SOSFul: Sensor Observation Service (SOS) for Internet of Things (IoT)

J. Pradilla, M. Esteve and C. Palau, *Senior Member,* IEEE

***Abstract*— Sensor networks have expanded their scope and capabilities, becoming a fundamental part within the Internet of Things (IoT). To be able to meet the interoperability needs, it is necessary to have a repository of the sensors' data and observations, with this end in mind, the Sensor Observation Service (SOS) has been created; however, after several years since its last update, many technological changes have accumulated. This paper proposes and evaluates a SOS with a REST architecture and a JSON over HTTP message exchange, which is used to interconnect sensors networks and, at the same time, improve the performance of these networks and facilitate their deployment within the IoT.**

***Keywords*— Sensor Observation Service (SOS), Internet of Things (IoT), Sensors Networks, Sensor Web Enablement (SWE).**

I. Introducción

LA INTERNET DE LAS COSAS (IoT) es un concepto en auge que engloba la capacidad de generar, transmitir, almacenar y procesar información sin la intervención humana [1]. Dentro de este concepto, las redes de sensores son una fuente importante de información, debido a que están compuestas por una gran cantidad de sensores, los cuales realizan medidas de forma periódica o atienden a circunstancias específicas (eventos), generando así un pequeño dato que circula por la red para ser almacenado y/o analizado.

A causa de la necesidad de interoperabilidad de los datos que circulan por las redes de sensores y con la finalidad de habilitar las capacidades de: almacenar, procesar y consultar estos datos, el Open Geospatial Consortium (OGC) crean el marco de estándares llamado Sensor Web Enablement (SWE) que agrupa servicios y modelos de datos. Dentro del SWE destaca el Sensor Observation Service (SOS), el cual brinda la posibilidad de interconexión entre dispositivos heterogéneos mediante interfaces SOAP bien conocidas y mensajes XML [2][3]. Estos mensajes son ricos semánticamente y basados en los modelos de datos del SWE, concretamente en: Sensor Model Language (SensorML) [4] para los sensores y Observations and Measurements (O&M) [5] para las observaciones.

Desde su concepción el SOS responde a una inmensa cantidad de casos de uso, siendo versátil y flexible; sin embargo, con el paso de los años se ha identificado que buena parte de los casos de los uso actuales para el SOS, pueden ser modelados como un subconjunto reducido de las posibilidades que brinda el estándar.

Dado que el estándar SOS lleva años en desarrollo, se puede acceder actualmente a aplicaciones maduras y probadas que lo implementan completamente; como también, a alternativas más recientes, que atienden a una necesidad de simplificación y uso de recursos limitados.

De esta forma, se extienden dos aproximaciones para el despliegue de un SOS: la primera de ellas utiliza un servidor centralizado que cuenta con amplios recursos computacionales y que es capaz de manejar la complejidad del estándar; mientras que la segunda, emplea de forma distribuida varios dispositivos con recursos limitados y una versión simplificada del estándar. Sin embargo, en estas dos aproximaciones se ha dejado de lado el hecho de que el ancho de banda suele estar restringido, bien sea por ahorro de potencia o debido al medio compartido que utilizan para la comunicación.

Así, ajustándose al panorama actual de la IoT y a las estimaciones de crecimiento para los próximos años, entre 50 y 100 billones de dispositivos conectados para el año 2020 [6]; al tiempo que, se considera la importancia de las redes de sensores dentro de la IoT, se hace evidente la necesidad una tecnología que pueda masificar el uso de las redes de sensores, la cual debe considerar el uso de dispositivos de bajo costo y un ancho de banda ajustado. Con esto en mente, un SOS que se ejecute en dispositivos distribuidos de recursos limitados, junto con un mejor aprovechamiento del ancho de banda, será un factor clave para permitir el crecimiento esperado de la IoT.

Para profundizar en esta propuesta, este artículo se estructura de la siguiente forma: en la sección II se presenta el trabajo relacionado explicando el estándar actual, los desarrollos de software basados en este, la opción de tener un estándar ligero y las nuevas tecnologías que pueden habilitar un SOS adaptado a la IoT; la sección III presenta el SOSFul , sus funcionalidades y desarrollo; por su parte, la sección IV evalúa las prestaciones del SOSFul y las compara frente a otras implementaciones; y finalmente, se presentan las conclusiones del trabajo realizado y las posibles líneas de trabajo futuro.

II. Trabajo relacionado

Repositorio de datos en la IoT

Son muchas las plataformas de IoT que se estantan gestando al rededor del mundo y casí todas definen la necesidad de tener un repositorio de datos y metadatos; sin embargo, la mayoría no han definido tecnologias propias para

J. Pradilla, Universidad Autónoma de Occidente (UAO), Cali, Colombia, jvpradilla@uao.edu.co

M. Esteve, Universitat Politècnica de Valencia (UPV), Valencia, Spain, mesteve@upvnet.upv.es

C. Palau, Universitat Politècnica de Valencia (UPV), Valencia, Spain, cpalau@dcom.upv.es

realizar tal labor. Siendo común el dejar este repositorio unicamente expresado como un componente de base de datos relacional, con PostgreSQL y MySQL a la cabeza, o no relacional, donde MongoDB es la tecnologia imperante.

Sin embargo, existen dos trabajos relevantes con respecto a los respositorios de datos en la IoT. El primero de ellos, es la especificación ontologica del estandar OneM2M[16] en la cual se definen cada un de los conceptos clave dentro de un modelo de IoT y de los cuales se comparte en buena medida la definición del sensor y la observación que se hace en este artículo. El segundo trabajo relevanete, es de la propia OGC, que ha tomado los estandares de SWE para realizar una adaptación de los mismos a la IoT [17], por lo reciente de esta publicción aun no se encuentran desarrollos, sin embargo, como en el caso anterior, la propuesta de este trabajo, esta en la línea que ha marcado el OGC.

Sensor Observation Service (SOS) y SOS Lightweight

En el 2012 el OGC publico la más reciente versión (2.0) del estándar SOS [3] dentro del marco del SWE. El estándar es modular compuesto de cuatro extensiones: Core, Enhanced, Transactional y Result Handling. Cada extensión agrupa un conjunto de operaciones para realizar acciones sobre sensores y observaciones (las mediciones de los sensores) y dependiendo de los casos de uso en los cuales se va a desplegar, una implementación basada en el estándar SOS puede tener una o varias extensiones, siendo el Core obligatoria en todos los casos [2].

En la actualidad, existen diversas implementaciones del SOS que se adaptan a una amplia variedad de casos. Por ejemplo, en una aproximación de servidor centralizado se puede optar por 52°North SOS, mientras que el SOSLite es ideal para una aproximación distribuida; en más detalle:

- 52°North SOS: es una iniciativa de código abierto con licenciamiento GPL (GNU General Public License); escrito en Java. Además, cuenta con una fuerte comunidad que brinda soporte y documentación. En la actualidad se encuentra en su versión 4.x y soporta las extensiones: Core, Enhanced, Transactional, and Result Handling, de la versión 2.0 [3] del estándar SOS. Es lo suficientemente madura como para ser utilizadas de forma comercial.
- SOSLite: es una implementación de cogido abierto escrita en PHP, con licencia GPL. Está basada en el estándar SOS 2.0 [3] y las buenas prácticas para un perfil SOS ligero [7]. Fue especialmente diseñado para funcionar en dispositivos de capacidades limitadas y con un rendimiento excepcional al desplegarlo sobre un servidor NGINX [2].

Puesto que el SOS puede exceder la complejidad necesaria para ser utilizado en múltiples casos de uso, el OGC recomienda una forma de simplificar las operaciones de las extensión Core [7], para esto, reduce la complejidad de las operaciones y se ajusta únicamente a los escenarios con sensores estacionarios; además, limitan los lenguajes para modelar los sensores (SensorML [4]) y las observaciones (O&M [5]). Con estas medidas, el OGC mantiene la compatibilidad con el estándar SOS versión 2.0 y hace que el SOS pueda desplegarse en dispositivo con unos recursos más limitados.

Tecnologías para un SOS adaptado a la IoT

El OGC desde el momento en que estandarizo el SOS [3] plasmo la intención de crear una versión o extensión basada en REST; con este fin, cita como referencia el articulo “A RESTful proxy and data model for linked sensor data” [8]. Es comprensible entonces, el pensar que ya se vislumbraba un SOS/REST dentro de los encargados de la especificación, en contraposición al SOS/SOAP de la actualidad.

Para empezar, tradicionalmente el SOS/SOAP ha funcionado utilizando el protocolo HTTP, aunque al ser un servicio sobre SOAP puede usar otros protocolos de comunicación para la transmisión de mensajes. HTTP (Hypertext Transfer Protocol) es un protocolo de comunicaciones sin estado, que funciona sobre TCP/IP, en el cual los mensajes se transmiten en texto plano y están constituidos por una línea inicial, unas cabeceras y un cuerpo del mensaje. La línea inicial en las peticiones está compuesta por el método (GET, POST, PUT, DELETE, etc.), el identificador del recurso URL (Uniform Resource Locator) y la versión del protocolo.

Por otra parte, REST (Representational State Transfer) es un estilo arquitectural para aplicaciones en sistemas hipermedia distribuidos y fue creado por uno de los principales autores de la especificación HTTP [9]. Por su parte, la identificación de recursos (elementos de información) que utiliza REST se realiza mediante Uniform Resource Identifier (URI), la cual es una cadena de caracteres que identifica un recurso de forma unívoca en toda la red, un ejemplo de URI es la URL empleada en HTTP.

Aunque REST es independiente del protocolo y por lo tanto no está acoplado a HTTP, muchos servicios que siguen la arquitectura REST utilizan HTTP como protocolo de transporte, esto no es una coincidencia dado que algunos métodos HTTP: GET, POST, PUT and DELETE, se corresponden directamente con las operaciones REST.

Por otra parte, son muchos los servicios REST/HTTP que utilizan JSON (JavaScript Object Notation) como lenguaje para el modelado de los datos transmitidos. JSON es un lenguaje independiente para el formato de datos que se deriva del lenguaje de programación JavaScript. En comparación con modelos de datos basados en XML, con JSON se obtiene una mayor sencillez sintáctica y un menor ancho de banda necesario para la transmisión de los mensajes. Aunque el estándar SOS no especifica ninguna intención de cambiar los lenguajes utilizados para modelar los sensores y las observaciones, comienza a ser evidente que modelos basados en JSON pueden remplazar el SensorML y el O&M.

SOSFul y otras implementaciones de SOS

La propuesta realizada en este artículo, es funcional en despliegues centralizados y distribuidos, siendo una

TABLA I.
COMPARACIÓN DE LAS CARACTERÍSTICAS DE LAS IMPLEMENTACIONES DE SOS

	52º North	SOSLite	SOSFul
Compatibilidad con el estándar SOS	Total	Parcial	Ninguna
Compatibilidad con el estándar SensorML	Total	Parcial	Ninguna
Compatibilidad con el estándar O&M	Total	Parcial	Ninguna
Consumo de recursos computacionales	Alto	Bajo	Bajo
Consumo de ancho de banda	Alto	Medio	Bajo

alternativa que mejora el rendimiento de forma considerable. Sin embargo, esta mejora conlleva algunas modificaciones con las cuales se pierde compatibilidad con el estar. La tabla I resume las principales características de las tres alternativas estudiadas.

III. SOSFUL

El SOSFul dentro de la IoT

El SOSFul se ha desarrollado para ser un servicio de almacenamiento de datos de las redes de sensores, que se caracteriza por ser distribuido y liviano, al tiempo que, tiene un bajo consumo de recursos. Dado la alta relevancia de las redes de sensores dentro de la IoT, es consecuente la integración del SOSFul dentro de esta.

Con esto en mente, se puede determinar el lugar que ocupa el SOSFul dentro de las arquitecturas propuestas para IoT. Para empezar, en la arquitectura de 3 capas de la Fig. 1.a [10] el SOSFul se ubica dentro de la capa de aplicación y se constituye como un servicio horizontal que puede ser aprovechado por otros componentes de esa capa. Después, en las arquitecturas de 5 capas de la Fig. 1.b [11] y de la Fig. 1.c [12] el SOSFul hace parte de la capa de "Middleware" y de "Object Abstraction" respectivamente, en estas capas se brinda una representación abstracta y homogénea de los sensores y las observaciones que puede ser utilizada por las capas superiores. Finalmente, en la arquitectura de la Fig. 1.d [13][14] el SOSFul hace parte la capa de "Processing" donde sirve como almacenamiento de los datos que provienen de las capas inferiores y permite que los servicios de análisis y

TABLA II.
OPERACIONES DEL SOSFUL Y EQUIVALENCIAS EN SOS 2.0.

IoTSOS					**SOS 2.0**	
URI	**Método**	**Consulta**	**Contenido**	**Funcionalidad**	**Operación**	**Extensión**
/	GET	- section	N/A	Meta información servicio	GetCapabilities	Core
/sensor	GET	- spatialFilter - observedProperty - featureOfInterest	N/A	Lectura de todos los sensores	DescribeSensor	Core
/sensor	POST	N/A	sensor	Creación de un sensor	InsertSensor	Transactional
/sensor/:id	GET	N/A	N/A	Lectura de un sensor	DescribeSensor	Core
sensor/:id	UPDATE	N/A	sensor	Actualización de un sensor	UpdateSensorDescription	Transactional
/sensor/:id	DELETE	N/A	N/A	Eliminación de un sensor	DeleteSensor	Transactional
/observation	GET	- spatialFilter - temporalFilter - sensor - observedProperty - featureOfInterest	N/A	Lectura de todas las observaciones	GetObservation	Core
/observation	POST	N/A	observación	Creación de una observación	InsertObservation	Transactional
/observation/:id	GET	N/A	N/A	Lectura de una observación	GetObservaitonById	Enhanced
/observation/:id	UPDATE	N/A	observación	Actualización de una observación	N/A	N/A
/observation/:id	DELETE	N/A	N/A	Eliminación de una observación	N/A	N/A
/feature	GET	- spatialFilter	N/A	Lectura de todas las áreas de interés	GetFeatureOfInterest	Enhanced
/feature	POST	N/A	área de interés	Creación un área de interés	N/A	N/A
/feature/:id	GET	N/A	N/A	Lectura de una área de interés	N/A	N/A
/feature/:id	UPDATE	N/A	área de interés	Actualización de un área de interés	N/A	N/A
/feature/:id	DELETE	N/A	N/A	Eliminación de un área de interés	N/A	N/A

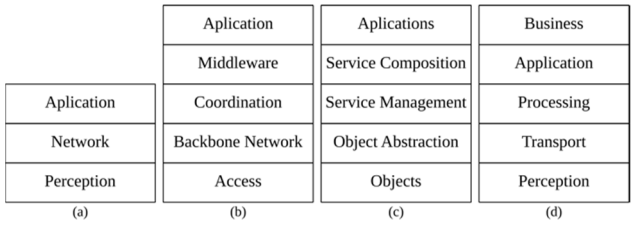

Figura 1. Arquitecturas de Internet de las Cosas (IoT).

procesamiento de información de su misma capa puedan abstraer las realidad física de la capa de percepción.

Funcionamiento del SOSFul

El SOSFul permite la gestión de los datos relacionados con las áreas de interés, los sensores y las observaciones [3]. Sobre estos datos permite las operaciones de: creación, lectura, actualización y eliminación (CRUD: Create, Read, Update and Delete) (Tabla II). Así, el SOSFul difiere parcialmente del estándar al ofrecer todas las operaciones como un núcleo y no como la unión de diversas extensiones (Tabla II). Además, ofrece las operaciones de actualización y eliminación de observaciones, las cuales no hacen parte del estándar SOS 2.0, al igual que con algunas operaciones para la gestión de las áreas de interés. El SOS Ful también brinda una operación equivalente a GetCapabilities (Tabla II) para obtener meta-información sobre el servicio.

Asimismo, el SOSFul es una implementación de SOS ligero que toma como referencia la simplificación planteada para la extensión core en la recomendación "OGC® Best Practice for Sensor Web Enablement Lightweight SOS Profile for Stationary In-Situ Sensors" [7] y la extiende a las otras extensiones del estándar SOS 2.0 [3].

Por consiguiente, para entender su funcionamiento y equivalencia con el SOS 2.0 (Tabla II) es necesario establecer un paralelismo entre las operaciones del SOS 2.0 y una dupla URI + método. De forma similar, los filtros disponibles en el estándar pasan a ser parte del parámetro de consulta en la URI, enviados en formato JSON. Así mismo, la descripción de las áreas de interés, los sensores y las observaciones hacen parte del contenido del mensaje HTTP. La siguiente URL (Tabla III) ejemplifica como obtener todas las observaciones de temperatura que se han realizado a la 1 p.m. del 30 de abril de 2016 dentro de un área geográfica cuadrada, utilizando el método GET de HTTP sobre una instalación de SOSFul en la maquina local.

Adicionalmente, los mensajes también se acogen a las simplificaciones planteadas en las buenas practicas publicadas por el OGC [7]; además, se abandonan los modelos de datos basados en XML (SensorML y O&M) en favor de unos modelos equivalentes en JSON. De esta forma, la descripción de las áreas de interés, los sensores y las observaciones ocupan menos bytes al momento de ser transmitidas y almacenadas, al tiempo que su procesamiento se facilita. Estas modificaciones aplicadas sobre los mensajes aumentan el rendimiento general de la red y disminuye el consumo de recursos computacionales.

TABLA III.
OBTENCIÓN DE OBSERVACIONES EN EL SOSFUL

```
http://localhost:3000/observation?q={
  "spatialFilter":{
    "operator": "within",
    "operand": "Polygon",
    "parameter": {
      "coordinates": [[[5, 7], [5,11], [6,11], [6,7], [5,7]]]
    }
  },
  "temporalFilter":{
    "operator": "equals",
    "operand": "TimeInstant",
    "parameter": {
      "attribute": "phenomenonTime",
      "time": ["2016-04-30T13:00:00.000Z"]
    }
  },
  "observedProperty": ["temperature"]
}
```

Entonces, partiendo de las especificaciones y buenas prácticas del OGC, se han creado modelos ricos semánticamente y que son acordes a un modelo ontológico para la IoT [15]. Los cambios en los modelos implican que todos los enlaces a otros modelos deben ser reverenciadas por un identificador único en el sistema el cual es asignado al ingresar estos modelos a priori, por ejemplo, las áreas de interés de una observación deben ingresarse antes mediante el uso de POST + /feature; de la misma forma, las estructuras de agrupación, cuya etiqueta en SOS 2.0 suele tener el sufijo "List", se transforman en arreglos JSON, esto se aplicaría por ejemplo a las "keywords" de los sensores, también, los "Term" pasan a ser un objeto JSON con un "label" y un "value", como en el caso de la "identification" del sensor.

De igual forma, existen otras modificaciones que permiten simplificar y optimizar los modelos sin perder riqueza semántica, la mejor forma de apreciar las ventajas que esto conlleva, es observar un ejemplo, el siguiente fragmento (Tabla IV) modela una observación de tipo texto y se puede comparar con la versión en O&M de las buenas practicas del OGC [7]:

TABLA IV.
OBSERVACIÓN DE TIPO TEXTO EN EL SOSFUL

```
{
  "type": "String",
  "sensor": "56ab90322c697f063d243bc4",
  "observedProperty": "def:phenomenon:OGC:species:flyingHorse",
   "featureOfInterest": "56ab900f2c697f063d243bc3",
   "phenomenonTime": "2005-01-11T16:22:25.00",
   "resultTime": "2005-01-11T16:22:25.00",
   "result": {
       "value": "I've observed a flying horse."
  }
}
```

Implementación

En la implementación del SOSFul se evaluaron diversas tecnologías para determinar: el lenguaje de programación, el servidor de aplicaciones y el gestor de base de datos. La selección de las tecnologías finalmente adoptadas se basó en tres criterios: las necesidades de eficiencia en uso de recursos computaciones, las potencialidades que brinda cada una de ellas y los datos que se almacenan y transmiten dentro de la IoT.

Así, Javascript se escogió como lenguaje de programación debido a que es un lenguaje funcional, que brinda desarrollos ágiles y que tiene una fuerte comunidad en la actualidad; lo que permite la existencia de excelentes librerías para desarrollar servicios con arquitectura REST y la manipulación de datos en formato JSON; al igual que, una amplia gama de conexiones a base de datos.

Del mismo modo, como gestor de base de datos se optó por MongoDB, que es ampliamente utilizado para almacenar datos en formatos no relacionales, concretamente en forma de documentos BSON (Binary JSON). Además, MongoDB se ha especializado en bases de datos distribuidas con alta redundancia, amplia flexibilidad y gran velocidad en operaciones de inserción y consulta; características que se ajustan a la amplia heterogeneidad de los datos que circulan por las redes de sensores y a las operaciones más empleadas por los sensores y las aplicaciones que funcionan dentro de la IoT.

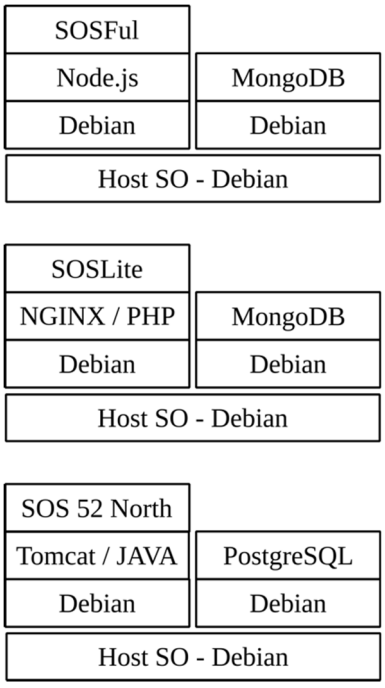

Figura 2. Esquema de los contenedores utilizados en el montaje.

Asimismo, se escogió Node.js como servidor de aplicaciones, el cual, brinda la posibilidad de utilizar Javacript como lenguaje de programación de aplicaciones y a su vez, tiene un alto desempeño con un consumo bajo de recursos computacionales. Node.js se ha desarrollado bajo un modelo de orientación a eventos, lo que le permite atender de forma simultánea a múltiples clientes sin ver afectado su rendimiento. Dada la naturaleza de las redes de sensores, donde muchos sensores reportan a un servidor centralizado en Internet o a un dispositivo que funciona como puerta de enlace a la IoT, un servidor que pueda recibir datos de diversas fuentes de forma eficiente es ampliamente recomendable.

IV. Evaluación

Entorno de Pruebas

Con el fin de evaluar el rendimiento del SOSFul en comparación con: SOSLite y SOS 52 North, se ha configurado un ambiente de pruebas sobre contenedores Docker, una aproximación más liviana al aislamiento que brindan las máquinas virtuales. Las pruebas se realizan de forma independiente, haciendo un reinicio programado con cada una de ellas, asegurando así, que el ordenador anfitrión siempre tiene la menor carga de trabajo posible antes de iniciar la ejecución de los contenedores.

Cada una de las implementaciones de SOS fue probada usando una composición de contenedores, donde en uno de ellos se ejecuta el SOS y en el otro su respectiva base de datos (Fig. 2). El ordenador anfitrión utiliza como sistema operativo la distribución Debian 8.3 con un kernel GNU/Linux 3.16, misma distribución que se utiliza en cada uno de los contenedores. Por su parte las versiones utilizadas dentro de las pruebas para los diferentes servicios son: SOSFul 1.0, SOSLite 1.0, SOS 52 North 4.3.6, JAVA 8u45, PHP 5, Tomcat 9.0, NGINX 1.6, Node.js 5.10, PosgreSQL 9.4 y MongoDB 3.2.

Como herramienta de generación de tráfico y medición de rendimiento se ha utilizado Apache JMeter (versión 2.13) y los conjuntos de extensiones “standard” y “extras” en su versión 1.3.1. De forma que, en el ordenador anfitrión se generan las operaciones y sus respectivos datos, al tiempo que, se miden los parámetros de red; mientras que en cada uno de los contendedores se despliega una sonda para monitorizar el uso de recursos computacionales de los servicios utilizados.

Las pruebas realizadas a cada una de las implementaciones del SOS evaluadas, consisten en la ejecución de las operaciones para insertar y obtener observaciones: InserObservation y GetObservation, o sus equivalentes en el SOSFul (Tabla I). Para esto se procede, en primer lugar, a borrar todos los contenedores y reiniciar el ordenador anfitrión; paso seguido, se inicia la composición de contenedores que va a ser probada; una vez los contenedores están en funcionamiento, se inicia el programa de pruebas y las sondas; por último, se ejecutan las pruebas y se almacenan los resultados para hacer analizados posteriormente. El ciclo descrito se realiza para cada una de las operaciones que se ejecutan sobre una implementación determinada.

Cada una de las pruebas están compuestas por 10 hilos, encargados de hacer las operaciones cada 100 ms, durante 300 segundos, obteniendo así un total de 30.000 muestras por cada una de las operaciones e implementaciones de SOS. Como ejemplo se puede tomar el caso del SOSFul (Fig. 3), donde las pruebas inician con la inserciones de 100 observaciones por segundo en un periodo de 300 segundos; una vez limpiado y reiniciado el ordenador anfitrión y los contenedores, se pasa a obtener 100 observaciones por segundo durante 5 minutos.

El tráfico modelado en las operaciones de inserción, coincide con una red de sensores que reporta las observaciones a una puerta de enlace con la IoT; aunque en los despliegues reales el tiempo entre cada una de las muestras suele ser mayor al escogido, el mismo es útil a fin de observar el comportamiento de las implementaciones ante condiciones de alta carga de la red. Por su parte las operaciones de consulta son propias de servicios de análisis de datos en tiempo real, que son comúnmente empleados en la IoT, como lo pueden ser los procesadores de eventos complejos (Complex Event Processing – CEP).

Resultados

Para determinar el rendimiento del SOSFul y compararlo con las otras opciones se miden: el uso de procesador (%), uso de memoria (%), throughpout (kbps), transacciones por segundo (#/s) y tiempo de respuesta (ms). Con las pruebas realizadas a cada una de las implementaciones evaluadas y para las dos operaciones escogidas, se presentan gráficamente los resultados obtenidos. En la Fig. 4 se condensan los resultados para las operaciones de inserción de observaciones, mientras que la Fig. 5 hace lo propio para la obtención de observaciones.

Como se puede observar el uso de procesador y memoria (Fig 4.a, 4.b, 5.a, 5.b) es menor en el SOSFul, seguido por el SOSLite y dejando como la implementación más demandante de recursos computacionales al SOS 52 North. Esta es una evidencia clara de la orientación hacia dispositivos limitados con el cual han sido diseñados el SOSFul y el SOSLite; y por ende, los presenta como las alternativas ideales en despliegues de SOS distribuidos.

El throughpout por su parte (Fig. 4.c, 5.c) es considerablemente menor en el SOSFul ; como en el caso

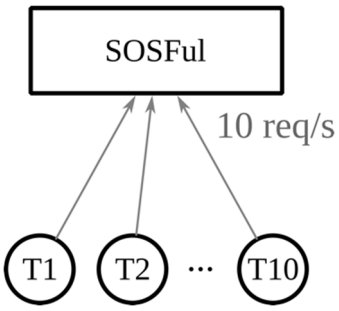

Figura 3. Esquema de las pruebas sobre el SOSFul.

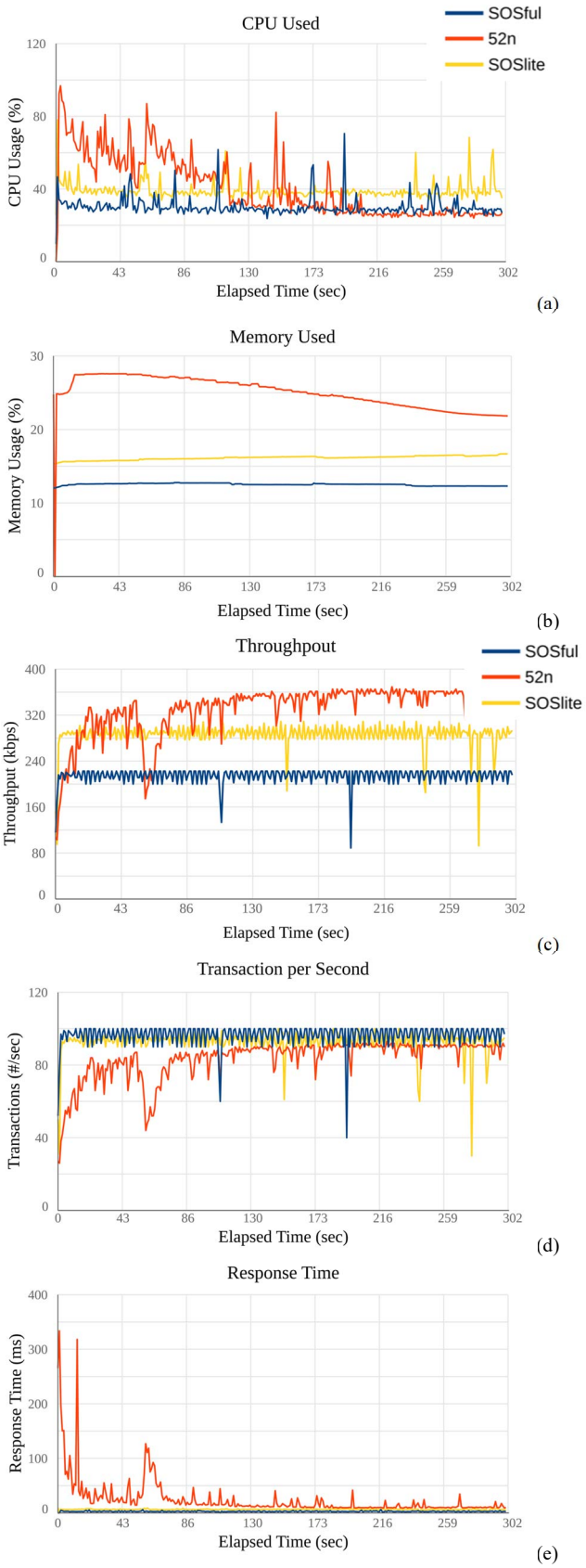

Figura 4. Resultados para la inserción de observaciones: (a) uso de procesador, (b) uso de memoria, (c) throughpout, (d) transacciones por segundo y (e) tiempo de respuesta.

anterior, el SOSLite sigue al desempeño del SOSFul . Estos resultados son consecuencia de las simplificaciones y adaptaciones realizados a los modelos de datos; que en el caso del SOSFul son mucho más condensados, permitiendo un menor consumo de ancho de banda y haciéndolo ideal para las redes de sensores de la IoT.

Así mismo, las transacciones por segundo (Fig. 4.d, 5.d), son muy similares en los tres casos, teniendo una mayor variabilidad en el caso del SOS 52 North pero estando muy cercanas al ideal de 100 transacciones por segundo que impone el tráfico modelado. Finalmente, el tiempo de respuesta (Fig. 4.e, 5.e) se encuentra por debajo de los 50 ms en todos los casos, siendo la menor la del SOSFul , seguida muy de cerca por el SOSLite, aunque este último presenta una mayor variabilidad. Es necesario recordar que debido a la naturaleza de las pruebas, no se cuenta con los tiempo de tránsito en red, con lo que los tiempos medidos están directamente relacionados con los tiempos de procesamiento; de esta forma, se obtiene un indicio claro de que el SOSFul y el SOSLite son más eficientes en el procesamiento de las operaciones, hecho que se puede relacionar con la reducción de los modelos de datos y en el caso del SOSFul , además, con la simplificación de las interfaces de comunicación mediante el uso de REST.

V. Conclusiones.

La Internet de las cosas (IoT) es actualmente la tendencia que marca el desarrollo tecnológico a nivel mundial y que se espera, permita revolucionar la interacción de los seres humanos con su entorno. Dentro de esta, las redes de sensores, son sin lugar a duda, el principal generador de información; hecho que conlleva a investigar la mejor forma de acoplar estas redes dentro de la IoT. Obteniendo así, un mejor uso de los recursos computacionales y de red, al tiempo que, se mejora la interoperabilidad en los entornos heterogéneos que las componen.

Por su parte, este artículo presenta un Sensor Observation Service (SOS) que adapta el estándar de la OGC para dar cabida a este servicio dentro de la IoT. Para esto, se simplifican las interfaces y los modelos de datos; y a su vez, se actualizan a tecnologías que son más fácil de utilizar en el desarrollo de programas y más eficientes al momento de ser procesadas.

Así, cada una de las decisiones de diseño tomadas en cuenta en el momento de desarrollar el SOSFul, han repercutido en un mejor desempeño en términos de consumo de recursos computacionales y aprovechamiento de ancho de banda. Haciendo del SOSFul una alternativa valiosa al momento de desplegar redes de sensores dentro de la IoT y como un punto de partida para la estandarización de futuras versiones del Sensor Observation Service.

Finalmente, se plantea continuar utilizando el SOSFul dentro de casos de uso de la IoT, como las ciudades inteligentes, de forma que se valide las bondades de este servicio y se concreten evidencias que se puedan presentar a la OGC con miras a una siguiente versión del estándar SOS.

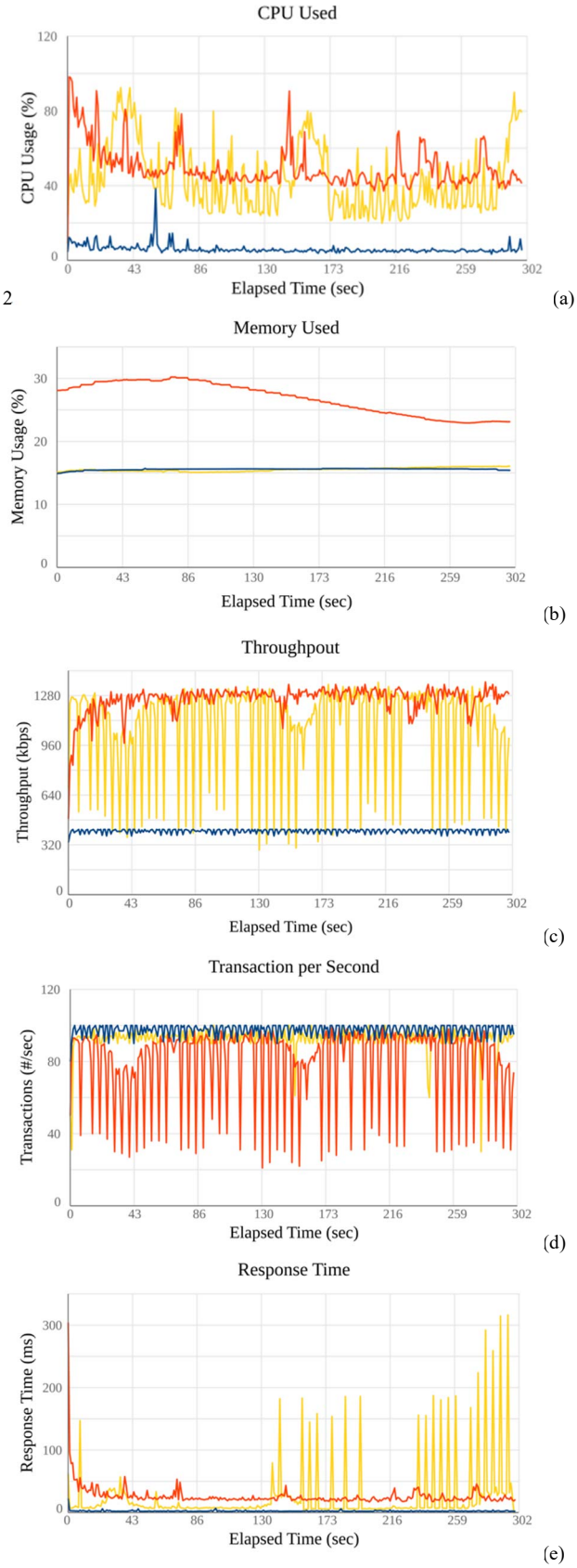

Figura 5. Resultados para la obtención de observaciones: (a) uso de procesador, (b) uso de memoria, (c) throughpout, (d) transacciones por segundo y (e) tiempo de respuesta.

REFERENCIAS

[1] C. Wang, M. Daneshmand, M. Dohler, X. Mao, S. C. Mukhopadhya, R. Q. Hu, and H. Wang, "Guest Editorial Special Issue on Internet of Things (IoT): Architecture, Protocols and Services," IEEE Sens. J., vol. 13, no. 10, pp. 3505–3510, 2013.

[2] J. Pradilla, C. Palau, and M. Esteve, "SOSLite: Lightweight Sensor Observation Service (SOS)," IEEE Lat. Am. Trans., vol. 13, no. 12, pp. 3758–3764, 2015.

[3] A. Bröring, C. Stasch, and J. Echterhoff, "OGC® Sensor Observation Service Interface Standard," 2012.

[4] M. Botts and A. Robin, "OGC® SensorML," 2014.

[5] S. Cox, "Observations and Measurements - XML Implementation," 2011.

[6] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and Challenges for Realising the Internet of Things," European Union, 2010.

[7] S. Jirka, C. Stasch, and A. Bröring, "OGC® Best Practice for Sensor Web Enablement Lightweight SOS Profile for Stationary In-Situ Sensors," 2014.

[8] K. Janowicz, A. Bröring, C. Stasch, S. Schade, T. Everding, and A. Llaves, "A RESTful proxy and data model for linked sensor data," Int. J. Digit. Earth, vol. 6, no. 3, pp. 233–254, 2013.

[9] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, 2000.

[10] Zhihong Yang, Yufeng Peng, Yingzhao Yue, Xiaobo Wang, Yu Yang, and Wenji Liu, "Study and application on the architecture and key technologies for IOT," 2011 Int. Conf. Multimed. Technol., pp. 747–751, 2011.

[11] L. Tan, "Future internet: The Internet of Things," 2010 3rd Int. Conf. Adv. Comput. Theory Eng., pp. V5–376–V5–380, 2010.

[12] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Comput. Networks, vol. 54, no. 15, pp. 2787–2805, 2010.

[13] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: The internet of things architecture, possible applications and key challenges," Proc. - 10th Int. Conf. Front. Inf. Technol. FIT 2012, pp. 257–260, 2012.

[14] M. Wu, T.-J. T. Lu, F.-Y. Ling, J. Sun, and H. Du, "Research on the architecture of Internet of Things," 2010 3rd Int. Conf. Adv. Comput. Theory Eng., pp. V5–484–V5–487, 2010.

[15] M. Ben Alaya, S. Medjiah, T. Monteil, and K. Drira, "Toward Semantic Interoperability in oneM2M Architecture," IEEE Commun. Mag., vol. 53, no. December, pp. 35–41, 2015.

[16] M. B. Alaya, S. Medjiah, T. Monteil and K. Drira, "Toward semantic interoperability in oneM2M architecture," in *IEEE Communications Magazine*, vol. 53, no. 12, pp. 35-41, Dec. 2015. doi: 10.1109/MCOM.2015.7355582

[17] S. Liang, C. Huang, and T. Khalafbeigi, "SensorThings API Part 1: Sensing" 2016.

Juan V. Pradilla received the Ph.D degree in Telecomunications from the Universitat Politècnica de Valencia (UPV) in 2016, M.Sc. degree in Communications Technologies, Systems and Networks from the UPV in 2013 and the M.Sc. degree in Computing and Telecomunications from the ICESI University in 2012. He is Full Professor in the Universidad Autónoma de Occidente, Cali, Colombia. His research activities and interests cover a wide range of subjects related to cyber-physical systems, Internet of Things, physical computer and wireless communications, particularly: sensor networks, cloud/fog computing, quantum communications, network security and next generation mobile networks.

Manuel E. Domingo received both his M.Sc. in computer engineering and his Ph.D. in telecommunication engineering (Dr.Ing.) from the Universitat Politecnica de Valencia in 1989 and 1994, respectively. He is Full Professor in the Escuela Técnica Superior de Ingenieros de Telecomunicacion at the Universitat Politecnica de Valencia (UPVLC), and he leads the Distributed Real-Time Systems research group. Prof. Manuel Esteve has more than 20 years of experience in the ICT research area in the area of Networking. Nowadays, he is managing several R&D projects at regional, national and international level. He has collaborated extensively in the R&D of projects for the government agencies, defence and EU-FP7 acting as chairman of the agreement between Spanish MoD and UPVLC. He is author and co-author of more than 100 research papers.

Carlos E. Palau received his M.Sc. and Ph.D. (Dr.Ing.) degrees, both in telecommunication engineering, from the Universitat Politecnica de Valencia in 1993 and 1997, respectively. He is Full Professor in the Escuela Tecnica Superior de Ingenieros de Telecomunicación at the Universitat Politecnica de Valencia. He has more than 18 years of experience in the ICT research area in the area of Networking. He has collaborated extensively in the R&D of multimedia streaming, security, networking and wireless communications for government agencies, defence and European Commission. He has been the main UPVLC researcher in the FASYS project, which has funded this work. He is author and co-author of more than 120 research papers and member of the TPC of several IEEE, ACM and IFIP conferences. He is Senior Member of IEEE.