva (solo recolectando información de los nodos) como de forma activa (cuando también envían órdenes a estos), pero su uso está limitado a entornos de laboratorio.

Tradicionalmente, un monitor puede clasificarse como *hardware* o *software* en función de su naturaleza. También se ha definido un enfoque híbrido, donde se combinan elementos hardware y elementos software en el proceso de monitorización. En los últimos tiempos se ha denominado como híbrido a cualquier propuesta que combine al menos dos enfoques de funcionamiento. En este sentido, existen contadas propuestas híbridas de monitorización para WSN, pero están enfocadas a un tipo específico de nodos, su funcionamiento es limitado (solo pueden recoger cierta información), y en ocasiones solo se han presentado como propuesta teórica, sin alcanzar la fase de implementación que proporcione un monitor operativo.

En esta tesis se presenta la propuesta de una Plataforma de Monitorización Híbrida, denominada HMP (*Hybrid Monitoring Platform*), orientada a la evaluación del comportamiento de cualquier WSN. Esta plataforma busca combinar los enfoques de monitorización *activo* y *pasivo* para aprovechar las ventajas de ambos al tiempo que se compensan sus inconvenientes. La arquitectura de la plataforma

sigue un Modelo de Referencia genérico para plataformas de monitorización distribuida. Este modelo busca que los sistemas o plataformas propuestos puedan ser aplicables a cualquier sistema distribuido, y específicamente a cualquier WSN, y que la independencia de niveles permita compartimentar la problemática de cada nivel, y por tanto los avances en un nivel del modelo no impliquen cambios en los otros niveles. Esta independencia permitiría, además, simplificar la interoperabilidad entre plataformas.

En la plataforma propuesta pueden encontrarse tres tipos de componentes principales: Los Nodos Monitores, que se conectan de forma activa a los nodos de la WSN; los Nodos Espías (*Sniffer* en inglés), que capturan en el medio compartido los mensajes enviados por la aplicación WSN a monitorizar; y finalmente un Servidor de Monitorización. Los dos primeros componentes registran los eventos observados de la aplicación WSN a monitorizar, a la par que registran el instante de tiempo en que éstos ocurren, y los hacen llegar al tercero, encargado de coordinar la recolección de los datos obtenidos por los otros componentes, y su procesado. El Servidor de Monitorización procesa los datos obtenidos, combinándolos en una única traza que refleja el comportamiento global de la aplicación. Esta traza puede ser analizada para evaluar las prestaciones, optimizar el funcionamiento o detectar problemas durante el diseño, despliegue, sintonización u operación de la aplicación basada en WSN.

Todo ello es posible gracias a múltiples avances alcanzados en distintos aspectos de la plataforma de monitorización. Uno de ellos es la definición de la arquitectura genérica que permite la interoperabilidad, reutilización, y modularidad en el desarrollo de los elementos. Otro es la reducción de la intrusión causada por la operación de monitorización sobre los nodos de la WSN, especialmente desde el punto de vista de las transmisiones al Nodo Monitor. Otra de ellas es el novedoso mecanismo de sincronización de las marcas de tiempo de los datos obtenidos. Este mecanismo, denominado GTSO (*Global Trace Synchronization and Ordering Mechanism*), realiza una sincronización fuera de línea (es decir después de haber terminado la monitorización). Su funcionamiento es simple pero efectivo, como se demuestra en las pruebas realizadas. Además, evita que existan cambios en el orden correcto de los eventos registrados.

La plataforma completa ha demostrado su utilidad mediante la monitorización de una WSN real, obteniendo resultados satisfactorios, tales como la obtención de una traza completa y ordenada de los eventos de la red que refleja el comportamiento del sistema, así como la detección de funcionamientos incorrectos en la misma en base al análisis de la traza obtenida. Como valor añadido, gracias al enfoque basado en una arquitectura estructurada en niveles independientes, HMP puede ser aplicada de forma modular y sencilla a diversos tipos de WSN.

ABSTRACT

For some years, Wireless Sensor Networks (WSN) have proven to be fundamental in the implementation of paradigms such as the Internet of Things (IoT) or the Smart Cities. However, due to their mode of operation and the environments where they work, this type of network is susceptible to errors or problems in their functioning, mainly due to the effects of environmental factors. In addition, as these are distributed systems where multiple nodes (in some cases heterogeneous) collaborate in the tasks of sensorization, information processing and transmission of the obtained data; it is extremely complex to evaluate their operation. The monitoring of the WSN, both during its development as in the deployment or throughout its useful life, is the best way to observe how they work, either for debugging purposes, verification of correct installation, or operation control.

Monitors or monitoring platforms are usually categorized according to their working approach, either *active* or *passive*. Active monitors require some level of modification in the monitored system and allow getting more accurate information about the operation of the WSN, but may interfere with its operation. On the other hand, passive monitors do not require modification of the observed system so they practically do not cause interference or intrusion in it, but the information obtained may not be sufficient. There are also *testbeds*, which can function both in passive way (only collecting information from the nodes) and in active way (when they also send orders to nodes), but their use is limited to laboratory environments.

Traditionally, a monitor can be categorized as *hardware* or *software* monitor depending on its composition. A hybrid approach has also been defined, where hardware elements and software elements are combined in the monitoring process. In recent times, many proposals that combine at least two operational approaches have been called "hybrid". In this sense, there are few hybrid monitoring proposals for WSN, but they are focused on a specific type of nodes, their operation is limited (they can only collect certain information), or they have only been presented as a theoretical proposal, without reaching the phase of implementation that provides an operational monitor.

This thesis presents the proposal of a Hybrid Monitoring Platform, called HMP, aimed at evaluating the behavior of any WSN. This platform seeks to combine *active* and *passive* monitoring approaches to exploit their advantages while compensating for their drawbacks. The architecture of the platform follows a generic reference model for WSN monitoring platforms. This model seeks that the proposed systems or platforms can be applicable to any distributed system, and specifically to any WSN, and that the independence of levels allows compartmentalizing the problems of each level, and therefore advances at a model level do

not imply changes in the other levels. This independence would also make it possible to simplify interoperability between platforms.

Three types of main components can be presented on the platform: The Monitors Nodes, which are actively connected to the WSN nodes; *Sniffer* Nodes, which capture from the shared media the messages sent by the WSN application to be monitored; and finally a Monitoring Server. The first two components register the observed events of the WSN application to be monitored, at the same time that they record the moment of time in which they occur, and they reach them to the third component in charge of coordinating the collection of the data obtained by the other components, and its processing. In turn, the Monitoring Server processes the obtained data, combining them in a single trace that reflects the global behavior of the application. This trace can be analyzed to evaluate the performance, optimize the operation or detect problems during the design, deployment, synchronization or operation of the WSN-based application.

All this is possible thanks to multiple advances achieved in different aspects of the monitoring platform. One of them is the definition of the generic architecture that allows interoperability, reuse, and modularity in the development of the elements. Another one is the reduction of the intrusion caused by the monitoring operation on the WSN nodes, especially from the point of view of the transmissions to the Monitor Node. Another is the novel synchronization mechanism of the time stamps of the data obtained. This mechanism, called GTSO (*Global Trace Synchronization and Ordering Mechanism*), performs an off-line synchronization (that is, after the monitoring is finished). Its operation is simple but effective, as demonstrated in the tests carried out. In addition, it prevents changes in the correct order of recorded events.

The complete platform has proven its usefulness by monitoring a real WSN, obtaining satisfactory results, such as obtaining a complete and orderly trace of network events that reflects the behavior of the system; as well as the detection of incorrect operations in the system, based on the analysis of the obtained trace. As an added value, thanks to the approach based on an architecture structured in independent levels, HMP can be applied in a modular and simple way to different types of WSN.

RESUM

Des de fa uns anys les xarxes sense fils de sensors (WSN per les seves sigles en anglès) han demostrat ser fonamentals en la implementació de paradigmes com el anomenat Internet de los Coses (IoT per les seves sigles en anglès) o les ciutats intel·ligents (*Smart Cities*). No obstant això, per la seva forma de funcionament i els entorns on treballen, aquets tipus de xarxes són susceptibles a errors o problemes durant la seva operació, degut principalment a factors de l'entorn en el que funcionen. A més, en tractar-se de sistemes distribuïts on múltiples nodes (en alguns casos, heterogenis) col·laboren en les tasques d'obtenir les mesurades dels sensors, processar la informació i transmissió de les dades obtingudes, resulta extremadament complex avaluar el seu funcionament. El monitoratge de les WSN, tant durant el seu desenvolupament, com en el seu desplegament o al llarg de la seva vida, és la millor forma amb la qual es pot observar el seu funcionament, ja sigui amb finalitats de depuració, verificació de la seva correcta instal·lació, o control de la seva operació.

Els monitors o plataformes de monitoratge solen classificar-se segons el seu enfocament, ben *actiu* o ben *passiu*. Els monitors actius requereixen algun nivell de modificació en el sistema analitzat i permeten adquirir informació més precisa de l'operació de la WSN, però poden interferir amb el seu funcionament. D'altra banda, els monitors passius no requereixen modificació del sistema observat pel que pràcticament no causen interferència o intrusió en el mateix, però la informació obtinguda pot no ser suficient. Existeixen també els bancs de proves (*testbeds* en anglès), que poden funcionar tant de forma passiva (solament recol·lectant informació dels nodes) com de forma activa (quan també envien ordres a aquests), però el seu ús està limitat a entorns de laboratori.

Tradicionalment, un monitor pot classificar-se com a maquinari o programari en funció de la seva naturalesa. També s'ha definit un enfocament híbrid, on es combinen elements maquinari i elements programari en el procés de monitoratge. En els últims temps s'ha denominat com a híbrid a qualsevol proposta que combini almenys dos enfocaments de funcionament. En aquest sentit, existeixen poques propostes híbrides de monitoratge per WSN, però estan enfocades a un tipus específic de nodes, el seu funcionament és limitat (solament poden recollir certa informació), i en ocasions solament s'han presentat com a proposta teòrica, sense aconseguir la fase d'implementació que proporcioni un monitor operatiu.

En aquesta tesi es presenta la proposta d'una Plataforma de Monitoratge Híbrid, denominada HMP (*Hybrid Monitoring Platform*), orientada a l'avaluació del comportament de qualsevol WSN. Aquesta plataforma busca combinar els enfocaments de monitoratge actiu i passiu per aprofitar els avantatges d'aquests al

mateix temps que es compensen els seus inconvenients. L'arquitectura de la plataforma segueix un model de referència genèric per a plataformes de monitoratge de WSN. Aquest model busca que els sistemes o plataformes proposats puguin ser aplicables a qualsevol sistema distribuït, i específicament a qualsevol WSN, i que la independència de nivells permeti compartimentar la problemàtica de cada nivell, i per tant els avanços en un nivell del model no impliqui canvis en els altres nivells. Aquesta independència permetria, a més, simplificar la interoperabilitat entre plataformes.

En la plataforma proposada poden trobar-se tres tipus de components principals: Els Nodes Monitors, que es connecten de forma activa als nodes de la WSN; els Nodes Espies (*Sniffer* en anglès), que capturen en el mitjà compartit els missatges enviats per l'aplicació WSN a analitzar; i finalment un Servidor de Monitoratge. Els dos primers components registren els esdeveniments observats de l'aplicació WSN a analitzar, alhora que registren l'instant de temps en el quals ocorren, i els fan arribar al tercer, encarregat de coordinar la recol·lecció de les dades obtingudes pels altres components, i el seu processament. Llavors, el Servidor de Monitoratge processa les dades obtingudes, combinant-los en una única traça que reflecteix el comportament global de l'aplicació. Aquesta traça pot ser analitzada per avaluar les prestacions, optimitzar el funcionament o detectar problemes durant el disseny, desplegament, sintonització o operació de l'aplicació basada en WSN.

Tot això és possible gràcies a múltiples avanços aconseguits en diferents aspectes de la plataforma de monitoratge. Un d'ells és la definició de l'arquitectura genèrica que permet la interoperabilitat, reutilització, i modularitat en el desenvolupament dels elements. Un altre és la reducció de la intrusió causada per l'operació de monitoratge sobre els nodes de la WSN, especialment des del punt de vista de les transmissions al Node Monitor. Una altra és el nou mecanisme de sincronització de les marques de temps de les dades obtingudes. Aquest mecanisme, denominat GTSO (*Global Trace Synchronization and Ordering Mechanism*), realitza una sincronització fora de línia (és a dir després d'haver acabat el monitoratge). El seu funcionament és simple però efectiu, com es demostra en les proves realitzades. A més, evita que existeixin canvis en l'ordre correcte dels esdeveniments registrats.

La plataforma completa ha demostrat la seva utilitat mitjançant el monitoratge d'una WSN real, obtenint resultats satisfactoris, com ara l'obtenció d'una traça completa i ordenada dels esdeveniments de la xarxa que reflecteix el comportament del sistema, així com la detecció de funcionaments incorrectes en la mateixa en base a l'anàlisi de la traça obtinguda. Com a valor afegit, gràcies a l'enfocament basat en una arquitectura estructurada en nivells independents, HMP pot ser aplicada de forma modular i senzilla a diversos tipus de WSN.

Capítulo 1

Introducción

El desarrollo en los últimos años del Internet de las Cosas (IoT) es sin duda tan significativo que algunos autores lo consideran la tecnología más relevante del siglo XXI. El paradigma del IoT abarca diversas tecnologías, cuyo despliegue también se ha acelerado (Tuwanut & Kraijak 2015; Al-Fuqaha et al. 2015). La diversidad en cuanto a arquitecturas y tecnologías es tal que actualmente el desafío radica en plantear propuestas de interoperabilidad (Kotsev et al. 2015; Giancarlo Fortino et al. 2016).

Dentro del conjunto de tecnologías que abarca IoT, las Redes Inalámbricas de Sensores (WSN) es una de las más difundidas y estudiadas. Las WSN están conformadas por pequeños nodos –que cuentan con uno o más sensores– de bajo costo, bajo consumo energético, y comunicación inalámbrica a corta distancia; y uno o más sumideros que recolectan la información enviada por los nodos. Su campo de uso es muy amplio, aunque se destacan el área industrial (Xu et al. 2014), agricultura (Ojha et al. 2015), y en medicina y salud (Atanasov 2013).

Sin embargo, a pesar del desarrollo alcanzado, las implementaciones de WSN no están exentas de problemas en su funcionamiento. Ya sea por problemas en la comunicación inalámbrica, malfuncionamiento de algún nodo, factores del entorno, u otros, la operación de una red inalámbrica de sensores puede verse afectada y aparecer errores (Mahapatro & Khilar 2013; Raposo et al. 2017), o sufrir ataques de seguridad (Patel & Aggarwal 2013).

Existen varias formas de evaluar y diagnosticar el funcionamiento de una WSN, y eso puede realizarse durante el desarrollo, despliegue u operación de la

red. Por ejemplo, se puede utilizar simuladores o emuladores, que son utilizados generalmente para probar nuevas funcionalidades, aunque tienen varias limitaciones (Dwivedi & Vyas 2011).

La forma más frecuente de evaluar el funcionamiento de una WSN para detectar estos inconvenientes y descubrir su origen, es monitorizar o depurar el funcionamiento de la misma. Esto se puede realizar durante cualquiera de las etapas antes mencionadas.

1.1. Monitorización para Redes Inalámbricas de Sensores.

La evaluación de una WSN puede realizarse antes o después de su despliegue. Cuando se realiza antes –además de poder utilizar simuladores o emuladores– se utilizan Bancos de Pruebas (*Testbeds*). Los *Testbeds* para WSN permiten emular un ambiente real de funcionamiento a pequeña escala, y probar o depurar la operación de los nodos. Tienen la ventaja de permitir evaluar varios aspectos de la WSN de forma mucho más realista que en un simulador. Existen varios *Testbeds* disponibles para WSN, unos de enfoque abierto y otros cerrados, que consideran diferentes técnicas y enfoques de prueba, y que pueden estar orientados a una arquitectura concreta de nodos –la mayoría de las propuestas– o trabajar con un protocolo de comunicación estandarizado que permita evaluar distintos tipos de nodos (Xiao et al. 2014; Horneber & Hergenroder 2014).

A pesar de las ventajas de los *Testbeds*, pueden presentar inconvenientes para trabajar en redes ya implementadas. Además, la mayoría solo permite hacer evaluaciones en ciertos entornos, y por lo general la topología y las condiciones reales de entorno terminan siendo diferentes a lo probado.

Cuando se quiere evaluar una WSN que ya está funcionando en un entorno real, las herramientas utilizadas se denominan Sistemas o Plataformas de Monitorización, o simplemente *Monitores*. Los Monitores pueden centrarse en muchos parámetros de operación, como el rendimiento, el *jitter* (variación del retardo de transmisión), el tiempo de respuesta o la fiabilidad, e incluso para la seguridad y la detección de intrusos en la red. A pesar de que su uso es mayoritariamente post-despliegue, también pueden utilizarse en la fase de pruebas de una WSN, o en entornos controlados. Las herramientas o plataformas de monitorización pueden tener varios enfoques en su forma de operar. Por ejemplo, hay plataformas de monitorización *online* que ofrecen visualizar la información a medida que se va obteniendo, es decir mientras está funcionando la red; mientras que las *offline* se basan principalmente en guardar la información en forma de trazas, para su posterior análisis o reproducción.

La principal clasificación de los Monitores los engloba/agrupa en *activos*, cuando interactúan directamente con la WSN –y pueden solicitar información a los

nodos–; y *pasivos*, cuando solo se basan en la información que transmiten los nodos, sin añadir información adicional. Los monitores activos pueden lograr obtener información más precisa del funcionamiento de la WSN, pero también generan intrusión que puede interferir con dicho funcionamiento. Además, necesitan agregar programación adicional en los nodos monitorizados. Por otra parte, los monitores pasivos no causan ninguna interferencia con la red o los nodos, pero solo pueden observar directamente la información que éstos transmiten en su funcionamiento/operación habitual, por lo que requieren aplicar algoritmos o técnicas de análisis para detectar problemas en su funcionamiento.

Así mismo, la monitorización puede ser *centralizada*, cuando la información recolectada de los nodos se procesa en un solo equipo (como por ejemplo cuando se analiza lo que llega al sumidero), independientemente del número de componentes de la plataforma de monitorización; o *distribuida*, cuando el procesamiento se distribuye entre los componentes del sistema. La monitorización centralizada permite obtener una imagen global más coherente de toda la red, aunque puede requerir tráfico adicional para obtener toda la información necesaria. Por otro lado, la monitorización distribuida reparte la tarea de detección de errores entre sus componentes, lo que reduce el tráfico entre los mismos, aunque podría no reflejar una visión global del funcionamiento de la WSN (Schoofs et al. 2012).

Otro enfoque, presentado por Rodrigues et al. (2013), clasifica las herramientas de monitorización de acuerdo al origen de los datos en los que se basan para detectar errores. Los monitores *basados en tráfico* trabajan aplicando técnicas de análisis de datos sobre el tráfico de la red para encontrar errores. Los monitores de *estado de nodo* emplean componentes agregados a los nodos, o interactúan con éstos, para encontrar y analizar errores. Los monitores de *estado global* combinan información de múltiples nodos para encontrar errores, identificando estados no válidos a nivel de toda la WSN.

Además de todos estos enfoques, los sistemas de monitorización pueden estar basados principalmente en *software* o en *hardware*. Los monitores basados en software se implementan mediante un código, aplicación o *plug-in* agregado al código o sistema del nodo, que recolecta información específica sobre el estado o funcionamiento del mismo. Permiten una depuración más profunda, pero pueden interferir con la operación normal del nodo. Los monitores basados en hardware consisten en dispositivos conectados al nodo, que recolectan información del mismo. Son menos intrusivos pero implican componentes físicos adicionales.

Tradicionalmente, cuando se habla de monitorización híbrida se hace referencia a una combinación de *hardware* y *software* (Ferrari et al. 1983). Sin embargo, a nivel de sistemas de computadoras, en los últimos años se ha utilizado el término *híbrido* para cualquier sistema que utilice dos enfoques distintos, por ejemplo cableado-inalámbrico, local-global, fijo-móvil, o activo-pasivo. Incluso a nivel de

monitorización ya se han presentado propuestas híbridas. Las propuestas presentadas por Khan & Zulkernine (2014) –que combina enfoques de monitorización por eventos y por tiempo para detectar vulnerabilidades en software– o por Kim et al. (2017) –un sistema de monitorización activo-pasivo para WSN– son ejemplos de monitorización híbrida que no aplican la combinación de enfoque hardware-software.

A pesar del número de propuestas presentadas para monitorizar las WSN, no se ha encontrado una herramienta o plataforma que pueda trabajar de forma estandarizada para la mayoría de implementaciones de WSN. La mayoría se enfocan a un tipo concreto de nodos o arquitectura, o utilizan –en el caso de los monitores activos– alguna interfaz específica que puede no estar disponible en todas las WSN. Tampoco existen modelos que detallen la arquitectura de las distintas plataformas, salvo la propuesta de Capella et al. (2016).

El análisis realizado por Schoofs et al. (2012) concluye que, en este campo, hay oportunidades para los sistemas híbridos, que permitan reducir las desventajas de los distintos enfoques y aprovechar sus ventajas. Sin embargo, se han encontrado muy pocas propuestas de sistemas híbridos de monitorización, y las encontradas están enfocadas a un tipo específico de nodos, su funcionamiento es limitado (solo pueden recoger cierta información), o solo se han presentado como propuesta teórica, sin ofrecer un sistema funcional terminado.

1.2. Objetivos de la Tesis.

El objetivo principal de esta tesis consiste en definir una plataforma de monitorización híbrida para evaluar WSN, que aproveche las ventajas de los enfoques activo y pasivo, y que esté basada en hardware y software. Esta plataforma deberá ser aplicable a distintos tipos y arquitecturas de redes de sensores. La propuesta presentada podrá tomar criterios de otros trabajos anteriores, pero con un enfoque mejorado u optimizado. Dentro de este objetivo principal están comprendidos los siguientes objetivos específicos:

- Analizar las propuestas de sistemas y plataformas de monitorización existentes, para determinar sus ventajas, desventajas, y características más destacadas.
- Diseñar la arquitectura, componentes, y protocolos de la plataforma de monitorización híbrida, buscando un enfoque novedoso en más de un aspecto.
- Implementar los elementos de la plataforma de monitorización híbrida, en base a una arquitectura estandarizada.
- Evaluar el funcionamiento y efectividad de la plataforma de monitorización, aplicándola sobre una WSN real.

1.3. Organización de la Tesis.

Después de haber hecho una breve introducción al tema, explicando las motivaciones del trabajo y los objetivos, el resto de la tesis se organiza de esta manera:

En el Capítulo 2 se realiza una revisión del estado del arte de los sistemas y plataformas de monitorización activas, pasivas, e híbridas; analizando su funcionamiento y características. También se revisa cómo se realiza –o se puede implementar– la sincronización entre los elementos de la plataforma de monitorización.

En el Capítulo 3 se presenta la propuesta de una nueva plataforma de monitorización híbrida. Se detallan los componentes y funcionamiento de forma global, así como de cada uno de sus elementos. También se explica el mecanismo de sincronización de las trazas obtenidas por la plataforma.

El Capítulo 4 abarca las pruebas de evaluación realizadas tanto a los componentes por separado, como a toda la plataforma en su conjunto. La evaluación de la plataforma completa ha sido realizada sobre una WSN real.

Por último el Capítulo 5 presenta las conclusiones de la tesis, el trabajo futuro previsto, y refleja las publicaciones realizadas en base al trabajo realizado.

Capítulo 2

Estado del Arte sobre Sistemas y Plataformas de Monitorización

En este apartado se presenta una revisión del estado del arte en lo que respecta a las propuestas de plataformas de monitorización para WSN encontradas en la literatura. Han sido categorizadas en función de su interacción con la WSN monitorizada, es decir en *Activas*, *Pasivas* e *Híbridas*. Si bien se han analizado una gran cantidad de referencias bibliográficas y propuestas en esta área, se mencionarán y comentarán únicamente aquellas consideradas como más importantes o relevantes. Un análisis más detallado de su funcionamiento ha sido realizado por Capella et al. (2016), como base para el planteamiento de un modelo de referencia para la monitorización de aplicaciones de IoT basadas en WSN.

Además, también se ha hecho una revisión de los mecanismos de sincronización aplicables a los componentes de una plataforma de monitorización. Dado que la propuesta que se presenta es básicamente un sistema distribuido de obtención de datos, se considera la correcta sincronización entre sus elementos como una parte imprescindible para su adecuado funcionamiento.

2.1. Sistemas y Plataformas Activas

Los Sistemas y Plataformas de Monitorización Activas –denominados desde aquí simplemente como *Monitores Activos*– comprenden software o hardware adicional que se agrega a la red a monitorizar. Normalmente, los Monitores Activos implican alguna modificación de los nodos de la WSN, en menor o mayor medida de acuerdo a las funcionalidades de monitorización o depuración que se quieran obtener. En algunos casos, los Monitores Activos permiten no solo la monitoriza-

ción, sino incluso acciones relativas a la administración de los nodos (como, por ejemplo, cambiar la configuración o actualizar el código que éstos ejecutan.)

En los siguientes puntos se analizan varios Monitores Activos, presentados en orden cronológico de publicación.

2.1.1. Sensor Network Management System (SNMS)

SNMS (Tolle & Culler 2005) es uno de los primeros sistemas de monitorización con amplia difusión, y quizás uno de los más destacados. SNMS es básicamente un sistema de gestión de red muy completo, que proporciona un conjunto de servicios: la recolección de datos de prestaciones y estado de la red mediante consultas, y el registro persistente de eventos. También se cita con el nombre de *Nucleos* en algunas referencias.

Está construido sobre TinyOS, un sistema operativo de código abierto diseñado para dispositivos de baja potencia (Levis et al. 2005), y aprovecha las características de este sistema para poder revisar el estado de un nodo e incluso guardar la información del mismo. Los objetivos de diseño de SNMS se resumen en ocupar muy poco espacio en la RAM del nodo, que su código sea muy pequeño, que solo genere tráfico adicional en respuesta a un requerimiento de un administrador humano, y que dependa lo mínimo posible de la aplicación del nodo donde se implemente. Para que este último objetivo se cumpla, SNMS necesita de una red de comunicaciones que opere en paralelo a la red de la aplicación de la WSN. Esta red se utiliza para la recolección y difusión de los datos que maneja SNMS.

El componente de monitorización mediante la generación de eventos –el *Event Logging System*– permite obtener notificaciones programadas de eventos únicos. Los identificadores de estos eventos son cadenas de caracteres previamente definidos por el programador. Se guardan en memoria RAM hasta que el administrador solicite a los nodos el requerimiento para que éstos envíen el registro de eventos generado; aunque también se puede programar para que el registro sea enviado inmediatamente después de guardarse en memoria.

Las funcionalidades de SNMS lo hacen una opción muy interesante para monitorizar una WSN. Sin embargo, se orienta más a la gestión en general de la red que a la monitorización. La sobrecarga en el nodo parece ser relativamente baja –según sus autores, una aplicación nula más la versión mínima de SNMS ocupan 1281 bytes de RAM, de los cuales 620 son de SNMS–, aunque no indican cuál es el tamaño del código. No obstante, no deja de provocar una considerable intrusión en el sistema debido justamente a sus numerosas opciones. Además, el hecho de que esté basado en TinyOS –para aprovechar sus características– limita su aplicación en nodos que utilicen otros sistemas operativos.

2.1.2. Memento

Es un sistema de monitorización activo que utiliza codificación en patrones de bits para identificar el estado o información de varios nodos (Rost & Balakrishnan 2006). Memento agrega su propio protocolo a los nodos, pero utiliza la misma topología y red de comunicación de la WSN para enviar la información capturada.

Memento sintetiza el estado de los nodos de la red en forma de mapas de bits, con una semántica definida para cada síntoma de estado o "salud" determinado. La posición del bit en el patrón indica el número del nodo, mientras que el valor del bit (1 o 0) indica que si el nodo cumple con ese estado. Siguiendo una topología en árbol, en un nodo *n* el protocolo reúne la información de sus nodos hijos y la envía al padre. De esta forma un nodo podría detectar si hay algún problema en una parte de la red. La información de todos los nodos de la red es entregada a un nodo *Gateway*, que lo pasa a un servidor donde la información es transformada en un formato más amigable para su análisis.

El protocolo de Memento tiene tres módulos que permiten su funcionamiento: Un módulo de *administración de vecindario y caché*, un *módulo de sincronización*, y un módulo *detector de inconsistencias*. Los nodos informan de su estado mediante mensajes periódicos tipo "latido" (*heartbeats*). Un nodo puede asumir un error en uno de sus nodos hijos si transcurre un tiempo predeterminado desde el último *hertbeat* recibido de ese nodo.

El módulo *detector de errores* de Memento puede utilizar varios algoritmos para detectar errores. Cada uno de estos algoritmos resulta indicado para ciertos casos, por lo que sus autores recomiendan la elección de algoritmo de acuerdo a los requerimientos de monitorización.

A pesar de ser un monitor activo es relativamente ligero, ya que apenas ocupa 400bytes de RAM en cada nodo. Sin embargo su forma de funcionamiento puede dar lugar a falsos positivos, hecho detectado por sus propios autores, aunque con valores considerados aceptables. Así mismo, genera tráfico adicional en la red monitorizada. Si bien la sobrecarga en la red puede considerarse baja, ésta dependerá de los requisitos de monitorización.

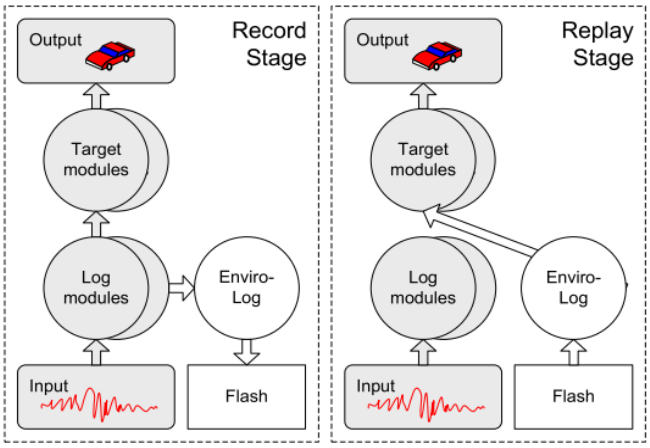
2.1.3. EnviroLog

EnviroLog es un sistema de monitorización que permite registrar eventos de una WSN de forma asíncrona para su posterior reproducción (Luo et al. 2006). Su principal justificación es que el comportamiento de los sistemas basados en eventos no es regular o predictible, y por lo tanto es más difícil de estudiar o simular.

La Figura 1 describe la idea básica de cómo funciona EnviroLog. La aplicación de los nodos de la WSN tiene varios módulos de funcionamiento. En modo *grabación*, un módulo *Log* que recibe una entrada –por ejemplo información de un

sensor o una llamada a una función–, envía la misma al componente de EnviroLog que registra este evento o anotación en memoria no volátil –que es parte del mismo nodo o ha sido añadida– y la pasa también al módulo destino, que a su vez la procesará hacia una salida, que bien podría ser la interfaz de red. Se puede registrar tanto eventos como el estado del nodo mediante el almacenamiento de valores de variables.

En modo *reproducción*, se recupera la información almacenada en la memoria no volátil, y se la transfiere hacia los módulos correspondientes, obteniendo una traza que permite reproducir el comportamiento del sistema a registrar. En este caso las entradas se encuentran deshabilitadas.



Fuente: (Luo et al. 2006)

Figura 1. Idea básica del funcionamiento de EnviroLog.

Para el funcionamiento de EnviroLog se agregan *anotaciones* –a modo de comentarios– en el código de los nodos. En modo normal (sin monitorización) estos comentarios son ignorados. En modo grabación, el código se pasa por un Preprocesador, que convierte estas anotaciones en código, antes de compilarlo y cargarlo en el nodo. Además, se dispone de una aplicación de control para el usuario que se comunica con un nodo principal –denominado *controlador de escenario*– y que puede ser utilizada en tiempo de ejecución, para observar y controlar el funcionamiento de EnviroLog.

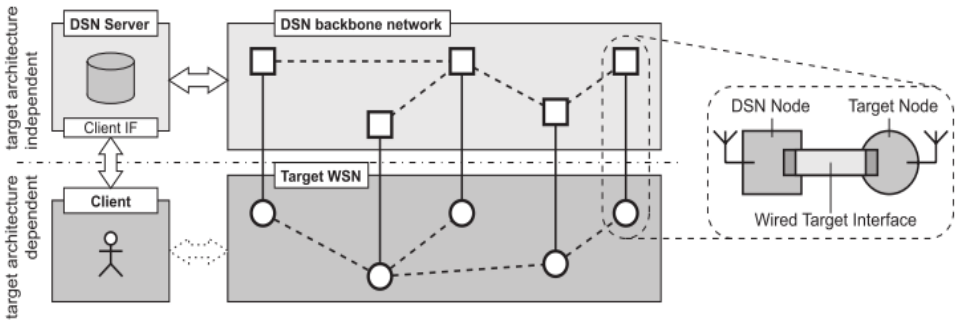
El mecanismo de EnviroLog es simple y útil. Además de ocupar poco espacio en el nodo (en pruebas realizadas las funciones de grabación y reproducción ocuparon 54bytes más el buffer temporal, mientras que para el controlador de escenario en una red multi-salto fueron 137bytes), su implementación es relativamente sencilla. Sin embargo, al ser el mismo nodo quien almacena el comportamiento de la

red, la frecuencia de registro de eventos y el tiempo durante el cual se puede grabar están limitados por las restricciones de recursos del nodo, tal como evidencian los resultados presentados por sus autores.

2.1.4. Deployment Support Network (DSN)

DSN es una plataforma para probar, controlar y monitorizar aplicaciones WSN en ambientes reales (Dyer et al. 2007). Este sistema tiene las funcionalidades de los *testbeds* (explicados más adelante) pero sin las limitaciones de estos.

La Figura 2 muestra el esquema conceptual del funcionamiento de DSN. Se compone de un conjunto de nodos conectados a nodos de sensores, todos ellos conectados a través de una red Bluetooth tipo *scatternet* en la que también se encuentra un servidor DSN. Los nodos DSN registran los eventos transmitidos por la aplicación de nodo –a través de una interfaz serie– y envían la información al servidor DSN, pero también pueden servir como una infraestructura para la programación remota, la transmisión de órdenes y la configuración dinámica de la supervisión.



Fuente: (Dyer et al. 2007)

Figura 2. Esquema del funcionamiento de DSN.

El servidor DSN ejecuta el servicio de registro de datos y eventos de la WSN monitorizada. Los registros de actividad de la red pueden ser enviados de forma proactiva por los nodos DSN, o solicitados explícitamente por el servidor. Además, el servidor DSN proporciona al administrador la información sobre los datos recolectados, así como del estado de la infraestructura DSN.

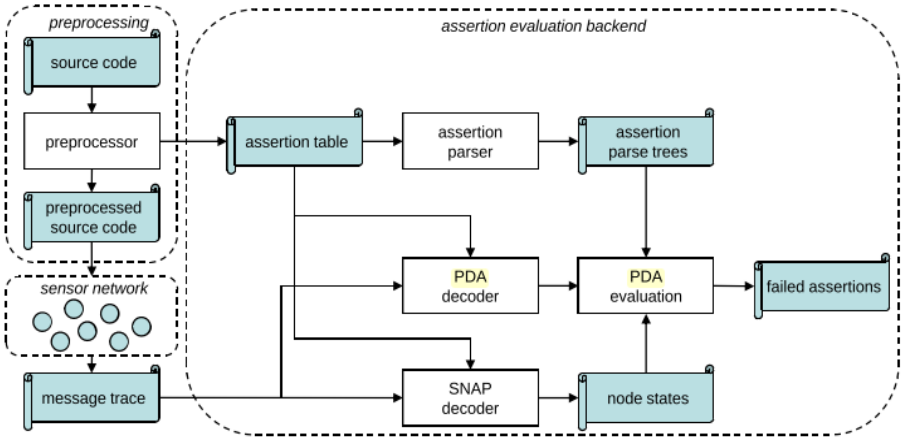
A pesar de las ventajas y posibilidades de DSN, la utilización de una red Bluetooth hace que esta plataforma adolezca de las limitaciones de esta tecnología inalámbrica en cuestiones tales como el alcance de los enlaces o la tasa de transmisión de datos. Los autores no proporcionan información respecto a la intrusión

causada sobre los nodos monitorizados al agregar el código para la transmisión de la información de los eventos a los nodos DSN.

2.1.5. Passive Distributed Assertions for Sensor Networks (PDA)

PDA permite formular *afirmaciones* (*assertions*) acerca del estado de un conjunto de nodos distribuidos, utilizando un lenguaje declarativo simple. Éstas obligan a que el nodo emita información para verificar las afirmaciones, que puede ser capturada y evaluada (Romer & Ma 2009). Estos mensajes pueden ser recogidos de forma pasiva por un *sniffer* implementado para este caso. Las afirmaciones distribuidas son similares a la macro *assert* del lenguaje de programación C.

En la Figura 3 se detalla la arquitectura y funcionamiento de PDA. Cada nodo ejecuta un código adicional que envía un pequeño mensaje cada vez que el valor de un atributo cambia (denominado mensaje de *instantánea* o SNAP) o cuando se cumple una afirmación distribuida (un mensaje PDA). Estos menajes pueden ser enviados por la misma red que conecta los nodos de la WSN, o por una vía diferente. Para ejecutar el envío de estos mensajes, el código de la aplicación del nodo pasa por un pre-procesador –de forma similar a EnviroLog– que agrega y optimiza las funciones propias de PDA.



Fuente: (Romer & Ma 2009)

Figura 3. Arquitectura de PDA.

Para recolectar los mensajes SNAP y PDA, los autores prevén varias alternativas. Sin embargo, en su implementación utilizan una red diferente a la del sistema monitorizado, donde varios sniffers capturan los mensajes, agregándoles una marca de tiempo. La traza de mensajes capturados es pasada a un *Backend* (Figura 3) donde se reconstruyen los atributos de los mensajes SNAP y se evalúan las afirma-

ciones distribuidas, con el objetivo de encontrar errores en el funcionamiento de la WSN. Este *Backend* dispone de una interfaz para mostrar los resultados de esta evaluación.

A pesar de su enfoque *pasivo*, la intrusión generada en los nodos por el envío de los mensajes –aunque reducida– implica una cierta afectación al funcionamiento del nodo monitorizado, por lo que se la considera una herramienta de monitorización activa. La obtención de información directamente de los nodos permite tener una visión más precisa del comportamiento de la WSN. Sin embargo, la precisión de PDA depende de que las afirmaciones distribuidas estén ubicadas adecuadamente, y que su número no sea excesivo, para evitar mucha sobrecarga en los nodos observados.

2.1.6. Lightweight tracing

De forma similar a Memento, *Lightweight tracing* (Sundaram et al. 2009) genera un registro de eventos utilizando una codificación muy ligera para los mismos, pero se diferencia en que guarda estos datos en memoria no volátil para su posterior reconstrucción y depuración del comportamiento del nodo y de la red. Lightweight tracing emplea una codificación muy compacta –3 bytes por evento– y un control de flujo de trazas que se adapta a las restricciones de los nodos.

Para cada evento que se quiera registrar, se coloca en el código de la aplicación "trampas" (*traps*) de software que permiten crear el registro. Varios eventos pueden ser comprimidos para que las trazas ocupen menos espacio, mediante un algoritmo de compresión en tres niveles. Cuando el nodo falla se envía la traza a la estación base para su análisis. El administrador del sistema también puede solicitar a un nodo el envío de su traza. Mediante la reproducción de la traza en un simulador o un depurador se pueden encontrar los errores.

El requerimiento de memoria –de acuerdo a las pruebas realizadas por sus autores en nodos con TinyOS– es de 315 bytes de memoria RAM. El ahorro de espacio en la memoria no volátil donde se guardan las trazas está entre el 72% y el 92% en estas mismas pruebas. Respecto al tamaño de código de programa adicional, este depende de los eventos que se quieran registrar.

Lightweight tracing ofrece un esquema ligero de generación y reproducción de trazas para encontrar errores. Sin embargo, de la forma en la que está concebido originalmente no permite hacer un diagnóstico global de toda la red, sino únicamente de errores a nivel de nodos. No se suele enviar la traza de todo el funcionamiento del nodo, desde su arranque, sino solo hasta donde ha ocurrido un error. Por tanto, no es posible la reproducción total de los eventos sucedidos en toda la red –que podría ser interesante en algunos casos– a no ser que una aplicación adicional coordine y reúna todas las trazas enviadas. Por otra parte, al estar integrado todo el

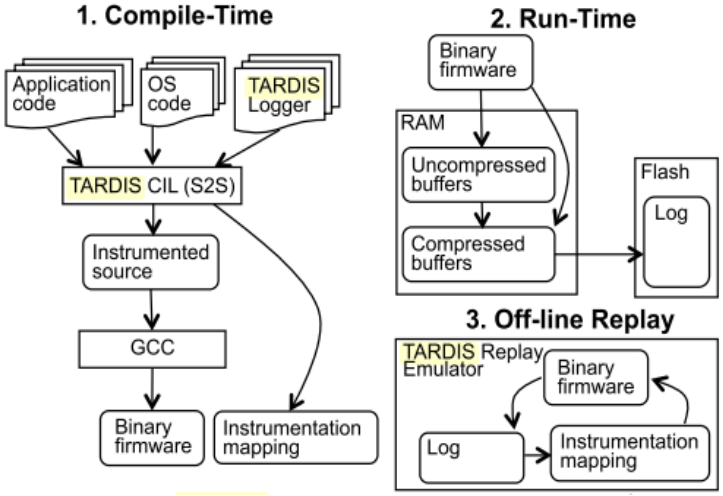
protocolo en el nodo –incluyendo el almacenamiento de las trazas– se corre el riesgo de perder la información de monitorización en caso de un fallo de hardware.

2.1.7. Trace And Replay Debugging In Sensornets (TARDIS)

TARDIS permite registrar eventos –tanto determinísticos como no determinísticos– en los nodos de una WSN, con un enfoque basado solo en software (Tancreti et al. 2015). Está diseñado para ser usado en nodos de WSN ya desplegadas, registrando eventos de varios tipos de fuentes.

La Figura 4 muestra el esquema de TARDIS. En primer lugar, se insertan las instrucciones para registrar los eventos requeridos en el código de los nodos, que a continuación es compilado y grabado. Posteriormente, en tiempo de ejecución, los eventos registrados se codifican, comprimen, y guardan en memoria no volátil. Por último, se reúnen todos los registros de los nodos para reproducir la secuencia de eventos registrados mediante un emulador –instalado en un computador de escritorio– que tiene una imagen del código binario del nodo.

TARDIS es similar a EnviroLog, aunque sus autores señalan diferencias: que EnviroLog no puede registrar (y por lo tanto reproducir) todas los tipos de eventos o condiciones de los nodos; y que TARDIS utiliza los contadores de programa (PC) y de ciclo, en lugar de marcas de tiempo, junto con el registro de un evento.



Fuente: (Tancreti et al. 2015)

Figura 4. Esquema del funcionamiento de TARDIS.

El enfoque de TARDIS le permite registrar mayor tipo de eventos y estados de un nodo sensor. Sin embargo, también causa una mayor intrusión a la red monitorizada. En las pruebas realizadas por sus autores el incremento de código de la

aplicación en el nodo fue de hasta un 25%, y de uso de CPU hasta cerca del 50%, mientras que en memoria RAM ocupó 2.6KB.

2.1.8. Otros sistemas de monitorización Activos

Existen otros sistemas de monitorización activos similares a los ya descritos, aunque con ciertas diferencias. Por ejemplo, NodeMD (Krunic et al. 2007) es un sistema en donde un nodo busca errores –mediante un código agregado en el mismo, similar a EnviroLog o PDA– e informa de ellos enviando una traza que almacena la secuencia de eventos que llevaron al error. Hecho esto, el nodo se pone automáticamente en cuarentena, entrando en un modo interactivo para permitir que el administrador de red determine las causas del error.

MARWIS (Wagenknecht et al. 2008) es un sistema con una arquitectura para administrar y monitorizar WSN –de forma similar a SNMS o DSN– , pero enfocado a redes de sensores heterogéneas. Para esto emplea una *red en malla* inalámbrica paralela, y divide a las WSN en sub-redes de sensores, de acuerdo a las características comunes de los nodos.

FAMoS (*Flexible Active Monitoring Service for WSN*) (Maerien et al. 2012) es un servicio flexible de monitorización activa para recopilar datos sobre el volumen y distribución de tráfico de una WSN. Este servicio se proporciona con una sobrecarga limitada y es aplicable en muchos contextos. Cada nodo recoge localmente datos sobre el tráfico de la red y luego transmite periódicamente –mediante la misma red– los datos a una aplicación *backend* para su posterior análisis y procesamiento.

2.2. Sistemas y Plataformas Pasivas

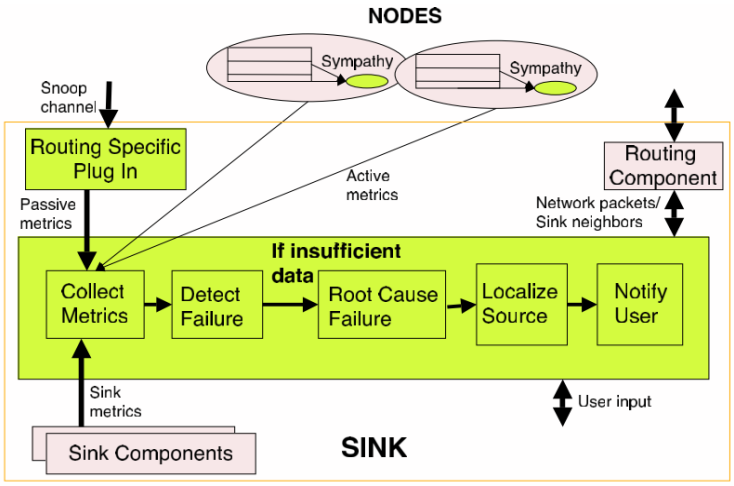
Los Monitores pasivos se basan únicamente en la observación de la información que envían los nodos de la WSN en su funcionamiento habitual, y por tanto sin interferir de ninguna manera en los mismos. En las propuestas se identifican dos enfoques diferentes: algunas de ellas trabajan directamente con la información que llega al sumidero, mientras que otras despliegan una red de *sniffers* que capturen las transmisiones en el aire. A continuación se analizan las contribuciones más relevantes de las encontradas durante esta investigación, de acuerdo al orden de publicación.

2.2.1. Sympathy

Sympathy (Ramanathan et al. 2005) actúa como un monitor pasivo que puede detectar errores en etapas de pre y post despliegue de una WSN mediante un análisis del tráfico que llega al sumidero de la red.

Su funcionamiento se basa en el principio de que los nodos transmiten un flujo predictible de datos utilizando mensajes periódicos. Cualquier cambio en este flujo es señal de un comportamiento anormal. Se puede establecer la causa principal de un problema en base a parámetros preestablecidos del comportamiento esperado de la red. Adicionalmente, Sympathy incluye transmisión de ciertas métricas por parte de los nodos monitorizados, con el fin de hacer más completo el análisis de los datos que llegan al sumidero. El funcionamiento básico de Sympathy en cada nodo consume aproximadamente 50 bytes de memoria RAM. En pruebas hechas por sus autores en nodos con TinyOS, Sympathy ocupó 47 bytes de RAM y 1558 bytes de ROM.

En la Figura 5 se muestran los componentes de Sympathy y su interacción. El código del sistema en los nodos solo transmite las métricas; toda la detección de errores se realiza en el sumidero. Cuando el sistema detecta que hay menos datos de los que se esperaba se detecta el error, y se determina la causa principal del error –por ejemplo, el bloqueo o reinicio de un nodo, no hay ruta disponible, ruta equivocada, entre otros– y el origen de la misma –problema en el nodo, en la ruta del nodo al sumidero, o en el sumidero–, y se notifica al usuario.



Fuente: (Ramanathan et al. 2005)

Figura 5. Resumen del esquema del sistema de Sympathy.

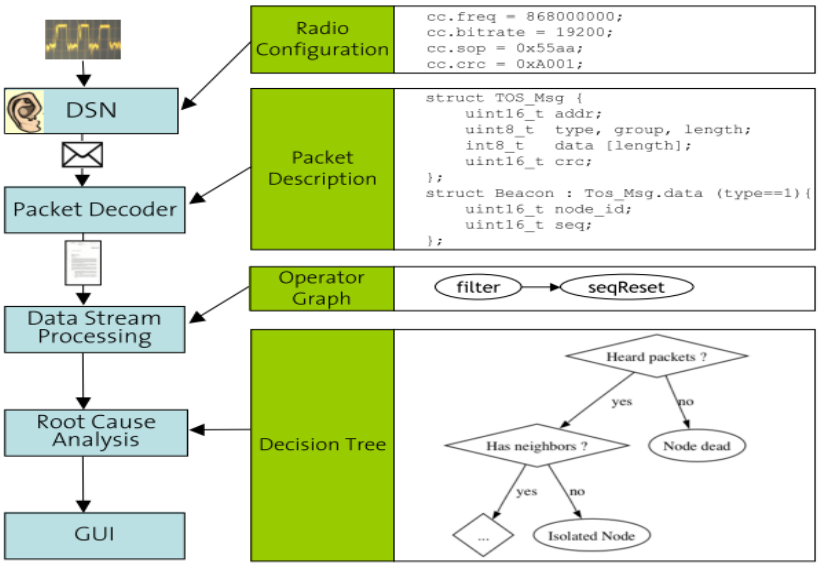
Sympathy ofrece un mecanismo sencillo para determinar la causa principal y origen de errores en una WSN, aunque requiere un conocimiento profundo del funcionamiento de la red a monitorizar. Sin embargo, no se desarrolló para detectar errores basados en eventos, perdiendo información importante y pudiendo reducir su precisión. Además, el envío de las métricas del sistema incrementa la sobrecarga en la red, aunque sea de forma leve.

2.2.2. Sensor Network Inspection Framework (SNIF)

SNIF (Ringwald et al. 2006) consiste en una red de *sniffers* que capturan las transmisiones de la WSN, para su análisis y depuración. Cada *sniffer* tiene dos interfaces de radio: una para monitorizar el tráfico; y otra para enviar la información capturada a un servidor, que en la implementación realizada por sus autores es de tipo Bluetooth.

Además de la red de *sniffers*, SNIF integra un marco de trabajo (*framework*) de detección de errores. De acuerdo a sus autores puede detectar problemas en los nodos, problemas de enlaces, problemas de rutas, y problemas globales en la red. Para esto se utilizan indicadores específicos para cada tipo de posible error, y se toman en cuenta los *flujos de datos* y los *registros* creados.

La arquitectura de los componentes de SNIF es mostrada en la Figura 6, y está compuesta de cuatro bloques principales. El primero es la Red de Asistencia de Despliegue *(Deployment Support Network* DSN), que consiste justamente en la red de *sniffers*. Estos deben estar sincronizados adecuadamente, ya que le agregan una marca de tiempo a los paquetes capturados (en el orden de milisegundos según sus autores.)



Fuente: (Ringwald & K Romer 2007)

Figura 6. Arquitectura de SNIF.

El segundo –el Decodificador de Paquetes– utiliza una estructura y configuración adecuada a la red a monitorizar, para mantener una descripción de los campos de los paquetes capturados. El Procesador de Flujo de Datos ejecuta operaciones

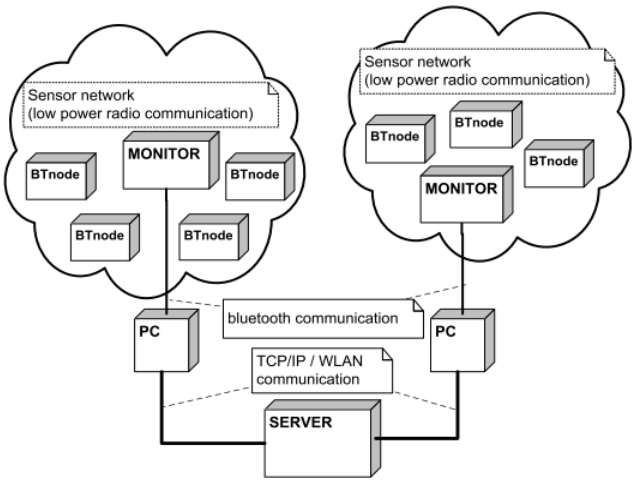
que toman como entrada un flujo de registros, y producen como resultado un flujo diferente. El Análisis de Causa Principal utiliza un árbol de decisión para establecer el estado de los nodos y detectar los problemas. Así mismo, SNIF provee una interfaz gráfica de usuario (GUI) para mostrar el estado de los nodos así como información adicional (Ringwald & K Romer 2007).

La mayor ventaja de SNIF, así como de otras propuestas similares, es que no producen ningún tipo de intrusión o interferencia en la red monitorizada, pues no hay que hacer ningún cambio en los nodos de la WSN. Sin embargo, tiene la desventaja de no poder detectar problemas que no influyan en las comunicaciones de los nodos. Además, el tener que desplegar otra red se podría ver como un inconveniente, aunque su reutilización en varias redes puede compensar el costo inicial.

2.2.3. Pimoto

Pimoto es una plataforma distribuida de monitorización pasiva para analizar y depurar WSN (Awad et al. 2008). De acuerdo a sus autores tiene una estructura jerárquica que le permite monitorizar diferentes redes de forma simultánea.

La Figura 7 muestra la estructura y los elementos de Pimoto. En la parte superior se puede ver que los nodos de monitorización –o monitores– se ubican dentro del rango de una WSN. A esto se le denomina *isla de monitorización*. Los nodos de monitorización tienen dos interfaces: Una de ellas implementa la misma tecnología que la WSN, y sirve para capturar las transmisiones de la red de sensores; la otra, de tipo Bluetooth, tiene como misión entregar la información capturada al siguiente nivel, un *PC Gateway*.



Fuente: (Awad et al. 2008)

Figura 7. Estructura jerárquica de Pimoto.

El PC Gateway es el segundo componente del sistema mostrado en la Figura 7. Se encarga de reenviar los datos capturados por los nodos de monitorización hacia el *Servidor central*, vía comunicación por TCP/IP. En el Servidor central se reúne toda la información, y se la visualiza mediante la aplicación Wireshark (Wireshark n.d.), con un *plug-in* adecuado al caso.

Al igual que otras redes de *sniffers*, Pimoto no causa ninguna interferencia en el funcionamiento de la red, pero también adolece de la dependencia solo de las transmisiones de los nodos para inferir problemas. Aunque cumple su función específica, el agregar un PC entre cada nodo de monitorización y el servidor, incrementa el costo del sistema, cuando estas funciones podrían incorporarse en uno de los otros dos componentes de Pimoto. La utilización de Wireshark resulta práctica, ya que es una herramienta ampliamente probada. Sin embargo, en el ejemplo mostrado por sus autores, solo muestra los campos de información generada por el nodo de monitorización, mas no el tipo de paquete capturado, por lo que haría falta otra herramienta adicional para este tipo de análisis.

2.2.4. LiveNet

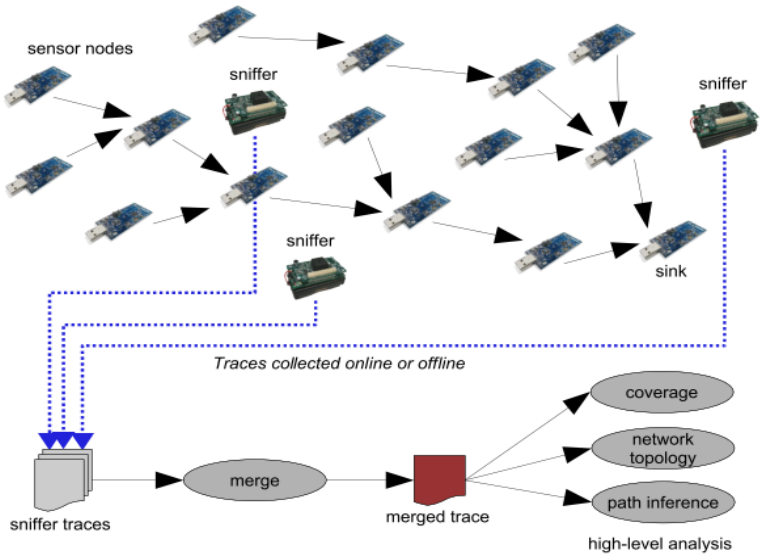
Sus autores definen a LiveNet como un conjunto de herramientas y técnicas para reconstruir el comportamiento de una red de sensores desplegada, basada en el uso de una infraestructura distribuida de *sniffers* que capturan las transmisiones de la WSN monitorizada (Chen et al. 2008). La captura se realiza en tiempo real, mientras que el procesamiento y reconstrucción se hace fuera de línea (*offline)*, de forma posterior a la captura.

La arquitectura de LiveNet, ilustrada en la Figura 8, consta de tres componentes. El primero es la red de *sniffers* desplegada junto a la WSN a monitorizar. Cada *sniffer* captura los paquetes transmitidos, y los empaqueta –junto con el ID del *sniffer* y la marca de tiempo cuando fue capturado– para pasarlos a un host, a través de un puerto serie. En la implementación hecha por sus autores este host es un PC, aunque indican que pueden utilizarse otras opciones

El segundo componente es el Proceso de Unión de las trazas generadas por los *sniffers*, para crear una traza única y global. Este componente tiene tres partes o etapas: El objetivo de la primera es sincronizar o corregir las marcas de tiempo de las trazas; en la segunda se van fusionando progresivamente las trazas; y finalmente en la tercera se gestionan las transmisiones duplicadas.

El último componente de LiveNet es el Análisis de la traza combinada. Para reconstruir el comportamiento de la red en base a la traza global, LiveNet utiliza varios algoritmos que determinan, por ejemplo, la tasa de tráfico, puntos de congestión, conectividad de la red, o la determinación del enrutamiento.

LiveNet puede ser utilizado tanto temporalmente para depuración, como de forma permanente, dada su nula intrusión en la red observada. Sin embargo, sigue dependiendo únicamente de los eventos de transmisión para realizar su análisis.



Fuente: (Chen et al. 2008)

Figura 8. Arquitectura de LiveNet.

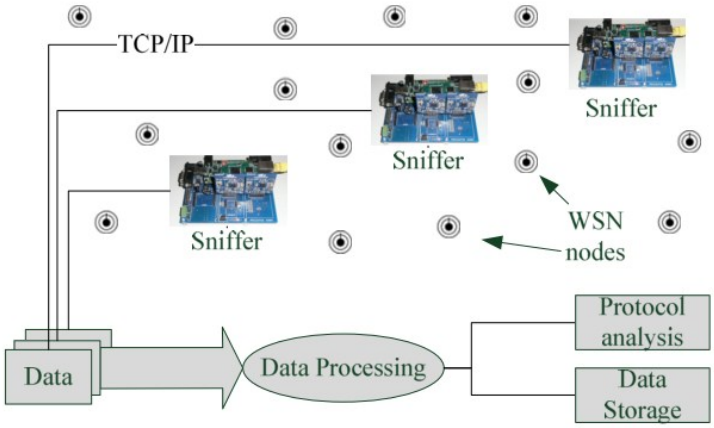
2.2.5. Sensor Network Distributed Sniffer (SNDS)

SNDS funciona de manera similar a SNIF, Pimoto o LiveNet –una red de *sniffers* desplegada junto a la WSN para capturar los mensajes en el aire–, pero utiliza conexiones Ethernet en lugar de Bluetooth para transmitir las capturas a un servidor central (Kuang & Shen 2010). SNDS está enfocado en realizar tanto la captura como el análisis de datos en tiempo real.

SNDS tiene dos componentes (Figura 9). El primero es la red de *sniffers* que monitorizan la WSN. Los *sniffers* funcionan de forma similar a los de otras propuestas, con la diferencia de que tienen una interfaz Ethernet para enviar los datos que capturan al dispositivo que los procesa.

Para transmitir los datos los *sniffers* no necesitan tener configurado previamente los parámetros de direccionamiento del protocolo IP, sino que utilizan UDP para buscar el host donde reside el programa servidor, y TCP para establecer una conexión con el mismo con el fin de transmitir datos. Así mismo, aprovecha las posibilidades de Ethernet y TCP/IP para realizar la sincronización de los *sniffers*, como se explicará más adelante.

El segundo componente –el Programa Servidor– consta de varios módulos que permiten recibir los datos y procesarlos, configurar los *sniffers* y mostrar información, entre otras tareas. En la implementación de SNDS, este programa se ejecuta en un PC.



Fuente: (Kuang & Shen 2010)

Figura 9. Esquema de la estructura de SNDS

SNDS no provoca interferencia con el funcionamiento de la red monitorizada, y su funcionamiento en tiempo real permite descubrir problemas de forma inmediata. Aunque sigue padeciendo de la limitación de otros sistemas similares, al depender solo de las transmisiones de la WSN.

2.2.6. Passive Diagnosis for WSN (PAD)

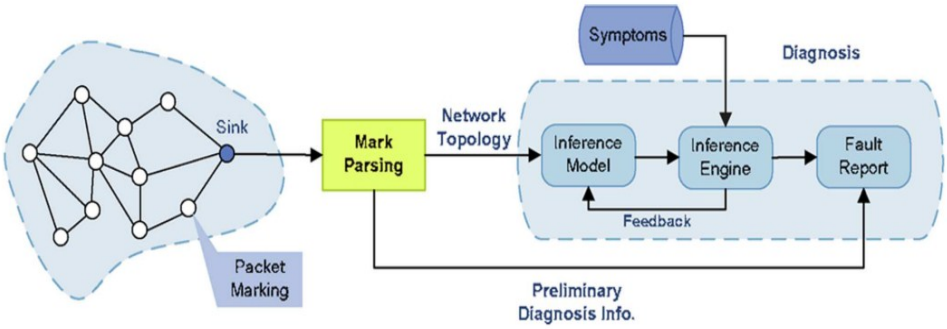
El Diagnóstico Pasivo para WSN (PAD) es un sistema de monitorización en línea con un enfoque de diagnóstico probabilístico para inferir en la causa de los errores en una WSN (Liu et al. 2010). A pesar de que PAD se presenta como un monitor pasivo, genera una intrusión mínima en los nodos monitorizados, de forma similar a Sympathy.

PAD no genera tráfico adicional, sino que agrega un código *sonda* en cada nodo que marca los paquetes con datos relevantes con muy poca sobrecarga (apenas 2 bytes), aunque no se indica la cantidad de memoria adicional requerida en el nodo para generar esta sonda. Este marcado permite revelar de forma dinámica las dependencias internas en la WSN, y permite descubrir errores en la red.

No obstante, la información de marcado es insuficiente, por lo que PAD emplea un modelo probabilístico para detectar *síntomas* que pueden ser "buenos" –

funcionamiento correcto– o "malos" –mal funcionamiento–, para lo que se crea un modelo de inferencia jerárquico, una *red de creencias* (red Bayesiana).

La Figura 10 muestra el funcionamiento de PAD de forma resumida. El *mark parsing* se ubica en el sumidero, y se encarga de extraer y analizar las marcas recibidas en los paquetes –así como de la topología de los nodos en la red– para realizar un diagnóstico preliminar. El módulo de diagnóstico permite inferir las posibles causas de los errores y emitir reportes de los mismos. Este módulo se retroalimenta para mejorar la detección de errores.



Fuente: (Liu et al. 2010)

Figura 10. Resumen de funcionamiento del sistema PAD.

La intrusión de PAD es mínima en la WSN, y no causa mayor sobrecarga en la red. Sin embargo, PAD requiere la transmisión de un mensaje de aplicación para enviar información de forma anexa, y puede que no sea posible determinar cuándo ha ocurrido un error. Si bien constituye un muy buen ejemplo de la utilidad de la inteligencia artificial en la detección de errores, aparecen falsos positivos en las pruebas realizadas por sus autores, aunque con una tasa baja.

2.2.7. Network Monitoring and Packet Sniffing Tool for WSN (NSSN)

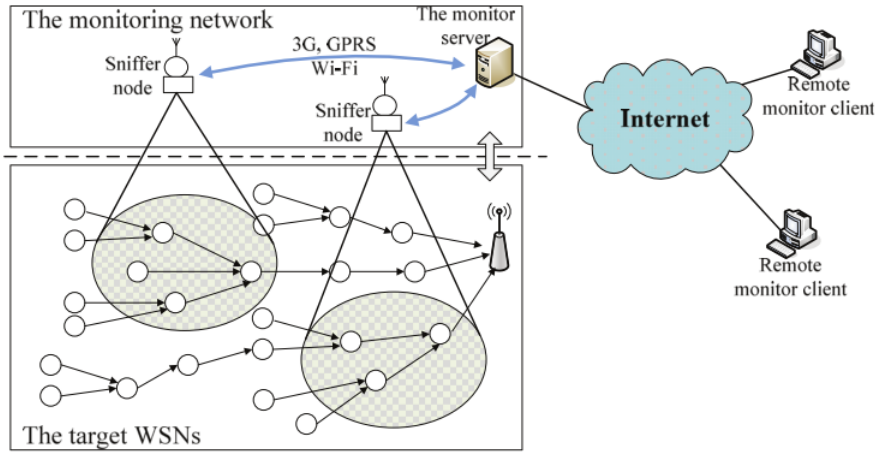
NSSN es una herramienta para capturar paquetes y monitorizar WSN en tiempo real (Zhao et al. 2012). La estructura de NSSN es similar a la de propuestas anteriores –una red de *sniffers* que capturan paquetes de la WSN monitorizada y envían la información a un servidor– pero con ciertas diferencias.

La primera es que los *sniffers* pueden detectar automáticamente el canal de transmisión de WSN objetivo, aunque este parámetro también puede ser configurado manualmente. Estos nodos también soportan varias frecuencias de trabajo para la captura de tramas.

Otra diferencia es que los datos obtenidos por los *sniffers* se pueden enviar a través de 3G inalámbrico, GPRS, o Wi-Fi –o cualquier tecnología que soporte

TCP/IP–; aunque en la implementación sus autores emplean Ethernet. Los datos recopilados se envían a un Servidor de monitorización que analiza, procesa y almacena la información obtenida en una base de datos.

La Figura 11 muestra la arquitectura de NSSN. En esta figura se aprecia en esquema los elementos ya explicados de la plataforma. Los clientes de NSSN tienen un software que permite acceder y visualizar o analizar la información almacenada en el servidor. La comunicación entre los clientes y el servidor se puede realizar a través de Internet.



Fuente: (Zhao et al. 2012)

Figura 11. Arquitectura de NSSN

Aunque la opción de múltiples canales de radio de los *sniffers* en NSSN es una característica interesante, esto no evita los posibles problemas de diferencias en circuitos y señales de radiofrecuencia, tal como sus autores reconocen. Además, sigue dependiendo de la transmisión de mensajes para poder monitorizar la red.

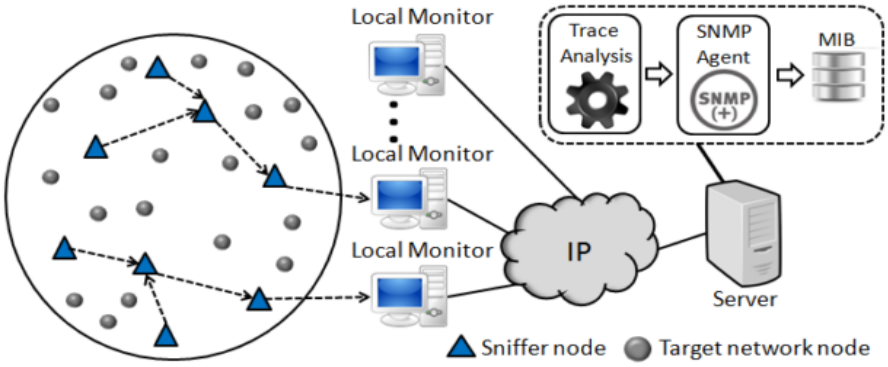
2.2.8. Energy-efficient Passive MOnitoring SysTem for WSN (EPMOSt)

EPMOSt es un sistema de monitorización pasivo basado en una red de *sniffers*, especialmente centrado en la reducción del consumo de energía de la red de monitorización (Garcia et al. 2014). Con este objetivo, EPMOSt selecciona los nodos cuyos paquetes serán capturados por cada *sniffer*, con lo que se consigue un menor consumo de energía al evitar la duplicidad de paquetes capturados.

La Figura 12 muestra la estructura de EPMOSt. Entre los *sniffers* y el Servidor principal se encuentran dispositivos denominados Monitores Locales, similares a los Gateways en Pimoto. Un Monitor Local recibe los mensajes capturados por varios *sniffers*, e ingresa la información en un archivo de traza localizado en el

Servidor. Este Servidor analiza la traza generada por uno o más monitores locales, y la almacena en una Base de Información de Administración (MIB), donde puede ser accedida. Tanto para transmitir información entre sus componentes como para ejecutar ciertos procedimientos, EPMOSt utiliza un agente SNMP (*Simple Network Management Protocol*).

Al iniciarse, EPMOSt realiza la selección de los nodos que cada *sniffer* observará, ejecutando el mecanismo entre los *sniffers* y el monitor local. La comunicación entre los monitores locales y el servidor es vía TCP/IP, aunque no se explica cómo se comunican los *sniffers* con el monitor local. El servidor de EMPOSt provee de mecanismos para analizar las trazas obtenidas.



Fuente: (Garcia et al. 2014)

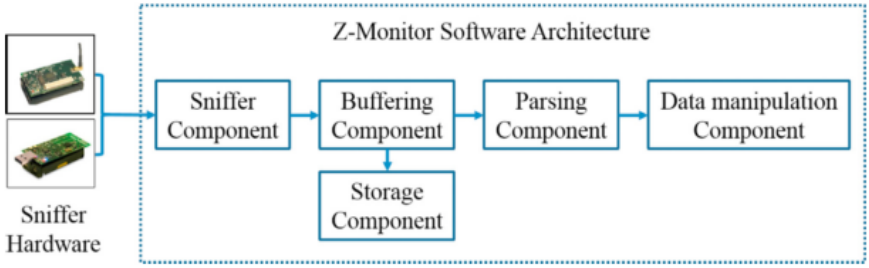
Figura 12. Estructura de EPMOSt.

Aunque en las pruebas realizadas por sus autores EPMOSt consigue el objetivo de reducir el consumo energético entre un 69% y 78%, esta ventaja resulta relevante únicamente cuando los *sniffers* tienen restricciones de alimentación en relación al tiempo que van a estar funcionando. Además, el mecanismo de selección empleado hace que la cantidad de paquetes capturados sea ligeramente menor que si no se aplica este mecanismo. Este fenómeno acentúa el inconveniente de depender solo de los mensajes transmitidos por la WSN, y de obtener una visión incompleta del comportamiento de la red.

2.2.9. Z-Monitor

Z-Monitor es una plataforma abierta de bajo costo de análisis para WSN con tecnologías estándar (Tennina et al. 2015). Una red distribuida de *sniffers* captura las transmisiones de la red monitorizada, y envían esa información a un servidor que dispone de una interfaz gráfica (GUI) amigable para visualizar los datos.

En la Figura 13 se muestra un diagrama de bloque de la arquitectura de Z-Monitor. La parte del de hardware la componen básicamente los *sniffers*, mientras que a nivel de software se definen varios módulos, que pueden estar en el mismo *sniffer* o en el servidor, denominado Z-Server.



Fuente: (Tennina et al. 2015)

Figura 13. Arquitectura de Z-Monitor.

El hardware de los *sniffers* es similar a los de otras propuestas, aunque puede funcionar de dos formas. La primera es en modo *independiente*, en la que el componente *sniffer* se conecta directamente a un computador mediante una interfaz USB, y transfiere los datos capturados directamente a la aplicación de Z-Monitor. La segunda –el modo *distribuido*– es la forma tradicional en que varios *sniffers* capturan mensajes y los pasan a un servidor. La conexión entre los *sniffers* y el Z-Server puede ser vía LAN inalámbrica o Ethernet. La aplicación cliente de Z-Monitor permite acceder a los datos guardada en el Z-Server –visualizando información de la captura–, así como configurar los parámetros de los *sniffers*. La GUI es además multiplataforma, ya que está desarrollada en el lenguaje Java.

Si bien Z-Monitor ofrece características interesantes –como la posibilidad de personalización o adaptación, dado que es de código abierto–, sigue dependiendo únicamente de los mensajes transmitidos por la WSN para su funcionamiento.

2.2.10. Otros sistemas de monitorización Pasivos.

Existen otras plataformas de monitorización pasivas similares a las ya mencionadas, aunque con algunas diferencias. En DIMO (Meier et al. 2008), los monitores –distribuidos a través de la WSN y conectados por una red secundaria– controlan la transmisión periódica de señales de actividad (*heartbeats*) desde los nodos. Solo se envía información al sumidero si un nodo potencialmente ha fallado, lo que se determina mediante un algoritmo desarrollado para el caso.

FlashBox (Choudhuri 2009) es un sistema basado en almacenamiento en memoria Flash –no integrada en el nodo, tal como en el caso de EnviroLog o TARDIS– para registrar eventos no deterministas en sistemas embebidos que pueden

ayudar en la investigación de errores y repetición de eventos. Si bien sus autores lo consideran un sistema de monitorización mínimamente invasivo, reportan una sobrecarga de entre 10 y 23%. Además, se podría considerar un sistema híbrido ya que implica hardware y software.

2.3. Sistemas y Plataformas Híbridas y Testbeds

Se consideran sistemas o plataformas híbridas de monitorización a aquellos que combinan al menos dos enfoques en su funcionamiento. Tal como se explicó antes, a pesar de que en monitorización se considera tradicionalmente como híbrido al sistema que combina hardware y software, en la actualidad se utiliza este término en varios casos donde se combinan dos enfoques de funcionamiento (Navia, Bonastre, et al. 2015). A pesar de haber hecho una búsqueda intensa de este tipo de sistemas, se han encontrado muy pocas propuestas en esta línea. Por otro lado, los *Testbeds* (bancos de pruebas) son plataformas que permiten evaluar el funcionamiento de una WSN bajo condiciones controladas, y que, en algunos casos, podrían trabajar tanto en un modo pasivo como en activo, de acuerdo a los requisitos de la campaña de monitorización.

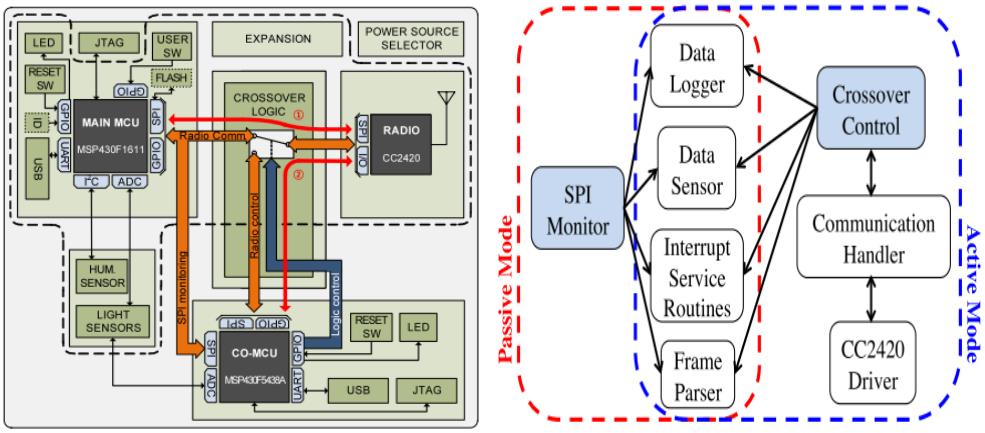
A continuación se detallan las plataformas híbridas de monitorización más relevantes, así como algunos de los *testbeds* más referenciados en la literatura.

2.3.1. Spi-Snooper

Spi-Snooper es una plataforma híbrida de monitorización de WSN con un enfoque hardware-software, aunque sus autores no utilizan el término híbrido (Hossain et al. 2012). La arquitectura de hardware une al nodo sensor y el monitor, denominado coprocesador, en una misma pieza de forma transparente. La Figura 14 muestra la arquitectura de Spi-Snooper tanto a nivel de hardware como de software.

El diagrama de la arquitectura de esta plataforma se muestra en la Figura 14a. El monitor de Spi-Snooper "espía" la interfaz SPI (*Serial-Peripheral Interface*) que utiliza el microcontrolador del nodo para comunicarse con el componente de radio. Además del hardware propio del nodo, existe un controlador de lógica de cruce, que permite conmutar las señales SPI.

Además del combinar hardware y software, Spi-Snooper tiene un enfoque híbrido desde el punto de vista de su forma de operación, al funcionar de modo activo y pasivo (Figura 14b). En modo pasivo, el monitor registra principalmente la comunicación a través del bus SPI y comprueba los datos del nodo. En modo activo asume el control de SPI y la interfaz de radio, ya sea para transmitir algún tipo de información o para funcionar como nodo enrutador.



Fuente: (Tennina et al. 2015)

Figura 14. (a) Diagrama de Bloque y (b) Arquitectura de Software de Spi-Snooper.

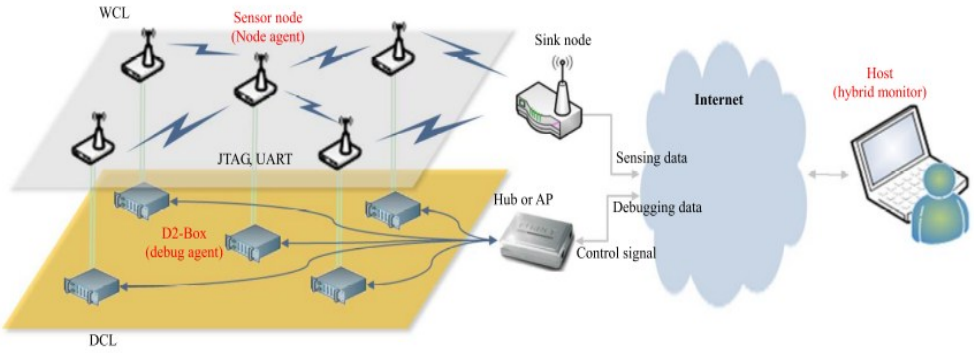
La capacidad de tomar el control para retransmitir mensajes en una WSN, cuando el nodo monitorizado ha dejado de funcionar, es una interesante y valiosa característica. Sin embargo, este enfoque sólo puede utilizarse en nodos sensores utilizan un módulo de radio interconectado con el microcontrolador por medio de un bus –como el SPI– y no aquellos que tienen integrado el módulo de radio al microcontrolador, lo que es usual hoy en día. Además, sólo los datos transmitidos o recibidos a través de esta interfaz pueden ser monitorizados, pero no otro tipo de actividad o eventos en el nodo.

2.3.2. Hybrid Debugging Framework for distributed network environments (HDF)

Según sus autores, HDF combina un enfoque activo –la ejecución de órdenes y consultas del estado de un nodo monitorizado– y uno pasivo –la escucha de información por parte de un dispositivo unido al nodo de la WSN–, para efectuar la depuración de una WSN en tiempo real (Kim et al. 2017). Entre sus funcionalidades están la depuración basada en consultas (*queries*), trazado en tiempo real, monitorización de procesos y memoria, y visualización.

La Figura 15 muestra el funcionamiento y componentes de esta plataforma. HDF agrega una red de nodos –denominados *D2-Box*– que están conectados por medio de dos tipos de interfaces a los nodos desplegados: una tipo serie UART (*Universal Asynchronous Receiver/Transmitter*) para monitorización, y una JTAG (*Joint Test Action Group*, norma IEEE 1149.1) para depuración y control. Los D2-Box se conectan vía Ethernet, para enviar los datos a un computador –denominado

Monitor Híbrido– que permite la visualización de los datos recolectados por los D2-Box, y enviar órdenes a los nodos de la WSN.



Fuente: (Kim et al. 2017)

Figura 15. Diagrama de funcionamiento de HDF.

El monitor híbrido de HDF permite también visualizar la información recolectada por el sumidero de la WSN. Al combinar esta información con la de la depuración se puede obtener una visión más completa del funcionamiento de la red monitorizada.

Esta plataforma puede ofrecer un profundo conocimiento acerca del funcionamiento de una WSN. Sus autores afirman que no se añade sobrecarga a la red original. Sin embargo, la intrusión causada en los nodos monitorizados es considerable, debido al componente de software que se ejecuta en cada nodo de la WSN. Aunque sus autores han implementado un prototipo del D2-Box, los beneficios de la plataforma han sido estimados solo de forma teórica.

2.3.3. Testbeds

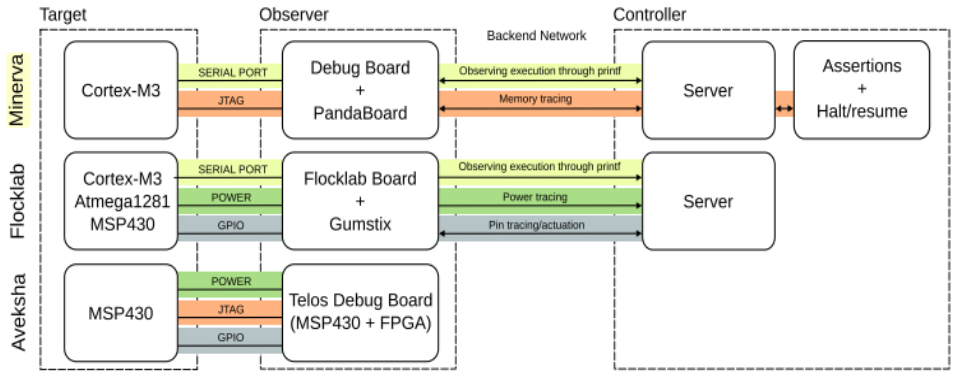
Además de monitorizar el funcionamiento de los nodos de una WSN, los *Testbeds* pueden permitir la configuración de los mismos, o incluso solicitar información sobre el estado del nodo. Una característica común en estas plataformas es que los nodos de monitorización suelen disponer de un sistema de alimentación de energía continua.

La plataforma Aveksha proporciona seguimiento de eventos y puntos de interrupción en un nodo, conectando una placa de depuración con el nodo sensor bajo observación (Tancreti et al. 2011). Aveksha aprovecha la interfaz de depuración ya disponible en los nodos. Define tres modos de depuración –dos de los cuales no causan intrusión–, además de permitir también monitorizar el consumo de energía. Esta característica hace que las modificaciones a nivel de software en el nodo observado sean mínimas. Sin embargo, no se define un método para relacionar los

registros de eventos obtenidos en cada nodo por separado, ya que cada monitor funciona de forma local, y Aveksha no define un esquema o protocolo de interconexión de los monitores.

Minerva y FlockLab extienden el enfoque inicial de Aveksha al conectar los nodos de depuración (monitores) a un servidor. Minerva es un *testbed* que utiliza un puerto de depuración (JTAG) y uno serie conectado al nodo sensor para observar el comportamiento del mismo (Sommer & Kusy 2013). Los nodos Minerva están conectados a través de una red Ethernet. Un servidor controla la plataforma y recopila datos de los nodos. Los usuarios pueden leer y escribir la memoria del nodo del sensor en tiempo de ejecución de forma no intrusiva, y detener / reanudar la red sincrónicamente. Si bien Minerva proporciona información útil sobre el estado global de la red de sensores, su arquitectura –que busca ser de bajo costo– no está destinada a depurar el código de bajo nivel. Además, el detener y reanudar el funcionamiento de la red puede dar lugar a inconsistencias, debido a los posibles retardos en la difusión de las órdenes.

FlockLab es también un *testbed* (Lim et al. 2013), aunque con algunas diferencias respecto a Minerva. Sus componentes pueden utilizar Ethernet o conectividad inalámbrica. FlockLab no utiliza un puerto de depuración, sino más bien pines digitales GPIO (*General-Purpose Input/Output*), un puerto serie, y un puerto para monitorizar el consumo de energía. Los pines digitales se utilizan para registrar eventos y generar una traza, así como para enviar señales al nodo monitorizado, como por ejemplo para detenerlo o reiniciarlo. Los eventos se guardan en archivos, que son enviados al servidor de la plataforma. El uso de pines digitales hace que la intrusión sea mínima. Sin embargo, el número de pines disponibles limita los tipos de eventos que se puedan registrar, que son cinco (5) en la implementación del nodo observador de FlockLab.



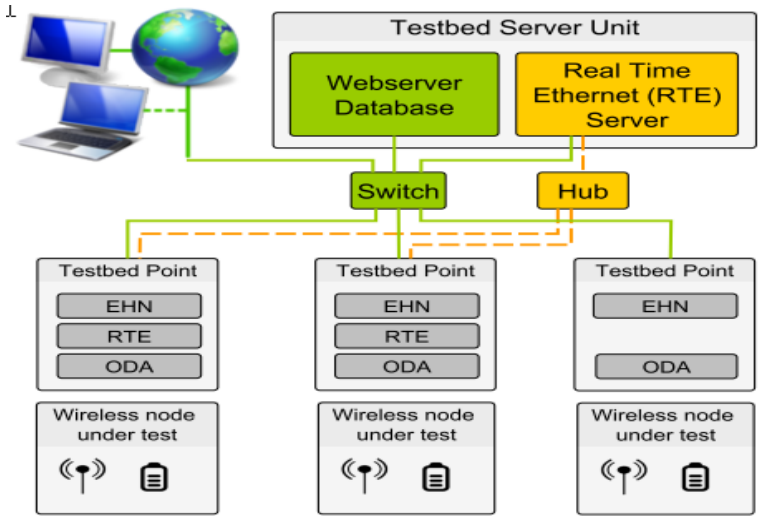
Fuente: (Sommer & Kusy 2013)

Figura 16. Esquema comparativo entre Aveksha, FlockLab y Minerva.

La Figura 16 muestra una comparación de las tres plataformas recién mencionadas. Mientras que Aveksha solo define el nodo observador de forma local, FlockLab y Minerva también incluyen un Servidor, desde el que se puede controlar los nodos observadores, y recolectar la información de monitorización obtenida por estos. FlockLab ha sido evaluado en nodos con distintos tipos de microcontroladores. Minerva por otra parte, incluye también la posibilidad de utilizar *afirmaciones* para determinar el estado global de la red

TWECIS es un *testbed* en tiempo real con enfoque industrial (Potsch et al. 2014), que combina características como generación de trazas, posibilidad de depuración, y mediciones sincronizadas y precisas de consumo energético. En la Figura 17 se muestra el esquema propuesto por TWECIS. Los nodos observadores tienen tres componentes: el nodo embebido (EHN), la adquisición de datos (ODA), y el sistema de Ethernet en tiempo-real (RTE). Este último se emplea para la sincronización, aunque sus autores lo ponen como opcional cuando no se requiere alta precisión en la observación.

La conexión de los nodos observadores hacia el servidor se realiza por Ethernet. En el Servidor del testbed se encuentran el servicio de RTE y la aplicación de base de datos y servidor web, para visualizar la información de la plataforma. A pesar de la funcionalidad de TWECIS, los costos de los nodos pueden ser un poco elevados en comparación con otras propuestas.



Fuente: (Potsch et al. 2014)

Figura 17. Estructura de TWECIS.

2.4. Sincronización de componentes de una plataforma de monitorización

Muchos de los sistemas de monitorización de WSN analizados consisten en plataformas distribuidas de nodos que adquieren información de forma individual –a manera de trazas o registros– que luego se centralizan en un servidor para analizarla. Esta información normalmente incluye el instante del tiempo (marca de tiempo o *time-stamp*) en que se ha producido el evento registrado. Los registros o trazas generados en las Plataformas Distribuidas de Monitorización (PDM) deben tener marcas de tiempo basadas en un tiempo de referencia. De lo contrario, no será posible establecer relaciones entre los eventos registrados en diferentes nodos. Es por esto que las PDM enfocadas en WSN deben implementar mecanismos de sincronización entre sus componentes para trabajar adecuadamente, aunque esto no siempre se especifica, tal como se verá más adelante.

Los mecanismos de sincronización se pueden enfocar en sincronizar los relojes de los nodos o directamente en las trazas. La sincronización de tiempo entre los nodos en una plataforma distribuida es un problema que ha sido ampliamente estudiado. Los relojes internos de los nodos de una plataforma distribuida pueden sufrir desviaciones debido a problemas de hardware o condiciones ambientales (Castillo-Secilla et al. 2017). Los mecanismos centrados en la sincronización (ajuste) de los relojes de los nodos pueden ser aplicables en este contexto. Sin embargo, esta solución no siempre es la mejor cuando se requiere sincronizar trazas, debido a que el reloj local de un nodo podría retroceder en el tiempo, cuando ha sido más rápido que otros relojes en la plataforma, introduciendo errores en las marcas de tiempo de los registros. Estos errores también pueden cambiar el orden de algunos eventos registrados localmente, derivando en conclusiones falsas después del análisis de los datos obtenidos (Becker, Rabenseifner, et al. 2008).

2.4.1. El problema de la sincronización de trazas

Cuando se monitoriza una WSN con un PDM, es necesario realizar capturas de trazas de forma paralela en cada uno de los componentes de la plataforma. Los nodos de monitorización registran la secuencia de eventos que se consideran relevantes en el sistema a observar. Cada evento se almacena con una información de marca de tiempo que refleja tanto la secuencia como el tiempo transcurrido entre los eventos. Sin una sincronización correcta, no es posible una fusión correcta de las trazas en el procesamiento posterior, porque el orden de los eventos se puede alterar, y por lo tanto el comportamiento del sistema no se puede estudiar adecuadamente.

Como ejemplo de este efecto, se considera el caso de una plataforma donde un nodo monitor registra dos eventos (*event1* y *event2*) que ocurren secuencialmente

con un tiempo entre ellos de 600µs (Figura 18). En el caso de un reloj perfecto, o cuando no es necesario comparar una traza con un tiempo de referencia (Figura 18a), un punto de sincronización llega 400µs después de *event1*, y 200µs antes del *event2*. A pesar de la cercanía de estos eventos, no hay inversión de orden de eventos. Sin embargo, el reloj interno del nodo del monitor puede acumular un desfase –por ejemplo de +700µs– desde el último punto de sincronización. Se considera ahora un evento denominado *punto de sincronización*, que consiste en la llegada de un mensaje que permite el ajuste del reloj local en base a un tiempo de referencia. La Figura 18b ilustra lo que podría suceder en este caso. *Event1* ocurre 400µs antes del punto de sincronización, y se registra con un tiempo (*te1*). Después de 400µs, llega el punto de sincronización *tr*. El nodo del monitor se da cuenta de que su tiempo debe ser de 700µs menos y disminuye su reloj interno en 700µs. Cuando ocurre *event2* –en un tiempo 200µs más tarde– la marca de tiempo de *event2* (*te2*) debe ser igual a *te1* + 600µs o *tr* + 200µs. Sin embargo, debido a que el reloj interno se ajustó 700µs hacia atrás –y si no se han tenido en cuenta otras correcciones– *te2* sería igual a *te1* - 100µs. Cuando se procesen las trazas, *event1* aparecerá después de *event2*.

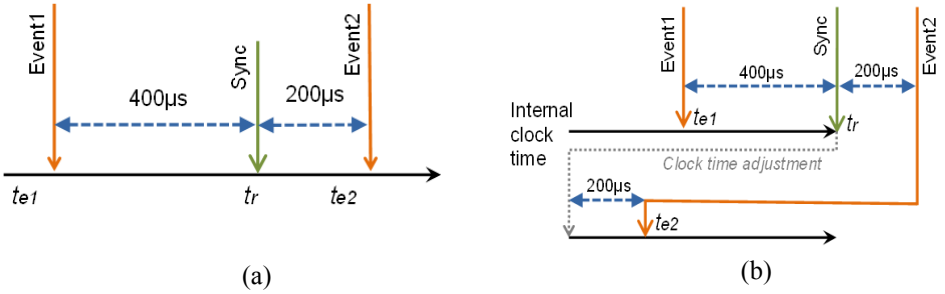


Figura 18. Registro casi concurrente de eventos y mensaje de sincronización: (a) secuencia real; (b) secuencia con ajuste de reloj hacia atrás en el tiempo.

En el caso de una PDM real, el efecto se multiplica por la existencia de numerosos nodos de monitorización. En este contexto, es común que muchos eventos relacionados se puedan observar en dos o más elementos de la plataforma. La Figura 19 muestra el esquema de una PDM con tres nodos monitores acoplados a tres nodos de una WSN. Cada monitor registra los eventos de su nodo sensor adjunto y envía esta información a un Servidor. Se pueden considerar dos escenarios: el primero supone que dos nodos registran el mismo evento, por ejemplo la llegada de un mensaje de difusión. Tanto A como B registran el evento a través de sus nodos monitor adjuntos, generando una nueva entrada en sus trazas. Una sincronización errónea entre nodos de la plataforma puede hacer que –al momento de procesar las trazas– la diferencia de marcas de tiempo de este evento en ambos nodos sea tan alta que parezca dos eventos diferentes.

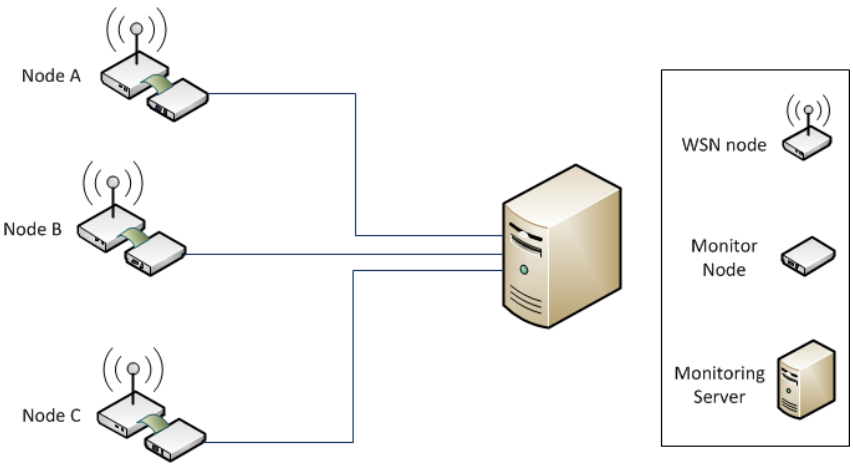


Figura 19. Esquema de una PDM para WSN. La conexión entre los nodos de monitorización y el servidor no es relevante para el caso considerado.

El segundo escenario está relacionado con la secuencia correcta de los eventos de rastreo. Por ejemplo, el nodo B recibe datos del nodo C, agrega sus propios datos y luego los reenvía al Nodo A. Si el reloj del nodo B está adelantado en comparación con el reloj del nodo C, al procesar las trazas, puede parecer que el Nodo B recibió los datos antes de que el nodo C los enviara, y por lo tanto el mecanismo de agregación no funcionaría adecuadamente. Estas incoherencias en las trazas pueden hacer que la reproducción del comportamiento de la WSN, principalmente con fines de depuración, sea inexacta.

La diferencia de velocidad de los relojes y su deriva ya se han estudiado y evaluado previamente (Navia et al. 2016). En este estudio se demostró que la velocidad del reloj no siempre es estable, incluso cuando se utiliza una fuente de reloj de alta calidad. Este problema ya ha sido mencionado antes en otros estudios (Yoon et al. 2007; Kim et al. 2012). Varios estudios han analizado antes la influencia de los factores ambientales –por ejemplo la temperatura– sobre el sesgo en las medidas de la deriva de los relojes (Engel & Koch 2015; Castillo-Secilla et al. 2017).

La Figura 20 muestra el diagrama de dispersión de las desviaciones medidas de los relojes de cuatro nodos –que tienen el mismo hardware– utilizando fuentes de reloj de baja calidad y alta calidad habitualmente utilizadas en sistemas empotrados. La deriva se calculó en intervalos de tiempo de 10s durante una hora de prueba. Como se puede ver, la deriva puede variar hasta 0.4 ppm (partes por millón) del valor más bajo al más alto para el mismo reloj con una fuente de reloj de alta calidad (Figura 20a). Esto puede causar variaciones de reloj de decenas de microsegundos en un rango de 60s. En el caso de una fuente de reloj de baja cali-

dad, la variación de la deriva alcanza casi 10000 ppm de variación (Figura 20b). Durante largos períodos de tiempo este tipo de fuente de reloj puede dar lugar a una diferencia de deriva considerable entre dos intervalos de tiempo diferentes. En este caso, las variaciones pueden llegar a 600 ms cada 60 s.

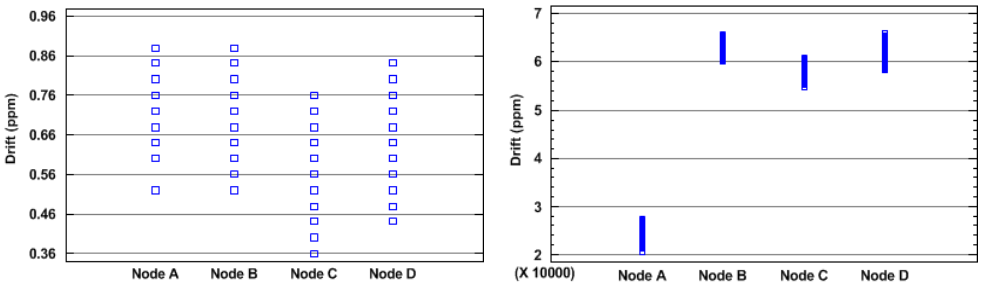


Figura 20. Diagrama de dispersión de variación de la deriva en cuatro nodos (a) con una fuente de reloj de alta calidad, (b) con una fuente de reloj de baja calidad

Todos los aspectos mencionados hacen que sea necesario un adecuado mecanismo de sincronización de los registros en las PDM para WSN, sobre todo si se considera que los nodos sensores realizan tareas en una fracción de segundo.

2.4.2. Sincronización en plataformas distribuidas de monitorización

Si bien es cierto que se han analizado plataformas de monitorización para WSN –por ser el objetivo de esta tesis–, existen PDM para otras áreas que también crean trazas de la monitorización de un sistema y que pudieran servir de referencia para otros trabajos. La sincronización de trazas en una PDM puede ser *online* u *offline*. La diferencia entre ambos enfoques de sincronización es similar a la explicada para la monitorización. En la sincronización de trazas *online* se genera la marca de tiempo ajustada o corregida mientras la traza es generada. De esta forma, se obtiene una traza sincronizada directamente desde los nodos de la plataforma.

La sincronización *offline* realiza la sincronización de la traza al final de su generación, y después de que se hayan recopilado todas las trazas. En el enfoque *offline* se aplican algoritmos y modelos matemáticos para estimar el desplazamiento y deriva de los relojes (Ashton 1995). Así, no es obligatorio sincronizar las trazas inmediatamente mientras se generan, sino que se pueden aplicar mecanismos más precisos. Además, es el enfoque más utilizado, de acuerdo a la literatura estudiada.

Existen varios algoritmos para realizar la sincronización *offline*. Estos algoritmos implican cálculos matemáticos que tienen cierto nivel de complejidad (Ashton 1995). Los algoritmos más utilizados y referenciados son los basados en la *regresión lineal* y en el método *Convex-Hull*s. Ambos se basan en la comparación

de tiempo entre dos eventos relacionados, que son por lo general el envío y recepción de un mensaje.

La regresión lineal consiste en estimar una línea en un plano de dos dimensiones, donde los puntos de tiempo pueden ser mapeados. Esta línea se puede definir de acuerdo a la Ecuación 1, donde a es la deriva, y b es la compensación inicial. Con esta línea, se puede estimar la corrección del tiempo (Jabbarifar 2013). Sin embargo, no hay garantía de que no existan inversiones en el orden de los eventos (Poirier et al. 2010).

$$y = ax + b \qquad \textbf{Ecuación 1.}$$

El enfoque *Convex-Hulls* parece ser el mejor para el manejo errores, no solo para la generación de trazas sino para cualquier medición de red. Este algoritmo consiste en asumir retrasos mínimos y máximos para la transmisión de mensajes, encuentra el área que tiene latencia mínima, e ignora los valores atípicos. La línea provista por *Convex-Hulls* es más precisa que la regresión lineal, y es capaz de evitar inversión de orden de eventos (Li Zhang et al. 2002). El algoritmo *Convex-Hulls* también se puede usar para la sincronización *online*. Sin embargo, ambos algoritmos dependen de los eventos regulares de recepción y transmisión para realizar la sincronización entre dos nodos, y la ausencia de estos eventos puede afectar la precisión del mecanismo.

Hofman & Hilgers (1998) presentan un método para estimar el tiempo global en trazas generadas en sistemas distribuidos y paralelos con comunicación bidireccional. Se basa en la causalidad de dos eventos entre dos nodos –es decir, un evento *i* en un nodo *A* causa un evento *j* en un nodo *B*–, y considera el envío y la recepción de un mensaje como estos dos eventos. Con un conjunto de marcas de tiempo de este par de eventos –y aplicando de ciertos algoritmos y procedimientos matemáticos– la deriva de reloj entre ambos nodos se puede estimar con gran precisión. Sus autores afirman que este método necesita menos cómputo que el método *Convex-Hulls*. Sin embargo, al igual que en los casos antes vistos, aún depende de los eventos de transmisión/recepción para funcionar.

Otra propuesta para evitar la inversión de orden de las marcas de tiempo –enfocada en sistemas paralelos– se basa en la corrección de una marca de tiempo posterior de un evento de *recepción*, en comparación con el evento de *envío* correspondiente, y una *amortización* de tiempo de los eventos cercanos a la recepción (Becker, Linford, et al. 2008; Becker et al. 2009). La idea básica es que la recepción de un mensaje no puede ser anterior en el tiempo respecto al envío del mismo mensaje. Cuando se recopilan trazas individuales de diferentes componentes del sistema paralelo, este mecanismo elimina las incoherencias de orden al avanzar las marcas de tiempo de recepción donde fuera necesario en la traza. Aunque corrige

las incoherencias entre las trazas y evita la inversión en el orden de los eventos, podría añadir un desfase de tiempo en algunos casos, dando una visión incorrecta del comportamiento del sistema monitorizado.

Khlifi & Grégoire (2006) proponen otras dos técnicas para la eliminación del sesgo de los relojes en la sincronización offline. El primero –denominado algoritmo *promedio*– se basa en el cálculo del retardo promedio para dos intervalos de paquetes consecutivos; del comienzo y final de la traza. La precisión de esta técnica aumenta al aumentar el tamaño de los intervalos. La segunda –técnica de *eliminación directa del sesgo*– es un algoritmo de fuerza bruta que no estima en sí el valor de sesgo, sino que tiene como objetivo eliminar su impacto en la traza, para lo cual analiza la totalidad de la misma. Sus autores afirman que estas técnicas son menos complejas que el algoritmo *Convex-Hulls*, pero no demuestran que se evite la inversión de orden de eventos, ni indican cual es la precisión obtenida.

Para mejorar la sincronización de trazas distribuidas, Poirier et al. (2010) presentan una propuesta que usa eventos a nivel de núcleo (*kernel*) de un sistema operativo. Su principal objetivo es reducir la latencia de generación de marcas de tiempo con baja intrusión. Este mecanismo utiliza un conjunto de herramientas a nivel de *kernel* –desarrollado para sistemas Linux– para marcar el tiempo de los eventos de envío y recepción de mensajes. Después de generar la traza, se aplica el algoritmo *Convex-Hulls* para la sincronización de mismas. Para las pruebas de sincronización, se utilizó el intercambio de datos con mensajes UDP y TCP, a través de conexiones Fast-Ethernet y Gigabit-Ethernet. Sus autores afirman que se garantiza la no inversión de mensajes y proporciona una buena precisión en cualquier punto de la traza. Sin embargo, este mecanismo necesita nodos con sistemas operativos que soporten la generación de trazas a nivel de *kernel*. Además, estos resultados se han obtenido en equipos de alto rendimiento, con procesadores enfocados en aplicaciones de servidor.

Jabbarifar et al.(2012) proponen un modelo *offline* para la sincronización de trazas, para usar en clústeres de computadores. Utiliza el mismo kit de herramientas que el último mecanismo explicado (Poirier et al. 2010). La arquitectura de este modelo tiene cuatro módulos que reciben entrada de uno o más módulos y envían salidas a otros módulos. La entrada de esta arquitectura consta de dos o más archivos de trazas no sincronizados, generados con el conjunto de herramientas mencionado. Esta propuesta tiene en cuenta algunos aspectos de la comunicación de clústeres para la sincronización, como el recuento de saltos, el número de paquetes intercambiados y otros. El enfoque de este mecanismo está orientado a los sistemas de clústeres, que tienen nodos de alto rendimiento y conexiones de alta velocidad; Por lo tanto, al igual que el anterior, está fuera del foco de las PDM para WSN.

Un mecanismo para detectar errores en WSN –mediante la aplicación de técnicas de minería de datos– es propuesta por Khan et al. (2014). En esta propuesta,

un dispositivo difunde su tiempo local a los nodos de monitorización de la plataforma que registra los eventos de una WSN. Los nodos almacenan este tiempo de referencia con su respectiva marca de tiempo de llegada. Posteriormente, las marcas de tiempo de los eventos registrados se pueden ajustar con esta información. El objetivo principal de este mecanismo es serializar los eventos registrados en el tiempo, evitando la sobrecarga causada por otros mecanismos de sincronización propuestos. Sus autores no explican qué algoritmo se aplica para la sincronización o ajuste de las marcas de tiempo, ni mencionan la precisión o exactitud alcanzada. No obstante, esta puede ser una técnica simple pero útil.

A pesar de los algoritmos disponibles y las propuestas de sincronización de trazas en PDM, no se ha encontrado una plataforma de monitorización –ni *online* ni *offline*– que aplique algunos de ellos, como se explicará a continuación. Por lo tanto, hay oportunidades para usarlos, especialmente para monitorización *offline*.

Además, hay que considerar que si bien una WSN es un sistema distribuido, no necesariamente tiene el mismo comportamiento que los sistemas distribuidos tradicionales –por ejemplo no todos los nodos se comunican entre sí–, por lo que la monitorización no necesariamente puede ser realizada de igual forma. Por otro lado, hay PDM para WSN que utilizan nodos tipo *sniffers*, que no transmiten sino que solo capturan mensajes, por lo que las propuestas como regresión lineal o *convex-hulls* no podrían aplicarse a ellos sin algún tipo de modificación. Además, algunas de las propuestas de sincronización encontradas tienen requisitos especiales, ya que están orientadas en sistemas de alto rendimiento, como por ejemplo grandes clústeres de computadoras.

2.4.3. Sincronización en plataformas de monitorización de WSN

De las propuestas de PDM para WSN encontradas y analizadas, algunas no describen los mecanismos de sincronización que aplican a sus componentes. Ciertas plataformas usan un protocolo estándar o conocido para sincronizar, mientras que otras proponen un mecanismo diferente y específico para llevarlo a cabo. A continuación se citan los mecanismos de sincronización más relevantes de entre los empleados por las plataformas de monitorización distribuidas para WSN.

EnviroLog utiliza una modificación del protocolo FTSP (*Flooding Time Synchronization Protocol*) (Maróti et al. 2004), en la que se deshabilita en envío periódico de mensajes para la sincronización. En la misma, la difusión de los mensajes de sincronización se realiza cuando se quiere enviar una orden a un nodo. Para evitar que la ejecución del mismo –y su registro– se den en forma no sincronizada, se envía la orden junto con el mensaje de sincronización a todos los nodos. Toda la red se sincroniza, pero solo el nodo al que se quería enviar la orden la ejecuta justo después de ajustar su reloj. Este método se utiliza sobre todo cuando

EnviroLog trabaja en redes multi-salto, ya que también permite a los nodos determinar la ruta hacia el nodo principal.

Los nodos *sniffer* de SNIF y PDA utilizan un mecanismo de sincronización desarrollado por Ringwald & Kay Romer (2007) para nodos Bluetooth. Este mecanismo explota algunas características y funciones disponibles de la conexión Bluetooth, intercambiando los desfases de los relojes de las interfaces Bluetooth, y comparándolas. Sin embargo, solo alcanza una precisión de pocos milisegundos y solo funciona en nodos Bluetooth. Aunque en el caso de PDA, sus autores afirman que cualquier otro mecanismo de sincronización puede ser utilizado.

LiveNet trata de normalizar (sincronizar) las marcas de tiempo de varios *sniffers* tomando uno de ellos como base para el tiempo de referencia. Divide las trazas en intervalos y calcula el desfase de cada intervalo, en función de eventos comunes y únicos – transmisiones de mensajes – registrados en un par de nodos. Con el desfase estimado, se corrigen las marcas de tiempo para cada intervalo. Dado que LiveNet necesita estos eventos comunes para realizar la sincronización, es posible que las trazas de algunos *sniffers* no puedan sincronizarse adecuadamente, ya que no todos pueden capturar los mismos eventos que el *sniffer* de referencia, tal como admiten sus autores.

Muchos nodos de monitorización con capacidad TCP/IP –como el PC Gateway de Pimoto, los sniffers de NSSN o de Z-Monitor, y los componentes de Minerva o de FlockLab– usan el *Network Time Protocol* (NTP) para la sincronización. NTP realiza la sincronización mediante el intercambio de mensajes con un servidor NTP, y no necesita más hardware que una interfaz de red. Sin embargo, fue diseñado para comunicaciones de Internet de área amplia, por lo que alcanza una precisión de pocos milisegundos y no garantiza que los relojes de los nodos no puedan retroceder en el tiempo (Mills 1991).

Pimoto usa un truco para sincronizar las trazas de los nodos de monitorización: estos cuentan los milisegundos desde que comienzan a funcionar –utilizando un contador de 4 bytes–, y colocan este parámetro como una marca de tiempo en los paquetes que capturan. El PC Gateway en Pimoto procesa estas marcas y los convierte en una marca de tiempo sincronizada, basada en el primer paquete enviado por el nodo. Dado que los Gateways se sincronizan mediante el uso de NTP, la precisión y exactitud de la sincronización de las trazas se limitan a las ofrecidas por este protocolo. Además, el mecanismo de Pimoto no tiene en cuenta que la velocidad del reloj puede variar de un monitor a otro.

Los *sniffers* de SNDS también disponen de TCP/IP, pero en cambio para sincronizarse usan el *Precision Time Synchronization Protocol* (PTP) (IEEE Standards Association 2008). PTP está orientado a sistemas de medición y control, y proporciona una precisión teórica de microsegundos a sub-microsegundos. Sin

embargo, para implementarlo se requiere hardware que soporte PTP, aunque se podría implementar con hardware de bajo costo, siempre que no se requiera una precisión muy alta. Además el periodo de actualización de la sincronización es de alrededor de 2 segundos (Edison 2005).

Los componentes de TWECIS aprovechan su conexión troncal y usan *Real Time Ethernet* (RTE) para sincronizar las mediciones. El uso de RTE es una muy buena alternativa para obtener una sincronización de alta precisión, pero implica hardware adicional o especial que puede ser costoso.

Algunas plataformas, como DSN, mencionan el uso de protocolos de marca de tiempo de alta precisión, pero no describen cómo funcionan. Sus autores solo mencionan que la precisión de este protocolo es mayor que el tiempo de llegada de los mensajes de difusión de órdenes, pero no dicen cómo obtienen dicha precisión. Mientras tanto, otras plataformas, como EPMOSt, no indican nada sobre la sincronización entre sus componentes, aunque afirman usar marcas de tiempo en su operación.

2.5. Conclusiones

Existen sistemas y plataformas para monitorizar o evaluar el funcionamiento de las WSN, que caben principalmente en uno de dos enfoques: activo o pasivo. Las plataformas o sistemas activos suelen ser más precisos en la obtención de información, pero generan intrusión en la red monitorizada, lo que puede generar imprecisiones en el comportamiento observado y por lo tanto resultados erróneos. Las plataformas o sistemas pasivos no generan ningún tipo de intrusión, pero no logran observar completamente el funcionamiento de los nodos de la red, ya que dependen solo de los mensajes transmitidos, lo que dificulta poder encontrar errores o anomalías en su operación.

A pesar de las oportunidades para utilizar sistemas híbridos (Schoofs et al. 2012), se han encontrado muy pocas propuestas en este sentido. Y las encontradas presentan limitaciones en su funcionamiento, como estar enfocadas en un tipo específico de nodos, o no tener una plataforma completa funcionando.

La mayor parte de las PDM para WSN realizan una sincronización en línea entre sus nodos; y solo unos pocos, como LiveNet o Pimoto, realizan una especie de sincronización fuera de línea de los datos capturados. Por lo tanto, pueden producirse imprecisiones en la sincronización debido a problemas de sincronismo.

Además, a pesar de existir mecanismos para sincronizar las trazas generadas por un sistema de monitorización que no se centran en el ajuste de relojes locales, las PDM enfocadas en WSN prácticamente no las aprovechan.

Del análisis realizado respecto a la sincronización de trazas de un PDM para WSN, se determina que deben considerarse los siguientes aspectos al diseñar un

protocolo efectivo para la sincronización global de trazas de una plataforma distribuida:

- La operación de monitorización necesita sincronizar la información obtenida –las trazas– con relación a un tiempo de referencia. Sin un orden apropiado en la traza, el análisis posterior puede dar lugar a conclusiones falsas.
- Los mecanismos centrados en la sincronización de nodos, que cambian el reloj interno pero no calculan la escala del desplazamiento del reloj o la deriva para corregirlos, deben descartarse porque pueden dar lugar a imprecisiones (Navia et al. 2018).
- En otros entornos, la sincronización de trazas se ha resuelto mediante el uso de métodos más complejos, que se basan en la regresión lineal o en el algoritmo *Convex-Hulls* (Ashton 1995). Estos mecanismos son útiles cuando la sincronización puede deducirse a partir de eventos relacionados –como la transmisión y recepción de mensajes– que están separados por una cantidad de tiempo desconocida y variable. Por otro lado, el comportamiento del tráfico de una WSN común –topología en árbol hacia un nodo sumidero– puede no producir suficientes eventos relacionados para la sincronización de todas las trazas en el sistema. Además, en el caso de nodos tipo *sniffers* para la monitorización –que no tienen eventos de transmisión– la aplicación de estas técnicas no es sencilla.

Capítulo 3

Plataforma de Monitorización Híbrida

En este capítulo se describe la propuesta de una Plataforma de Monitorización Híbrida para redes inalámbricas de sensores, denominada HMP de aquí en adelante. En primer lugar se describe la arquitectura y el esquema de funcionamiento de la misma. Luego se detalla cada uno de sus elementos, tanto en la parte de hardware como en software, haciendo especial hincapié en el problema de sincronización de las trazas generadas por la plataforma

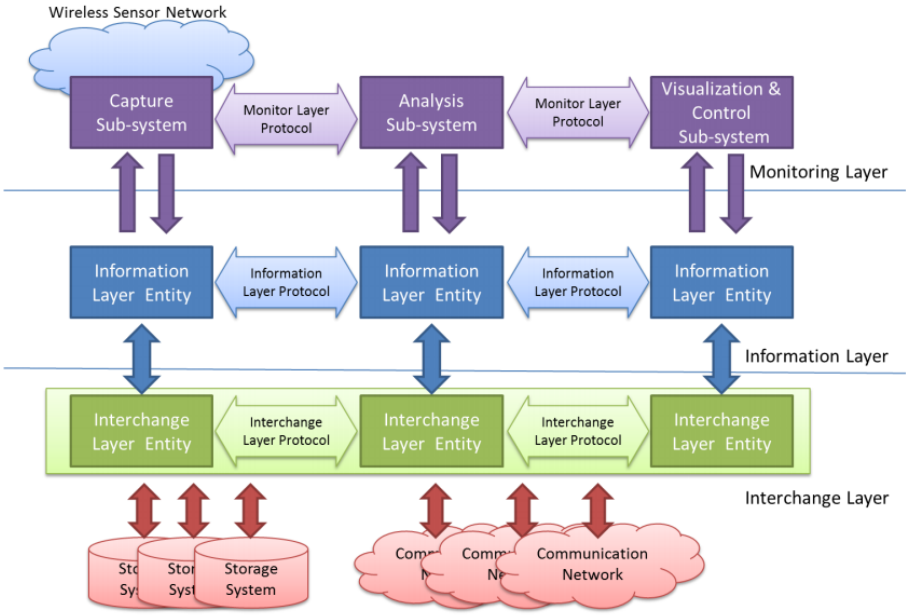
3.1. Funcionamiento general y arquitectura de la Plataforma.

El inicio de esta propuesta reside en un primer esquema de una plataforma de monitorización híbrida (Navia, Bonastre, et al. 2015). En dicho desarrollo se puso de manifiesto la necesidad de una aproximación sistemática al problema, que se consiguió con la arquitectura de referencia (Capella et al. 2016), y en base a esto se ha desarrollado esta nueva plataforma denominada HMP.

3.1.1. Arquitectura de la Plataforma

La arquitectura de los elementos de la plataforma se basa en el modelo propuesto por Capella et al. (2016), y desarrollado dentro del mismo grupo y marco de trabajo que esta tesis. Este modelo está ilustrado en la Figura 21, y consta de tres capas: capa de Monitorización, capa de Información, y capa de Intercambio. La capa de Monitorización –que se ubica en la parte superior del modelo– se encarga de los aspectos de la red observada, la definición de los datos que deben adquiridos y cómo deben ser mostrados al usuario. La capa de Información –ubicada debajo de la de Monitorización– provee el soporte para codificar de una forma estandarizada la información obtenida, así como aspectos relacionados al tiempo de monitorización, como cuándo capturarlos y las marcas de tiempo de los datos capturados. La capa de Intercambio –en la parte inferior del modelo– permite que la informa-

ción capturada en diferentes partes de la plataforma sea almacenada y transferida. Las capas superiores recuperan esta información para que sea analizada y/o visualizada por la capa de Monitorización.



Fuente: (Capella et al. 2016)

Figura 21. Modelo de referencia para la plataforma de monitorización.

Cada capa define interfaces para comunicarse con la capa superior y/o inferior del modelo. Una de las ventajas de trabajar con una arquitectura definida por capas es que los cambios o desarrollos en una capa no deberían afectar a las otras. Por lo tanto, cualquier mejora en una parte de la plataforma debe ser fácil de desarrollar e implementar.

En el modelo se definen la función de las "sondas" (*probes*) –ubicadas en la capa de Monitorización–, que son las que permiten obtener la información de la WSN. Estas *probes* pueden ser funciones de *trap* en el nodo sensor (*probes* software) o componentes físicos que capturen datos (*probes* hardware). Normalmente –aunque no siempre–, las *probes* software se emplean en monitorización activa, mientras que las *probes* hardware son usadas en monitorización pasiva.

En la capa de Información se le da un formato estructurado a la información pasada "en crudo" desde las *probes*. Por lo general este formato sigue algún estándar, aunque también se puede utilizar un formato propio de la PDM. Entre los formatos que podrían utilizarse están XML (*eXtensible Markup Language*), JSON

(*JavaScript Object Notation*) y EXI (*Efficient XML Interchange*), entre otros. Además en este nivel se define cómo se van a sincronizar las marcas de tiempo de los datos obtenidos de la WSN, pudiendo aplicarse una sincronización de los elementos de la plataforma, o directamente un mecanismo de sincronización de los registros de datos capturados.

La transmisión de los datos entre las entidades de la PDM a través de la capa de Intercambio puede darse de dos maneras. La primera es utilizando un medio de almacenamiento no volátil, en el que se guarda la información para ser descargada o transportada más adelante. La segunda es utilizar una red de comunicación –cableada o inalámbrica– para transmitir la información desde un componente de la plataforma a otro.

En base a estos criterios se han diseñado los componentes de la HMP.

3.1.2. Funcionamiento de la Plataforma

La plataforma consta de tres componentes básicos: El Nodo Monitor (NM), el Nodo *Sniffer* (NS), y el Servidor de Monitorización (SM). En la Figura 22 se muestran los componentes de la HMP y su ubicación en relación a una WSN monitorizada. Tanto los NM como los NS recogen datos que posteriormente hacen llegar al servidor de monitorización, tal como muestra la Figura 22.

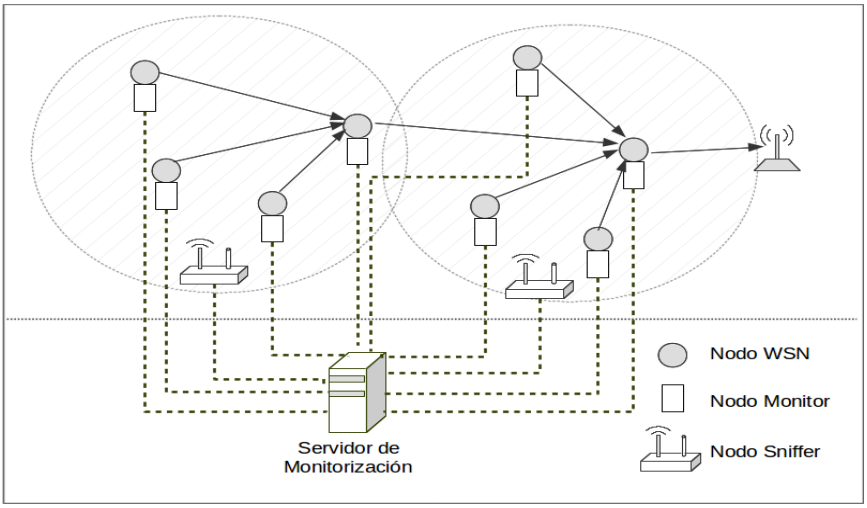


Figura 22. Esquema de los componentes de la HMP

Para facilitar el despliegue de la HMP, se considera en la propuesta inicial que los Nodos *Sniffer* (NS) actúen como recolectores de la información de los Nodos Monitor (NM) y posteriormente la retransmitan al Servidor. No obstante, por simplicidad este detalle no aparece en la Figura 22.

Un Nodo Monitor acompaña a cada uno de los nodos de la WSN observada, incluyendo –si es necesario– el sumidero de la red. La conexión entre ambos nodos se detalla más adelante. Cada NM registra los eventos de interés del nodo al que se conecta, les agrega una marca de tiempo y un formato estructurado, y los envía al SM.

Cada NS cubre un área determinada de la WSN, siendo capaz de capturar los mensajes que son transmitidos por los nodos de la WSN. En este sentido no se diferencia mucho de los *sniffers* que utilizan otras propuestas, si bien integrados en el resto de la HMP. El Servidor de Monitorización –o Servidor Monitor (SM)– reúne la información obtenida por los diferentes elementos de la plataforma para su procesamiento. Además el SM podría tener a su cargo otras funciones, como la de la sincronización de los datos obtenidos.

El funcionamiento de la plataforma es el siguiente: En primer lugar, el diseñador analiza la WSN que desea monitorizar, seleccionando qué parámetros debe observar para obtener el fin propuesto (diagnóstico, despliegue, depuración, etc.). Una vez establecidos los parámetros e indicadores, se diseñarán las sondas, o *probes*, adecuadas para la obtención de los mismos. Asimismo, se estudiará de entre los interfaces disponibles en el nodo a monitorizar cuál es el más adecuado para la comunicación con el nodo monitor, en función de, entre otros aspectos, la intrusión causada en el nodo. Esta intrusión ha sido estudiada en artículos anteriores (Navia, Campelo, et al. 2015), conociéndose su impacto en el nodo observado. Cabe destacar que, para las opciones más frecuentes, la intrusión es despreciable, tanto en tiempo como en requisitos de memoria y en consumo de potencia, tal como se presenta en la referencia anterior y posteriormente se detalla en el Capítulo 4.

Para el estudio de monitorización, en los nodos a observar se incluirán las rutinas –denominadas *traps* software– que permitirán informar al nodo monitor sobre el estado actual del nodo a observar.

A continuación se conectarán los nodos monitores a los nodos a monitorizar a través del interfaz seleccionado. En paralelo, se analizará la cobertura de la red para seleccionar las ubicaciones óptimas para los nodos *sniffers*. Una vez completados estos pasos, únicamente resta proporcionar la conectividad entre los NS y NM con el SM.

Una vez implantada la plataforma, se puede poner en marcha la campaña de monitorización correspondiente. La WSN bajo observación realizará sus funciones de la forma habitual, mientras que los NM y los NS recogerán una traza de los parámetros a monitorizar, incluyendo su temporización.

El paso final consiste en la confluencia de los datos sobre el SM, que empleando los algoritmos de sincronización necesarios generará una traza unificada

de los eventos sucedidos en la WSN a observar que permitirá detallar su funcionamiento.

Una de las principales ventajas de esta plataforma de monitorización, que responde a una arquitectura estructurada, consiste en la reutilización de los elementos de monitorización. En gran parte de las plataformas estudiadas en el Capítulo 2 el diseño del monitor está estrechamente ligado al propio sistema distribuido a monitorizar. Sin embargo, la plataforma propuesta puede adaptarse de forma sencilla a cualquier WSN. Así, tras la campaña de monitorización tanto los NM como los NS pueden ser retirados para su uso en otra campaña.

3.1.3. Monitorización activa y pasiva

La HMP combina tanto mecanismos de monitorización activos como pasivos. La Figura 23 ilustra los componentes de la plataforma que funcionan en cada uno de estos modos, así como las operaciones básicas y el flujo de datos entre estos.

El componente activo de la plataforma es el nodo monitor. El NM se conecta directamente a cada nodo de la WSN observada, e interactúa con una poción muy pequeña de código que se agrega al nodo monitorizado, para obtener información de este. La conexión entre ambos se realiza a través de alguna de las interfaces estándar disponibles en el nodo sensor. El NM provee formato y le agrega la marca de tiempo correspondiente a los datos recibidos. Esta información pasa al servidor de monitorización para su procesamiento.

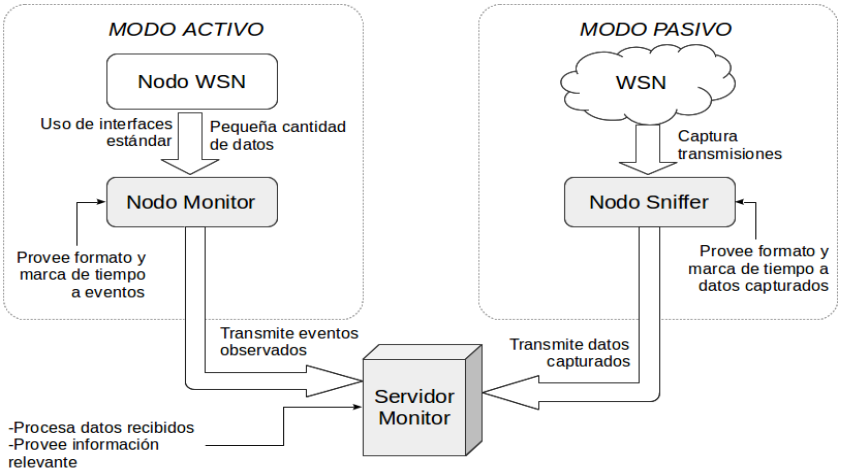


Figura 23. Diagrama de funcionamiento y comunicación de la HMP.

El NS es el componente pasivo de la plataforma. El NS guarda cada mensaje obtenido con un formato predefinido que incluye la marca de tiempo del momento cuando fue capturado. Al igual que el NM, el NS envía los mensajes capturados al

servidor de monitorización. Con la información que va obteniendo y generando cada elemento de la HMP, se va creando una traza individual de los eventos relevantes de la WSN.

El SM procesa los datos recibidos por los elementos activos y pasivos de la plataforma, los unifica, y los pone a disposición del usuario que está realizando la monitorización. La información procesada por el SM puede ser visualizada en el mismo servidor, aunque también podría ser utilizada para otros fines, por ejemplo como base para realizar una simulación.

3.2. Diseño de la plataforma

3.2.1. Diseño del Nodo Monitor

El nodo monitor (NM) es el elemento activo de la plataforma. Se puede decir también que es un monitor híbrido ya que tiene un componente de software y uno de hardware. Los NM de la HMP, presentada en esta Tesis, son una evolución de las primeras propuestas presentadas previamente (Navia, Campelo, et al. 2015).

En la Figura 24 se muestra la funcionalidad del NM, basado en la arquitectura descrita previamente. La capa de Monitorización está constituida por un conjunto de rutinas software agregadas al nodo sensor (*traps* de software), para enviar información de los eventos internos relevantes en el mismo hacia el nodo monitor.

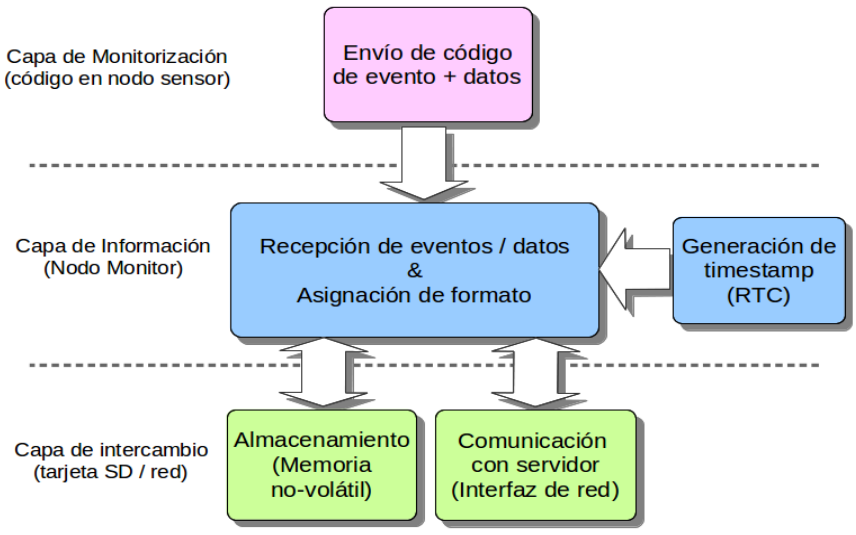


Figura 24. Arquitectura del Nodo Monitor.

Este código deber ser necesariamente de tamaño reducido, para minimizar la intrusión causada al nodo sensor, tanto a nivel de consumo de recursos –memoria o energía–, como en cuanto a tiempo de operación del nodo. Estas rutinas contemplan una función de inicialización de la interfaz utilizada y una función de transmisión de información que será utilizada para cada uno de los eventos que quieran registrarse.

La capa de Información se implementa en la parte del hardware del NM, y está constituido por las operaciones de recepción de datos, la generación de marcas de tiempo, y la asignación del formato con el que se almacenan los datos. Las marcas de tiempo son generadas mediante un Reloj de Tiempo Real (RTC) incluido en el NM. El formato de datos –incluyendo las marcas de tiempo– debe ser legible no solo por el SM, sino también por cualquier otra aplicación que pudiera hacer uso de la información obtenida por los NM.

En la plataforma propuesta para la capa de Intercambio del NM se contemplan dos opciones, que pueden ser utilizadas individualmente o de forma simultánea. La primera es almacenar los datos en una memoria no volátil –una tarjeta *Secure Digital* (SD) por ejemplo–, cuyo contenido (traza de eventos) posteriormente será entregado al SM de forma manual. La segunda opción contempla una interfaz de red que permite enviar los datos –con el formato establecido– directamente al SM. Esta segunda opción permite una monitorización en tiempo real, mientras que la primera, evidentemente, solo acepta análisis *offline*. Por el contrario, un flujo continuado de datos hacia el SM puede perder datos si su procesado se ralentiza. El uso simultáneo de ambas opciones permite tanto proporcionar inmediatez como garantizar que no se pierden datos.

La Figura 25 muestra un ejemplo de funcionamiento de un NM. En este caso, cuando el nodo sensor pasa a estado activo (se despierta), informa al NM de los eventos que se consideran relevantes, como son el despertar, la lectura de los sensores, la transmisión, u otros, a través de la interfaz entre ambos, denominada interfaz Mon-Inf (Monitorización-Información). La interfaz Mon-Inf puede ser cualquier posibilidad de transmisión que esté disponible en el nodo sensor, preferiblemente una interfaz estándar (transmisión serie, puertos paralelo, buses I^2C o SPI, entre otros.)

Por otra parte, la interfaz entre las capas de Información e Intercambio –denominada Inf-Int– puede implementarse mediante sockets para comunicación por la red (preferiblemente UDP para evitar retrasos y reducir la sobrecarga), o mediante librerías o controladores para el almacenamiento en memoria no volátil.

Figura 25. Esquema de ejemplo de operación del nodo monitor.

3.2.2. Diseño del Nodo Sniffer

El nodo *Sniffer* –el componente pasivo de la plataforma–tiene cierta similitud con los *sniffers* de otras plataformas de monitorización pasiva para WSN citadas en el capítulo 2. La principal diferencia es que el NS está diseñado de acuerdo a la arquitectura descrita, y por lo tanto se integra perfectamente en la plataforma HMP. La arquitectura del NS –ilustrada en la Figura 26– se basa en la presentada en el apartado 3.1.1. Es posible encontrar en cada capa los componentes de este nodo.

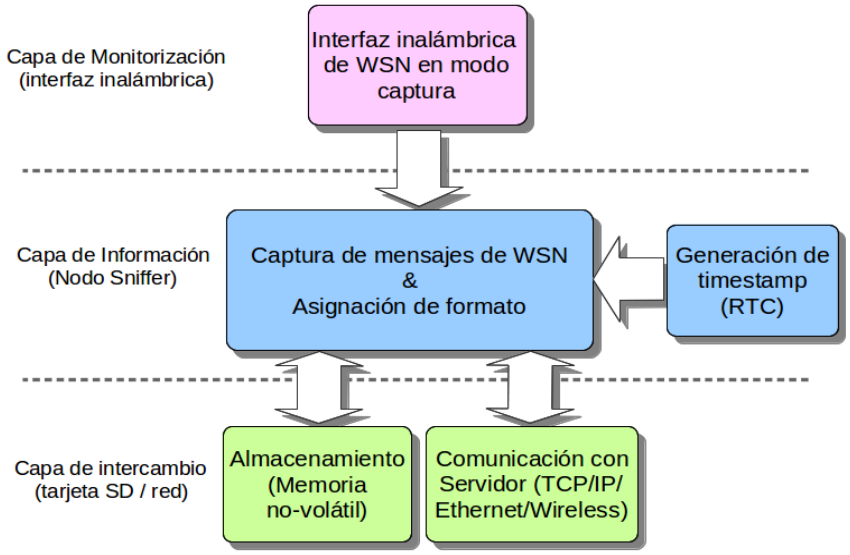


Figura 26. Arquitectura del Nodo Sniffer.

En el nivel de Monitorización, implementado mediante un *probe* hardware, consistente en una interfaz inalámbrica del mismo tipo que la WSN que funciona en modo promiscuo. Esta interfaz debe funcionar con la misma configuración que

la red monitorizada, es decir, en la misma frecuencia, canal, modulación de señal, entre otros aspectos de operación.

Los paquetes recibidos a través de la interfaz inalámbrica pasan a la capa de Información del NS, donde se les asigna una marca de tiempo. El NS también cuenta con un RTC para la generación de estas marcas. De forma similar que el NM, la capa de Información del Nodo *Sniffer* aplica un formato preestablecido que incluye una marca de tiempo a los mensajes capturados.

Al igual que en el caso anterior, la capa de Intercambio puede implementarse tanto mediante una interfaz de red como mediante soporte en memoria no volátil. Para el primer caso se contemplan adaptadores tanto inalámbricos tipo IEEE 802.11, como Ethernet, o cualquier otra opción que soporte la pila de protocolos TCP/IP. Gracias a ellos es posible la comunicación del NS con el Servidor de Monitorización.

3.2.3. Diseño del Servidor de Monitorización

El SM es el encargado de la recolección, procesamiento y visualización de los datos recolectados por el resto de la plataforma. Si bien para la tesis se han desarrollado algunas aplicaciones específicas para el SM, también podrían utilizarse otras aplicaciones o sistemas existentes –junto con un *plug-in* adecuado–, gracias a la utilización de formatos estandarizados para la información generada, que puede ser fácilmente tratada por estas aplicaciones estándar. Un enfoque realista partiría de la personalización de herramientas estándar que podrían complementarse con otras aplicaciones ad-hoc directamente relacionadas con el sistema a monitorizar. A largo plazo, estas aplicaciones ad-hoc podrían centralizarse en un repositorio de forma que quedaran a disposición del colectivo de usuarios como software libre.

En la Figura 27 se muestra el esquema de la arquitectura del SM. La capa superior se implementa a través de una interfaz de usuario que permite manejar los procesos del SM, así como visualizar la información generada por la HMP.

En la capa de Información –la parte medular del SM– se ejecutan varias tareas: El procesamiento de los datos, la sincronización de trazas (cuyo mecanismo se explicará detalladamente más adelante), y la unión de las trazas de cada componente en una única traza global.

Al igual que en los componentes anteriores, las funciones de la capa de Intercambio se pueden implementar tanto mediante una interfaz de red como mediante lectores que permitan la obtención de las trazas guardadas en memoria no volátil.

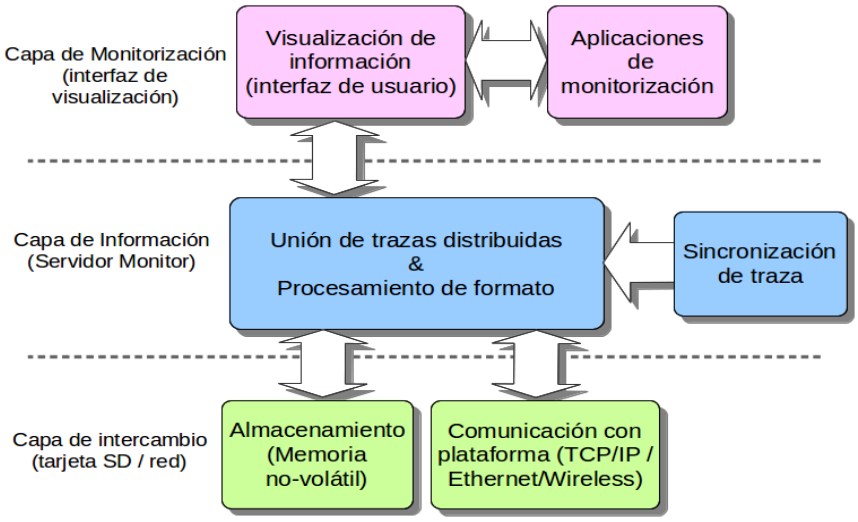


Figura 27. Arquitectura del Servidor de Monitorización.

3.2.4. Mecanismo de sincronización de trazas

Para la sincronización de las trazas de la HMP se ha planteado un nuevo mecanismo denominado GTSO (Navia et al. 2018). GTSO aplica un procedimiento de sincronización *off-line* o fuera de línea (sincronización *post-facto*). Se basa en la inclusión de eventos comunes globales –puntos de sincronización– registrados por todos los nodos de la plataforma, tanto NM como NS, y por tanto incluidos dentro de las trazas generadas por mismos durante la campaña de monitorización, junto con los eventos relevantes capturados. Estos puntos de sincronización no interfieren con la WSN observada, ya que las señales de sincronización que los generan se difunden a través de medios alternativos –como la red de la plataforma de monitorización– con el objetivo de que se reciban de forma simultánea en todos los nodos. En cada uno de ellos, la llegada de estas señales de sincronización se registra en la traza local como un punto de sincronización con su respectiva marca de tiempo local. Esto hará posible corregir las marcas de tiempo de los eventos registrados en las trazas.

GTSO puede considerarse un enfoque híbrido entre la sincronización en línea de los nodos del monitor y el procesamiento fuera de línea de las trazas. No existe un reloj común para todos los nodos de la PDM, pero se recibe y registra un punto de sincronización periódico en todos estos nodos simultáneamente. De esta forma, el rendimiento de GTSO solo está limitado por la precisión y la variabilidad del tiempo de registro en las trazas de este punto de sincronización, debido a la deriva, y a la variabilidad e inestabilidad de la deriva.

En GTSO hay un nodo especial –denominado *SyncRoot*– que genera y envía las señales de sincronización periódicamente, mediante la transmisión de tramas dedicadas a la sincronización. Cada señal de sincronización se numera como única y el *SyncRoot* almacena el punto de sincronización correspondiente en un archivo con la marca de tiempo local.

Cuando finaliza el experimento de monitorización, todas las trazas se entregan al SM –que también podría hacer la función de *SyncRoot*– y se aplica el mecanismo GTSO, tal como se define en el apartado 3.3.4, para corregir las marcas de tiempo de cada traza. Se estima la deriva del reloj entre dos puntos de sincronización, y la marca de tiempo de cada evento registrado entre estos dos puntos se recalcula proporcionalmente de acuerdo con la deriva calculada. Si dos puntos de sincronización están suficientemente cerca en el tiempo, la deriva de la velocidad de reloj en un nodo puede considerarse constante. Esto pone de manifiesto la relevancia del período de sincronización en la precisión de GTSO: si el período de sincronización es demasiado grande en relación con la estabilidad del reloj, la variabilidad de la deriva puede aumentar el error de sincronización (Navia et al. 2018).

Finalmente, todas las trazas deben fusionar en una única traza ordenada, que incluye todos los eventos y sus parámetros, su marca de tiempo corregida y el nodo que proporcionó cada evento. Esta traza reflejará fielmente la evolución del sistema observado.

3.3. Implementación y operación de la Plataforma

En este apartado se detallan algunos aspectos relativos a la implementación concreta de la plataforma HMP, presentada en esta Tesis. Así, se traslada la propuesta a un entorno concreto con el propósito de demostrar la viabilidad y documentar el proceso de diseño e implementación de la propuesta. Como toda implementación, puede no ser única y es susceptible de ampliaciones dentro del marco propuesto. Por ello, las decisiones tomadas no son excluyentes, es decir, otras plataformas compatibles podrían asumir otros modos de funcionamiento dentro de la arquitectura de referencia.

3.3.1. Implementación del Nodo Monitor

La función principal del nodo monitor es recoger los eventos generados por las *traps* software que se ejecutan en el nodo a monitorizar y que éste transmite a través de la interfaz Mon-Inf.

En el Algoritmo 1 se muestra un ejemplo del código correspondiente a dichas *traps*, que se agrega al nodo sensor como parte de la capa de Monitorización. Una función –denominada *WriteLog*– envía la información byte a byte al componente

hardware del NM. El parámetro *DataLong* indica si se pretenden enviar datos adicionales junto al código del evento, siendo mayor que cero cuando es así. El primer byte enviado lleva dos campos: el primer bit indica si hay datos adicionales, y los siete restantes corresponden al código del evento (argumento *code*). De haber datos adicionales al código de evento, el segundo byte transmitido indica su tamaño en bytes. A continuación se transmite esta información adicional, contenida en el argumento *msg*.

Algoritmo 1. Ejemplo de código de monitorización implementado en el nodo sensor.

```
void WriteLog(uint8_t DataLong, uint8_t code, char msg[]){
  if (DataLong>0){
    msgTx = (uint8_t) (0x80 | code);
    Serial0_send(msgTx);
    Serial0_send(DataLong);
    for (ind=0; ind<DataLong; ind++)
        Serial0_send(msg[ind]);
  }else{
    msgTx = 0x7F & code;
    Serial0_send(msgTx);
  }
}
```

```
void main (void) { /*Aplicación principal del nodo sensor */
…
  WriteLog(0,Log_Reset,message); // Registrar evento inicio
…
  temperature_frame_send (message);
  WriteLog(12,Log_TxData,message); // Registrar evento TxData
…
} /*Fin de la aplicación principal del nodo sensor */
```

En ciertas ocasiones es suficiente registrar un evento solo con un código –como por ejemplo cuando recién se inicia o reinicia el nodo sensor–, pero en otros casos puede ser necesario registrar información adicional sobre el evento –por ejemplo el contenido de un mensaje que se transmite– para un análisis más preciso. La función de paso de datos hacia el NM se ha implementado de forma que pueda ser flexible en ambos casos.

La Tabla 1 muestra un ejemplo de la definición de los códigos de evento de los nodos sensor. Si bien se tienen 7 bits para definir los códigos de evento –lo que en teoría da para registrar hasta 128 tipos diferentes–, en la mayoría de casos con 4 bits es suficiente para registrar todos los eventos que pueden resultar interesantes. Se han utilizado dos bits más, para poder definir eventos relacionados a la operación del nodo *sniffer*, tal como lo muestra la última entrada de la Tabla 1.

Tabla 1. Ejemplo de Definición de códigos de eventos del nodo sensor.

Definición de código		Significado (evento)
#define Log_Reset	0x00	Reset/inicialización del nodo
#define Log_Sense0	0x01	Leer del sensor 0 (primero o único)
#define Log_Sense1	0x02	Leer del sensor 1 (segundo, si hubiera)
#define Log_TxData	0x03	Se transmite información
#define Log_TxRoute	0x04	Se transmite información de enrutamiento
#define Log_RxData	0x05	El nodo recibe información
#define Log_RxACK	0x06	El nodo recibe un ACK
#define Log_Sleep	0x07	El nodo entra en modo Sleep
#define Log_Stop	0x08	El nodo entra en modo Stop
#define Log_WakeUp	0x09	Despertar del nodo desde Sleep/Stop
#define Log_LowBat	0x0A	Indicador de batería baja
#define Log_ReRoute	0x0B	Retransmite (si es nodo enrutador)
#define Log_Temper	0x0C	Temperatura del nodo
#define Log_Error	0x0F	Error en el nodo
#define Log_SnifMsg	0x20	Trama capturada por nodo sniffer

En cuanto a la implementación del interfaz Mon-Inf, el NM está diseñado para utilizar tanto una interfaz paralela, mediante pines GPIO, como una interfaz serie, usando un bus UART o SPI. Cada evento requiere una única invocación de la rutina *trap*, que internamente gestiona la transmisión de los diferentes bytes que pueden contener el evento. La selección del hardware para la interfaz Mon-Inf entre ambos nodos tendrá un papel relevante a la hora de generar una mayor o menor intrusión, que podría distorsionar las medidas obtenidas.

En la implementación propuesta, cuando el NM recibe los datos transmitidos por el nodo sensor a través de la interfaz Mon-Inf, se activa una interrupción. En el Algoritmo 2 se muestra un ejemplo de la función de interrupción *MonInf_IRQHandler* que gestiona la llegada del evento. En primer lugar, esta interrupción genera la marca de tiempo del instante de llegada del evento, mediante una llamada a la función *TimeStamp*.

El código presentado en el Algoritmo 2 distingue entre la trasmisión de un evento simple, identificado solo con el código, y otro que lleva información adicional. Tanto los datos recibidos como las marcas de tiempo –que, como se comentará más adelante, en esta plataforma tienen una precisión a nivel de microsegundos– son almacenados en listas. Para una gestión rápida y eficiente de la información

recibida, se utilizan técnicas de Acceso Directo a Memoria (DMA) en el NM, mediante un buffer circular.

Algoritmo 2. Código de interrupción de recepción de evento en el NM.

```
void MonInf_IRQHandler(void){
uint8_t lectura, x, y;
    TStmp1 = TimeStamp(&SubSec1);
    y = MyBuffer.head;
    x = MyBuffer.tail;
    for (y; y<x; y++) {
        if ((ring_buffer_pop(&MyBuffer, &lectura)!=OK))
            printf("\nBuffPop Error");
        if (VectorPos==0) {
            TimeStampVector[EvPos] = TStmp1;
            SSecVector[EvPos] = SubSec1;
            EvPos=count;
            count++;
            if (lectura<0x80)
                VectorLong=0;
            else
                VectorLong=1;
        }
        LogVector[VectorPos][EvPos]=lectura;
        if (VectorPos==1)
            VectorLong=(lectura+1);
        if (VectorLong>0)
            VectorPos++;
        if (VectorPos>VectorLong)
            VectorPos=0;
    }
    ClearITPendingBit(MonInf, IT_RXNE);
}
```

Periódicamente, el NM verifica si hay datos recibidos desde la *probe* –revisando el contador *count* en el ejemplo mostrado en el Algoritmo 2– y si es así aplica las funciones del nivel de Información, aplicando el formato preestablecido. A continuación emplea los servicios del nivel de Intercambio para hacerlo llegar hasta el SM, bien mediante acceso directo en red o bien de forma diferida mediante almacenamiento en la memoria SD, o por ambos medios. Tras ello, resetea de nuevo el contador. Para hacer más versátil la asignación de formato de acuerdo a la definición de los eventos, la plantilla de formato es definible para cada campaña de monitorización, y puede proporcionarse al nivel de Información a través de un fichero. Si existe este fichero, se carga la plantilla en él almacenada y se aplica para cada registro, antes de entregarlo al nivel de Intercambio. En su ausencia, al evento se le da un formato genérico.

Se ha propuesto una estructura de tipo XML como formato de los datos obtenidos mediante los componentes de la plataforma propuesta, tanto el NM como el NS. En la Figura 28 se muestra un ejemplo de plantilla para un evento de transmisión registrado en el nodo monitor. Para cada evento observado se reemplazan las ocurrencias de los caracteres "%s" por los valores correspondientes a los campos: el NM origen que recibe el evento, la marca de tiempo correspondiente, y como valor adicional los valores transmitidos por el nodo sensor que corresponden con los datos enviados desde el nodo monitorizado.

```
<HEADER>
  <TYPE>SEND</TYPE>
  <ORIGEN>%s</ORIGEN>
  <DESTINO>ESP.VAL.ITACA.WISFAS.SERV</DESTINO>
  <TIME>%s</TIME>
</HEADER>
<SEND>
  <NAME>Transmission</NAME>
  <VALUE>%s</VALUE>
  <UNITS>Frame-Data</UNITS>
</SEND>
```

Figura 28. Ejemplo de definición de formato de datos para evento Transmisión.

La estructura generada se envía a la capa de intercambio, ya sea para ser transmitida al SM, o guardada en la tarjeta SD. En la implementación realizada del nivel de Intercambio se decidió aplicar ambas opciones. Para la transmisión hacia el SM se utilizan datagramas UDP, teniendo cada NM pre-configurado los parámetros de direccionamiento IP. Las trazas también se almacenan en una memoria SD, en formato ASCII, a modo de respaldo, por si no fuera correcta la comunicación hacia el SM.

3.3.2. Implementación del Nodo Sniffer

El NS ha sido implementado sobre la misma base del NM, concretamente a nivel de hardware. A nivel de la capa de Monitorización el NS implementa una *probe* hardware, que consiste en una interfaz inalámbrica –la misma que la WSN monitorizada– en modo promiscuo. La forma de operación del NS es parecida al del NM, con la diferencia de que lo que recibe son mensajes capturados por la *probe*. Cuando un mensaje es capturado, se activa una interrupción. La función de manejo de esta interrupción –similar a la que se muestra en el Algoritmo 2– también genera una marca de tiempo de cuando el mensaje es capturado por el NS.

A nivel de la capa de Información, el NS revisa regularmente si hay mensajes capturados. Si es así, al igual que el NM, les da un formato predeterminado –que

incluye la marca de tiempo– y los pasa al nivel de Intercambio. A diferencia del NM –donde hay que definir los varios tipos de eventos que quieran registrarse–, en el NS solo es necesario establecer un formato para la información de los mensajes capturados. La estructura de este formato es similar a la mostrada previamente en la Figura 28, es decir una estructura XML, con la única diferencia de que todos los eventos registrados por el NS son de tipo "*MensajeCapturado*".

Al igual que en el NS, el nivel de intercambio se ha implementado utilizando las dos opciones disponibles, esto es la transmisión por UDP al SM, y el almacenamiento en una memoria SD.

3.3.3. Implementación del Servidor de Monitorización

Las funciones principales del SM se centran en el nivel de Monitorización, especialmente en el procesamiento de las trazas obtenidas. Este procesado suele incluir almacenamiento de las trazas, sincronización de las mismas mediante el algoritmo GTSO citado anteriormente, análisis automáticos de la traza unificada y visualización tanto gráfica como textual de los resultados. En caso de sistemas dotados de una capa de Intercambio que lo permita, el SM podrá implementar mecanismos de análisis y visualización *online*, así como módulos de gestión de alarmas. Adicionalmente, también puede actuar a nivel de Información como controlador de la sincronización (*SyncRoot*). Todas estas tareas se pueden controlar desde una GUI de la aplicación del servidor.

3.3.3.1 Interfaz gráfica del Servidor de Monitorización

La GUI –que constituye el nivel de Monitorización en el Servidor– permite controlar las opciones principales del SM propias de la plataforma de monitorización. La mayor parte de la interfaz fue implementada con el lenguaje de programación Python sobre Ubuntu Linux. Las aplicaciones de envío de señales para sincronización y de recepción de mensajes se crearon con el lenguaje C y el compilador GCC, y son llamadas desde la interfaz correspondiente.

La Figura 29 muestra la ventana principal de la aplicación del SM. De forma predeterminada la ventana que aparece es la del procesamiento de datos posterior a la monitorización. Sin embargo, desde aquí –a través del menú– se puede acceder al resto de opciones de la aplicación.

Cuando se escoge la opción de Nueva Monitorización aparece otra ventana (Figura 30), que permite iniciar un nuevo proceso de monitorización en el que se pueden recibir los datos para las trazas a través de la red, y además integra el proceso de envío de señales para la función de sincronización. En esta interfaz se especifica en qué directorio se guardarán las trazas individuales, el periodo de sincronización, y el fichero donde se guardarán las marcas de las señales de

sincronización. Si no se especifican todos estos parámetros no se puede iniciar el servicio.

Figura 29. Ventana principal de la interfaz gráfica del SM.

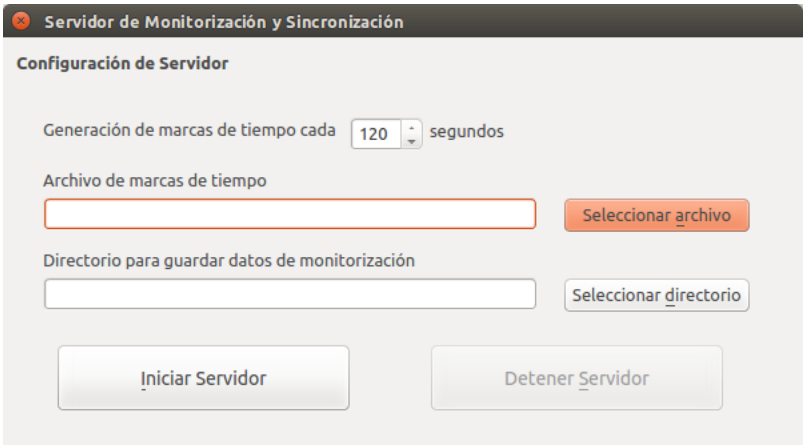


Figura 30. Ventada para iniciar monitorización.

Al iniciarse el servicio de monitorización, se empiezan a ejecutar las aplicaciones de recepción de datos desde la plataforma y de envío de señales de sincronización (Figura 31). Al finalizar la monitorización (opción "Detener Servidor" en la figura) estas aplicaciones se cierran. La información generada por la plataforma –es decir el conjunto de las trazas individuales de cada elemento de la misma– queda guardada en el directorio especificado. De igual manera, queda almacenado el archivo de marcas de tiempo maestro que se utilizará en la sincronización de las trazas.

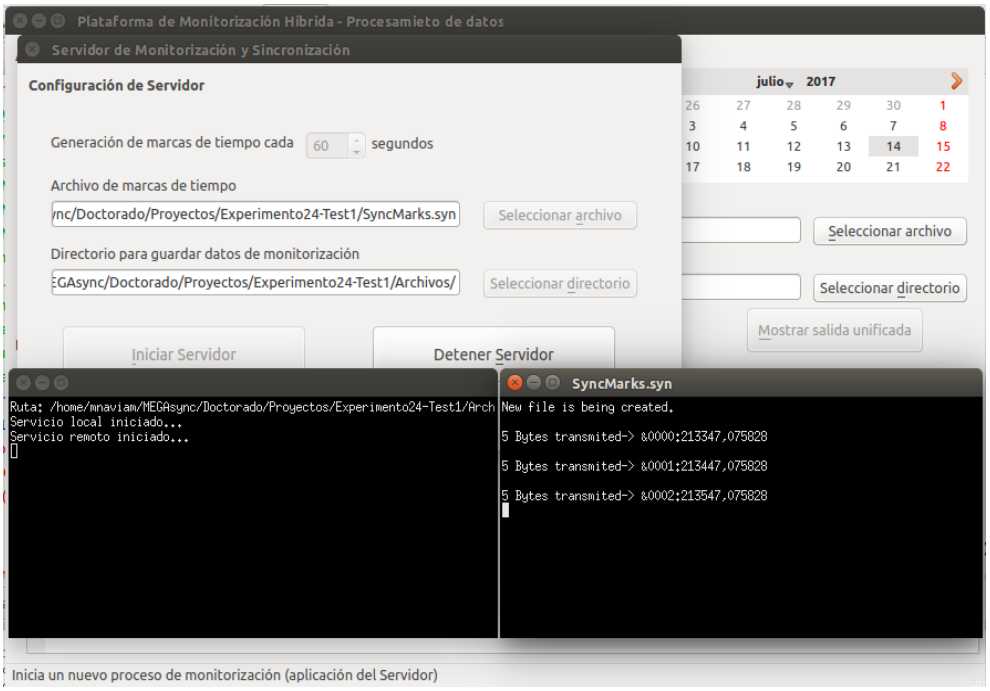


Figura 31. Procesos de monitorización en el SM.

3.3.3.2 Procesamiento de los datos obtenidos

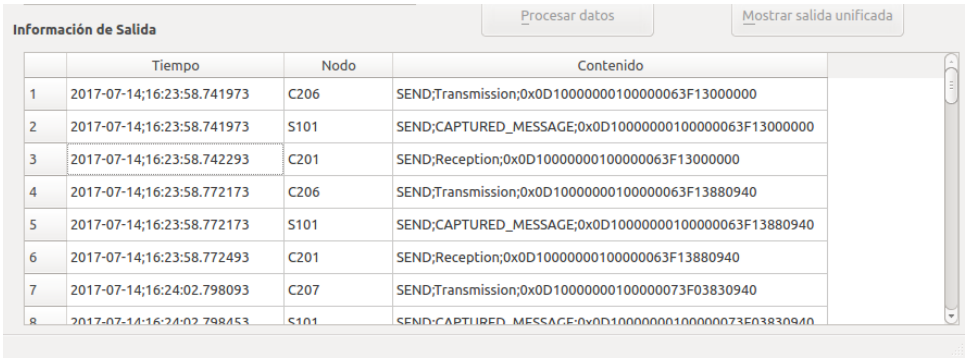
Las funciones de procesamiento de datos –el nivel de Información del SM– se han desarrollado en el lenguaje de programación Python, al igual que la GUI. El procesamiento se realiza en dos partes: la sincronización de las trazas individuales obtenidas en cada elemento de la plataforma, y la unión de trazas.

Para el procesamiento de los datos es necesario especificar los parámetros indicados anteriormente –archivos de trazas individuales, archivo para sincronización, directorio de salida–, pudiendo añadir opcionalmente la fecha en que se realizó la monitorización. Por defecto se asume la fecha actual del sistema.

En primer lugar sincroniza las trazas individuales obtenidas por cada componente de la plataforma, y después se las unifica en una sola traza global de forma ordenada. Este proceso crea archivos de salida de cada traza individual sincronizada en el directorio de salida, que luego pueden ser utilizados para otro fin si se lo requiere. En la Figura 32 se muestra un ejemplo del resultado de este proceso: la tabla que contiene la traza global unificada.

Una vez sincronizadas las trazas individuales, éstas se pueden combinarse en una única traza global. Los eventos en esta traza global –visualizados en la GUI en

la Figura 32– son ordenados de acuerdo a la marca de tiempo registrada. Es posible definir si los eventos duplicados –es decir, que se han registrado en más de un nodo monitor y/o *sniffer*– deben aparecer una única vez en la traza (facilitando la legibilidad de la misma), o si deben mantenerse los distintos eventos individuales para resaltar algún aspecto concreto de la transmisión (por ejemplo, mantener los eventos de transmisión de la trama en un nodo y el de recepción de la misma en otro para evaluar el tiempo de transmisión).



	Tiempo	Nodo	Contenido
1	2017-07-14;16:23:58.741973	C206	SEND;Transmission;0x0D10000000100000063F13000000
2	2017-07-14;16:23:58.741973	S101	SEND;CAPTURED_MESSAGE;0x0D10000000100000063F13000000
3	2017-07-14;16:23:58.742293	C201	SEND;Reception;0x0D10000000100000063F13000000
4	2017-07-14;16:23:58.772173	C206	SEND;Transmission;0x0D10000000100000063F13880940
5	2017-07-14;16:23:58.772173	S101	SEND;CAPTURED_MESSAGE;0x0D10000000100000063F13880940
6	2017-07-14;16:23:58.772493	C201	SEND;Reception;0x0D10000000100000063F13880940
7	2017-07-14;16:24:02.798093	C207	SEND;Transmission;0x0D10000000100000073F03830940
8	2017-07-14;16:24:02.798453	S101	SEND;CAPTURED_MESSAGE;0x0D10000000100000073F03830940

Figura 32. Resultado de Procesamiento de datos de la plataforma.

La traza global obtenida puede exportarse la en diferentes formatos, como por ejemplo formato de valores separados por comas (CSV), con una estructura definible por el usuario, en función de qué utilidad vaya a tratar dicha traza.

3.3.3.3 Recepción de datos obtenidos (trazas)

Por debajo del nivel de Información, la capa de intercambio deberá recibir las trazas mediante medios compatibles con los citados anteriormente para Nodos Monitores y Nodos *Sniffer*.

En el caso de almacenamiento en memoria no volátil, cada memoria es introducida en el SM, procediéndose al volcado automático de la traza. El contenido de la etiqueta <ORIGEN> de la estructura XML permite etiquetar de forma correcta cada fichero.

Como alternativa, si el nivel de Intercambio lo permite, la creación de los ficheros de trazas se hace durante la campaña de monitorización, puesto que cada evento detectado en un nodo se envía directamente al SM a través de la red de comunicaciones. Al igual que en el caso anterior, el contenido de la etiqueta <ORIGEN> permite clasificar los eventos recibidos y generar archivos individuales para cada elemento de captura.

3.3.4. Implementación del mecanismo de sincronización

3.3.4.1 Funcionamiento de GTSO

La Figura 33 y la Figura 34 muestran gráficamente el funcionamiento del mecanismo. En ambas figuras, el equipo que asume el rol de *SyncRoot* transmite periódicamente las señales de sincronización (Figuras 33a y 34a). El tiempo t_{r0} es el tiempo inicial de referencia, en el que se envía el primer mensaje de sincronización, y $t_{r0} + nT$ el tiempo en el que se envían los mensajes de sincronización sucesivos. El nodo monitor (o *sniffer*) inserta en su traza un nuevo evento –o punto de sincronización– que registra los tiempos de llegada de la señal de sincronización (t_{lx} en las Figuras 33b y 34b), de acuerdo con su reloj local. Estas marcas de tiempo pueden adelantarse o retrasarse en relación con el tiempo de referencia en *SyncRoot*. En el caso de la Figura 33b, la velocidad del reloj local es más rápida que *SyncRoot* –los valores de marca de tiempo son mayores que el tiempo *SyncRoot*–, mientras que en la Figura 34b las marcas de tiempo se retrasan, ya que la velocidad del reloj local es más lenta que *SyncRoot*.

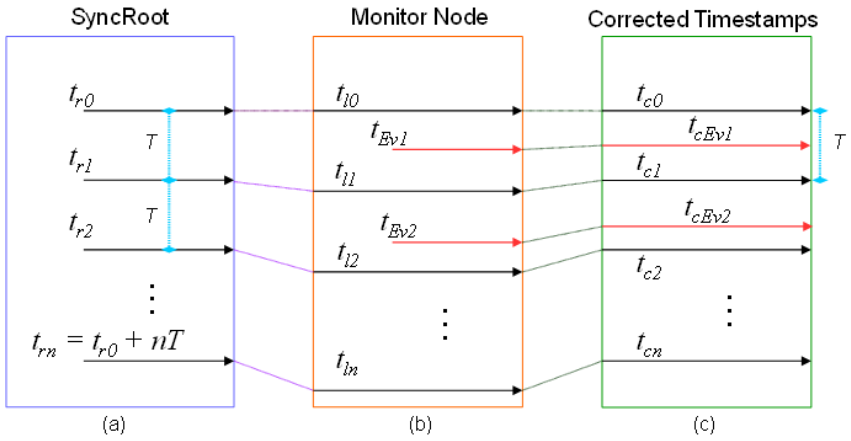


Figura 33. Esquema de funcionamiento de GTSO, para un nodo con reloj más rápido que el tiempo de referencia

Como se indicó anteriormente, al final de la operación de monitorización, las trazas se procesan en el SM. Como se ve en las Figuras 33c y 34c, los tiempos de las marcas de sincronización locales t_{lx} se corresponden con los puntos t_{rx} de *SyncRoot*, y se indican como t_{Cx}. Después de eso, las marcas de tiempo t_{Evx} de los eventos registrados se recalculan proporcionalmente, según los puntos de sincronización más cercanos (anterior y posterior). Las nuevas marcas de tiempo se indican como t_{cEvx} una vez recalculadas.

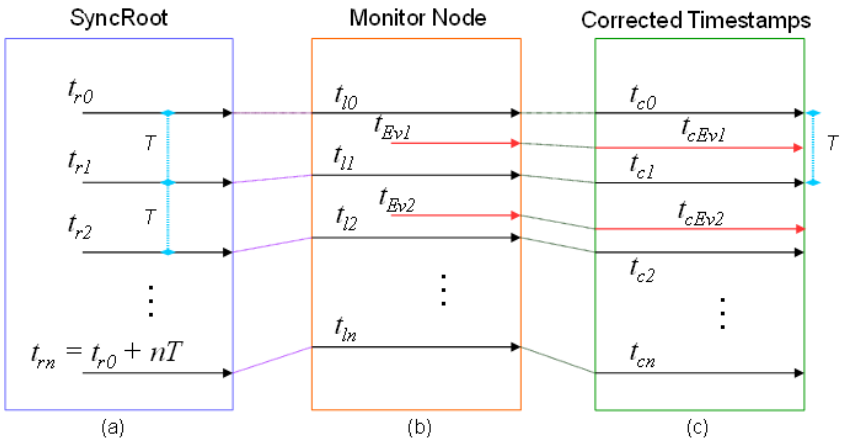


Figura 34. Esquema de funcionamiento de GTSO, para un nodo con reloj más lento que el tiempo de referencia

La corrección de marcas de tiempo se realiza por medio del siguiente procedimiento, publicado por Navia et al. (2018): t_{Evm} denota la marca de tiempo de un evento *m*, que se ha grabado en un nodo de monitor entre dos puntos de sincronización locales consecutivos con marcas de tiempo t_{ln-1} y t_{ln}, eso es, $t_{ln-1} \leq t_{Evm} \leq t_{ln}$, con $t_{ln-1}<t_{ln}$. La marca de tiempo corregida del evento *m*, indicada como t_{CEvm}, se puede calcular con la Ecuación 2, donde t_{Cn-1} es la marca de tiempo en *SyncRoot* del punto de sincronización registrado en t_{ln-1}, y *T* es el período de transmisión de los puntos de sincronización.

$$t_{CEvm} = t_{Cn-1} + \frac{t_{Evm} - t_{ln-1}}{t_{ln} - t_{ln-1}} T \qquad \textbf{Ecuación 2.}$$

Como se indicó anteriormente, el *SyncRoot* numera secuencialmente las señales de sincronización generadas y las registra con el formato "<número>, <tiempo>" (*####, hhmmss.µssec*) para el procesamiento posterior de las trazas. El primer campo de este formato (####) se refiere al número de secuencia de la señal en notación hexadecimal, que si bien se busca que sea único para cada señal en un experimento –dado que se tendrían hasta 2^{16} números–, podría volver a empezar en cero cuando se alcance su número máximo. El tiempo incluye horas (hh), minutos (mm), segundos (ss) y seis dígitos con los microsegundos (µsec). El número de secuencia de la señal de sincronización se envía a los nodos, que registran el evento de sincronización en sus trazas –con su número de secuencia– como un punto de sincronismo, incluida su marca de tiempo local. En un funcionamiento normal, dos eventos de sincronización consecutivos en un nodo de monitor corresponden a dos señales de sincronización en *SyncRoot*, por lo que se puede aplicar la Ecuación 2. Sin embargo, si un NM o un NS pierde uno o más puntos de sincronización, la

sincronización de trazas puede aproximarse mediante la Ecuación 2, utilizando los dos puntos de sincronización más cercanos (antes y después) de cada evento en la traza. En este caso, el valor de *T* en la Ecuación 2 deberá corresponder con la diferencia de tiempo entre estas dos señales de sincronización en el *SyncRoot*.

3.3.4.2 Consideraciones del mecanismo

En trabajos anteriores (Navia et al. 2016) se ha demostrado que en sistemas empotrados autónomos la velocidad del reloj y la deriva no son estables a lo largo del tiempo. Por tanto, el valor del período de transmisión de señales de sincronización (*T*) debe ajustarse a las características de los relojes locales. Como se analizó en (Navia et al. 2018), un menor periodo ofrece mejores prestaciones, pero incrementa la sobrecarga de la monitorización. Por tanto, más adelante se analiza cómo *T* puede aumentarse o disminuirse según la calidad de la fuente de reloj local y la precisión requerida por la monitorización.

Para la implementación de la plataforma HMP se ha optado por, además del almacenamiento de memorias SD, la utilización de una red de monitorización (Ethernet) que transmite las señales de sincronismo mediante tramas de tipo difusión (*broadcast*). La solución propuesta supone que las señales de sincronización requieren un tiempo de propagación muy pequeño, y prácticamente igual para alcanzar todos los nodos de la plataforma. También se considera que los tiempos de acceso al medio y de recepción son constantes y despreciables frente al resto de tiempos. Esto es factible porque la red de supervisión secundaria tiene solo un transmisor (*SyncRoot*) para todos los mensajes del punto de sincronización, y todos los receptores usan el mismo hardware. La utilización de concentradores –*hubs*– sin almacenamiento permite suponer que la red Ethernet se comporta como un medio compartido de difusión, y la llegada del mensaje es prácticamente igual en todos los receptores (nodos). Esta electrónica de red no agrega retrasos adicionales debido al almacenamiento en búfer o conmutación de la red. Además, se ha asignado la más alta prioridad en los nodos a los eventos de sincronización (llegada de tramas con señales de sincronización). A diferencia de otras propuestas (Maróti et al. 2004; Ganeriwal et al. 2003; Akhlaq & Sheltami 2013) no resulta necesario modificar las tramas a nivel de MAC. Finalmente, la red secundaria se puede usar también para recopilar todas las trazas en el SM al finalizar la monitorización.

GTSO incluso puede expandir su funcionamiento a una estructura de árbol. En una plataforma jerárquica, cada nodo puede actuar como dispositivo *SyncRoot* para sus nodos hijos. En esta situación, un nodo Monitor recibiría y procesaría las trazas de monitorización de los nodos debajo de él, y luego transmitiría la traza conjunta a los nodos de nivel superior, hasta que las trazas lleguen a la raíz del árbol. En este caso, se requiere la adición de un indicador de nivel para los puntos de sincronización, tal como lo hacen otros mecanismos.

3.4. Conclusiones

En este capítulo se ha presentado la propuesta de la plataforma de monitorización híbrida denominada HMP. El objetivo de esta plataforma es combinar las ventajas de los enfoques de monitorización activo y pasivo, para la evaluación del funcionamiento de una WSN.

El componente activo –el Nodo Monitor– tiene una estructura híbrida –hardware y software–, y registra los eventos del nodo monitorizado con un enfoque de baja intrusión.

El nodo *Sniffer* –el componente pasivo de HMP– captura las transmisiones realizadas por los nodos de la WSN monitorizada, complementando la información capturada por los Nodos Monitores. Ambos componentes emplean el mismo formato de datos, es decir, una estructura predeterminada para las trazas –en formato XML–, así como marcas de tiempo de los eventos o mensajes registrados.

El Servidor de Monitorización realiza las tareas de procesamiento y sincronización de los datos obtenidos por la plataforma. También integra las funciones del *SyncRoot* –enviar los mensajes de señales de sincronización–, aunque esta función podría ejecutarse en otro equipo por separado.

El mecanismo de sincronización de trazas desarrollado para la plataforma –denominado GTSO– es de tipo *offline*, y ejecuta la sincronización de las trazas al finalizar la monitorización, después de haber reunido la información de cada componente de la plataforma. Es un mecanismo relativamente sencillo pero efectivo, tal como se demostrará en el siguiente capítulo.

Capítulo 4

Evaluación y pruebas

Este capítulo presenta la evaluación de la plataforma de monitorización híbrida y las pruebas realizadas con el fin de caracterizar y validar su funcionamiento.

A este respecto, dos de los aspectos a considerar son el estudio del nivel de intrusión y la validez de la unificación temporal de las trazas.

Uno de los principios básicos de todo proceso de medida es que el posible error cometido en este proceso sea mínimo. En este sentido, resulta deseable que la inclusión del sistema de monitorización no altere de forma significativa el comportamiento del sistema a monitorizar. Puesto que este objetivo puede resultar imposible de cumplir en ocasiones, al menos es imprescindible que el error introducido por el proceso de medida sea cuantificado y se añada a la incertidumbre de la misma. Es por ello que se han realizado estudios detallados de la intrusión generada por la plataforma sobre la WSN a observar.

Como se ha indicado en el apartado 3.3.4, la temporización de los eventos en los diferentes nodos y sus requisitos de precisión dependen de las características del sistema a observar y de las capacidades de los NM y NS. El algoritmo GTSO propuesto en dicho apartado deberá ajustarse en función de la implementación hardware.

Por todo ello se ha considerado necesario reevaluar los estudios previos cuando se aplican a una implementación real de la plataforma HMP.

Este capítulo comienza centrándose en el nodo monitor, para a continuación estudiar con más detalle el mecanismo de sincronización de trazas. Finalmente, la información obtenida desde los nodos monitores se complementa con la procedente de los Nodos *Sniffer* –cuyo funcionamiento es análogo al de un Nodo Monitor–, de forma que la última sección se centra en un ejemplo de uso mediante una prueba de plataforma completa aplicada a una monitorización real.

4.1. Evaluación de Nodo Monitor

Como ya se ha indicado en el Capítulo 2, existen varias propuestas que introducen los Nodos *Sniffer* para la observación de WNS. Una de las principales aportaciones de la plataforma HMP propuesta consiste en poder contrastar la información así obtenida con los procesos internos de los nodos de la WSN a monitorizar. Por ello, el primer paso en el diseño de la plataforma HMP consistió en poner en marcha los mecanismos que permiten la obtención de información interna de los nodos a observar.

Como se ha indicado anteriormente, este proceso conlleva tres tareas fundamentales: Introducción de las *traps software* en los nodos a monitorizar, estudio del soporte físico al interfaz Mon-Inf y diseño e implementación del hardware del Nodo Monitor. En este apartado se describen estos procesos, a la par que se incluyen los resultados obtenidos.

4.1.1. Monitor software en los nodos sensores

Como ya se ha dicho, la inclusión de las sondas de observación (*probes*) en los nodos sensores a monitorizar requiere la inclusión de pequeñas rutinas, denominadas *traps*, que actúen como puntos de traza del estado interno del sistema observado. Mediante estas *traps* se consigue que cuando el código ejecutado en el nodo sensor alcanza determinado punto, se informe de ello al exterior a través de algún interfaz de comunicaciones.

En muchos casos, las *traps* pueden hacer llegar al nodo monitor información de estado adicional, como puede ser el número de secuencia del paquete transmitido, el valor proporcionado por el sensor o la cantidad de energía residual al despertar el nodo. Por ello, es necesario permitir la transmisión de datos, generalmente en pequeño volumen, tras el código de *trap*.

La implementación de las *traps software* en el nodo monitorizado se realizó mediante una librería que contiene las funciones de inicialización de interfaz y de paso de datos al NM. El fichero de librería *monitor.h* es genérico para cualquier nodo monitorizado; mientras que la implementación de las funciones de esta librería –el fichero *monitor.c*– depende de la WSN monitorizada.

La Figura 35 muestra un ejemplo de la librería agregada a los nodos monitorizados. Parte del contenido de esta librería –concretamente la definición de los códigos de evento– ya ha sido mostrada previamente en la Tabla 1, en la sección 3.3.1.

```
/*****************************************************************
 * @file   monitor.h
 * @author  Marlon Navia Mendoza - ITACA – UPV
...
*/
/*Librería de definición de tipos*/
#include "common_defs.h"
/* Exported constants --------------------------------------------------------*/
/*Códigos de eventos*/
#define Log_Reset     0x00
#define Log_Sense0    0x01
#define Log_Sense1    0x02
#define Log_TxData    0x03
#define Log_TxRoute  0x04
#define Log_RxData    0x05
...
/* Declared functions ------------------------------------------------------*/
void InitMonPort(void);
void WriteLog(uint8_t DataLong, uint8_t code, char msg[]);
/**** EOF ****/
```

Figura 35. Contenido de la librería monitor.h.

4.1.2. Prototipos de Nodos Monitores

Uno de los primeros estudios que se realizaron al inicio de este trabajo profundizaba en la integración entre el monitor software basado en *traps*, y el nodo monitor encargado de recoger estas *traps* de la forma más eficiente y, simultáneamente, menos costosa para el nodo observado. El estudio se centró en analizar y caracterizar el uso de las distintas interfaces que pueden resultar útiles para la comunicación de estas *traps*. Este análisis es parte fundamental de los estudios de intrusión de la herramienta de monitorización.

En un primer prototipo la investigación se centró en el funcionamiento básico del mecanismo de captura, por lo que no se incluyeron mecanismos de sincronización ni un formato estándar XML para almacenar los datos. Este primer prototipo está basado en un microcontrolador STM32F051 –un ARM Cortex M0– incorporado en una placa STM32F0Discovery (STMicroelectronics n.d.), que almacena la información capturada en una memoria SD. Las pruebas realizadas con este dispositivo fueron publicadas en (Navia, Campelo, et al. 2015). Una imagen de este primer prototipo se puede observar en la Figura 36. A la izquierda de la imagen se muestra el componente hardware del NM, mientras que a la derecha se aprecia el nodo sensor monitorizado.

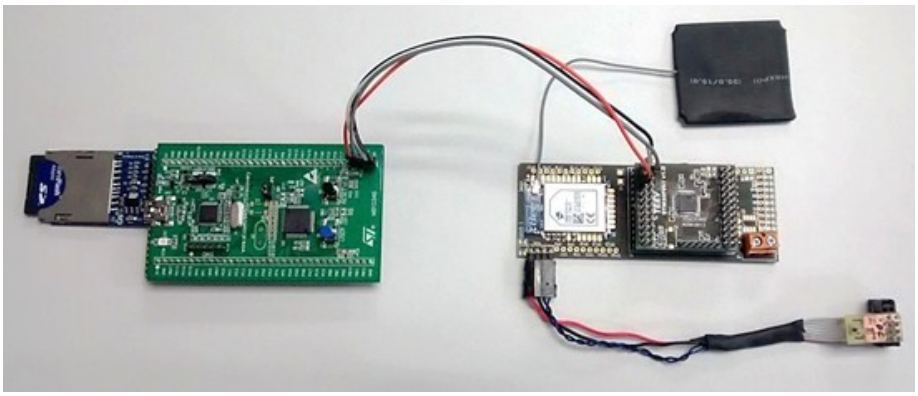


Figura 36. Fotografía del primer prototipo del NM

Tras este primer prototipo y la experiencia adquirida en los estudios realizados, se pasó a realizar una implementación definitiva de nodo monitor. En éste ya se integraron todos los mecanismos descritos anteriormente (Capítulo 3). Se decidió recurrir a un microcontrolador más potente, el STM32F407 –un ARM Cortex M4– incorporado en la placa de evaluación STM32F4Discovery (STMicroelectronics n.d.), junto con una placa de expansión STM32F4DisBB (Embest Technology Co. 2012). Esta placa de expansión incorpora, entre otras funcionalidades, una memoria micro-SD para el almacenamiento no volátil de la traza generada, así como conectividad Ethernet. Esta capacidad se empleó para proporcionar una red secundaria de monitorización. Esta red proporciona soporte al protocolo de sincronización –envío de las señales de sincronización– y permite la comunicación de las trazas capturadas hacia el servidor de monitorización. La Figura 37 muestra el hardware definitivo del NM –a la izquierda de la figura– conectado a un nodo sensor.

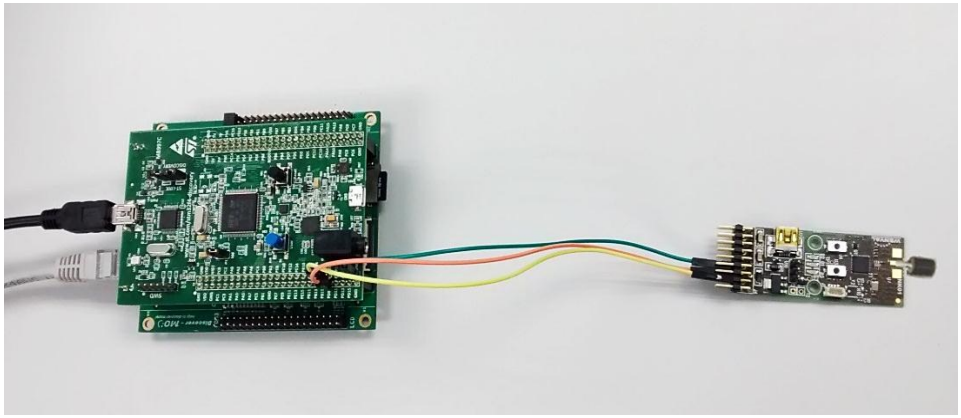


Figura 37. Fotografía del NM incorporado a la plataforma

La programación del funcionamiento del NM se realizó tal como se describe en el capítulo anterior. Con el objetivo de evaluar la intrusión generada por el NM sobre el nodo monitorizado, se procedió a caracterizar la sobrecarga producida por las interfaces más comunes que se pueden encontrar en el ámbito de los sistemas empotrados que constituyen los nodos habituales en las redes de sensores. Por ello, se evaluó tanto la interfaz en paralelo como las interfaces serie UART y SPI. Se habilitó el almacenamiento en la memora micro-SD, así como la comunicación Ethernet, junto con la pila de protocolo LwIP (*lightweight IP*) adaptada para el microcontrolador seleccionado.

4.1.3. Análisis de la intrusión

Como se ha puesto de manifiesto en capítulos anteriores, la monitorización pasiva, es decir, sin afectar al sistema observado, tiene limitaciones en cuanto a la observación de parámetros internos de funcionamiento. Es por ello que, para aumentar la información disponible sobre el sistema, la plataforma HMP propuesta incluye elementos de monitorización activa, las denominadas *traps* software, que hacen visible los eventos internos de los nodos a monitorizar. Sin embargo, esta funcionalidad implica restar recursos al sistema monitorizado. Esta merma de recursos es lo que se define como "intrusión". Esta intrusión puede modificar el comportamiento del sistema observado, y por tanto es imprescindible hacer un esfuerzo para, por un lado, minimizarla en el mayor grado posible, y por otro, cuantificarla para conocer la incertidumbre de las medidas obtenidas.

Generalmente, la monitorización puede requerir dos tipos de recursos al sistema observado: espacio en memoria y tiempo de proceso. Por ello, estos son los dos componentes de la intrusión que deben considerarse principalmente. En el entorno de las WSN, también resultan muy escasos los recursos energéticos. Sin embargo, la intrusión generada en este concepto está muy relacionada con la correspondiente a tiempo de proceso, aunque ha sido considerada expresamente en análisis anteriores (Navia, Campelo, et al. 2015).

Para estas pruebas se consideraron dos aplicaciones WSN distintas a monitorizar con el MN. En ambos casos se implementó una librería de monitorización específica para cada uno y se incluyó en el código de la aplicación. La intrusión en los nodos de la WSN depende fundamentalmente de cómo se implementen las *traps* software, y no tanto del componente hardware del NM.

El primer experimento se realizó con un nodo sensor dotado de un anemómetro con veleta, que se denominó Nodo1. Este nodo está implementado mediante un microcontrolador ARM Cortex M0 STM32F050 con una frecuencia de funcionamiento de 48MHz. El nodo envía las medidas obtenidas a través de una interfaz serie a un transmisor de radio conectado al mismo. Para proporcionar soporte al interfaz Mon-Inf, que comunica el nodo sensor con el NM, se consideraron tres

opciones: transmisión en paralelo (utilizando 16 pines GPIO agrupados), UART y SPI. Dentro de este experimento se evaluó la intrusión a nivel de código compilado, así como de tiempo de ejecución, y se constató que la intrusión a nivel de consumo energético está directamente relacionada a la intrusión en tiempo.

El segundo experimento consistió en la monitorización de un nodo sensor, denominado Nodo2, basado en el microcontrolador CC1110F32 de Texas Instruments, un microcontrolador modelo 8051 a 26MHz (Texas Instruments n.d.), que utiliza unas librerías abiertas, denominadas SimpliciTI (Texas Instruments n.d.), para proporcionar las funciones de comunicación. La terminología SimpliciTI considera tres tipos de nodos: Dispositivo final (ED), Repetidor o ampliador de rango (RE), y Punto de acceso o Recolector (AP).

El nodo evaluado, configurado como ED, dispone de un sensor de temperatura, mediante el cual obtiene un valor que envía cada cierto tiempo. Otro nodo –basado en el mismo hardware– adopta el papel de RE, y reenvía las medidas hacia un nodo recolector, configurado como punto de acceso (AP). Para este experimento se incluyó el código de monitorización en la librería SimpliciTI. La monitorización se realizó en los tres nodos implicados: Nodo sensor (ED), repetidor (RE) y recolector (AP).

Dado que cada nodo funciona de forma diferente, los eventos registrados en cada uno difieren. En el caso del nodo sensor se registran los cuatro eventos que se consideran significativos: transmisión de datos (*Transmission*), recepción de datos (*Reception*), error de transmisión (*Log_Error*) y despertar (*WakeUp*). Éste último evento no es aplicable al nodo reemisión, que permanece activo de forma continua, por lo que registra los eventos de transmisión, error en transmisión y recepción. Finalmente, el nodo recolector únicamente registra los eventos de recepción e inicio de funcionamiento (*Reset*).

4.1.3.1 Intrusión en código

Para determinar la intrusión o incremento a nivel de código se ha tomado como medida el tamaño en bytes del código compilado. Por ello se ha comparado el tamaño antes y después de incluir las funciones de monitorización. Puesto que el número de eventos registrados –es decir, el número de llamadas a *traps* introducidas– es relevante, se ha analizado el incremento para un número variable de eventos a observar.

En el nodo anemómetro (Nodo1) se obtuvieron los resultados de incremento a nivel de código que pueden verse en la Tabla 2. Es apreciable que la mayor parte del incremento se debe a las rutinas de comunicaciones e inicialización, por lo que, una vez incluidas, para cada evento se requiere añadir únicamente 10bytes. Lógicamente, el código de inicialización de las interfaces serie (sean SPI o UART) es

mayor, que el requerido en la inicialización de los pines GPIO para implementar una comunicación en paralelo.

En cualquier caso, el código necesario es muy pequeño en relación al de una aplicación típica. En el caso del nodo anemómetro el tamaño de la aplicación original no alcanzaba los 8KB, por lo que en el peor caso de los analizados en la Tabla 2 la intrusión en espacio se sitúa alrededor del 6%.

Tabla 2. Intrusión en código en el Nodo1 (valores en bytes)

Interfaz	Código original (no monitor)	Incremento para 2 eventos	Incremento para 3 eventos	Incremento para 4 eventos
Paralelo	8144	406 (4.99%)	416 (5.11%)	426 (5.23%)
SPI/UART	8144	478 (5.87%)	488 (5.99%)	498 (6.11%)

En el caso del segundo experimento (Nodo2), la Tabla 3 muestra los valores de la intrusión generada en los nodos con el protocolo SimpliciTI. Se efectuó la evaluación solo con la interfaz serie SPI debido a que el nodo sensor no dispone de suficientes pines libres como para realizar la transmisión en paralelo. La interfaz UART emplea las mismas funciones que la SPI para su configuración el envío de datos, solo que su velocidad máxima es menor que esta última.

En este caso el porcentaje de intrusión en memoria que se obtiene es similar al mostrado en la Tabla 2, esto es, alrededor del 6% para registrar 3 eventos. Se nota que el incremento de código en el Recolector es menor que en el nodo sensor y el nodo reemisor, como consecuencia de la distinta funcionalidad de este dispositivo. Por otra parte, porcentualmente, el incremento de memoria utilizada –mostrado en la columna *xdata*– es similar para todos los casos, y no alcanza el 5% en relación al espacio requerido por la aplicación original del nodo.

Tabla 3. Intrusión en código y memoria en Nodo2 (valores en bytes)

Nodo	Sin monitorización		Con monitorización		Incremento	
	Código	xdata	Código	xdata	Código	xdata
Sensor	14697	1423	15645	1489	948 (6.33%)	66 (4.64%)
Repetidor	14996	1421	15927	1487	931 (6.21%)	66 (4.64%)
Recolector	16602	1644	17231	1710	629 (3.79%)	66 (4.01%)

En base a los resultados obtenidos se puede generalizar una expresión para predecir la intrusión introducida sobre el nodo sensor a nivel de código. Esta fórmula se muestra en la Ecuación 3, donde $S_{intrusion}$ es la intrusión generada a nivel de código en bytes por el monitor, S_{init} es el tamaño del código en bytes necesario para la función de inicialización y para transmitir un solo evento, a es el tamaño en by-

tes del código necesario para añadir un nuevo registro de eventos, y *n* es el número de eventos que se quieren registrar. Los valores de S_{init} y de *a* pueden variar de un nodo monitorizado a otro. En el caso del Nodo1, *a* sería igual a 10, mientras que en el Nodo2 sería 17. Esta diferencia se debe a que son distintas arquitecturas de microcontroladores, y distintos los compiladores que se utilizaron.

Ecuación 3.

$$S_{Intrusion} = S_{init} + a \times (n - 1)$$

Con estos valores se podría estimar la intrusión a nivel de código, generada por en NM. En la Figura 38 se muestra la estimación de la intrusión a nivel de código tanto para el Nodo1 como para el Nodo2. Los valores de la intrusión se calcularon con la Ecuación 3 y con los valores obtenidos en las mediciones previas. Se estimaron el registro de 8 eventos distintos en cada caso. No se muestra el valor obtenido en el Nodo2 para el tamaño de la aplicación de 8KB, debido a que en este nodo el protocolo de comunicación está añadido al de la aplicación, por lo que es muy difícil obtener un tamaño de código menor a 12KB.

Aunque para los valores más altos de la Figura 38 la intrusión puede superar un 6% u 8%, en los casos del Nodo1 y Nodo2 respectivamente, aún siguen siendo relativamente bajos, ya que para el peor caso de intrusión en el Nodo1 –con la configuración indicada– es de apenas 538bytes, mientras que en el Nodo2 es de 1016bytes. Incluso en este último caso, la cantidad de código compilado es mucho menor que el que necesitan otras propuestas. Por ejemplo, Sympathy necesita 1558 bytes de ROM en TinyOS (Ramanathan et al. 2005), EnviroLog puede llegar a necesitar 15160 bytes (Rodrigues et al. 2013), y en las pruebas de Lightweight Tracing se requirieron al menos 4KB (Sundaram et al. 2009).

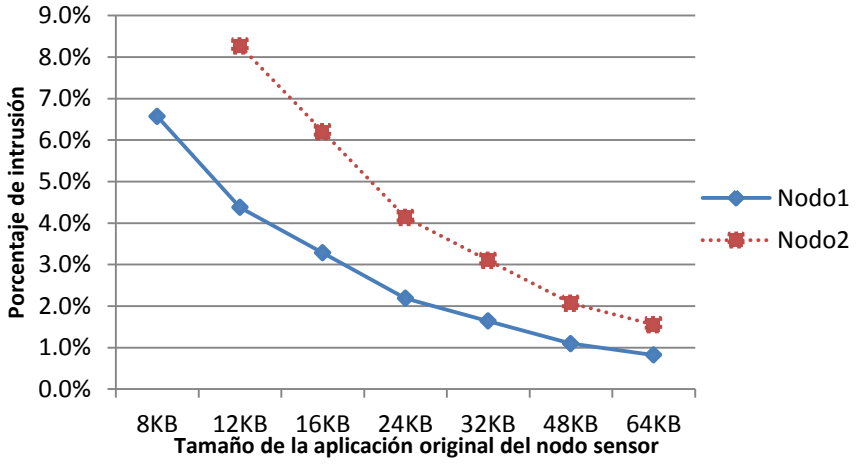


Figura 38. Estimación de intrusión a nivel de código

4.1.3.2 Intrusión en tiempo

Para evaluar la intrusión en tiempo se midieron de forma experimental las diferencias de tiempo de ejecución en ambos nodos sensores de experimentos anteriores (Nodo1 y Nodo2). Para ello, se modificó brevemente la aplicación de los mismos, añadiendo la configuración de una línea de salida en cada uno de ellos de forma que su valor se activa (pasa a 1) al iniciarse la función *trap*, y se desactiva (pasa a 0) tras la ejecución de dicha instrucción. La medición de estos tiempos se realizó con un osciloscopio digital, modelo DSO6014A de la marca Keysight, que según sus especificaciones tiene una capacidad de hasta 10Giga muestras por segundo, con un error en las medidas de tiempo de ±0.005% en la lectura ± 0.1%del ancho de la pantalla del osciloscopio (Keysight Technologies 2017).

Se ha contrastado que los tiempos de ejecución de las *traps* son dependientes del tamaño de datos adicionales que debe registrar el NM. Por tanto, se consideraron diversos casos: una *trap* sencilla que únicamente envía su código (1 byte), una *trap* extendida (2 bytes) y una *trap* extendida con 16 bits adicionales (4 bytes).

Al igual que en el caso anterior, se han considerado los interfaces disponibles en los microcontroladores. El Nodo1 permite la transmisión por UART, por SPI y en paralelo mediante GPIO, mientras que en el Nodo2 únicamente se encontraba disponible el interfaz serie SPI.

La Tabla 4 muestra los resultados obtenidos para el Nodo1. Se puede apreciar que para la transmisión en paralelo utilizando GPIO los valores son ligeramente superiores, como consecuencia del protocolo de acuse de recibo (*handshake*) empleado, implementado en las propias rutinas de la librería de monitorización. Por otra parte, al utilizar interfaces serie estándar –como SPI o UART– el tiempo de transmisión no se incrementa notablemente en relación a la cantidad de bytes transmitidos. Esto es debido a la existencia de controladores específicos de comunicaciones para estos interfaces que hacen uso de DMA, para descargar al procesador principal de estas tareas. Este paralelismo justifica que los tiempos de ejecución de la *trap* medidos para UART o SPI son prácticamente los mismos, e incluso los controladores prosiguen su funcionamiento aunque la unidad de proceso pasé a modo de bajo consumo (*sleep*). Asimismo, aunque históricamente los canales SPI soportan mayores velocidades de transmisión que los canales UART, es cada vez más habitual que ambos se puedan configurar a velocidades similares de varios Mbps, con lo que el tiempo de comunicación es prácticamente equivalente. Sólo cuando el canal serie UART mantiene los valores estándares –hasta 115.2 kbps – sería apreciable la mayor velocidad del SPI, y por tanto implicaría una mejora sustancial en el tiempo de transmisión. Por tanto, el canal SPI se muestra como la mejor alternativa como soporte al interfaz Mon-Inf entre el nodo sensor y el NM.

Tabla 4. Intrusión en tiempo en nodo anemómetro (valores en microsegundos)

Interfaz	1 byte	2 bytes	4 bytes
Paralelo	3.7	3.8	7.3
SPI/UART	3.2	3.3	3.4

Para ponderar de forma adecuada estos tiempos es necesario compararlos con el ciclo de trabajo del nodo sensor, y particularmente con el tiempo de trabajo en que se encuentra activo. Para esto, se tomó como base los resultados de la Tabla 4, respecto al primer nodo sensor donde se evaluó el NM. Se tomaron como referencia el registro de 8 eventos, con una transmisión de 8 bytes cada uno (uno de código, uno de longitud de datos, y seis de datos adicionales). Este tamaño de datos intenta reflejar la información que se puede obtener de una aplicación típica de redes de sensores. Los tiempos para transmitir los 8 bytes se determinaron en base a los valores de la Tabla 4. Se tomaron varios posibles ciclos de trabajo del nodo sensor, con tiempos activos entre 10 y 70 milisegundos. La Figura 39 muestra los resultados de la intrusión con estas estimaciones. Para el peor caso –10ms de tiempo de trabajo, utilizando comunicación en paralelo con GPIO– la intrusión en tiempo apenas supera el 1%. A partir de 30ms no llega al 0.4%, y utilizando una interfaz serie (UART o SPI) y DMA no pasa del 0.3% en los casos analizados.

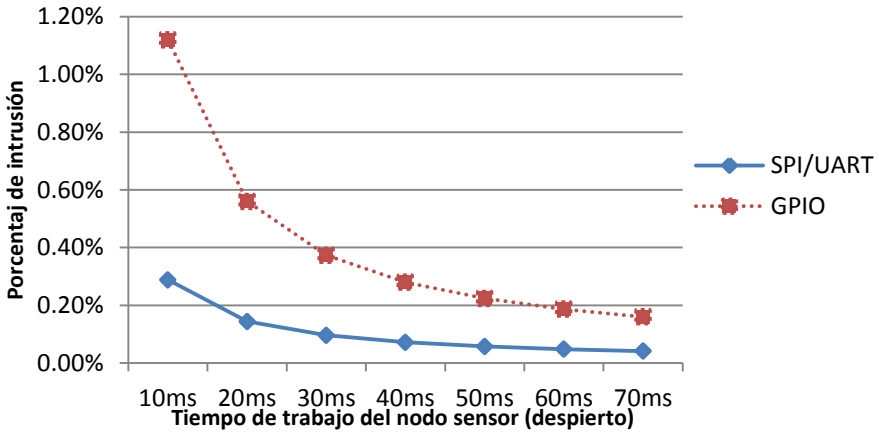


Figura 39. Estimación de intrusión a nivel de tiempo de trabajo

Los valores de tiempo de trabajo del nodo con los que se estimaron los valores de la Figura 39 están de acorde a aplicaciones reales. Por ejemplo, Gharghan et al. (2014) presentan un nodo inalámbrico para medir el rendimiento en una bicicleta de pista, con un tiempo de trabajo del nodo de 30ms. Otros casos como el de Molina-Garcia et al. (2012) –monitorización de cargas de enfriamiento y calenta-

miento ambiental– o de Lee et al. (2012) –un nodo sensor inalámbrico activado por alimentación inalámbrica–, el tiempo en que el nodo pasa activo puede llegar a los 100ms. Por tanto, en estos casos reales la intrusión en tiempo no alcanzaría más de 1.2% para el peor caso.

4.2. Evaluación de resincronización de trazas

Para sincronización de las trazas generadas por los diversos componentes de la plataforma, éstos implementan el mecanismo GTSO descrito en el Capítulo 3. Este mecanismo, que puede ser implementado en cualquier entorno que requiera sincronizar y procesar de trazas *offline*, se ha adaptado al funcionamiento de la HMP.

Se han considerado dos parámetros importantes a la hora de validar el correcto funcionamiento del mecanismo. En primer lugar, el objetivo es evaluar la precisión de tiempo del mecanismo, mientras que en segundo se establecen las condiciones necesarias para garantizar la corrección de la secuencia temporal de los eventos adquiridos.

En estas pruebas, se compara GTSO con uno de los enfoques de sincronización más sencillo, y aun así bastante utilizado, que denominaremos de "ajuste de reloj interno". Este mecanismo se basa difusión de mensajes de sincronización que obligan a todos los dispositivos a modificar su reloj para ajustarse instantáneamente al valor contenido en el mensaje. En este método no se calculan ni se estiman la deriva o el desfase de reloj para ajustar las marcas de tiempo.

Para la valoración de las características de GTSO, se han propuesto varios escenarios de prueba con nodos monitores reales. Tras los diferentes experimentos, se obtuvo un conjunto de trazas, que fueron tratadas con ambos métodos para comparar los resultados. Se han considerado tanto el error medio como su varianza como criterios de calidad de los ajustes.

4.2.1. Configuración y hardware utilizado

La generación de señales de reloj en los sistemas empotrados se basa en un circuito resonante, que puede estar dotado de un cristal de cuarzo para añadir estabilidad. Cada nodo monitor (NM) incluye un Reloj de Tiempo Real (RTC) que permite la generación de un reloj. Los microcontroladores actuales que disponen de un RTC pueden emplear más de una fuente de reloj. La precisión de las marcas de tiempo generadas por los NM depende de esta fuente (oscilador con o sin cristal, interno o externo).

Para evaluar GTSO se utilizó el NM desarrollado previamente. En el caso del hardware del NM, la fuente de reloj RTC puede ser interna, en este caso un cristal Interno de Baja Velocidad (LSI), o externa, un cristal Externo de Alta Velocidad

(HSE). El usuario puede seleccionar una u otra de acuerdo a las necesidades de implementación. La posibilidad de tener más de una fuente para el RTC es algo común actualmente en los sistemas empotrados comerciales. En el hardware NM empleado, el cristal externo proporciona una mejor precisión que el LSI interno. Usando el cristal externo, la frecuencia más alta que se puede aplicar al RTC es igual a 1MHz. Con esta frecuencia, el menor ciclo o *quantum* de tiempo del reloj es igual a 40μs, debido a las características del microcontrolador. Por otro lado, cuando se usa LSI (un oscilador de 32 KHz) como fuente de tiempo, la deriva es mayor, la estabilidad de la frecuencia del reloj es peor y por ello se considera que no es adecuada para aplicaciones que requieren diferenciar en el tiempo eventos que ocurren con una diferencia por debajo de un milisegundo.

Las pruebas realizadas previamente (Navia et al. 2016) demuestran que la fuente de reloj más adecuada para la generación de marcas de tiempo es el HSE, en la medida en que proporciona un período más pequeño y una frecuencia de reloj más constante. Este aspecto ya ha sido explicado previamente en el Capítulo 2. Sin embargo, también sería posible usar el LSI como fuente de tiempo para monitorizar entornos donde una precisión de milisegundos podría ser aceptable, como se explica más adelante. En cualquier caso, con el objetivo de evaluar el funcionamiento de GTSO se ha considerado interesante realizar también el estudio con el LSI, es decir, ver en qué márgenes GTSO seguirá ofreciendo un comportamiento adecuado.

En la Figura 40 se presenta el esquema del hardware utilizado en las pruebas. Resulta muy similar a una campaña de monitorización, excepto por el hecho de que las *probes* software no son reales, sino que son simuladas por un dispositivo que generará señales equivalentes a las generadas por los eventos que deben ser registrados por los NM.

Como puede apreciarse, los NMs se conectan a una red Ethernet que también incluye un computador. Este computador actúa tanto como Servidor de Monitorización (SM) así como de servidor de sincronización (*SyncRoot*). El *SyncRoot* envía señales de sincronización periódicamente cada *p* segundos. En este prototipo de pruebas la señal de sincronización se ha implementado mediante una trama Ethernet de difusión, que llega simultáneamente a todos los NMs

Para evitar el retraso que puede introducirse por la operación de almacenamiento en búfer en las redes Ethernet conmutadas, se utilizó un concentrador – *hub* – Ethernet, aunque en pruebas posteriores con un *switch* Ethernet los resultados no variaron significativamente. Cuando *SyncRoot* envía una trama de sincronización, el concentrador la transmite a través de todos sus puertos sin almacenamiento en búfer u otras demoras. Al usar este esquema, el retardo de extremo a extremo se mantiene constante y muy pequeño. Se verificó que *SyncRoot* solo estaba enviando tramas de sincronización y no había tráfico adicional en el sentido de SM a NMs.

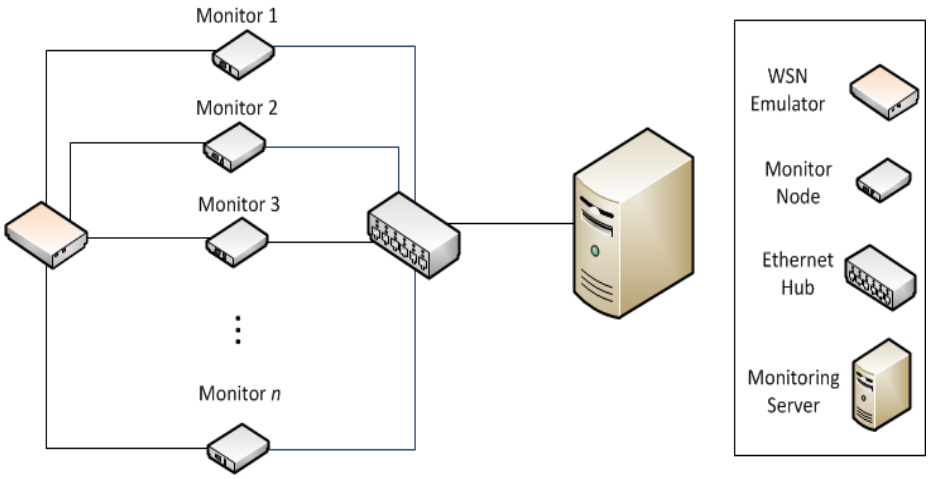


Figura 40. Arquitectura y componentes de la evaluación de GTSO

Se ha seleccionado una implementación basada en Ethernet del mecanismo para los experimentos de evaluación porque resulta muy útil, ya que la red se puede usar para tanto para enviar mensajes de sincronización a lo largo de la campaña de monitorización, como para la recopilación de las trazas al final del experimento.

Además de Ethernet, la señal de sincronización podría enviarse a través de líneas digitales –utilizadas en pruebas preliminares–, pero los resultados han demostrado que la elección no influye en la precisión de la propuesta, siempre que cumpla con los requisitos establecidos anteriormente respecto al tiempo de propagación y retardos. Sin embargo, se ha decido utilizar Ethernet, ya que esta situación se puede considerar más realista en un sistema de monitorización distribuido real.

Por otro lado, es cierto que este enfoque basado en Ethernet no siempre es adecuado para monitorizar una WSN ya desplegada, sino que es más adecuada para emplearla en condiciones de laboratorio. Sin embargo, se podría utilizar otro medio para enviar los mensajes de sincronización, como por ejemplo una red inalámbrica de difusión –un enfoque que ya aplican otras propuestas de sincronización (Li et al. 2011; Chen et al. 2014; Hao et al. 2014)– que cumpla con los requisitos mínimos de GTSO.

La WSN monitorizada se simula por medio de otra placa STM32F4-Discovery (en el lado izquierdo de la Figura 40). Este dispositivo (llamado *Emulador WSN*) reproduce una traza real obtenida de un conjunto de nodos WSN que ejecutaban una aplicación de monitorización de temperatura, generando los eventos que los NMs deben registrar. Para los experimentos de evaluación, cada NM está conectado al emulador de WSN por medio de líneas digitales. Se utilizó una línea por monitor para el primer experimento, evaluación de precisión, y dos líneas –una

para cada evento: Transmisión y Recepción– para el segundo, evaluación de secuenciación.

4.2.2. Evaluación de precisión

Sabiendo que la frecuencia del reloj no siempre es constante en un nodo y puede variar durante los intervalos de tiempo (Kim et al. 2012; Castillo-Secilla et al. 2013; Navia et al. 2016), los primeros experimentos se diseñaron para evaluar la precisión del mecanismo de sincronización. En este escenario, con el fin de evaluar el comportamiento de la plataforma cuando el mismo evento es registrado en varios NMs, el emulador de WSN se ha programado para simular un evento cada 8s y entregarlo como una señal digital a seis nodos Monitores diferentes al mismo tiempo. Con este propósito, se conectaron seis líneas digitales del mismo puerto de E/S a una línea de interrupción de los seis nodos del monitor. Dado que todas las líneas se activan con la misma instrucción de escritura en el registro interno del microcontrolador del emulador WSN, y que la longitud de los cables es la misma, no se espera ningún retraso entre los eventos detectados por los NM. Cada nodo de monitor crea su propia traza, registrando los eventos observados y la señal de sincronización con sus respectivas marcas de tiempo.

Se realizaron dos series de experimentos, utilizando como fuente del RTC tanto el cristal HSE externo como el oscilador LSI, y se compararon sus resultados. Como se ha indicado previamente, el objetivo de utilizar ambos cristales fue evaluar el funcionamiento de GTSO bajo diferentes circunstancias de calidad de reloj, no solo en cuanto a deriva sino también en cuanto a variabilidad de la misma. Las limitaciones del microcontrolador proporcionan una granularidad de 40μs por ciclo cuando se utiliza el HSE, y 125μs cuando se utiliza el LSI, aunque como se indica en (Navia et al. 2016) la variabilidad del *clock rate* del LSI es muy superior a la del HSE.

Para la evaluación de la precisión del mecanismo sobre la traza corregida respecto al periodo de sincronismo, es decir, cada cuánto se inserta una marca de tiempo, se consideraron varios valores de este parámetro, desde 5s hasta 600s (5, 10, 30, 60, 120, 180, 240, 300, 450 y 600s). Cada prueba se ejecutó durante aproximadamente 60 minutos, y se registraron más de 500 eventos en cada caso.

Posteriormente, los archivos de las trazas se enviaron al SM para la sincronización de las marcas de tiempo utilizando el algoritmo GTSO, empleando la Ecuación 2. En cada prueba se detectaron y registraron aproximadamente 500 eventos en cada uno de los seis nodos. Para cada evento generado, se promediaron las seis marcas de tiempo obtenidas en los distintos NM, y el error se calculó como la desviación de cada marca de tiempo frente a este promedio. El intervalo de confianza de la media de error se obtuvo mediante la Ecuación 4, donde S es la desviación estándar de los datos obtenidos para cada prueba, $\overline{x}$ es la media de las mediciones

(errores), α es el nivel de significancia (5%), y n es el número de mediciones en cada prueba:

$$\bar{x} \pm t_{\alpha/2} \frac{s}{\sqrt{n}} \qquad \textbf{Ecuación 4}$$

4.2.2.1 Precisión empleando el reloj externo

En el primer conjunto de pruebas de evaluación se usó el HSE como fuente del RTC, y sus resultados se reflejan en la Tabla 5. Como se puede observar en esta tabla, los períodos de sincronización de hasta 300s garantizan un error por debajo de la granularidad del RTC. Esto significa que los errores registrados no causarían ninguna diferencia en las marcas de tiempo de la traza. No se asegura que para períodos de sincronización superiores a este valor la deriva sea más o menos constante a lo largo del período, y que por lo tanto el error de estimación de deriva no sea despreciable. Esto puede introducir un error en el proceso de corrección de marca de tiempo y puede causar una incorrecta sincronización de la traza.

En la medida en que se pretende garantizar el correcto funcionamiento de la sincronización de las marcas de tiempo, los valores de período de sincronización superiores a 300s no se consideraron para las siguientes pruebas, a pesar de que los errores en las marcas de tiempo son muy escasos y podrían ser asumibles, como se muestra más adelante.

Tabla 5. Precisión de GTSO usando el HSE como fuente del RTC

Periodo	Menor error (µs)	Mayor error (µs)	Media (µs)	Mediana (µs)	Desviación estándar	Intervalo de confianza al 95%	Porcentaje menor o igual a 40µs
p = 5 s	0	23	10	11	5.31	10.11 ± 0.44	100.0%
p = 10 s	0	25	10	10	4.97	9.80 ± 0.42	100.0%
p = 30 s	1	24	10	10	4.63	10.22 ± 0.39	100.0%
p = 60 s	1	23	10	10	4.34	10.12 ± 0.37	100.0%
p = 120 s	0	27	11	11	4.64	11.09 ± 0.40	100.0%
p = 180 s	0	31	10	10	4.71	10.29 ± 0.40	100.0%
p = 240 s	1	27	11	10	4.85	10.56 ± 0.41	100.0%
p = 300 s	1	25	11	10	5.23	10.54 ± 0.43	100.0%
p = 450 s	0	55	12	11	7.02	12.05 ± 0.61	98.3%
p = 600 s	1	49	11	10	5.59	11.18 ± 0.48	99.4%

La Figura 41 muestra un gráfico de caja de la distribución de los valores de error promedio obtenidos en cada prueba de sincronización de trazas para los diferentes períodos de sincronización. El gráfico de caja ilustra el primer y el tercer cuartil –el rectángulo gris–, la media –una cruz roja dentro del rectángulo– y la mediana –la línea vertical azul en el rectángulo– como los principales parámetros estadísticos. Los valores atípicos se representan con cuadrados pequeños fuera de la línea. Cuando estos valores atípicos afectan la media, por ejemplo, durante períodos de 450s y 600s, se agrega una pequeña cruz dentro del cuadrado.

Como se esperaba, hay una tendencia a que aparezcan mayores valores de error cuando se usan períodos de sincronización largos, algunos de ellos por encima de la duración del ciclo de reloj. Sin embargo, aun con un período de sincronización que puede considerarse muy largo (600s), los resultados de error obtenidos pueden considerarse aceptables para aplicaciones con una restricción de precisión media. La mediana y la media apenas se ven afectadas por el período de sincronización. No obstante, a medida que aumentan los períodos de sincronización, se detectan valores de error más altos. La explicación a este fenómeno reside en las variaciones de la deriva de reloj, que causa que aparezcan valores atípicos, pero estos no afectan significativamente la precisión del mecanismo. El error promedio del mecanismo propuesto, cuando el período de sincronización no supera los 300s se puede determinar en 10±1μs.

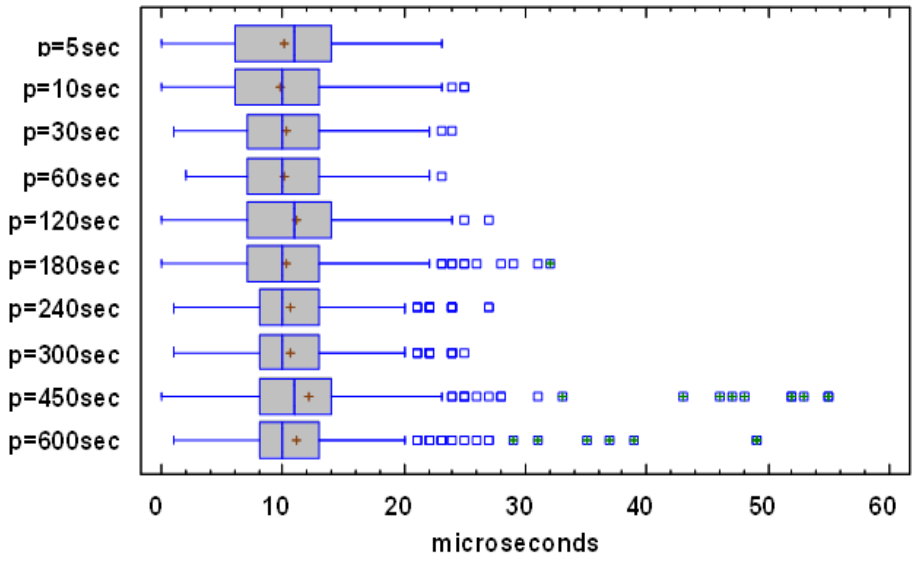


Figura 41. Gráfico de Caja de los valores obtenidos en la prueba de precisión con el HSE

Tal como se indicó anteriormente, con el fin de poner de relieve las ventajas del mecanismo propuesto, se procesaron las mismas trazas aplicando el método de sincronización por ajuste de reloj interno, detallado anteriormente. Por ello, tras el mensaje de sincronización todos los nodos configuran su reloj interno con el mismo valor que el tiempo de *SyncRoot*, y por tanto las marcas de tiempo de los siguientes eventos se corrigieron en consecuencia. Se considera que no existe retraso de propagación relevante (se estima como despreciables los tiempos de propagación de las señales y acceso al medio).

La Tabla 6 muestra los resultados de esta prueba. Como antes, los valores (error medio) están en microsegundos y se obtuvieron como la desviación de cada marca de tiempo de los NM para cada evento registrado. Se puede observar que para el período de 5s los valores obtenidos son todavía aceptables, ya que el error medio es relativamente bajo (16μs) y los errores son iguales o menores que la granularidad de RTC; pero empeoran con períodos superiores a 10s. La media y la mediana para cada período de sincronización están muy cerca. Por lo tanto, este método de sincronización –que solo ajusta el reloj interno del nodo– no puede proporcionar la misma precisión para la sincronización de trazas que la propuesta presentada, incluso con una fuente de reloj altamente precisa como HSE, ya que la deriva de la frecuencia de reloj entre los puntos de sincronización no se compensa.

Tabla 6. Precisión del Ajuste de reloj interno usando el HSE como fuente del RTC

Periodo	Menor error (μs)	Mayor error (μs)	Media (μs)	Mediana (μs)	Desviación estándar	Porcentaje menor o igual a 40μs
p = 5 s	0	40	16	15	9	100.0%
p = 10 s	0	60	26	30	14	88.6%
p = 30 s	0	150	71	70	39	31.1%
p = 60 s	0	285	140	140	80	15.5%
p = 120 s	0	575	277	265	161	6.8%

La Figura 42 muestra la comparación gráfica del error medio entre el mecanismo de sincronización de trazas propuesto y el de ajuste de reloj interno. En esta figura se aprecia aún mejor las diferencias entre los valores medios de error presentados en las Tablas 5 y 6. Mientras que GTSO los valores de error medio están entre 10 y 12μs o menos, para el mecanismo de sincronización de reloj interno el error medio aumenta continuamente, llegando a exceder 250μs con un período de sincronización de 120s.

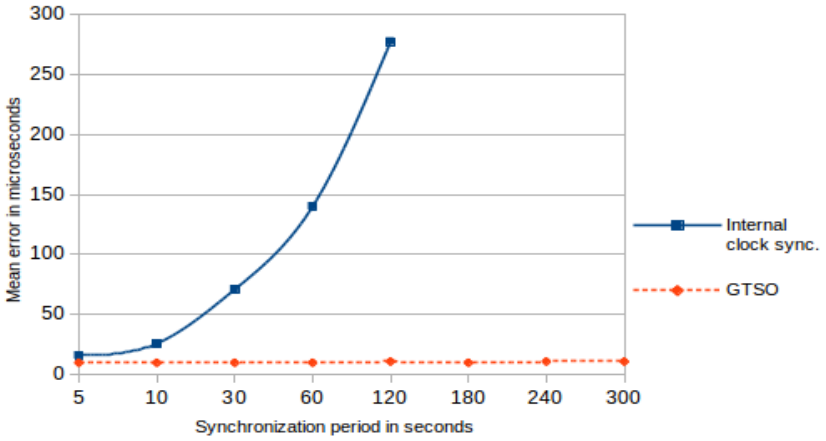


Figura 42. Comparación del error medio de precisión con el HSE

4.2.2.2 Precisión empleando el reloj interno

No en todos los casos de implementación de NM se podrá contar con un reloj externo con una buena estabilidad. En ocasiones, otros factores (coste, experiencia, disponibilidad, etc.) puede ser necesario implementar la plataforma empleando microcontroladores sin esta posibilidad, y que por tanto están limitados a su propio reloj interno. Este segundo experimento de precisión se realizó con el fin de evaluar el funcionamiento de GTSO en estas condiciones más adversas. Así, se han repetido el experimento anterior empleando en los NM una fuente de reloj de baja calidad, como el LSI incorporado (32 KHz). El uso del LSI como fuente de RTC causa una disminución de precisión y un aumento significativo de la variabilidad de la velocidad de ciclo de reloj. En estas circunstancias, como se describió anteriormente, la precisión máxima del microcontrolador que se puede obtener con este esta fuente es de125µs. Las condiciones de las pruebas fueron las mismas que las descritas anteriormente: el emulador WSN generó un evento cada 8s, y los experimentos se realizaron durante 60min con períodos de sincronización que varían desde p=5s hasta p=60s. Esta fuente de reloj no es lo suficientemente estable como para proporcionar una precisión mayor que a nivel de milisegundos, debido a que los fabricantes suelen incluir un cristal barato y de baja calidad, que no está orientado a altas prestaciones. Es por eso que los valores de error medio por debajo de un milisegundo –es decir 1000µs– se han considerado como satisfactorios.

La Tabla 7 muestra los resultados de las pruebas de precisión después de aplicar el mecanismo GTSO. Como se indicó anteriormente, el error se calculó como la desviación de cada marca de tiempo con la media de todos los nodos del monitor. Con el periodo de sincronización p=5s, los valores obtenidos pueden considerarse buenos, ya que el error estuvo por debajo de un milisegundo en casi todas las

mediciones, con un error promedio de aproximadamente 0.4ms. Para mayores períodos de sincronización, las variaciones en la deriva de los relojes se vuelven más visibles, y su efecto sobre la precisión es más relevante. Con el período p=10s, los valores empeoran ligeramente, pero el error medio permanece por debajo del milisegundo, así como dos tercios de los valores obtenidos. Para períodos de sincronización más largos, los resultados no son buenos, aunque permanecen en el rango de unos pocos milisegundos.

Tabla 7. Precisión de GTSO usando el LSI como fuente del RTC

Periodo	Menor error (µs)	Mayor error (µs)	Media (µs)	Mediana (µs)	Desviación estándar	Porcentaje menor o igual a 1000µs
p = 5 s	0	1204	397	363	238	98.40%
p = 10 s	41	2858	826	779	458	66.49%
p = 30 s	52	9353	2588	2460	1574	16.76%
p = 60 s	0	86917	5519	5242	4544	5.72%

Es claro que los resultados en la Tabla 7 son peores que los que se muestran en la Tabla 6. Esto se puede justificar debido a la baja calidad de LSI. Experimentos previos (Navia et al. 2016) han mostrado una deriva de hasta 10% cuando el LSI era la fuente de tiempo para el RTC. Sin embargo, esta podría ser la única opción cuando el hardware de los nodos de monitorización solo puede ofrecer este tipo de fuente RTC

La Tabla 8 muestra los resultados de la misma prueba cuando se aplica la sincronización de ajuste de reloj interno. En todos los casos, los valores medios de error obtenidos son de varios milisegundos, incluso de más de un segundo cuando el período aumenta a p=60s. Además, el porcentaje de marcas de tiempo que pueden considerarse sincronizadas –es decir con un error inferior a un milisegundo– fue menor que el 1% de todos los casos analizados. Aún más, la desviación estándar es muy alta para todos los casos, lo que indica una alta variabilidad de los datos obtenidos.

Tabla 8. Precisión del Ajuste de reloj interno usando el LSI como fuente del RTC

Periodo	Menor error (µs)	Mayor error (µs)	Media (µs)	Mediana (µs)	Desviación estándar	Porcentaje menor o igual a 1000µs
p = 5 s	0	303190	168688	169490	235771	0.71%
p = 10 s	6104	612083	320838	320885	176321	0.00%
p = 30 s	6990	1 867750	925267	935208	543465	0.00%
p = 60 s	0	3 685792	1 819828	1 820896	1 068465	0.36%

En la Figura 43 se muestra una comparación gráfica del error promedio de ambas pruebas usando el LSI como fuente de RTC, para diferentes períodos de sincronización. Estos resultados corroboran el hecho de que un enfoque de sincronización de ajuste de reloj interno es completamente imposible cuando los nodos monitores no ofrecen una fuente de tiempo de alta calidad para el RTC; mientras que el mecanismo GTSO para sincronización de trazas puede obtener resultados aceptables en este caso si el período de sincronización es bajo, alrededor de 5s. Sin embargo, no se recomienda el uso de este tipo de fuente de reloj, especialmente para aplicaciones que requieren una alta precisión o un grano de tiempo muy fino.

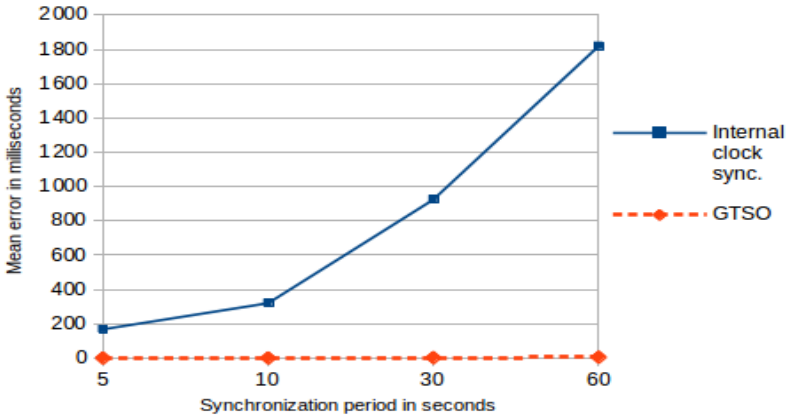


Figura 43. Comparación del error medio de precisión con el LSI

4.2.3. Evaluación de secuenciación de eventos

El objetivo de este estudio consiste en evaluar la exactitud de la secuencia de eventos de la traza final, especialmente cuando esta secuencia de eventos se registra en diferentes NM. Por ejemplo, considérese la transmisión de una trama entre un nodo sensor de origen A y un nodo sensor B de destino. Esta transmisión generará dos eventos relacionados en la plataforma de monitorización: el NM de A debe registrar un evento de *Transmisión*, mientras que el NM de B debe registrar un evento de *Mensaje Recibido*. Ambos eventos están relacionados con la misma acción –la transmisión de un mensaje específico– y deben estar temporalmente en orden: el evento de recepción en el nodo de destino debe registrarse en un corto período de tiempo –correspondiente a la latencia de transmisión– más tarde que el evento de transmisión en el nodo de origen.

Para verificar este aspecto se planteó un experimento en el cual la plataforma debe registrar la evolución de un mensaje en una WSN multisalto. Para ello se simula una comunicación de varios saltos desde el nodo 1 al nodo recolector. Los datos transmitidos se reenvían a través de los nodos 2 a 12. Este último nodo 12

finalmente entrega el paquete de datos al nodo recolector. Cada nodo *n* (del nodo 2 al nodo 11) genera dos eventos, que corresponden a la recepción desde el nodo *n-1* y la transmisión al nodo *n+1*, es decir registran dos eventos: *Recepción* y *Transmisión*. El nodo 1 solo generará el evento de transmisión, mientras que el nodo 12 solo registrará el evento de recepción.

El esquema de hardware de las pruebas es similar al presentado en la Figura 40: El Emulador WSN genera los eventos equivalentes a los doce nodos sensores que son registrados por los NM (Figura 44). El Emulador WSN ejecuta una traza sintética que simula el comportamiento del algoritmo de reenvío de múltiples saltos de una WSN. Esta traza sintética está basada en los eventos obtenidos de una campaña de captura real con nodos reales. La Figura 44 representa el comportamiento simulado de la WSN, es decir, la ruta multisalto desde el nodo 1, que genera la información, hasta el nodo 12 que la entrega al recolector.

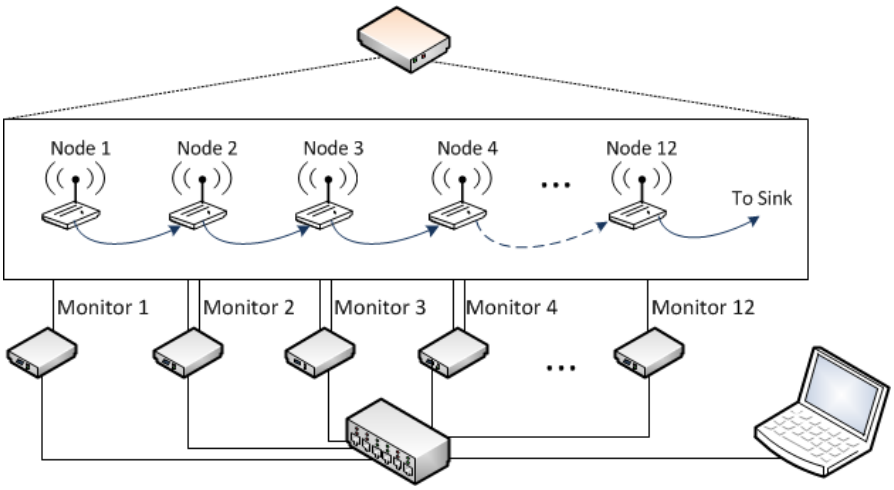


Figura 44. Esquema del segundo grupo de pruebas de GTSO

La traza sintética utilizada para los experimentos incluye un intervalo de tiempo de 480µs entre los eventos de transmisión y recepción. Este valor se obtuvo mediante la observación de una WSN real. Con estos tiempos el uso de LSI no se considera aceptable –debido a que están por debajo del milisegundo–, por lo que se utilizó como fuente del RTC el oscilador HSE. Se asume que podría introducirse un error de ciclo en la fuente de RTC en cada par de NM, uno positivo y uno negativo. Por lo tanto, la diferencia entre las marcas de tiempo del evento de transmisión (nodo *n-1*) y del evento de recepción (en un nodo *n*) debe ser de 480±40µs, ya que el HSE ofrece una granularidad de tiempo de 40µs por ciclo.

El experimento se realizó para diferentes valores del período de transmisión de las señales de sincronización. Los valores utilizados fueron p=30s, 60s, 120s,

180s, 240s, y 300s, es decir, los valores considerados en el primer grupo de pruebas. Cada escenario se simuló durante 120min, y se observaron más de 700 ciclos de operación de toda la WSN. Al final del experimento, todos los archivos de trazas se enviaron al SM, y se aplicó el método de sincronización GTSO. Para cada par de eventos relacionados en dos nodos diferentes, como son la transmisión de un mensaje y la recepción del mismo en el nodo siguiente, se calculó la diferencia entre sus marcas de tiempo. Se considera que los valores con correctos si esta diferencia es positiva –el segundo evento no puede ocurrir antes que el primero– y su valor debe estar cerca del valor simulado en la traza sintética (480μs).

La Tabla 9 resume los resultados de las pruebas de ordenamiento, después de haber aplicado el mecanismo GTSO. Aunque la mediana permanece constante en el valor óptimo (480μs), el error medio aumenta cuando el período de sincronización es mayor que *p*=180s, de forma similar a la desviación estándar y el rango del intervalo de confianza. El porcentaje de valores dentro del rango considerado aceptable (480±40μs) está por encima del 93% hasta un período *p*=120s, pero comienza a disminuir desde *p*=180s. La última columna de la Tabla 9 muestra el porcentaje de eventos ordenados erróneamente, es decir, el porcentaje de eventos que tienen un registro con un tiempo de *Recepción* menor que el tiempo de *Transmisión*. Una vez más, debido las variaciones en la frecuencia del reloj, existen valores que afectan el intervalo de confianza y los valores porcentuales en el rango de diferencia de tiempo aceptado.

Tabla 9. Resultados de evaluación de ordenamiento con GTSO

Periodo	Media (μs)	Mediana (μs)	Desviación estándar	Intervalo de confianza al 95%	Porcentaje en rango 480±40μs	Cambio de orden (%)
p = 30 s	481	480	22.31	480.96±1.01	93.48%	0.00%
p = 60 s	481	480	21.85	480.60±0.97	93.65%	0.00%
p = 120 s	482	482	27.29	482.03±1.21	93.35%	0.00%
p = 180 s	498	483	184.91	497±11.21	91.48%	0.00%
p = 240 s	518	482	483.88	517.69±22.06	90.23%	0.00%
p = 300 s	774	479	1848.42	774.40±129.97	85.71%	2.50%

A partir de estos resultados, es posible determinar que –para un funcionamiento adecuado de GTSO– el período de sincronización sea entre 60 y 180s, siendo el óptimo 120s. Con estos valores, se obtiene un error medio de 10±1μs. Si el sistema de monitorización permite una precisión menor, se puede usar un período ligeramente más largo, como se muestra en la Tabla 5.

La Tabla 10 muestra los resultados de la segunda prueba al aplicar el mecanismo de sincronización de ajuste de reloj interno. Con este método, solo con periodos de 30 y 60s, el error medio y la mediana se mantienen en valores cercanos a 480μs, pero el porcentaje de valores en el rango de 480±40μs no alcanza el 50%. A partir de p=120s, los valores de la media y la mediana se desvían del valor óptimo, el porcentaje de valores dentro del rango aceptable disminuye, y los valores del intervalo de confianza aumentan. En todos los casos, la desviación estándar es relativamente alta, en comparación con GTSO. Además, el elevado porcentaje de eventos ordenados erróneamente demuestra la inaplicabilidad de estos métodos de sincronización tradicionales basados en el simple ajuste de los relojes internos y que no estiman parámetros como la deriva o el desfase. Hay que destacar que aunque en la Tabla 10 se muestran valores para los que no hay o hay pocos cambios de orden, concretamente en las dos primeras filas, como se ha indicado previamente (Tabla 6) menos de la mitad de las medidas tienen la precisión adecuada.

Tabla 10. Resultados de evaluación de ordenamiento con Ajuste de reloj interno

Periodo	Media (μs)	Mediana (μs)	Desviación estándar	Intervalo de confianza al 95%	Porcentaje en rango 480±40μs	Cambio de orden (%)
p = 30 s	485	480	122.16	485.39±5.54	47.06%	0.00%
p = 60 s	490	480	269.01	490.09±11.92	32.23%	6.65%
p = 120 s	501	520	559.70	500.784±24.91	17.01%	13.20%
p = 180 s	710	520	746.77	710.21±44.91	12.52%	16.69%
p = 240 s	719	520	1140.44	719.36±117.73	12.70%	20.10%
p = 300 s	1446	440	11250.00	1446.24±682.09	11.96%	32.92%

Finalmente, la Figura 45 muestra la comparación en porcentaje de los errores de secuencia –es decir los cambios en el orden de los eventos– para ambos mecanismos. Cuando se aplica solo el ajuste de reloj interno, solo durante un período de 30s el porcentaje es cero, pero aumenta hasta alcanzar casi el 33% para el caso de 300s. Por otro lado, cuando se utiliza GTSO, el porcentaje es cero para casi todos los casos, con la excepción del periodo de 300s, donde un 2.5% de las veces la secuencia registrada no es correcta. En este caso, sería necesario utilizar un período de sincronización más corto para evitar este tipo de errores.

Esta conclusión no es exclusiva de este trabajo. De acuerdo con varias propuestas, la precisión de un método de sincronización global de trazas puede mejorarse al disminuir el período de sincronización, aumentando así el número de señales de sincronización en la traza. El inconveniente es que esto aumenta la carga de trabajo para ser asumida por los nodos (NM y NS, así como *SyncRoot*), y por tanto

el coste de implementación de estos dispositivos. Este enfoque ha sido utilizado por otras propuestas, especialmente para la sincronización en línea, como por ejemplo (Gong & Sichitiu 2015; Lenzen et al. 2015; Zou & Lu 2012), que envían los mensajes de sincronización con mayor frecuencia, incluso con un período de 1s. GTSO, que ha sido evaluado en una plataforma de real en condiciones de laboratorio, ha demostrado una buena precisión y eficiencia con una frecuencia de sincronización inferior –cuando se aplica la sincronización fuera de línea– a pesar de la variación de la deriva. Además, en GTSO el período de sincronización utilizado puede ajustarse en función de la calidad de la fuente de reloj RTC, especialmente su estabilidad en frecuencia. Por otro lado, GTSO no depende de los eventos transmisión-recepción para realizar la sincronización, como si lo hacen otras propuestas tales como las de Edison (2005); Hofman & Hilgers (1998); Becker et al. (2009). Esta ventaja permite sincronizar las trazas de los nodos de monitorización que no transmiten, por ejemplo, los nodos tipo *sniffer*.

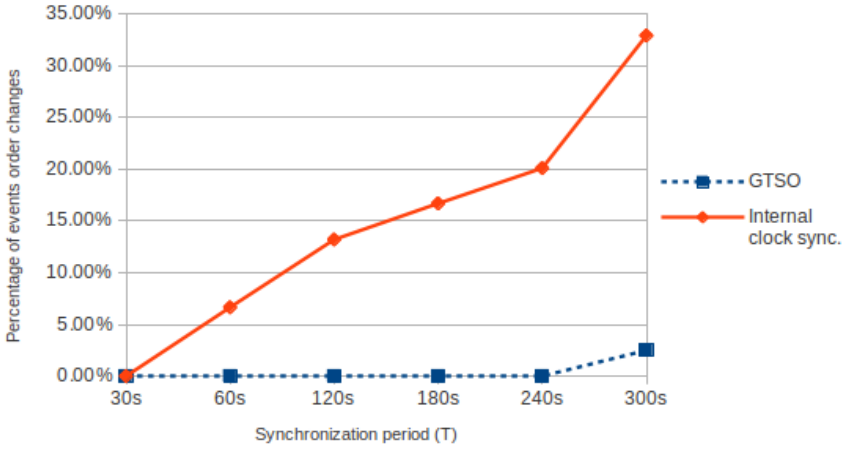


Figura 45. Comparación del porcentaje de errores en ordenación de eventos

4.3. Aplicación de la Plataforma sobre una WSN

Una vez evaluados tanto los NM como el mecanismo de sincronización de trazas, se ha procedido a evaluar el conjunto de la HMP sobre una WSN real. El interés de este punto es demostrar la funcionalidad de la propuesta realizada para un caso real. Para esto se emplearon los nodos que utilizan el protocolo SimpliciTI descritos anteriormente en el apartado 4.1.3.

4.3.1. Configuración de la WSN utilizada en la evaluación

Para este apartado se ha observado, mediante la plataforma HMP, la WSN ya empleada en el análisis de intrusión (Apartado 4.1.3), presentado como Nodo2.

Brevemente, el sistema consiste en un nodo sensor (ED) que periódicamente –cada 10 segundos– sale del modo bajo consumo, obtiene el valor de temperatura de un sensor y lo transmite hacia un nodo recolector (AP). Sin embargo, para alcanzar su destino el mensaje debe ser reenviado por un nodo repetidor o interpuesto (RE) que a su vez también actúa como nodo sensor (ED). Éste último dispositivo no pasa a modo bajo consumo para poder proporcionar continuamente los servicios de reenvío. Cabe destacar que el reenvío es transparente, por lo que el nodo sensor más alejado no es consciente del mismo para acceder al recolector. Todo este esquema descrito se muestra en la Figura 46.

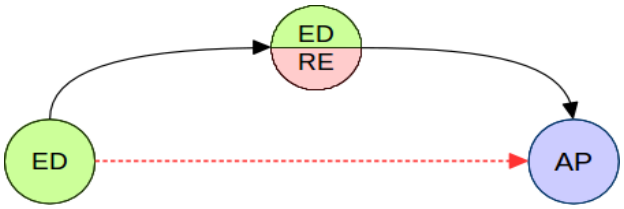


Figura 46. Esquema de la comunicación de la WSN monitorizada

La WSN opera en la banda de frecuencias de 868MHz, su velocidad de transmisión es de 250Kbps, y se emplean tramas SimpliciTI de 20Bytes. Por tanto, el tiempo de transmisión teórico de cada trama es de aproximadamente 320 microsegundos, sin contar los tiempos de acceso y procesamiento en emisor y receptor respectivamente. Aunque estos últimos tiempos –como se comprobará más tarde– no deberían trascender en los resultados de la monitorización

Se procedió a agregar la librería de monitorización correspondiente a los nodos de la WSN –incluido el recolector–, así como las funciones de registro de eventos respectivos – *traps* de software– dentro del código de cada nodo para cada evento a registrar: transmisión de datos (*Transmission*), recepción de datos (*Reception*), y despertar (*WakeUp*), este último únicamente en el nodo sensor. Para complementar la tarea de la plataforma, se añadió un nodo *Sniffer* con la misión de capturar los paquetes transmitidos por cualquier nodo. Éste sensor registra como eventos las tramas capturadas (*CAPTURED_MESSAGE*).

La Figura 47 muestra una fotografía de parte del sistema propuesto. En la parte superior aparecen el nodo sensor y el nodo repetidor, dotados con su correspondiente alimentación. Los NM añadidos se encuentran conectados a ellos mediante sendos interfaces serie a 1Mbps, y disponen a su vez de su cable de alimentación y conectividad Ethernet. El nodo *Sniffer*, que aparece en la fotografía entre ambos nodos –en la parte inferior– está basado en el mismo hardware del NM, y utiliza un nodo SimpliciTI configurado en modo promiscuo para este fin. Se emplea almacenamiento en DMA tanto en el caso de los NM como en el NS, para optimizar el envío-recepción de los datos de monitorización entre ambos elementos. La conec-

tividad Ethernet, tanto de los NMs como del NS, permite recibir las señales de sincronización desde el SM, y permite la descarga de las trazas hacia el Servidor. La electrónica de red consistía en un *switch* para la comunicación entre los elementos de la plataforma HMP.

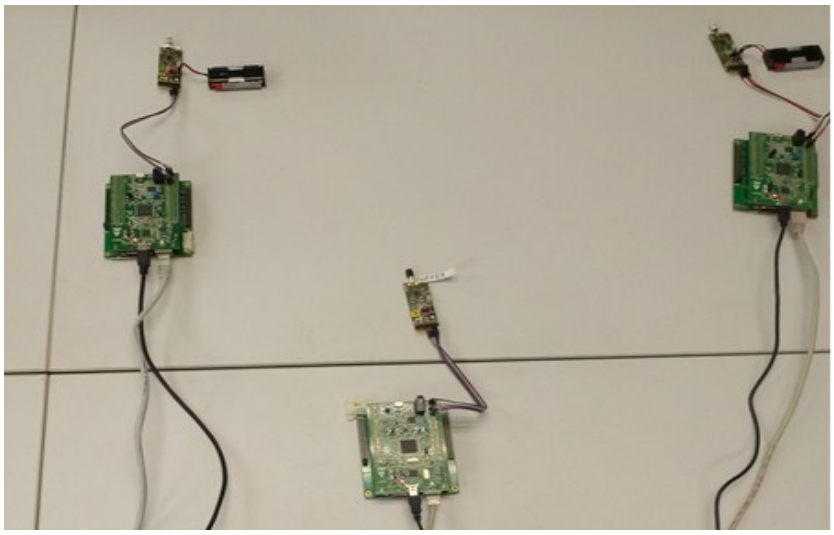


Figura 47. Conexión de los nodos de la WSN con los componentes de la HMP

4.3.2. Evaluación y resultados

Cada sesión experimental tuvo una duración de una hora aproximadamente, con lo que se obtuvieron unos 3500 registros de eventos entre los tres NMs y el NS. Tras el procesado de estas trazas obtenidas de la monitorización empleando GTSO, se generó un único archivo de datos, mediante la aplicación del SM, tal como se describe en el capítulo anterior. Estas mismas trazas, por otro lado, fueron procesadas para ajustar los tiempos de los eventos mediante la técnica ya citada de sincronización de reloj interno. Se realizaron sesiones con periodos de sincronismo 30s, 60s y 120s, tanto mediante GTSO como mediante el ajuste de reloj interno. Finalmente, se realizó un último procesado de unificación de trazas, de forma independiente a ningún mecanismo de sincronización, entre elementos de la plataforma, contando únicamente con una sola señal que indica en todas las trazas el tiempo de inicio de operación.

El criterio de comparación adoptado para discriminar la valoración entre las tres opciones (GTSO, ajuste de reloj interno y sin sincronismo) ha sido el número de cambios de orden de eventos. Es decir, en cuantas ocasiones un evento *Reception* o *CAPTURED_MESSAGE* era registrado antes del evento *Transmission* para la misma trama.

La Figura 48 muestra un caso de cambio de orden de eventos, al aplicar la sincronización de relojes internos. En este caso las marcas de tiempo del nodo C206 –un ED/RE–están un poco adelantadas, y la recepción del mensaje que le envía el nodo C207 aparece antes de que sea enviado. Incluso la diferencia entre el tiempo de envío desde el nodo C206 y la recepción en el sumidero (C201) es pequeña para el tiempo teórico de transmisión en el aire.

17:15:59,060813	C207	WakeUp	0x3000000000				
17:15:59,064093	C206	**Reception**	0x0D	0x10000000	0x10000007	0x3F03C1	0x0947
17:15:59,064133	C207	**Transmission**	0x0D	0x10000000	0x10000007	0x3F03C1	0x0947
17:15:59,064413	S101	CAPTURED_MESSAGE	0x0D	0x10000000	0x10000007	0x3F03C1	0x0947
17:15:59,068733	C206	Transmission	0x0D	0x10000000	0x10000007	0x3F12C1	0x0947
17:15:59,068773	C201	Reception	0x0D	0x10000000	0x10000007	0x3F12C1	0x0947
17:15:59,069293	S101	CAPTURED_MESSAGE	0x0D	0x10000000	0x10000007	0x3F12C1	0x0947

Figura 48. Parte de las trazas procesadas: Cambio de orden de eventos

En la Tabla 11 se resume el porcentaje de veces que hubo un cambio en el orden de los eventos registrados. Aplicando el mecanismo propio de la HMP –esto es GTSO– se aprecia que no existen cambios en el orden correcto de eventos. Sin embargo, con la sincronización por ajuste de relojes internos si aparecen cambios en el orden correcto de los eventos registrados, aún con el periodo más bajo T=30s –con el 1.57%– llegando hasta casi el 19% para el periodo T=120s. En el caso de no aplicar ningún tipo de sincronización entre los componentes de la plataforma de monitorización, el porcentaje de cambios en el orden de eventos sube al 44.09%.

Tabla 11. Cambio de orden de los eventos por estrategia de sincronización

Estrategia de sincronización	Cambio de orden de eventos (%)
GTSO T=30s	0.00%
GTSO T=60s	0.00%
GTSO T=120s	0.00%
Ajuste de relojes internos T=30s	1.57%
Ajuste de relojes internos T=60s	9.49%
Ajuste de relojes internos T=120s	18.83%
Monitorización sin sincronización	44.09%

Las diferencias existentes entre la Tabla 10 y la Tabla 11 pueden ser provocadas, principalmente, al menor tiempo de transmisión de trama en este experimento (320µs en lugar de 480µs), por lo que hay mayor probabilidad de que el desfase supere este tiempo de transmisión y por tanto cause una inversión del orden.

La diferencia entre los casos analizados en este último experimento se aprecia mejor en la Figura 49, que representa el número acumulado de errores en el orden

de eventos registrados. Se puede observar como aumenta el número de cambios de orden de eventos a medida que el periodo para la sincronización por ajuste de reloj interno es mayor. Mientras que utilizando GTSO no se observan este tipo de errores, aunque se aumente el periodo de sincronización. Por otra parte, en ausencia de un mecanismo de sincronización los errores se producen de forma considerable.

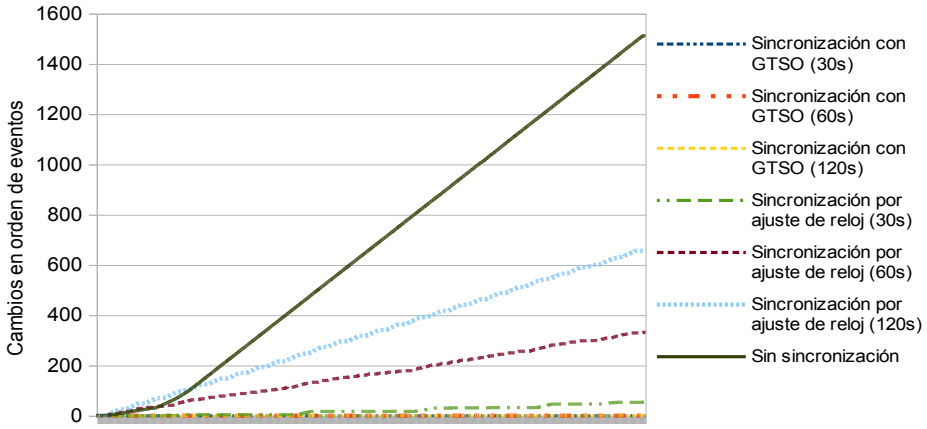


Figura 49. Número acumulado de cambios de orden de eventos

Evidentemente, es necesario adoptar algún tipo de sincronización entre los componentes de la plataforma de monitorización para evitar un enorme número de errores. La sincronización por ajuste de los relojes de los nodos únicamente puede considerarse una solución aceptable cuando pueden tolerarse un porcentaje pequeño de errores y la frecuencia de sincronización entre elementos de la plataforma es muy alta (periodos por debajo de 30s.)

Como ejemplo de uso, la utilidad de la HMP puede demostrarse en el diagnóstico de problemas –anomalías o errores reales– en el funcionamiento de la WSN. Estos problemas pueden deberse a algún error concreto en el funcionamiento de algún nodo, o simplemente ser un error en la configuración o programación de los nodos que no afecte sustancialmente a la operación de la WSN.

Como ejemplo del primer caso, en la Figura 50 se aprecia una parte del listado de eventos capturados por la plataforma, una vez unificadas las trazas recolectadas por los elementos de la misma. Se ha resaltado el camino que debe llevar una trama desde el nodo sensor (nodo C207), pasando por el nodo repetidor o interpuesto (nodo C206), hasta llegar al recolector o AP (nodo C201). En la parte resaltada de la figura se observa la transmisión del nodo sensor, la recepción y retransmisión del nodo interpuesto, así como las capturas en el NS (nodo S101) para ambas transmisiones; pero no se observa la recepción en el recolector. En este caso se debe a

algún problema en el recolector, ya que se comprueba que el mensaje sí fue enviado por el nodo repetidor.

17:14:47,297537	C206	Transmission	0x0D	0x10000000	0x10000006	0x3F13C1	0x0A4D
17:14:47,297709	S101	CAPTURED_MESSAGE	0x0D	0x10000000	0x10000006	0x3F13C1	0x0A4D
17:14:47,297725	C201	Reception	0x0D	0x10000000	0x10000006	0x3F13C1	0x0A4D
17:14:53,685924	C207	WakeUp	0x0000				
17:14:53,688590	C207	Transmission	0x0D	0x10000000	0x10000007	0x3F03BA	0x0947
17:14:53,688889	S101	CAPTURED_MESSAGE	0x0D	0x10000000	0x10000007	0x3F03BA	0x0947
17:14:53,688899	C206	Reception	0x0D	0x10000000	0x10000007	0x3F03BA	0x0947
17:14:53,692593	**C206**	Transmission	0x0D	0x10000000	0x10000007	**0x3F12BA**	**0x0947**
17:14:53,692743	**S101**	CAPTURED_MESSAGE	0x0D	0x10000000	0x10000007	**0x3F12BA**	**0x0947**
17:14:55,332607	C206	Transmission	0x0D	0x10000000	0x10000006	0x3F13C2	0x0A4D
17:14:55,332743	S101	CAPTURED_MESSAGE	0x0D	0x10000000	0x10000006	0x3F13C2	0x0A4D
17:14:55,332770	C201	Reception	0x0D	0x10000000	0x10000006	0x3F13C2	0x0A4D
17:15:01,785189	C207	WakeUp	0x0000				

Figura 50. Parte de las trazas procesadas: Mensaje perdido

En segundo lugar, como de ejemplo de la detección de errores de funcionamiento o programación de algún nodo, la Figura 51 muestra un posible problema en la programación de los nodos de la WSN o algún defecto de hardware. Se han resaltado los eventos de *Transmission* por parte del nodo sensor (nodo C207) y el nodo interpuesto (nodo C206). De acuerdo a las especificación de la aplicación de los nodos, cada uno debería transmitir cada 10 segundos la temperatura que mida del sensor incorporado al mismo. En el caso del nodo intermedio se aprecia una diferencia de tiempo entre ambas transmisiones de 9.999800s, es decir 200µs menos que los 10s establecidos de acuerdo a la sincronización realizada. Por otro lado, el nodo sensor demora aproximadamente 10.081000s, unos 81ms más que los 10 segundos previstos. En otras palabras, el nodo interpuesto trabaja algo más rápido que el nodo sensor.

16:57:19,900167	C207	WakeUp	0x0000				
16:57:19,903447	**C207**	Transmission	0x0D	0x10000000	0x10000007	**0x3F0352**	0x0947
16:57:19,903813	S101	CAPTURED_MESSAG	0x0D	0x10000000	0x10000007	0x3F0352	0x0947
16:57:19,903853	C206	Reception	0x0D	0x10000000	0x10000007	0x3F0352	0x0947
16:57:19,908493	C206	Transmission	0x0D	0x10000000	0x10000007	0x3F1252	0x0947
16:57:19,908653	S101	CAPTURED_MESSAG	0x0D	0x10000000	0x10000007	0x3F1252	0x0947
16:57:19,908720	C201	Reception	0x0D	0x10000000	0x10000007	0x3F1252	0x0947
16:57:19,949330	**C206**	Transmission	0x0D	0x10000000	0x10000006	**0x3F1359**	0x0A41
16:57:19,949490	S101	CAPTURED_MESSAG	0x0D	0x10000000	0x10000006	0x3F1359	0x0A41
16:57:19,949517	C201	Reception	0x0D	0x10000000	0x10000006	0x3F1359	0x0A41
16:57:29,949130	**C206**	Transmission	0x0D	0x10000000	0x10000006	**0x3F135A**	0x0A41
16:57:29,949330	S101	CAPTURED_MESSAG	0x0D	0x10000000	0x10000006	0x3F135A	0x0A41
16:57:29,949371	C201	Reception	0x0D	0x10000000	0x10000006	0x3F135A	0x0A41
16:57:29,981188	C207	WakeUp	0x0000				
16:57:29,984507	**C207**	Transmission	0x0D	0x10000000	0x10000007	**0x3F0353**	0x094D
16:57:29,984848	S101	CAPTURED_MESSAG	0x0D	0x10000000	0x10000007	0x3F0353	0x094D
16:57:29,984888	C206	Reception	0x0D	0x10000000	0x10000007	0x3F0353	0x094D

Figura 51. Parte de las trazas procesadas: Diferencias de tiempo entre nodos

Si bien en esta aplicación de demostración podría no suponer un grave error, o podría asumirse sin mayores problemas, este tipo de desfase podría acarrear consecuencias. Por ejemplo, impediría que el nodo interpuesto planteara pasar a modo de bajo consumo entre periodos de muestreo, ya que el desfase acumulado haría que, paulatinamente, el nodo sensor despertara cada vez más lejos de la ventana de reenvío. La detección de estos problemas permitiría, en ese caso, incorporar mecanismos para corregir la sincronización entre los nodos de la WSN. La herramienta, por tanto, se muestra válida para detectar también este tipo de anomalías.

4.4. Conclusiones

En este capítulo se han presentado una serie de experimentos y pruebas sobre los componentes de la plataforma de monitorización híbrida presentada en esta tesis, tanto a nivel individual –en lo que respecta a sus dos componentes más novedosos– como en conjunto. Los resultados de estas pruebas demuestran la utilidad y funcionalidad de la HMP.

En el primer bloque de experimentos se ha demostrado que la intrusión causada por el NM es muy baja, provocando una intrusión mínima en el nodo monitorizado, a la par que permite obtener información muy valiosa respecto al funcionamiento interno del nodo. Además, el hecho de que los NMs empleen interfaces estándar, hace que un NM puede emplearse en la monitorización de múltiples implementaciones de nodo sensor, permitiendo su reutilización en sucesivas campañas de monitorización. En los experimentos citados se ha probado la relación entre intrusión y el interfaz Mon-Inf empleado. A ese respecto, se han evaluado varias opciones, siendo la que causa una menor intrusión el uso del bus SPI.

El mecanismo de sincronización propuesto –GTSO– permite obtener trazas ordenadas, con un error medio de entre 10 y 12μs para el hardware en el que se implementó. Este mecanismo demostró ser efectivo incluso en condiciones adversas, como cuando se utiliza una fuente de reloj de muy baja calidad.

El funcionamiento completo de la plataforma permite registrar tanto eventos internos del nodo, como transmisiones en el aire. Esto permite combinar información para poder realizar un análisis más completo del comportamiento de la WSN monitorizada. La traza final combinada además podría servir de base para realizar una reproducción del comportamiento o funcionamiento de la WSN, ya sea para análisis o para otro fin.

La plataforma HMP ofrece características novedosas muy interesantes. Su enfoque híbrido permite registrar eventos internos del nodo sensor –técnica activa que tradicionalmente sobrecarga al nodo observado con tareas de almacenamiento y procesado que pueden distorsionar las medidas– pero lo añadiendo un dispositivo específico (NM) que minimiza la intrusión en el nodo observado.

Otro de los aspectos destacados es que no requiere una sincronización total de los NM –que requiere mecanismos más complejos–, sino que basta con la introducción en la traza de puntos de sincronismo que posteriormente –*offline*– permitirán reconstruir una temporización común. Esto permite a los componentes de la plataforma –los NM y los NS– centrarse en el registro de eventos.

El uso de interfaces estándar, junto a la introducción de *traps* software muy simples en los nodos, permite que la plataforma sea utilizada en distintos tipos de WSN. El costo inicial de la plataforma se amortiza rápidamente por la reutilización de todos los componentes en múltiples campañas de monitorización de WSN.

Por tanto, la plataforma HMP debe considerase una herramienta muy potente para el análisis de una WSN. Puede emplearse en cualquier fase del ciclo de vida de la red: diseño, implementación, despliegue, operación, reposición, entre otros. Proporcionando la posibilidad de depuración, sintonización, ubicación, diagnóstico y verificación de las mismas. Las trazas obtenidas reflejan fielmente el funcionamiento del sistema, y por tanto pueden utilizarse como carga sintética en estudios analíticos.

La Tabla 12 resume las características más destacadas de la plataforma HMP, las que ya han sido explicadas anteriormente.

Tabla 12. Resumen de características de la HMP

Característica / Capacidad	Cumple
Registro de eventos internos en nodo	Si (según programación)
Captura en el aire de mensajes transmitidos	Si (mediante *sniffers*)
Detección de errores	Si (análisis de trazas/otros)
Sincronización de componentes	No
Sincronización de trazas generadas	Si (*offline*)
Formato predefinido de traza	Si (configurable)
Red independiente para monitorización	Si
Intrusión causada en la red monitorizada	Muy baja
Nodos reutilizables	Si
Aplicable distintos tipos de nodos/WSN	Si

Finalmente, esta plataforma abre la puerta a la posibilidad de certificación de WSN. Mediante HMP es posible evaluar de forma objetiva los parámetros deseados de una WSN. Por tanto, una vez fijados por los organismos correspondientes, resulta posible medir y certificar el cumplimiento de estos estándares por parte de una WSN.

Capítulo 5

Conclusiones, publicaciones realizadas, y trabajo futuro

5.1. Conclusiones de la Tesis

Una vez finalizada la ejecución del trabajo planificado en la Tesis, se puede afirmar que se alcanzaron todos los objetivos propuestos en la planificación.

Las plataformas y sistemas de monitorización y evaluación de WSN ofrecen ventajas e inconvenientes, en función de su enfoque de funcionamiento respecto a la red observada (activo o pasivo). El enfoque activo permite obtener información más precisa, pero provoca intrusión en la red y los nodos, lo que podría incidir en la información obtenida. El enfoque pasivo no provoca intrusión, pero depende únicamente de la información transmitida por los nodos, con lo que se obtiene información parcial e incompleta del comportamiento de la red. Hay muy pocas propuestas de plataformas de monitorización para WSN con un enfoque híbrido, que pueden aprovechar las ventajas de cada enfoque al tiempo que compensan sus desventajas.

Tras un estudio de las plataformas y sistemas de monitorización y evaluación de WSN, en el que se analizaron sus ventajas e inconvenientes, se procedió a proponer una nueva plataforma que recoge las mejoras, propone nuevos mecanismos y supera los inconvenientes detectados en las propuestas existentes, algunas meramente teóricas.

La plataforma propuesta se basa en una arquitectura que permite interoperabilidad y flexibilidad en el desarrollo de sus componentes. El uso de un formato es-

tándar para almacenar la información recopilada permite compartir esta información con otras aplicaciones.

Dos de los elementos propuestos en la plataforma HMP resultan notablemente novedosos desde el punto de vista de la monitorización distribuida. El primero es el NM, que es de naturaleza híbrida, hardware-software. Siguiendo la arquitectura de monitorización propuesta, el NM combina la obtención de información desde el nodo sensor, con un mecanismo de asignación de formato estándar, marcado de tiempo, y almacenamiento de datos, liberando de estas tareas al nodo monitorizado y por tanto reduciendo considerablemente la intrusión en el nodo observado. El segundo elemento es el mecanismo de sincronización de trazas, un mecanismo sencillo de implementar a la par que efectivo, susceptible de ser aplicado a cualquier sistema de monitorización con sincronización de trazas fuera de línea.

La aplicación de la HMP sobre una WSN real ha permitido evaluar su funcionalidad. Las aplicaciones implementadas en el SM permiten ejecutar una monitorización configurable, y un procesamiento ágil de los datos obtenidos. Se puede comprobar los beneficios proporcionados por la plataforma, tanto para obtener una traza correctamente ordenada del comportamiento del sistema, como para poder detectar errores o anomalías en el funcionamiento de los nodos.

Los resultados y contribuciones más relevantes recogidas en la presente Tesis pueden resumirse en las siguientes:

- Revisión de las principales propuestas de monitorización y evaluación de redes inalámbricas de sensores, divididas de acuerdo a su enfoque de funcionamiento, incluyendo las pocas encontradas que trabajan con un enfoque híbrido.
- Revisión y análisis de los mecanismos de sincronización de plataformas distribuidas de monitorización, con énfasis en las enfocadas en WSN.
- La propuesta de un nodo Monitor híbrido hardware-software (Navia, Campelo, et al. 2015), que con muy baja intrusión puede registrar distintos tipos de eventos de un nodo monitorizado. Si bien este nodo ha sido ideado como parte de la HMP para WSN, podría utilizarse en la monitorización de nodos de otros tipos de redes de control.
- Propuesta de un mecanismo de sincronización fuera de línea de trazas generadas por una plataforma de monitorización distribuida (GTSO), que asegure el correcto orden de los eventos (Navia et al. 2018). Este mecanismo podría ser aplicado a cualquier otra plataforma distribuida, que no requiera sincronización en línea.
- Desarrollo de un conjunto de aplicaciones para un servidor de monitorización, que ejecute las tareas correspondientes al proceso de sincronización, recolección y procesamiento de trazas de una plataforma distribuida.

5.2. Publicaciones relacionadas con la Tesis

Publicaciones en congresos:

- Navia, M.; Campelo, J.C.; Bonastre, A.; Serrano, J.J. Low Intrusion Active Hybrid Monitor for Nodes of Sensor Networks. In Proceedings of the Workshop on Innovation on Information and Communication Technologies (ITACA-WIICT), Valencia, Spain, 4 July 2014; pp. 111–120.
- Navia, M.; Bonastre, A.; Campelo, J.C. Hybrid Monitoring Proposal for Wireless Sensor Network. In Proceedings of the 2015 Asia-Pacific Conference on Computer Aided System Engineering, Quito, Ecuador, 14–16 July 2015; pp. 320–324.
- Navia, M.; Campelo, J.C.; Bonastre, A.; Ors, R.; Capella, J.V. Drift clock analysis on distributed embedded systems for IoT applications. In Proceedings of the Workshop on Innovation on Information and Communication Technologies (ITACA-WIICT 2016), Valencia, Spain, 17 June 2016; pp. 42–49.

Publicaciones en revistas:

- Navia, M.; Campelo, J.C.; Bonastre, A.; Ors, R.; Capella, J.V.; Serrano, J.J. Active Low Intrusion Hybrid Monitor for Wireless Sensor Networks. *Sensors*, **2015**, vol.15, num. 9, pp.23927–23952, DOI: 10.3390/s150923927, factor de impacto 2.033 (2015), Q1 en índice JCR.
- Navia, M.; Campelo, J.C.; Bonastre, A.; Ors, R. GTSO: Global Trace Synchronization and Ordering Mechanism for Wireless Sensor Network Monitoring Platforms. *Sensors*, **2018**, vol. 18, num. 1, p.28, DOI: 10.3390/s18010028, factor de impacto 2.677 (2016), Q1 en índice JCR.

Publicaciones en proceso de revisión en revistas:

- Navia, M.; Bonastre, A.; Campelo, J.C. A Hybrid Monitoring Platform for WSN evaluation.

5.3. Trabajo futuro sobre el tema

En base a los resultados obtenidos en este trabajo, se pueden proponer y realizar otras investigaciones relacionadas al tema. Aunque parezca que el aspecto de la monitorización de WSN es un área ya muy desarrollada, se puede hacer aún contribuciones que faciliten el avance de las mismas, sobre todo considerando la importancia de las mismas en el paradigma actual del IoT.

Los trabajos futuros que pudieran realizarse son entre otros:

- La ampliación del nodo monitor, donde pueda, además de las funciones ya establecidas, actuar sobre el nodo monitorizado, como por ejemplo reini-

ciarlo cuando se detecte un error que no permita su funcionamiento normal, siguiendo el enfoque de muy baja intrusión. Esto es algo que ya hacen algunas propuestas, pero por lo general la intrusión provocada es considerable, o se limita a ciertos nodos.

- La reprogramación remota de la plataforma, es decir, poder reconfigurar la forma de funcionamiento de los nodos NM y NS, así como la estructura y formato de almacenamiento de los datos obtenidos.
- La mejora del mecanismo de sincronización de trazas GTSO. Una tarea que queda pendiente sobre el mismo es la evaluación de la ampliación de su funcionamiento a una red multi-salto. Además, queda por evaluar su operación en condiciones ambientales adversas, como podría ser la transmisión inalámbrica de las señales de sincronización en entornos con interferencias.
- La adaptación del modo de funcionamiento de la plataforma para que pueda trabajar en tiempo real, es decir procesando y mostrando la información de forma casi inmediata en relación al tiempo en que se genera. Para esto se debería tomar en cuenta la realización de la sincronización de trazas a medida que se generen los puntos de sincronización.
- La combinación del principio de GTSO (envío de señales periódicas para la sincronización) con el mecanismo *Convex-hulls*. La disponibilidad de una señal periódica que pueda ser recibida por todos o por la mayoría de nodos, permitiría un funcionamiento más óptimo de este último. El mecanismo resultante podría, además, ser aplicado de forma más eficaz en la sincronización en línea.
- Adicionalmente, se podría desarrollar un analizador sintáctico (*parser*) que permita transformar los datos obtenidos en formato XML a un formato que pueda ser interpretado por un simulador, como OMNeT o NS2.

Bibliografía

Akhlaq, M. & Sheltami, T.R., 2013. RTSP: An accurate and energy-efficient protocol for clock synchronization in WSNs. *IEEE Transactions on Instrumentation and Measurement*, 62(3), pp.578–589.

Al-Fuqaha, A. et al., 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, 17(4), pp.2347–2376.

Ashton, P., 1995. *Algorithms for off-line clock synchronisation*,

Atanasov, S., 2013. An overview of wireless communication technologies used in wireless sensor networks. In *International Scientific Conference eRA-8*. pp. 11–18. Available at: http://era.teipir.gr/sites/default/files/ictisession.pdf.

Awad, A. et al., 2008. On the Need for Passive Monitoring in Sensor Networks. In *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*. Parma, Italy: IEEE, pp. 693–699. Available at: http://ieeexplore.ieee.org/document/4669304/.

Becker, D., Linford, J.C., et al., 2008. Replay-Based Synchronization of Timestamps in Event Traces of Massively Parallel Applications. In *2008 International Conference on Parallel Processing - Workshops*. IEEE, pp. 212–219. Available at: http://ieeexplore.ieee.org/document/4626803/.

Becker, D. et al., 2009. Scalable timestamp synchronization for event traces of message-passing applications. *Parallel Computing*, 35(12), pp.595–607. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0167819109000155.

Becker, D., Rabenseifner, R. & Wolf, F., 2008. Implications of non-constant clock drifts for the timestamps of concurrent events. In *2008 IEEE International Conference on Cluster Computing*. IEEE, pp. 59–68. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4663756.

Capella, J. et al., 2016. A Reference Model for Monitoring IoT WSN-Based Applications. *Sensors*, 16(12), p.1816. Available at: http://www.mdpi.com/1424-8220/16/11/1816.

Castillo-Secilla, J. et al., 2017. Homomorphic Filtering for Improving Time Synchronization in Wireless Networks. *Sensors*, 17(4), p.909. Available at: http://www.mdpi.com/1424-8220/17/4/909.

Castillo-Secilla, J., Palomares, J. & Olivares, J., 2013. Temperature-Compensated Clock Skew Adjustment. *Sensors*, 13(8), pp.10981–11006. Available at: http://www.mdpi.com/1424-8220/13/8/10981/.

Chen, B. et al., 2008. LiveNet: Using Passive Monitoring to Reconstruct Sensor Network Dynamics. In *International Conference on Distributed Computing in Sensor Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 79–98. Available at: http://link.springer.com/10.1007/978-3-540-69170-9_6.

Chen, Y., Qin, F. & Yi, W., 2014. Guard Beacon: An Energy-Efficient Beacon Strategy for Time Synchronization in Wireless Sensor Networks. *IEEE Communications Letters*, 18(6).

Choudhuri, S., 2009. FlashBox: a system for logging non-deterministic events in deployed embedded systems. In *Proceedings of the 2009 ACM symposium*. pp. 1676–1682. Available at: http://portal.acm.org/citation.cfm?id=1529657.

Dwivedi, A.K. & Vyas, O.P., 2011. An Exploratory Study of Experimental Tools for Wireless Sensor Networks. *Wireless Sensor Network*, 03(07), pp.215–240. Available at: http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/wsn.2011.37025.

Dyer, M. et al., 2007. Deployment Support Network. In K. Langendoen & T. Voigt, eds. *Wireless Sensor Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 195–211. Available at: http://link.springer.com/10.1007/978-3-540-69830-2_13.

Edison, J., 2005. IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems - A tutorial. , pp.1–94.

Embest Technology Co., 2012. *STM32F4DIS-BB User Manual* www.element14.com/, ed., www.element14.com/.

Engel, A. & Koch, A., 2015. Accelerated Clock Drift Estimation for High-Precision Wireless Time-Synchronization. In IEEE, ed. *Local Computer Networks Conference Workshops (LCN Workshops)*. pp. 627–635.

Ferrari, D., Serazzi, G. & Zeigner, A., 1983. *Measurement and Tuning of Computer Systems*, Prentice-Hall.

Ganeriwal, S., Kumar, R. & Srivastava, M.B., 2003. Timing-sync protocol for sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems - SenSys '03*. New York, New York, USA: ACM Press, pp. 138–149. Available at: http://portal.acm.org/citation.cfm?doid=958491.958508.

Garcia, F. et al., 2014. EPMOSt: An Energy-Efficient Passive Monitoring System for Wireless Sensor Networks. *Sensors*, 14(6), pp.10804–10828. Available at: http://www.mdpi.com/1424-8220/14/6/10804/.

Gharghan, S., Nordin, R. & Ismail, M., 2014. Energy-Efficient ZigBee-Based Wireless Sensor Network for Track Bicycle Performance Monitoring. *Sensors*, 14(8), pp.15573–15592. Available at: http://www.mdpi.com/1424-8220/14/8/15573/.

Giancarlo Fortino et al., 2016. Interoperability in the Internet of Things | Guest Editors' Introduction. *Computing now*. Available at: https://www.computer.org/web/computingnow/archive/interoperability-in-the-internet-of-things-december-2016-introduction [Accessed August 21, 2017].

Gong, F. & Sichitiu, M., 2015. CESP: A Low-power, High-accuracy Time Synchronization Protocol. *IEEE Transactions on Vehicular Technology*, 65(4), pp.1–10. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7070715.

Hao, T. et al., 2014. WizSync: Exploiting Wi-Fi infrastructure for clock synchronization in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 13(6), pp.1379–1392.

Hofman, R. & Hilgers, U., 1998. Theory and tool for estimating global time in parallel and distributed systems. In *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing - PDP '98 -*. IEEE Comput. Soc, pp. 173–179. Available at: http://ieeexplore.ieee.org/document/647195/.

Horneber, J. & Hergenroder, A., 2014. A survey on testbeds and experimentation environments for wireless sensor networks. *IEEE Communications Surveys and Tutorials*, 16(4), pp.1820–1838.

Hossain, M.S., Lee, W.S. & Raghunathan, V., 2012. SPI-SNOOPER: A Hardware-Software Approach for Transparent Network Monitoring in Wireless Sensor Networks. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES+ISSS '12*. New York, New York, USA: ACM Press, p. 53. Available at: http://dl.acm.org/citation.cfm?doid=2380445.2380460.

IEEE Standards Association, 2008. IEEE 1588-2008 International Standard.

Jabbarifar, M., 2013. *On line Trace Synchronization for Large Scale Distributed Systems*. École Polytechnique de Montréal.

Jabbarifar, M. et al., 2012. Optimum off-line trace synchronization of computer clusters. *Journal of Physics: Conference Series*, 341, p.012029. Available at: http://stacks.iop.org/1742-6596/341/i=1/a=012029?key=crossref.c3b85f8a8ad5d7022a2a4649002d9e97.

Keysight Technologies, 2017. 6000 Series Oscilloscopes Data Sheet.

Khan, M.M.H. et al., 2014. Troubleshooting interactive complexity bugs in wireless sensor networks using data mining techniques. *ACM Transactions on Sensor Networks*, 10(2), pp.1–35. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84893359386&partnerID=tZOtx3y1.

Khan, M.U. & Zulkernine, M., 2014. A Hybrid Monitoring of Software Design-Level Security Specifications. In *2014 14th International Conference on Quality Software*. IEEE, pp. 111–116. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6958394.

Khlifi, H. & Grégoire, J.-C., 2006. Low-complexity offline and online clock skew estimation and removal. *Computer Networks*, 50(11), pp.1872–1884. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1389128605002689.

Kim, H., Ma, X. & Hamilton, B.R., 2012. Tracking Low-Precision Clocks With Time-Varying Drifts Using Kalman Filtering. *IEEE/ACM Transactions on Networking*, 20(1), pp.257–270. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5934391.

Kim, Y.-J., Song, S. & Kim, D., 2017. HDF: Hybrid Debugging Framework for Distributed Network Environments. *ETRI Journal*, 39(2), pp.222–233. Available at: http://doi.wiley.com/10.4218/etrij.17.2816.0108.

Kotsev, A. et al., 2015. Architecture of a Service-Enabled Sensing Platform for the Environment. *Sensors*, 15(2), pp.4470–4495. Available at: http://www.mdpi.com/1424-8220/15/2/4470/.

Krunic, V., Trumpler, E. & Han, R., 2007. NodeMD. In *Proceedings of the 5th international conference on Mobile systems, applications and services - MobiSys '07*. San Juan, Puerto Rico, USA: ACM Press, p. 43. Available at: http://dl.acm.org/citation.cfm?id=1247660.1247669.

Kuang, X. & Shen, J., 2010. SNDS: A Distributed Monitoring and Protocol Analysis System for Wireless Sensor Network. In *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*. Wuhan, Hubei, China: IEEE, pp. 422–425. Available at: http://ieeexplore.ieee.org/document/5480839/.

Lee, D.S., Liu, Y.H. & Lin, C.R., 2012. A wireless sensor enabled by wireless power. *Sensors (Switzerland)*, 12(12), pp.16116–16143.

Lenzen, C., Sommer, P. & Wattenhofer, R., 2015. PulseSync: An Efficient and Scalable Clock Synchronization Protocol. *IEEE/ACM Transactions on Networking*, 23(3), pp.717–727. Available at: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6777583.

Levis, P. et al., 2005. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*. Berlin/Heidelberg: Springer-Verlag, pp. 115–148. Available at: http://www.springerlink.com/index/10.1007/b138670.

Li, L. et al., 2011. Exploiting FM radio data system for adaptive clock calibration in sensor networks. In *Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11*. New York, New York, USA: ACM Press, p. 169. Available at: http://dx.doi.org/10.1145/1999995.2000012.

Li Zhang, Zhen Liu & Honghui Xia, C., 2002. Clock synchronization algorithms for network measurements. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE, pp. 160–169. Available at: http://ieeexplore.ieee.org/document/1019257/.

Lim, R. et al., 2013. FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proceedings of the 12th international conference on Information processing in sensor networks*. New York, New York, USA: ACM Press, pp. 153–165. Available at: http://dl.acm.org/citation.cfm?id=2461402.

Liu, Y., Liu, K. & Li, M., 2010. Passive Diagnosis for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 18(4), pp.1132–1144. Available at: http://ieeexplore.ieee.org/document/5356174/.

Luo, L. et al., 2006. Achieving Repeatability of Asynchronous Events in Wireless Sensor Networks with EnviroLog. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE, pp. 1–14. Available at: http://ieeexplore.ieee.org/document/4146767/.

Maerien, J. et al., 2012. FAMoS: A Flexible Active Monitoring Service for Wireless Sensor Networks. In *Distributed Applications and Interoperable Systems*. Springer, Berlin, Heidelberg, pp. 104–117. Available at: http://link.springer.com/10.1007/978-3-642-30823-9_9.

Mahapatro, A. & Khilar, P.M., 2013. Fault Diagnosis in Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials*, 15(4), pp.2000–2026. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6489879.

Maróti, M. et al., 2004. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*. New York, New York, USA: ACM Press, pp. 39–49. Available at: http://dl.acm.org/citation.cfm?id=1031501.

Meier, A. et al., 2008. DiMo. In *Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems - MSWiM '08*. New York, New York, USA: ACM Press, p. 117. Available at: http://doi.acm.org/10.1145/1454503.1454526.

Mills, D.L., 1991. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications*, 39(10), pp.1482–1493.

Molina-Garcia, A. et al., 2012. Development and Assessment of a Wireless Sensor and Actuator Network for Heating and Cooling Loads. *IEEE Transactions on Smart Grid*, 3(3), pp.1192–1202. Available at: http://ieeexplore.ieee.org/document/6217294/.

Navia, M., Campelo, J., et al., 2015. Active Low Intrusion Hybrid Monitor for Wireless Sensor Networks. *Sensors*, 15(9), pp.23927–23952. Available at: http://www.mdpi.com/1424-8220/15/9/23927/.

Navia, M. et al., 2016. Drift clock analysis on distributed embedded systems for IoT applications. In C. Fernandez-Llatas & M. Guillen, eds. *Workshop on Innovation on Information and Communication Technologies (ITACA-WIICT 2016)*. ITACA-UPV, pp. 42–49.

Navia, M. et al., 2018. GTSO: Global Trace Synchronization and Ordering Mechanism for Wireless Sensor Network Monitoring Platforms. *Sensors (Switzerland)*, 18(1), p.28. Available at: http://www.mdpi.com/1424-8220/18/1/28.

Navia, M., Bonastre, A. & Campelo, J.C.J.C., 2015. Hybrid Monitoring Proposal for Wireless Sensor Network. In *2015 Asia-Pacific Conference on Computer Aided System Engineering*. IEEE, pp. 320–324. Available at:

http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7287039.

Ojha, T., Misra, S. & Raghuwanshi, N.S., 2015. Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges. *Computers and Electronics in Agriculture*, 118, pp.66–84. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0168169915002379.

Patel, M.M. & Aggarwal, A., 2013. Security attacks in wireless sensor networks: A survey. In *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)*. IEEE, pp. 329–333. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6526929.

Poirier, B., Roy, R. & Dagenais, M., 2010. Accurate offline synchronization of distributed traces using kernel-level events. *ACM SIGOPS Operating Systems Review*, 44(3), p.75. Available at: http://portal.acm.org/citation.cfm?doid=1842733.1842747.

Potsch, A. et al., 2014. TWECIS: A testbed for wireless energy constrained industrial sensor actuator networks. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. Barcelona, Spain: IEEE, pp. 1–4. Available at: http://ieeexplore.ieee.org/document/7005274/.

Ramanathan, N. et al., 2005. Sympathy for the sensor network debugger. In *Proceedings of the 3rd international conference on Embedded networked sensor systems - SenSys '05*. New York, New York, USA: ACM Press, p. 255. Available at: http://portal.acm.org/citation.cfm?doid=1098918.1098946.

Raposo, D. et al., 2017. A Taxonomy of Faults for Wireless Sensor Networks. *Journal of Network and Systems Management*, 25(3), pp.591–611. Available at: http://link.springer.com/10.1007/s10922-017-9403-6.

Ringwald, M. & Romer, K., 2007. Practical time synchronization for Bluetooth Scatternets. In *2007 Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS '07)*. IEEE, pp. 337–345. Available at: http://ieeexplore.ieee.org/document/4550453/.

Ringwald, M. & Romer, K., 2007. Snif: A comprehensive tool for passive inspection of sensor networks. *6. Fachgespräch Sensornetzwerke*, pp.12–15. Available at: https://vs.inf.ethz.ch/publ/papers/mringwal-snif-kuvs.pdf.

Ringwald, M., Römer, K. & Vialetti, A., 2006. *SNIF: Sensor Network Inspection Framework*, Zurich, Switzerland. Available at: http://www.vs.inf.ethz.ch/publ/papers/snif-techreport.pdf.

Rodrigues, A. et al., 2013. Diagnostic Tools for Wireless Sensor Networks: A Comparative Survey. *Journal of Network and Systems Management*, 21(3), pp.408–452. Available at: http://link.springer.com/10.1007/s10922-012-9240-6.

Romer, K. & Ma, J., 2009. PDA: Passive distributed assertions for sensor networks. In *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on*. San Francisco, CA, USA: IEEE, pp. 337–348. Available at: http://ieeexplore.ieee.org/document/5211917/.

Rost, S. & Balakrishnan, H., 2006. Memento: A Health Monitoring System for Wireless Sensor Networks. In *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*. IEEE, pp. 575–584. Available at: http://ieeexplore.ieee.org/document/4068315/.

Schoofs, A., O'Hare, G.M.P. & Ruzzelli, a. G., 2012. Debugging low-power and lossy wireless networks: A survey. *IEEE Communications Surveys & Tutorials*, 14(2), pp.311–321. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5729264.

Sommer, P. & Kusy, B., 2013. Minerva: Distributed Tracing and Debugging in Wireless Sensor Networks. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems - SenSys '13*. New York, New York, USA: ACM Press, pp. 1–14. Available at: http://dl.acm.org/citation.cfm?doid=2517351.2517355.

STMicroelectronics, STM32F0DISCOVERY - Discovery kit with STM32F051R8 MCU. Available at: http://www.st.com/en/evaluation-tools/stm32f0discovery.html [Accessed January 15, 2015a].

STMicroelectronics, STM32F4DISCOVERY Discovery kit with STM32F407VG MCU. Available at: http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f4discovery.html [Accessed May 16, 2016b].

Sundaram, V., Eugster, P. & Zhang, X., 2009. Lightweight tracing for wireless sensor networks debugging. In *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks - MidSens '09*. Urbana Champaign, Illinois, USA: ACM Press, pp. 13–18. Available at: http://doi.acm.org/10.1145/1658192.1658195\nhttp://dl.acm.org/citation.cfm?id=1658195.

Tancreti, M. et al., 2011. Aveksha: A Hardware-Software Approach for Non-intrusive Tracing and Profiling of Wireless Embedded Systems. In ACM, ed. *Proceedings of*

the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11. New York, New York, USA: ACM Press, p. 288. Available at: http://dl.acm.org/citation.cfm?doid=2070942.2070972.

Tancreti, M. et al., 2015. TARDIS: Software-Only System-Level Record and Replay in Wireless Sensor Networks. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks - IPSN '15*. New York, New York, USA: ACM Press, pp. 286–297. Available at: http://dl.acm.org/citation.cfm?doid=2737095.2737096.

Tennina, S. et al., 2015. Z-Monitor: A protocol analyzer for IEEE 802.15.4-based low-power wireless networks. *Computer Networks*, 95, pp.77–96. Available at: http://www.sciencedirect.com/science/article/pii/S1389128615004788.

Texas Instruments, CC1110-CC1111 Sub-1 GHz wireless MCU with up to 32 kB Flash memory. Available at: http://www.ti.com/product/CC1110-CC1111 [Accessed August 4, 2017a].

Texas Instruments, SIMPLICITI SimpliciTI Compliant Protocol Stack. Available at: http://www.ti.com/tool/SimpliciTI [Accessed August 4, 2017b].

Tolle, G. & Culler, D., 2005. Design of an application-cooperative management system for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.* IEEE, pp. 121–132. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1462004.

Tuwanut, P. & Kraijak, S., 2015. A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends. In *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015)*. Institution of Engineering and Technology, p. 6 .–6 . Available at: http://digital-library.theiet.org/content/conferences/10.1049/cp.2015.0714.

Wagenknecht, G., Anwander, M. & Braun, T., 2008. MARWIS: a management architecture for heterogeneous wireless sensor networks. *Wired/Wireless Internet ...*, pp.177–188. Available at: http://www.springerlink.com/index/3t408278288104n2.pdf.

Wireshark, Wireshark · Frequently Asked Questions. Available at: https://www.wireshark.org/faq.html [Accessed August 28, 2017].

Xiao, J., Zeng, P. & Yu, H., 2014. A survey of testing and debugging testbeds for sensor networks. In *Proceeding of the 11th World Congress on Intelligent Control and Automation*. IEEE, pp. 150–155. Available at:

http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7052704.

Xu, L. Da, He, W. & Li, S., 2014. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4), pp.2233–2243.

Yoon, S., Veerarittiphan, C. & Sichitiu, M.L., 2007. Tiny-sync: Tight Time Synchronization for Wireless Sensor Networks. *ACM Transactions on Sensor Networks*, 3(2), p.8–es. Available at: http://portal.acm.org/citation.cfm?doid=1240226.1240228.

Zhao, Z., Huangfu, W. & Sun, L., 2012. NSSN: A network monitoring and packet sniffing tool for wireless sensor networks. In *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, pp. 537–542. Available at: http://ieeexplore.ieee.org/document/6314261/.

Zou, C. & Lu, Y., 2012. A Time Synchronization Method for Wireless Sensor Networks. In B. Liu, M. Ma, & J. Chang, eds. *Information Computing and Applications ICICA 2012. Lecture Notes in Computer Science*. Chengde, China: Springer, Berlin, Heidelberg, pp. 221–228. Available at: http://link.springer.com/10.1007/978-3-642-34062-8_29.