Sistema automatizado para detección de anomalías en redes de sensores inalámbricas

André Rodrigues[1], Jorge Sá Silva[2], Fernando Boavida[2]
[1]ISCAC – Coimbra Business School
[2]Departamento de Ingeniería Informática
Universidad de Coimbra
3030-290 Coimbra, Portugal
{arod, sasilva, boavida}@dei.uc.pt

***Resumen*- A medida que las redes de sensores inalámbricas (*Wireless Sensor Networks*, WSN) cobran impulso, la monitorización de este tipo de redes se está convirtiendo en un aspecto crucial, a fin de garantizar que las anomalías sean rápidamente detectadas. Las actuales soluciones de monitorización de WSN tienen varias limitaciones entre las que destacan el ser diseñadas para aplicaciones específicas, requerir *hardware* dedicado o específico, consumir demasiada energía y/o recursos de procesamiento o depender de la intervención manual o fuera de línea. En este trabajo se propone un método para la detección de anomalías en redes de sensores inalámbricas que se ocupa de estas limitaciones. El método se basa en dos indicadores muy simples: una herramienta de registro (*logging*), y un algoritmo de minería de datos. Gracias a este método, se logra un consumo de recursos muy bajo, independencia de las aplicaciones, muy buena potencial para el control y la monitorización de múltiples WSN y simplificación del proceso de detección. El sistema propuesto ha sido validado mediante una implementación donde se ha demostrado las capacidades que ofrece el método para la detección de varias anomalías típicas.**

***Palabras Clave*- redes de sensores inalámbricas, detección de anomalías, monitorización**

I. INTRODUCCIÓN

La detección y el diagnóstico de problemas en redes de sensores inalámbricas (WSN) se considera ahora esencial, debido al hecho de que el despliegue de este tipo de redes está creciendo a un ritmo rápido. Existen numerosas implementaciones, ya sean experimentales o de naturaleza comercial.

Un ejemplo de ello, dentro del área de investigación, es la implantación reciente de Intel, que equipó unos pocos cientos de hogares con dispositivos que permitieron la recogida de un conjunto de valores de parámetros (ambientales, fisiológicos y de comportamiento), con el objetivo de evaluar el potencial de la tecnología WSN en el estudio del envejecimiento y las enfermedades crónicas [1]. Una de las conclusiones del estudio es la necesidad de disponer de herramientas para gestionar este tipo de infraestructuras, que se caracterizan por un gran número de instalaciones geográficamente dispersas, en donde sus usuarios directos (en este caso los ancianos) no tienen suficiente experiencia tecnológica para ayudar en el diagnóstico y solución de problemas que se producen inevitablemente en los sistemas instalados.

En lo que se refiere a implementaciones comerciales, se espera que la tecnología WSN sea ampliamente utilizada para dar soporte a la monitorización en tiempo real de múltiples instalaciones pertenecientes a las mismas o diferentes entidades. Por ejemplo, es fácil prever su uso en granjas para apoyar la monitorización de las condiciones de salud de los animales. Como ejemplo, un sistema basado en WSN podría apoyar la recogida de diversos parámetros y, basándose en sus valores y en un conjunto de reglas, generar alarmas si algo anormal ocurre. Tal sistema se podría desplegar fácilmente en varias instalaciones y, por supuesto, requeriría una supervisión adecuada para asegurar su correcto funcionamiento. Esto es claramente un ejemplo de un escenario en el que la entidad que comercializa o mantiene el sistema también debe tener la capacidad de monitorizar en tiempo real las instalaciones en funcionamiento en los distintos clientes.

Aunque los escenarios de múltiples instalaciones WSN y/o de instalaciones de gran escala estén emergiendo rápidamente, las herramientas existentes para su monitorización aún no cumplen los requisitos de detección de anomalías: simplicidad, eficacia, automatización, operación distribuida y generalidad. Típicamente, los instrumentos de monitorización están específicamente desarrollados para las aplicaciones, consumen recursos considerables, exigen una configuración compleja y su utilización se limita a una única WSN.

El trabajo presentado en este artículo tiene por objetivo demostrar que es posible desarrollar sistemas de detección de anomalías que cumplan con los requisitos antes mencionados, utilizando dos indicadores simples: una herramienta de registro ya existente y un algoritmo de minería de datos.

El artículo se organiza de la siguiente manera. La sección II define el conjunto de requisitos que deben cumplir los instrumentos de monitorización de WSN. La sección III está dedicada a la presentación detallada de la propuesta, es decir, en lo que se refiere a su *hardware* y plataformas de *software*, indicadores utilizados, la recogida y el análisis de registros (*logs*), transformación de datos, y la detección y el diagnóstico. La implementación de prueba de concepto fue objeto de una evaluación en dos escenarios simples que comprenden varias condiciones anómalas. Los resultados de la evaluación se analizan en la sección IV. La sección V identifica el trabajo relacionado, con una breve presentación y discusión de un conjunto representativo de las herramientas de supervisión de WSN. La sección VI presenta las conclusiones y orientaciones para trabajo futuro.

II. REQUISITOS

En esta sección se presentan y discuten brevemente los requisitos que una herramienta de monitorización de WSN debería cumplir. Estos requisitos se dividen en dos categorías: requisitos relacionados con el escenario y requisitos de prestación/utilización.

A. Requisitos relacionados con el escenario

Escalabilidad – la herramienta debe ser capaz de escalar, tanto en términos del número de nodos por WSN como en el número de redes de sensores inalámbricas soportadas. Como se mencionó antes, los escenarios que comprenden la monitorización simultánea de distintas WSN van a ser frecuentes.

Homogeneidad inter-WSN y heterogeneidad intra-WSN – en cualquier WSN a menudo existe cierta heterogeneidad a nivel de *hardware* y *firmware*. Sin embargo, cuando se da el caso de múltiples WSN bajo la responsabilidad de una cierta organización, es frecuente que tengan las mismas aplicaciones en las mismas plataformas. Por lo tanto, las herramientas de monitorización deberían permitir explorar las ventajas de esta homogeneidad inter-WSN, y también lidiar con la heterogeneidad intra-WSN.

Soporte de WSN geográficamente dispersas – la existencia de redes de sensores inalámbricas situadas en varios lugares es un factor que debe ser tenido en cuenta en el diseño de una herramienta de monitorización, que debe apoyarse en mecanismos adecuados de comunicación.

Soporte de nodos móviles en WSN – es bastante común tener nodos móviles en WSN. Este tipo de nodos no tienen un impacto insignificante en las estrategias de recogida de datos y en las comunicaciones. Por este motivo, las herramientas de monitorización deben ser capaces de lidiar con este tipo de nodos.

B. Requisitos de prestación/utilización

Soporte de todos los paradigmas de aplicación – a fin de no verse limitada por el paradigma de la aplicación (es decir, *schedule-driven*, *query-driven*, o *event-driven*), la herramienta no debe depender de la existencia de patrones específicos de operación.

Soporte de distintas plataformas de *hardware* y sistemas operativos (SO) – la tecnología WSN está cambiando rápidamente, por lo que es importante asegurarse de que la herramienta se puede utilizar con varios sistemas operativos y plataformas, con el fin de hacer frente tanto a las necesidades actuales como las futuras.

Minimizar el uso de recursos de las WSN – por lo general, los nodos WSN tienen recursos limitados, por lo que las herramientas de monitorización deben reducir al mínimo el consumo de recursos, como la energía, el procesamiento y la memoria.

Fácil de instalar y de usar – el esfuerzo necesario para la integración de una herramienta de monitorización en una WSN o un conjunto de WSN debe ser minimizado. Lo mismo se aplica al esfuerzo requerido para utilizar la herramienta.

Flexibilidad y Extensibilidad – es importante que la herramienta incluya los mecanismos que permiten que el gestor adapte la herramienta a sus necesidades (en el despliegue y en tiempo de ejecución) a fin de permitir una aplicación más amplia.

III. SISTEMA PROPUESTO

Teniendo en cuenta los requisitos identificados en el apartado anterior, nos propusimos demostrar que es posible la construcción de una herramienta de detección de anomalías simple, eficaz y automatizada, e independiente de la aplicación.

Cabe señalar que el objetivo del trabajo presentado no fue la construcción de la herramienta en sí, sino la demostración de que una herramienta basada en indicadores simples, con una funcionalidad de monitorización ligera, y algunos algoritmos de minería de datos puede satisfacer un gran porcentaje de los requisitos identificados.

Con este propósito, hemos desarrollado una implementación de prueba de concepto. En esta sección se ofrece información detallada sobre esa implementación, es decir, el *hardware* utilizado, plataformas de *software*, y en el diseño general, incluyendo indicadores, recogida y análisis de registros, transformación de datos, detección de eventos anómalos y su diagnóstico.

A. Sistema operativo y plataforma seleccionados

El prototipo de herramienta se implementó usando TinyOS [2] en una nueva plataforma de *hardware* llamada Hermes [3]. Esta plataforma incluye un reciente MSP430, una interfaz radio de 868 MHz, un lector de tarjetas SD, un acelerómetro, un giroscopio, un termómetro, un receptor de ritmo cardíaco, y un sistema de gestión de energía.

Como se empleó un sistema operativo basado en eventos, los indicadores que se utilizarán como base para la detección de anomalías se recogerán en el curso de *event procedure instances*. Es preciso aclarar que se utiliza aquí la terminología ya utilizada en [4], según la cual un *event procedure instance* es la secuencia que comienza con una interrupción de *hardware* y termina con la ejecución del último código asociado con el evento inicial.

También hay que señalar que, a pesar de que son esenciales, los indicadores recogidos deben ser complementados con información adicional, con el fin de identificar eficazmente las razones de comportamiento anómalo. En el caso de esta implementación de prueba de concepto, se decidió recoger adicionalmente rastreos de llamadas para el código de la aplicación, y también registrar todas las tareas ejecutadas.

Como observación final, cabe señalar que a pesar de la elección específica de la plataforma de *hardware*, sistema operativo y la implementación, los principios que guiaron la construcción de la herramienta (es decir, los indicadores simples, la independencia de la aplicación, el registro de operaciones y la minería de datos) son de carácter general y se pueden implementar fácilmente utilizando otros sistemas operativos y plataformas.

B. Cálculo de los indicadores

Al considerar la cuestión de qué indicadores utilizar para caracterizar el comportamiento como normal o anormal, se debe tener en cuenta aspectos como el impacto en los recursos del nodo, la aplicabilidad a diversos paradigmas de aplicación, el poder descriptivo, y la especificidad de plataformas de *hardware* y sistemas operativos. Indicadores con las siguientes características deben evitarse: 1) dependientes de la aplicación, por ejemplo, utilizando estadísticas sobre eventos específicos de la aplicación; 2) que requieran registros detallados, por ejemplo, rastreos

completos de llamadas, o vectores de contadores de instrucciones como en [4]; 3) dependientes del paradigma de la aplicación, por ejemplo, contadores de tráfico; 4) que exijan *hardware* dedicado o soporte del sistema operativo, por ejemplo, dispositivos dedicados de medición de energía, mecanismos de registro integrados en el sistema operativo.

De lo anterior, se puede concluir que el tiempo transcurrido, el procesamiento, y la energía son buenos candidatos para caracterizar lo que ocurre entre dos eventos consecutivos de nivel de aplicación. Estas medidas son de carácter general, fáciles de calcular, y no requieren sofisticados mecanismos de apoyo.

En el caso de la implementación de prueba de concepto, los indicadores seleccionados fueron el procesamiento y la energía. La principal razón para la exclusión del tiempo transcurrido fue que, para aplicaciones WSN típicas, no proporciona mucha información acerca de los recursos de procesamiento utilizados, pues la mayoría de tiempo entre dos eventos de nivel de aplicación es periodo inactivo (*sleep time*). Esto también significa que, por ejemplo, una anomalía que resultara en un aumento del periodo activo podría ser fácilmente "ocultada" por una ligera variación en el período inactivo, sin mucho impacto en el valor del indicador de tiempo transcurrido.

Cada indicador se calcula de la siguiente manera. Al arrancar, el contador se pone a cero. Cuando ocurre el siguiente evento de aplicación, el valor del contador se registra y se asocia al evento anterior. Finalmente, el contador se pone de nuevo a cero.

El primer método para calcular el indicador de procesamiento fue contar las instrucciones de la MCU (o ciclos de la MCU) ejecutados entre dos eventos de nivel de aplicación consecutivos. Lamentablemente, la plataforma Hermes no admite este método directamente.

El método seguido fue contar los ciclos del *sub-main clock* (SMCLK) de la MCU (es decir, MSP430F2618) entre eventos consecutivos de nivel de aplicación. El SMCLK en Hermes se define basándose en el *master clock* (MCLK) dividido por 4. El temporizador *TimerA* de la MCU se configuró para utilizar el SMCLK, dando como resultado que cuando la MCU no está durmiendo el SMCLK está funcionando y *TimerA* se incrementa.

Para hacer frente a plazos de ejecución largos durante los cuales *TimerA* podría desbordarse, el contador utiliza una variable de 32 bits que se acumula el valor de *TimerA* cada vez que se desborda o que se lee el contador. Este es un indicador muy ligero, ya que el proceso sólo requiere leer o actualizar la variable asociada al contador. Por razones de simplicidad, a partir de este punto a este indicador se llamará ciclos de MCU.

En lo que se refiere al indicador de energía, en iCount [5] los autores explicaron cómo un regulador de conmutación cuidadosamente seleccionado, utilizado para proporcionar energía regulada a un nodo sensor, se puede usar para obtener mediciones de consumo de energía, casi de forma gratuita. Debido a que el regulador seleccionado utiliza modulación de frecuencia de pulso, la frecuencia de conmutación está casi directamente relacionada con la corriente de carga. Su idea fue conectar la salida del inductor del regulador a la línea MCU INCLK que se puede utilizar para el *TimerA*. De esta manera, cada vez que el voltaje en el inductor cruza el cero en la dirección ascendente, *TimerA* se incrementa pues un nuevo ciclo de conmutación se detectó.

Hermes utiliza un sistema de gestión de energía (*Power Management System*, PMS) que incluye dos reguladores de conmutación basados en la técnica de modulación de ancho de pulso (*Pulse Width Modulation*, PWM), lo que no permite utilizar directamente el método iCount. Sin embargo, con cargas ligeras, el PMS soporta un modo de ráfaga (*burst mode*) en donde la energía se proporciona en una ráfaga de pulsos para reducir al mínimo las pérdidas de conmutación. Durante este modo de operación es posible contar los impulsos (usando un método similar al iCount) para obtener una estimación de la energía consumida.

Este método tiene dos limitaciones. La primera es que cuando el nodo sensor requiere más energía (por ejemplo, cuando se utiliza el canal radio) el regulador cambia automáticamente a modo PWM. La segunda es que, cuando el nodo está conectado a través de USB, está siempre en el modo PWM (aunque esto no sea un problema en escenarios reales pues Hermes utiliza baterías Li-Poly).

A pesar de estas limitaciones, se tomó la decisión de probar la utilidad de un sistema de medición basado en este mecanismo. Cabe señalar que una estimación precisa de la energía consumida no es necesaria. En su lugar, lo que es importante es obtener mediciones que permitan diferenciar entre funcionamiento normal y anómalo. De ahora en adelante, por razones de simplicidad, a este indicador se llamará consumo de energía.

C. Recogida y análisis de registros

Los mecanismos de registro son necesarios para recoger los indicadores y la información sobre los eventos de aplicación (por ejemplo: llamadas, tareas) de los nodos sensores, y remitirlos al sistema de gestión con el fin de apoyar la funcionalidad de detección y diagnóstico.

El mecanismo de registro debe ser flexible y ampliable, no exigir modificaciones al código fuente de la aplicación, ser fácil de instalar y usar, aprovechar las capacidades de comunicación de las aplicaciones, introducir latencia pequeña, ser leve en términos de uso de los recursos, ser fácilmente portable a otras plataformas y/o sistemas operativos, y permitir heterogeneidad de los nodos.

Habiendo que decidir entre el desarrollo de un sistema de registro de acuerdo con los requisitos anteriores o utilizar uno ya disponible, la decisión fue utilizar LIS [6], ya que soporta la mayoría de los requisitos. Sus aspectos negativos son su flexibilidad limitada (ya que no permite configuración pos-despliegue) y soporte limitado de nodos heterogéneos. Sin embargo, ya que estos aspectos no eran esenciales para la implementación de prueba de concepto, no se consideraron un obstáculo para su utilización.

En LIS, los desarrolladores tienen que producir un guión (LIS *script*) utilizando un lenguaje declarativo, que describe los mecanismos de registro y su ubicación en el código fuente de las aplicaciones. Entonces, un motor de instrumentación operando en un PC modifica el código fuente de acuerdo con el guión LIS, con el fin de incluir declaraciones de registro. También se añade al código una biblioteca de tiempo de ejecución que soporta el registro de llamadas a funciones del nodo sensor, y un módulo de código de funcionalidad de almacenamiento y recuperación de datos.

Durante la ejecución, los rastros de ejecución y el estado se guardan en la memoria local y se pueden enviar al nodo

sumidero (*sink node*) utilizando comunicación por cable (es decir, SerialActiveMessages), o la comunicación inalámbrica (es decir, TinyOS CTP para *multi-hop*, o ActiveMessages para un solo salto). Cuando los paquetes llegan al *sink node* se descodifican usando un analizador genérico de LIS y el guión LIS, a fin de extraer la información significativa.

El lenguaje LIS se puede utilizar directamente o como un lenguaje intermedio permitiendo definiciones de tareas reutilizables y de alto nivel. Una de esas tareas de alto nivel es la monitorización de llamadas en una región de interés (*Region of Interest*, ROI), donde el desarrollador especifica una ROI (por ejemplo, un cierto subsistema) y el sistema genera el LIS *script* correspondiente, que permitirá crear un registro de las llamadas de función dentro ese subsistema. Esta funcionalidad es muy útil, ya que permite evitar la necesidad de crear manualmente los guiones LIS.

En el caso de la implementación de prueba de concepto actual, fue necesario modificar los scripts de Python asociados con la funcionalidad de análisis de ROI, a fin de permitir a LIS la generación automática del código fuente modificado de la aplicación, con el objetivo de generar rastreos de llamadas y registros de indicadores.

La primera modificación fue la introducción de una etapa inicial en la que se modifica el código de las componentes de eventos de aplicación para incluir, al comienzo de cada evento, una declaración relacionada con el indicador de interés, como en el ejemplo siguiente:

```
energy = call EnergyMeter.read();
```

Esta instrucción asigna a la variable "`energy`" la energía consumida entre el inicio del evento de nivel de aplicación presente y el comienzo del anterior. El segundo paso fue modificar el *script* de Python utilizado por la ROI para generar el *script* de LIS, para que este pueda generar automáticamente las declaraciones "`watch`" asociadas a la variable "`energy`", con el fin de que sea registrada.

Con estas dos actualizaciones del mecanismo ROI, los guiones LIS resultantes tienen todos los comandos requeridos por el motor de instrumentación del código fuente de la aplicación, con el fin de recoger los rastreos de llamadas y los registros de consumo de energía. Para desplegar esta funcionalidad en los nodos sensores, todo lo que el desarrollador tiene que hacer ahora es programar los nodos de Hermes, como de costumbre, y toda la funcionalidad de registro será incluida automáticamente.

Por último, la recogida de los registros se realiza mediante el comando de consola "`timestampedlisten`", proporcionado por TinyOS, que recoge los paquetes enviados por los nodos de sensores en cuyos el mecanismo de registro está en marcha. Los paquetes recogidos se someten al analizador de LIS, que hace uso de la información de los scripts de LIS para emitir una lista fácil de leer conteniendo los rastreos de llamadas y los indicadores recogidos.

D. Transformación de datos

El fichero de datos producido por el analizador de LIS se filtra para eliminar registros incompletos de eventos de aplicación, que hayan resultado de pérdidas de paquetes. Esto es necesario porque si algunos paquetes se pierden eso puede comprometer el análisis de las entradas de registro siguientes. LIS incluye un mecanismo para descartar entradas de registro incompletas. Sin embargo, fue necesario mejorar este mecanismo, pues se detectó información incorrecta de eventos de aplicación en los registros analizados.

Después de esta fase, los registros son procesados con el fin de generar un fichero conteniendo una lista de eventos de nivel de aplicación con el formato de datos deseado. Este nuevo fichero también contiene la información adicional necesaria para permitir el diagnóstico de los eventos anómalos. Para apoyar todas estas transformaciones, fue desarrollado un conjunto de scripts en Python.

Cada línea del fichero generado tiene el siguiente formato:

<class> <m1>:<v1> <m2>:<v2> # <event> <begin> <end>

En este formato, <class> designa la clase del evento, <m1>:<v1> designa un par indicador/valor, <event> es el nombre del evento de nivel de aplicación, y <begin> y <end> identifican, respectivamente, la línea del fichero donde comienzan las entradas del registro relacionadas con este evento, y la línea donde terminan.

Los datos antes de la "#" son utilizados por el algoritmo de clasificación para identificar eventos anómalos. Los datos después del "#" son ignorados por el algoritmo de clasificación, pero se utilizan para localizar, en el fichero, la información de registro asociada con los eventos anómalos detectados.

E. Detección y diagnóstico de eventos anómalos

El algoritmo de clasificación seleccionado se basa en una técnica de aprendizaje automático (*machine learning*), llamada *Support Vector Machines* (SVM) [7], que genera un modelo que se puede utilizar para predecir la clase de una instancia. En este caso, una 'instancia' representa una instancia de un evento de nivel de aplicación, que tiene valores de indicadores como atributos, y puede ser clasificada como normal o anómala.

En SVM, se crea un modelo durante una fase de entrenamiento basada en un conjunto de instancias de datos etiquetados. En esta fase, se hace un mapeo de las instancias de datos, de su espacio de entrada a un espacio de dimensión superior, para encontrar un hiper-plano que divide las instancias de datos en dos regiones. En la fase de prueba, una instancia de datos se asigna a un mismo espacio superior y se clasifica de acuerdo con el lado del hiper-plano en el que se encuentra.

En el escenario presente, la aplicación de la técnica SVM presenta dos limitaciones. En primer lugar, no está disponible un conjunto de entrenamiento etiquetado. En segundo lugar, se espera que los eventos anómalos representen una pequeña fracción de todos los eventos (ya que el objetivo es detectar problemas esporádicos).

Teniendo en cuenta este escenario, la solución utilizada (también seguida de [4]) fue recurrir a una variante llamada SVM de una sola clase (*one-class* SVM), y admitir que el conjunto de entrenamiento contiene únicamente los eventos normales, a sabiendas de que un pequeño porcentaje de ellos pueden haber sido mal clasificados como tal. Al definir el porcentaje de eventos clasificados erróneamente en el conjunto de entrenamiento, el método *one-class* SVM creará un modelo que pone la mayoría de los eventos en el lado de la clase normal del hiper-plano, mientras que los restantes se colocan en el lado de clase anómala. Este modelo se usa entonces para predecir la clase de los futuros eventos recibidos. El modelo puede ser actualizado periódicamente,

en caso de que se requiera que el mecanismo de detección de evento anómalo tenga cierta flexibilidad para adaptarse a los cambios del entorno/sistema.

Las razones para elegir esta técnica fueron las siguientes: no se requieren datos previamente etiquetados, se puede trabajar con conjuntos de datos no balanceados (es decir, conjuntos de clases desigualmente representadas), y la existencia de una biblioteca de códigos bien documentada y fácil de usar (LIBSVM [8]). Esta biblioteca incluye secuencias de comandos de Python para simplificar su uso, a saber, para la ampliación de los conjuntos de datos, la selección de parámetros optimizados, el entrenamiento del modelo y la prueba de los datos.

La salida de LIBSVM es una clasificación para cada evento de nivel de aplicación. Se desarrolló un script que utiliza esta información para localizar, en el fichero LIS analizado, la información registrada en relación con los eventos de nivel de aplicación clasificados como anómalos. Esto permite a la persona responsable su análisis, con el fin de identificar las posibles razones de su clasificación.

En la Fig. 1 se resumen las actividades involucradas en la funcionalidad de detección y diagnóstico de eventos anómalos presentada en esta sección.

IV. Evaluación

La presente implementación de prueba de concepto fue sometida a varias pruebas, para evaluar la eficacia y la eficiencia de los conceptos subyacentes. Esta sección comienza por describir y presentar los resultados de un conjunto de experimentos realizados para evaluar las capacidades de detección y diagnóstico de la herramienta. A continuación, la herramienta fue evaluada a la luz de los requisitos iniciales.

A. Resultados experimentales

Se llevaron a cabo dos conjuntos de pruebas. En el primer grupo de experimentos, el objetivo fue evaluar la capacidad del indicador 'ciclos de MCU' para detectar comportamientos anómalos.

La aplicación seleccionada fue RadioCountToLeds, una aplicación estándar de TinyOS en la que dos nodos difunden periódicamente paquetes que contienen un contador, y cada vez que un nodo recibe un paquete se muestran los últimos 3 bits del contador en el mostrador LED. Se eligió esta aplicación debido a que tiene un comportamiento simple y está disponible al público, lo que permite a la comunidad validar los resultados presentados en este artículo.

El código de la aplicación incluye los eventos MilliTimer.fired, Receive.receive y AMSend.sendDone. Estos se activan periódicamente durante la ejecución normal.

La aplicación se ha desplegado en ambos nodos presentados en la Fig. 2. Uno de ellos (llamado nodo local, *local node*) se conecta por USB a un PC para enviar los paquetes con los rastreos de llamadas y los indicadores, utilizando la UART del nodo sensor. En este caso, se seleccionó la capa de enlace de SerialActiveMessages. Otras opciones son CTP y ActiveMessages para transmisiones inalámbricas multi-salto o de un solo salto).

El código fuente de la aplicación se modificó automáticamente para incluir los mecanismos de registro necesarios para generar los rastreos de llamadas y los indicadores de ciclos de MCU para los eventos de nivel de aplicación, y para generar los rastreos de llamadas para las

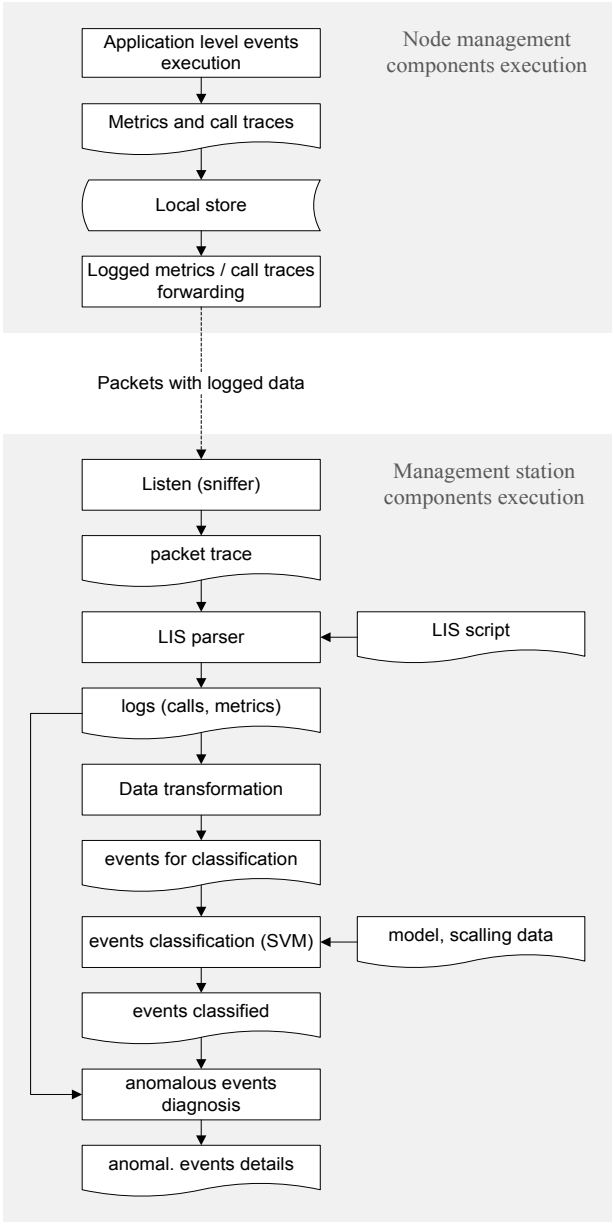

Fig. 1. Diagrama de flujo de detección y diagnóstico.

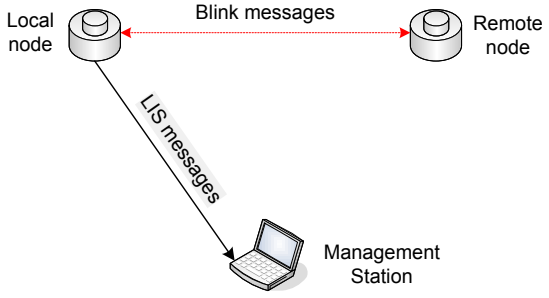

Fig. 2. Escenario de evaluación para el indicador 'ciclos de MCU'.

tareas ejecutadas. Los componentes que tienen sus funciones registradas fueron especificados en tiempo de compilación, evitando la modificación manual del código fuente de la aplicación.

Para la obtención del conjunto de entrenamiento, la aplicación RadioCountToLeds fue ejecutada durante 15 minutos en condiciones normales. En seguida el rastreo de

paquetes fue analizado por LIS, transformado para eliminar eventos de nivel de aplicación incompletos y para contruir la lista de eventos en el formato LIBSVM, y finalmente se sometió a la secuencia de comandos LIBSVM para construcción del modelo. El porcentaje de eventos clasificados erróneamente en el conjunto de entrenamiento se fijó en 1%. El conjunto utilizado para entrenar el clasificador contenía 1486 instancias de eventos.

Todos los experimentos tuvieron 5 minutos de duración y los resultados se presentan en la Tabla I. Antes de proceder al análisis de los resultados, dos aspectos deben ser destacados. En primer lugar, hay que señalar que, en la ausencia de problemas, el porcentaje de eventos clasificados como normales debe ser alrededor de 99% (ya que el umbral definido para los eventos clasificados erróneamente en la fase de entrenamiento fue de 1%). En segundo lugar, en la Tabla I, columna «Observaciones», se presentan los datos que aparecen en los registros de sucesos anómalos y que proporcionan pistas para la clasificación de los acontecimientos.

El experimento #1 fue diseñado para evaluar cómo la funcionalidad de detección de anomalías reacciona ante un fallo permanente (por ejemplo el nodo remoto bloquea y deja de transmitir sus mensajes). El bajo porcentaje de eventos normales (más de 40% de los eventos fueron clasificados como anómalos) apunta claramente a algún tipo de error. Este fallo fue diagnosticado fácilmente mediante la observación de la ausencia de eventos Receive.receive en los registros.

El objetivo de los experimentos #2 y #3 fue determinar si un error lógico (que resultó en la ejecución de código adicional) podría ser detectado. El código del evento MilliTimer.fired fue cambiado para que contuviese un ciclo que incrementaba un contador de 1 a 100 (o a 1000 en el caso del experimento #3). Este ciclo fue ejecutado en 10% de las ejecuciones de eventos MilliTimer.fired. En ambos casos, los eventos anómalos (un exceso de eventos MilliTimer.fired con un alto valor de ciclos MCU) fueron detectados, como se indica por un porcentaje de eventos normales por debajo de 99%.

El objetivo del experimento #4 fue ligeramente más ambicioso: determinar si otro tipo de error lógico podría ser detectado. Específicamente, en este caso, el contador fue incorrectamente incrementado dos veces en 10% de los casos. Esto se hizo mediante la modificación del código del evento MilliTimer.fired. El impacto en la aplicación que se ejecuta en el nodo local fue mínimo y, por lo tanto, no se detectó.

Otro conjunto de experimentos pretendió evaluar la

Tabla I
RESUMEN DE CLASIFICACIÓN (INDICADOR 'CICLOS DE MCU')

#	Condición	% eventos normales	Eventos (norm/total)	Observaciones
1	Nodo remoto apagado	58.21%	163/280	Ausencia de registros Receive.receive
2	100 i++	98.66%	443/449	Solo MilliTimer.fired > 21750
3	1000 i++	95.53%	406/425	Solo MilliTimer.fired > 23600
4	Extra op	99.04%	416/420	No detectado

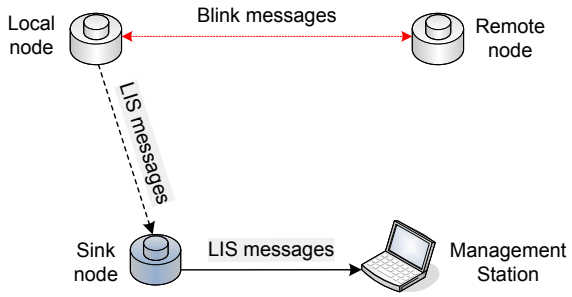

Fig. 3. Escenario de evaluación para el indicador 'consumo de energía'.

eficacia del indicador 'consumo de energía' en la detección de comportamientos anómalos. En estos experimentos la topología era una estrella con dos nodos que ejecutaban la aplicación RadioCountToLeds, y otro nodo cuya función era recoger los paquetes con los registros y enviarlos, a través de USB, a la estación de gestión (Fig. 3). La aplicación RadioCountToLeds fue modificada para escuchar la red en modo de bajo consumo (*Low Power Listening*) y el temporizador utilizado para enviar los mensajes se incrementó de 1,9 segundos a 5 segundos. La duración de los experimentos se aumentó a 10 minutos. La Tabla II presenta los resultados de los experimentos.

En el experimento #5 se reinicia un nodo 5 veces. Mediante el análisis de los registros del otro nodo, fue posible detectar la existencia de un comportamiento anómalo (como se indica por un porcentaje de eventos normales por debajo de 99%), y posteriormente identificar eventos anómalos AMSend.sendDone con valores de alta energía. Esto parecía indicar que algunos mensajes no fueron enviados por el nodo reiniciado o que se perdieron.

En el experimento #6 el porcentaje significativamente menor de eventos normales da una pista sobre alguna anomalía. Después de inspeccionar los registros para los eventos anómalos, se detectó un reinicio del nodo local (específicamente, uno de los eventos anómalos era un Boot.booted).

En el experimento #7, un nodo se movió lejos del otro, a un piso más arriba. El porcentaje relativamente bajo de eventos normales, junto con un menor número de eventos en el conjunto de datos del nodo que no se movía, era un indicio de problemas. Mediante la inspección de los registros de sucesos anómalos fue posible identificar varios eventos anómalos AMSend.sendDone con valores de alta energía. Esto sugirió que se habían producido pérdidas de paquetes, pero fue necesario un análisis de los registros del otro nodo para llegar a una conclusión.

El experimento #8 se diseñó para determinar si un estado

Tabla II
RESUMEN DE CLASIFICACIÓN (INDICADOR 'CONSUMO DE ENERGÍA')

#	Condición	% eventos normales	Eventos (norm/total)	Observaciones
5	5 reinicios remotos	98.77%	322/326	AMSend = 6.213.697
6	1 reinicio local	96.93%	347/358	Boot.booted
7	Un piso arriba	91.97%	229/249	AMSend > 6.000.000
8	Giroscopio encendido	75.36%	260/345	AMSend ~3.200.000

erróneo de energía en un dispositivo sería detectable. Específicamente, no se apagó el giroscopio en el momento de arranque con el fin de crear una anomalía. El porcentaje considerablemente bajo de eventos normales indica claramente que algo andaba mal. Además, la existencia en los registros de varios eventos AMSend.sendDone anómalos con los valores de energía mayor que 3.000.000 lo confirmó. Sin embargo, ya que este tipo de problema no afecta a la ejecución del programa, no hubo información de los registros que permitiera a su diagnóstico.

A la luz de los resultados obtenidos, es evidente que la utilización de indicadores simples combinados con el análisis de registros y la minería de datos permite la detección de condiciones más anómalas. Debe tenerse en cuenta que los objetivos de esta implementación de prueba de concepto y de los experimentos asociados fueron la evaluación de la eficacia de la detección automática de anomalías, no el diagnóstico en sí.

B. Análisis de requisitos

El objetivo de desarrollar y evaluar la implementación presentada era, por un lado, validar los conceptos en los que se basa – a saber, el uso de indicadores simples, el uso de una herramienta de registro ligera, y la minería de datos – y, por otra parte, para evaluar su capacidad para satisfacer las necesidades identificadas. En el apartado anterior, la herramienta se evaluó con respecto a la primera. En el actual sub-sección, se presenta un análisis para esta última.

Escalabilidad – En la implementación actual, el análisis, transformación de datos y las tareas de clasificación requeridas para procesar un registro de 10 minutos tardó menos de 1,5 s (1.295 s, 0,172 s, y s 0,023, respectivamente) por nodo sensor, en un ordenador Intel Core 2 Duo 2,4 GHz con 3 MBytes de memoria RAM. Este es un tiempo de procesamiento bajo. Para mantener bajos tiempos de análisis con un número elevado de nodos y con más componentes cuyas llamadas de funciones sean registradas, una implementación optimizada sólo debe enviar los indicadores en tiempo de detección, guardando localmente los registros de seguimiento de llamadas, para una inspección posterior a pedido.

Soporte de nodos sensores heterogéneos – la herramienta permite trabajar con dispositivos WSN con distinto *hardware* y *software*. Esto se realiza por agrupamiento de los registros, en el sistema de gestión, de acuerdo con su configuración de *software* y *hardware*. De esta manera, cada grupo de información de registro contiene datos de eventos de nodos sensores con el mismo tipo de *hardware*/*software*. Cada grupo se analiza entonces individualmente según el diagrama de flujo presentado en la Fig. 1.

Soporte de WSNs geográficamente dispersas – las herramientas implementadas de acuerdo con los principios presentados pueden trabajar de forma transparente con los datos de registro procedentes de múltiples redes de sensores inalámbricas, siempre que cada WSN esté conectada a Internet a través de una pasarela que se comunique con el sistema de gestión.

Soporte de nodos móviles – Las comunicaciones LIS pueden utilizar mensajes de TinyOS activas o CTP. Para aplicaciones WSN utilizando estos protocolos, la herramienta soportará el mismo patrón de movilidad que la aplicación. La inclusión de otros protocolos no es difícil, ya que el componente de recogida de registros es un componente separado, con interfaces bien definidas. En los experimentos que se llevaron a cabo no se evaluó el rendimiento de la herramienta en situaciones de movilidad de los nodos. Esto se dejó para trabajo futuro.

Independencia de los paradigmas de aplicación – la funcionalidad de detección se basa en la ocurrencia de eventos de nivel de aplicación en los nodos sensores monitorizados y utiliza indicadores generales. De esta manera, es compatible con todo tipo de paradigmas de aplicación. Sin embargo, para aplicaciones de redes de sensores inalámbricas donde los nodos de sensores están inactivos durante largos periodos de tiempo y sólo se activan raramente, este método no funcionará. Este es un problema común, no específico del método presentado, y la solución habitual es utilizar los mecanismos, ya sea iniciados por los nodos sensores o por el nodo *sink*, que permiten saber si un nodo sensor está activo o no. Si este tipo de mecanismo es soportado por la aplicación, la herramienta se beneficiará de él. La otra opción sería que el agente de gestión del nodo sensor forzara una recogida de indicadores si ningún evento se produce durante un período de tiempo predefinido.

Soporte de diversas plataformas de *hardware* y sistemas operativos – la mayor parte de la funcionalidad de detección de los nodos sensores se basa en LIS, siendo la excepción el cálculo de indicadores. En la actualidad, LIS es compatible con Mica2 / Z, TelosB y Hermes. Su uso con otros sistemas operativos, como Contiki, no aparenta ser complejo, dado que LIS opera mediante la modificación de las aplicaciones basadas en el lenguaje C.

Minimizar el uso de recursos – la MCU, RAM, ROM y el consumo se minimizan porque LIS es una herramienta de registro muy eficiente y porque el cálculo de indicadores es muy ligero. El impacto sobre la energía y ancho de banda depende del número de componentes que tienen sus llamadas a funciones registradas. Sólo el envío de registros de llamadas a pedido permitirá un mayor ahorro.

Ser fácil de instalar y de usar – en el caso de esta implementación de prueba de concepto, la implementación de la herramienta en un nodo sensor sólo necesita compilar la aplicación WSN con una opción que indica qué componentes deben tener su actividad registrada (en los experimentos de evaluación presentados, se trataba de la aplicación y del planificador). La facilidad de uso solamente puede ser evaluada con una plataforma integrada y no con una aplicación prototipo de prueba de concepto. Sin embargo, los experimentos no exigieron mucho trabajo de análisis.

Flexibilidad y extensibilidad – la implementación actual no permite la configuración posterior al despliegue de la funcionalidad de registro en los nodos sensores. La funcionalidad implementada se basa en una versión mejorada del mecanismo ROI existente en LIS. Este trabajo se puede extender fácilmente usando el lenguaje de script LIS, más indicadores y algoritmos adicionales de clasificación. Para eso sería necesario modificar el programa de análisis para lidiar con los nuevos indicadores, y en el desarrollo de scripts de Python para soportar los formatos de datos requeridos por los nuevos algoritmos de clasificación.

V. TRABAJOS RELACIONADOS

Varios trabajos han abordado el problema de la monitorización de WSN. Se mencionan brevemente los

principales en esta sección, con un enfoque en los que tuvieron un mayor impacto en la presente implementación de prueba de concepto.

MANNA [9] fue uno de los primeros sistemas de gestión de redes de sensores inalámbricas. A pesar de utilizar una arquitectura muy flexible y general, no incluye mecanismos de detección y diagnóstico para la gestión conjunta de múltiples redes de sensores inalámbricas.

SWARMS [10] se destina a la gestión de redes de sensores inalámbricas de área extensa, en distintas localidades, proporcionando funcionalidades de diagnóstico y programación. Fue diseñado para ser escalable, flexible y extensible. La mayor preocupación con esta arquitectura es el hecho de que se requiere que cada nodo sensor esté directamente conectado a un ordenador que ejecuta un proceso dedicado a ese nodo. Por otra parte, no comporta funcionalidad para permitir la detección y diagnóstico automáticos.

MARWIS [11] se dirige a la gestión de una WSN heterogénea dividiéndola en redes WSN homogéneas conectadas por una red de malla. Fue diseñado para ser escalable, flexible y extensible. El principal problema de esta arquitectura es que soportar heterogeneidad de nodos sensores dividiendo una WSN en un conjunto de redes de sensores inalámbricas homogéneas no encaja bien cuando hay necesidad de administrar varias redes de sensores inalámbricas heterogéneas pertenecientes a diversas organizaciones. Por otra parte, MARWIS asume que los nodos sensores están ejecutando aplicaciones basadas en Contiki y no existen disposiciones para soportar la detección y diagnóstico automáticos.

Sentomist [4] es una herramienta para la identificación de los posibles errores transitorios en aplicaciones WSN, que también utiliza SVM para identificar eventos anómalos. Sin embargo, el hecho de que se apoya en una métrica basada en la información de las instrucciones de procesador ejecutadas en cada evento, requiere el uso del emulador Avrora, lo que lo restringe a uso de laboratorio.

Por último, hay varias herramientas desarrolladas para ayudar a diagnosticar redes de sensores inalámbricas que operan en el campo. Hemos realizado un amplio estudio [12] que describe, analiza y compara un conjunto representativo de dichas herramientas. Ese trabajo nos ha guiado en el desarrollo del sistema actual para la detección de anomalías en redes WSN, y en la elección de LIS como herramienta base de esta implementación.

VI. CONCLUSIONES

Este artículo propone un método simple para la detección de anomalías en las redes de sensores inalámbricas, basado en el uso de dos indicadores generales, una estrategia de registro sencilla, y una técnica de aprendizaje automático. La tesis es que estos conceptos y soluciones son suficientes para desarrollar un sistema automatizado, independiente de la aplicación, ligero, capaz de monitorizar múltiples redes de sensores inalámbricas. Para evaluar esto, una implementación de prueba de concepto fue desarrollada y sometida a prueba. Los resultados han demostrado que el método propuesto tiene muy buen potencial y características, y es capaz de detectar anomalías de *hardware* y *software* de una manera muy eficaz, sin comprometer los requisitos identificados, tales como escalabilidad, heterogeneidad, generalidad y facilidad de uso.

El trabajo presentado en este artículo abre muchas líneas de trabajo futuro. En primer lugar, debe hacerse una evaluación más amplia y más general. Además, otros indicadores generales deben ser identificados y explorados. El desarrollo de una aplicación completa para uso en redes de sensores inalámbricas existentes también será muy interesante, así como el soporte para IPv6 (6LoWPAN) con el fin de aumentar la aplicabilidad de la herramienta.

AGRADECIMIENTOS

Trabajo parcialmente financiado por el proyecto CENTRO-07-ST24-FEDER-002003. Nos gustaría dar las gracias al Prof. Roy Shea, de UCLA, por aclarar varios aspectos de la operación de LIS, y a la Prof. Alicia Triviño-Cabrera, de la Universidad de Málaga, por sus comentarios, sugestiones y revisión del texto.

REFERENCIAS

[1] T. Hayes, M. Pavel, and J. Kaye, “Gathering the Evidence: Supporting Large-Scale Research Deployments,” Intel Technology Journal, 13(3), 2009.

[2] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, “Tinyos: An operating system for wireless sensor networks,” In W. Weber, J. Rabaey, and E. Aarts, editors, Ambient Intelligence. Springer-verlag, 2004.

[3] A. Rodrigues, M. Silva, T. Camilo, N. Blanco, J. Pedro, J. Martins, J. S. Silva, and F. Boavida, “Hermes: A versatile platform for wireless embedded systems,” Proceedings of the IEEE WoWMoM 2012, IEEE, San Francisco, CA, USA.

[4] Y. Zhou, X. Chen, M. Lyu, and J. Liu, “Sentomist: Unveiling Transient Sensor Network Bugs via Symptom Mining,” Proceedings of the IEEE ICDCS, pp. 784-794, 2010.

[5] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler, “Energy Metering for Free: Augmenting Switching Regulators for Real-Time Monitoring,” Proceedings of the IPSN 2008, IEEE, pp. 283-294, 2008.

[6] R. Shea, Y. Cho, and M. Srivastava, “LIS is More: Improved Diagnostic Logging in Sensor Networks with Log Instrumentation Specifications,” TR-UCLA-NESL-200906-01, 2009.

[7] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, “A Practical Guide to Support Vector Classification,” Technical Report, Department of Computer Science, National Taiwan University, (2010). [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[8] C.-C. Chang, and C.-J.Lin, “LIBSVM: A library for support vector machines,” ACM Transactions on Intelligent Systems and Technology 2(3), article no. 27, 2011.

[9] B. Ruiz, J. Nogueira, and A. Loureiro, “MANNA: a management architecture for wireless sensor networks,” IEEE Communications Magazine 41(2), pp. 116-125, 2003.

[10] C. Gruenwald, A. Hustvedt, A. Beach, and R. Han, “SWARMS: a sensornet wide area remote management system,” Proceedings of the TridentCom, 2007.

[11] G. Wagenknecht, M. Anwander, T. Braun, T. Staub, J. Matheka, S. Morgenthaler, “MARWIS: a management architecture for heterogeneous wireless sensor networks,” Proceedings of the WWIC, pp. 177-188, 2008.

[12] A. Rodrigues, T. Camilo, J. S. Silva, and F. Boavida, “Diagnostic Tools for Wireless Sensor Networks: A Comparative Survey,” Springer JNSM, Springer New York, 2012. doi: 10.1007/s10922-012-9240-6.