Integrating Sensors with the Cloud Using Dynamic Proxies

Wei Wang, Kevin Lee, David Murray
School of Information Technology, Murdoch University,
Murdoch 6150, Western Australia, Australia
{W.Wang, Kevin.Lee, D.Murray}@murdoch.edu.au

***Abstract*— Wireless Sensor Networks (WSNs) can have high demands for real-time data transmission and processing, but this is often constrained by limited resources. Cloud Computing can act as the backend for WSNs to provide processing and storage on demand. This paper proposes a generic architecture to support the integration of sensors with the Cloud. It uses a lightweight component model and dynamic proxy-based approach to connect sensors to the Cloud. The feasibility of this approach is evaluated experimentally.**

***Keywords*— Wireless Sensor Networks, WSN, Cloud Computing**

I. INTRODUCTION

The Cloud Computing paradigm is being applied to many domains, including, scientific computation [1], e-commerce [2], online games [3] and industrial design [4]. The core features of Cloud Computing include cost-saving, virtualization, elastic resources, self-service interface and pay-per-use pricing models. Conversely, WSNs have limitations:

- Limited computational resources that cannot meet the requirement of elastic demand.
- Finite battery power
- The heterogeneous nature of WSN platforms can cause interoperability problems.
- Due to the upfront investment, deploying short life-cycle applications of WSNs is financially prohibitive [5].

The complementary characteristics of Cloud Computing and WSNs indicate that there are many advantages to integrate WSNs with Cloud Computing.

This paper introduces the Tangible Cloud as a new concept, and extends prior work [6], enabling Cloud applications to interact with WSNs and the physical world. Equally, WSNs may also offload processing capabilities to Cloud resources.

This paper proposes a generic architecture for integrating WSNs with the Cloud. It utilises an established loosely coupled component model [7] with the addition of a dynamic proxy-based approach to connect sensor motes to the Cloud. Furthermore, it is demonstrated through evaluation that it is feasible to deploy a universal architecture to integrate lightweight sensors with the Cloud.

The remainder of this paper is structured as follows. Section II introduces the background of Cloud Computing, WSNs, and their integration methods. Section III describes the proposed architecture using dynamic proxies to support lightweight WSNs. Section IV evaluates the feasibility of deploying the proposed architecture through experiments. Finally, Section V presents some conclusions.

II. BACKGROUND

A. Wireless Sensor Networks

Wireless Sensor Networks (WSNs) consist of distributed nodes with embedded CPUs and low-power radios. WSNs are mainly used to monitor environmental conditions, such as temperature, sound, and vibration and pressure [8]. The applications of WSNs can be applied in many areas, such as forest fire detection [9], medical monitoring [10] and pollution monitoring [11]. A generic wireless sensor consists of several components:

- A low-power CPU, a small amount of memory, and a small solid-state storage.
- A radio transceiver with an antenna for transmitting and receiving data.
- A microcontroller for interfacing with the sensors.
- Energy sources like batteries or other power supply.

In WSNs, the gathered raw data may include a large amount of irrelevant information. In order to reduce this volume of data, this raw data needs to be filtered, aggregated and processed [12]. WSNs have the following challenging problems:

- The battery-powered sensors are resource-constrained and depletion times vary from days to years.
- Nodes can be difficult to maintain in some deployment scenarios, such as for seafloor temperature monitoring.
- In the event of sensor node failure, the network must adapt dynamically, often resulting in topology change.
- The integration of heterogeneous sensor nodes into WSNs is not standardised.
- Scalability issues in large deployments.
- Complexity is a hindrance to the ease of deployment.

B. Cloud Computing

Cloud Computing is a collection of virtualized resources that can be assigned on demand. They use a pay-per-use model with service license SLAs [2]. Cloud Computing can also be considered as the convergence of Grid computing and service-oriented computing. Cloud Computing has four characteristics: 'abstracted or virtualized resources', 'Elastic resource capacity', 'Programmable self-service interface' and 'Pay-per-use pricing model' [6].

Cloud services can be classified into three types: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [13], described as follows.

- IaaS provides low-layer processing or data storage (e.g. Amazon EC2 and Amazon S3).

- PaaS provides developers with a platform to design their applications according to some specifications without the concern of hardware layer.
- SaaS delivers various off-the-shelf services in a pay-per-use pricing model. The service providers own and manage the resources. The users just directly use the services without the concern of complex cooperation among multiple providers (e.g. Salesforce.com).

The focus of this paper is the computational resource advantages of integrating WSNs and the Cloud; it therefore falls into IaaS. Cloud Computing is used in various forms [2]:

- *Public Clouds* are owned by third parties and offer services to paying clients.
- *Private Clouds* are Clouds owned by individuals or organizations that have the characteristic of full control by the owner.
- *Hybrid Clouds* are partly private and partly public, they allow users to have the security of a private Cloud and the resource potential of a public Cloud.
- *Federated Cloud* describes the cooperation of multiple Clouds to provide a consistent service.

The main issues of using the Cloud as the backend for WSNs are latency and the ability of the Cloud to support periodic events. A private Cloud would lack elasticity if thousands of wireless sensors were transmitting data to the cloud simultaneously. Conversely, a public Cloud would have high latency for interactive applications. In certain scenarios, large latencies could have catastrophic consequences. For example, the temperature monitoring in nuclear plants needs very fast responses to emergencies. Thus, the hybrid cloud is the logical choice for the WSN back-end as it can eliminate both the disadvantages of public and private Clouds. This paper proposes an architecture utilizing a hybrid Cloud, Eucalyptus [14] and Amazon Web Services [15], to meet the dynamic computational needs of WSN.

The elastic resources capability of the Cloud is the main motivation for integration WSNs with the Cloud. In some situations, elasticity is an operational requirement rather than cost saving [2]. In the area of environmental monitoring, exhaustion of resources could lead to untimely flood prediction [16]. However, it is cost prohibitive to manually update or re-task the infrastructure of emergency prediction. From a software engineering perspective, large-scale sensing applications can be made a reality when data is processed and stored in the Cloud. Cloud Computing can provide elastic resources to WSNs. Experiments have shown that Cloud Computing services have sufficient elasticity to process the collected data from periodic WSN events [5].

III. INTEGRATION OF WSN AND THE CLOUD

The integration of WSNs and Cloud Computing can be seen as the integration of WSNs with the traditional Internet with the addition of load-balancing and resource elasticity. There are six approaches to integrate WSNs with the Internet [17].

The first approach is based on message-oriented communication that uses low-level application-specific APIs to exchange messages between sensors and motes running platforms such as TinyOS [18] and Contiki [19]. These are integrated with the Internet by using a lightweight IP stack. This approach provides communication infrastructure, but doesn't provide application-level service distribution.

The second approach is to use SOAP-based Web services such as Tiny web services [20]. The application layer functionality of web services can be accessed via ports. The ports are described using Web Service Description Language (WSDL). Sensor data is encapsulated in SOAP packets that lead to high complexity and resource requirements.

The third approach uses the HTTP RESTful paradigm to control the status of wireless sensors. An example is TinyREST [21] that uses a gateway to connect sensor nodes to the Internet by mapping messages to HTTP requests. Another example is CoAP [22], which reduces the HTTP overhead by using a subset of HTTP. This approach adds overhead from processing HTTP requests.

The forth approach is based on the Universal Plug and Play (UPnP) architecture. It allows wireless sensors to connect each other in a universal environment [23]. However, as it is based on TCP/IP, UDP and HTTP, it requires a complete networking stack be present on the sensors.

The fifth approach uses socket communication, which adds the minimum overhead. However, it is just a communication mechanism with no application support.

The last approach uses a component-based model that uses RPC invocations in WSNs. For example, NesC [24] is a component-based, event-driven model used to build applications for the TinyOS platform. In this model, components are statically bound together via their interfaces. Predefining the static components allows for better analysis of whole programs. However, NesC becomes a single monolithic blob of code at compile time and only the full system image can be replaced after compilation. In this way, the static components cannot be reconfigured on runtime, thus not suitable for the WSNs in dynamic environments. Compared with NesC, OpenCOM [25] can support runtime reconfiguration. It encapsulates the WSN resource into a reusable component with predefined interface for discovery and resource management. However, OpenCOM is designed for relatively resource-rich platforms which are unsuitable for lightweight sensor nodes. OpenCOM is based on traditional RPC binding requiring developers to build relationships between single nodes instead of multicasting relationships to a group of nodes. Another example is LooCI [7], which provides an extensible networking framework and an event bus abstraction to bind reusable components. LooCI has a key advantage that its communication is based on Inter Isolate RPC (IIRPC). Its macro-components can support multiple threads and utility libraries by running each macro component in isolation [7]. LooCI can implement multicast to multiple nodes network-wide. The LooCI middleware has been deployed on a variety of platforms such as the SunSPOT, Contiki and OSGi. The proposed architecture in this paper adopts LooCI as its supporting middleware.

III. PROPOSED ARCHITECTURE FOR INTEGRATION

A. Aims

This paper proposes a new architecture that will allow the integration of any lightweight sensors with the Cloud. The proposed architecture is inspired by the advantages of both the message-oriented and component-based approaches. It utilises

the publically available LooCI middleware for component management and adds support for dynamic proxies. The sensor components can be dynamically generated according to the data structure messages; advantages include:

- Supporting generic lightweight sensors. The proposed architecture completely moves the middleware to the local proxy, saving energy on resource constrained WSN nodes. Running the middleware on the proxy is independent of the WSN platform.
- Individual connectivity and global interoperability: each sensor component has its own individual connectivity and can be accessed via a global identifier, which is a combination of public IP address and component ID.
- High-level programming APIs can be used to deploy third-party components. The developers can design their own components for different purposes. For example, a third-party component is designed to convert temperature from Kelvin to Celsius.
- Runtime reconfiguration can improve resource management. As most wireless sensors are resource-constrained devices, developers can only deploy the required components and remove the unused components on runtime.
- Overheads are reduced when compared with the encapsulation/de-encapsulation of SOAP envelopes.

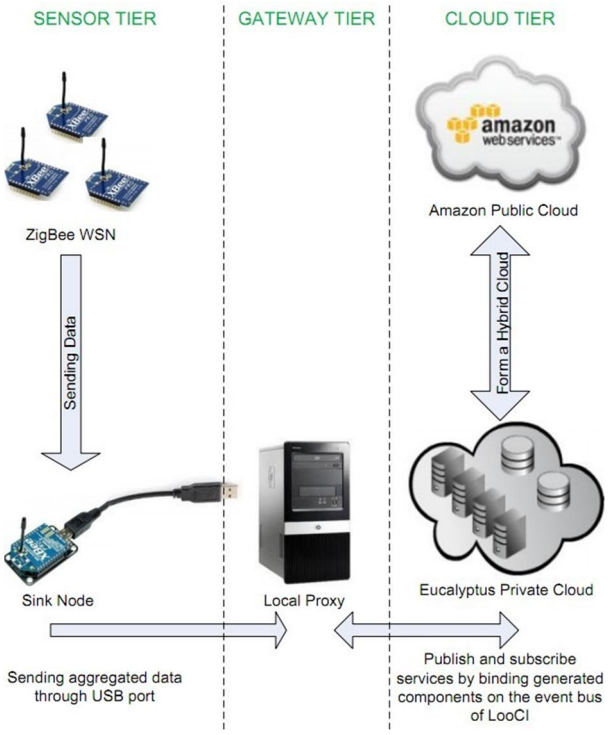

Fig. 1 The proposed architecture

B. The Overall Architecture

Fig. 1 illustrates the proposed generic architecture to integrate lightweight sensors with the Cloud. At the Sensor Tier, the sink node is used to gather the environmental data from a group of wireless sensors. The aggregated sensor messages are then sent to local proxy via a local communication mechanism. At the Gateway Tier, the local proxy parses received messages and then generates dynamic components for each new sensor detected. Components can be bound and unbound over the event bus. The gathered sensor data can be modelled and relayed to the Cloud. The Cloud Tier acts as a hybrid Cloud back-end of WSNs, combining Eucalyptus Private Cloud [14] and Amazon Web Services [15]. The Eucalyptus Private Cloud can integrate with Amazon Web Services via published APIs, thus integrating the private and public Clouds [26]. Both the Gateway Tier and Cloud Tier are implemented on the event bus of the LooCI middleware.

C. The Proposed Architecture

Fig. 2 illustrates an overview of the proposed architecture – based on the LooCI architecture with the addition of dynamic proxies, which enables any sensors to connect to the Cloud seamlessly – described as follows.

Component Factory: The purpose of Component Factory is to parameterize the gathered data and then dynamically generate node components for different types of sensors. In the proposed architecture, a group of sensors send data to a sink node, and then the sink node passes a stream to the Component Factory through a serial port. The gathered data contains multiple-lines of text. Each line is received from a unique sensor node with its own data structure. For example, a line of text "NodeID 3 Pressure 101325 Date 2011-09-06 Time 12:05:56" represent a meaning of "Node3's pressure was 101325KPa at 12:05:56 on 6th, September of 2011". The Component Factory can read real-time messages one at a time, and then generate dynamic components for each sensor node according to the data structures of each message.

These generated data architectures can be registered and put in "Component Templates" for reuse. The Component Factory generates sensor components in three ways:

- If the type of received message is predefined in Component Templates, the data will be sent to a related template and then some of its parameters will be replaced by the captured data (i.e. sensor ID and sensor name). A new sensor component will then be generated from the modified template.
- If the type of received message does not exist in Component Templates, a new type of template will be created and registered in the Component Templates. The new template then will be used to generate sensor components as well as the old templates.
- If the sensor node in the received message has already been activated and wired to Component Factory, the Component Factory just sends the required data to related component on event bus.

The generation of components also results in registration with the Reconfiguration Engine and Event Bus of the LooCI runtime. Once a component is registered, the Reconfiguration Engine issues an ID to the component. The new component is inactive when it is created, and can be activated. As the Component Factory is implemented above the level of the middleware, the middleware layer doesn't need to be modified and inherently supports the connections to other platforms, such as SunSPOT, Contiki and OSGi.

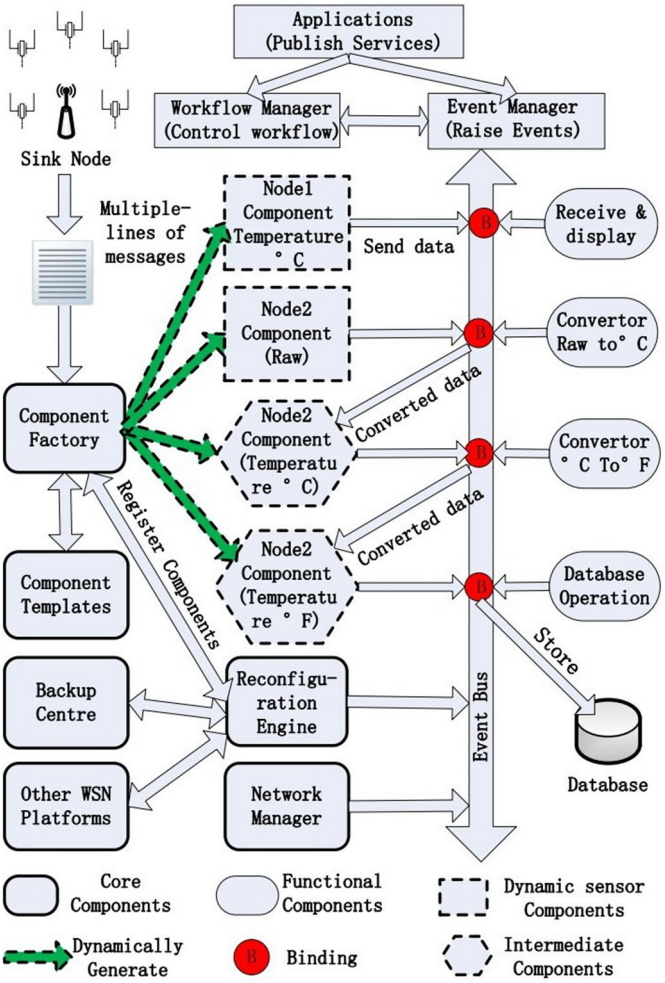

Fig. 2 The proposed architecture

Dynamic Proxy Components: In Fig.2, The dynamic proxy components are generated by the Component Factory and include Node Components and Intermediate components. They are both dynamically generated and terminated. WSN nodes have a high chance of failure, which could be hardware error or low battery. Subsequently, each component is designed with an expiration date. If the Component Factory has not received data from a registered sensor above a predefined period of time, the status of the sensor will be switched to "stop working". Then the Component Factory will send a failure report to the event bus. Consequently, the dynamic node component will be terminated and unregistered. The Intermediate Components only exist in an intermediate state, and then will be replaced by a new component. The generated Node Components can run in the proxy, and then transmit to the cloud. Running components in the proxy is beneficial. In some instances, local proxy based processing will reduce the expense of large uploads to the cloud.

Workflow Manager: Workflow management can also be achieved by rewiring the relationship between multiple components. In the proposed architecture, the Workflow Manager can coordinate and control the workflow of components over the event bus. For example, if the raw temperature data in Fig. 2's Node2_Component_(Raw) needs to be converted to Temperature °F, the raw data can be firstly converted to Temperature °C and then further be converted to Temperature °F. An intermediate component Temperature °C exits in the middle status, benefiting workflow management.

Reconfiguration Engine: The heart of the proposed infrastructure that can register, control and introspect all components [7]. Each component registers with a per-node reconfiguration engine, and these components can be started, stopped and resumed by invoking a set of standard methods in a generic component base-class. In order to easily implement run-time reconfiguration for all components, all the running components store their reference of reconfiguration interface in the Reconfiguration Engine. The Reconfiguration Engine can inspect information at the node, component and binding level over the event bus [7].

Backup Centre: As wireless sensors have a high chance of failure and new replacement, the Backup Centre is a recovery mechanism with fault-tolerance. If a sensor node fails and is replaced, the Backup centre can recover the previous reference to a replaced node.

Event: Each LooCI event has a globally unique identifier which can identify the event in a global descriptive hierarchy [7]. On the event bus, the Interfaces are responsible for publishing events and the Receptacles subscribe the events. In this publish-subscribe architecture, distributed Events can travel over the whole event bus along wirings, which can be seen as the logic connections between two components [7].

Event Manager: The Event Manager logically forms a distributed 'Event Bus' to connect LooCI components. A per-node instance of the LooCI Event Manager publishes and subscribes events based on topic and event type. The event system is a spanning tree structure to allow developers to easily discover third-party components [7]. For instance, "Event.Temperature.conventor.raw-_to_C" represents a tree architecture which contains a component "raw_to_C".

8) Network Manager: Network Manager is used to discover the address of central node. In this architecture, all lightweight sensors in a WSN share the same IP address with the central node (i.e. each WSN uses a unique IP address).

9) Wiring and Unwiring of components: A combination of component ID, event ID and network address are used to bind or unbind a component to a specified event [7].

IV. EVALUATION

To evaluate the feasibility of the proposed architecture in Section III, the following experiments were performed.

A. Experiment Setup

The proposed architecture is a generic architecture that supports the integration of any WSN with the Cloud. A Zigbee Xbee and an Arduino UNO were combined together to simulate a lightweight WSN with eight temperature sensors. The Arduino is an open-source microcontroller that has an Atmel AVR processor and on-board I/O, is programmed in C and used in multidisciplinary projects. ZigBee is a low-cost, low-power, lightweight wireless mesh network standard designed for personal wireless networks based on IEEE 802.15.4. "ZigBee + Arduino" is a platform that can support projects such as monitoring of temperature, light, pressure, and sound [27]. Messages from sensors were collected on the Arduino board, and sent via ZigBee to the sink node. Each message includes three elements: component ID, temperature

(°C) and sending time. Temperature (°C) can be further converted to temperature (°F) in the Cloud.

A ZigBee USB explorer was used to collect data from the WSN, and it also acts as the central node to bridge the WSN with local proxy. Ubuntu 10.10 was installed in two virtual machines on a local network. One machine was a proxy to receive data from the central node of the WSN and connect to the Cloud. Openjdk 1.6 and Java_Communications_3.0 was installed as the API to read data from the USB port of proxy. LooCI-OSGi v1.0 was installed on the two machines, which are used for sending and receiving data respectively.

The middleware (LooCI) was taken off-the-shelf to support the proposed architecture, and was not implemented specifically for it. In this experiment, two static LooCI components were created in Java. The first component has two functions: 1) Reading data from the central node of a ZigBee network; 2) Sending the data to the second component. The second component is used for receiving and displaying data. The sending delay was configured as 200 milliseconds on the Arduino board (i.e. sending data 5 times per second). The evaluation results illustrate that this approach can be used to successfully exchange data using Zigbee.

B. Individual Connectivity for Lightweight Sensors

Lightweight WSNs aggregate data on a central node. The advantage of lightweight WSNs is reduced energy consumption due to lower complexity [28]. Unfortunately, if individual sensors lose connectivity, it becomes difficult to manage each sensor's data. The purpose of this experiment is to evaluate the feasibility of establishing individual connectivity for each sensor in the proposed architecture.

In this experiment, the Component Factory was designed to generate sensor components dynamically. A Perl script was used to parse the real-time messages from the ZigBee network. Predefined component templates were stored in Java files. If the message is from a known type of sensor, the Component Factory will replace some parameters of the template with captured data from messages, such as the component ID and component name. Then the modified template will be used to create new components, such as Class files and Jar files. The evaluation shows these generated sensor components can be wired to the Component Factory and activated automatically. The dynamic components of each node were generated based on their data structures. The evaluation results shows that with registered component IDs and IP addresses, the dynamically generated components can also publish and subscribe to services on the event bus.

C. Overhead Testing

To evaluate the overhead of the proposed architecture, this experiment tested the latency between two dynamic components in two different virtual machines. The latency of socket communication was measured as a benchmark for comparison and repeated 20 times to get a precise result. Fig. 3 compares the latency for dynamic components, sockets and the average of dynamic components.

The latency of socket communication was 1ms. The latency between two dynamic components fluctuated between 1ms and 3ms. The average value was 1.6ms. It is observed that on average, the proposed architecture only adds 0.6ms of latency between two dynamic components. The additional 0.6 ms of latency can be explained by the need to first transfer the data to the core components [7].

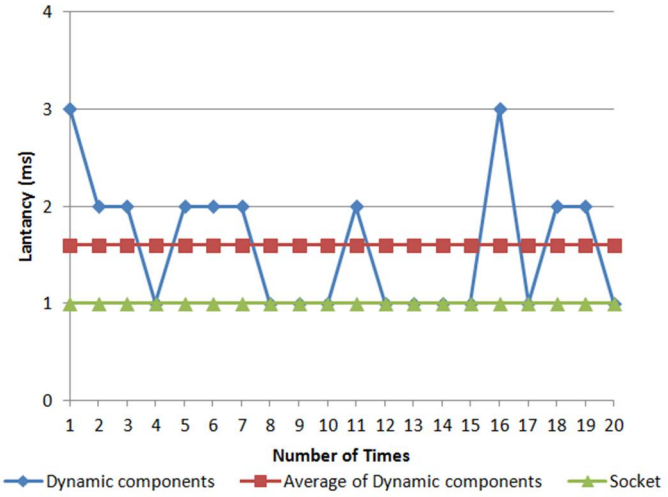

Fig.3 Latency comparison

D. Memory Footprint

A minimal memory footprint is critical to wireless sensors in embedded systems. Compared with the approach of running middleware on sensors, in the proposed architecture, it is running on the local proxy or in the Cloud. It can effectively avoid the constraints of memory and energy on sensors. However, the memory of the local proxy is still limited. The purpose of this experiment is to measure the runtime memories of the approach and the dynamically generated components. The runtime memory of the middleware and the JVM was observed as 18088 KB. This was tested using the 'ps' command in Linux. Eight dynamic components were loaded one after another. The evaluation result shows that upon activation of additional dynamic components, the runtime size of each component decreased. The first component used 196 KB to initialize loading components. After that, there was a slow trend of increasing from the second component (82 KB) and stabilising at the last component (28 KB).

The disparity of components' runtime size can be explained by the process of generating components from the same template, and resultant memory allocation of the JVM. Multiple components can share the same process in the memory. It can be inferred that each component size is no more than 28 KB when the number of dynamic component is greater than 8. This is acceptable for a local proxy. A computer with 8 GB of RAM can provide enough memory for running 20000 dynamic components. This assumption is based on the OS using 2 GB of memory. Fig. 4 illustrates the variation of runtime memory for dynamic components.

Compared with the dynamic components in the local proxy, previous experiments show they consume 44.5 KB of total memory for each sensor [7]. The component itself only has a footprint of 20.8 KB on one sensor, and the rest of 23.4 KB is used for networking. It can be observed that, on average, dynamic components consume less memory than running middleware on sensors. Most importantly, as the proposal is completely running on the local proxy, it can efficiently reduce the energy consumption for resource-constrained sensors.

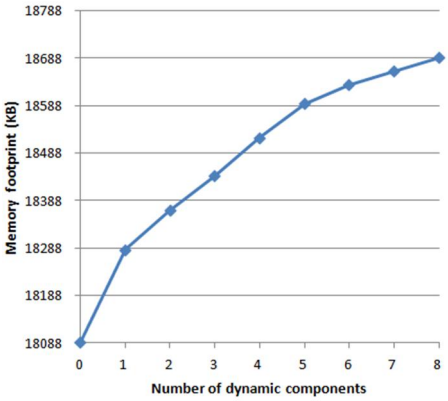

Fig. 4 Runtime memory of dynamic components

E. A Limitation and Optional Improvement

The evaluation revealed a problem when dynamically generating components. If a large number of sensor nodes were found by the Component Factory the process of generating components and compiling components may consume a lot of CPU time. As a result, the processing speed of the local proxy will be reduced. In this case, the performance can be improved by moving the process of generating components to the private Cloud.

V. CONCLUSIONS

This paper proposed a technique for the integration of lightweight WSNs and Cloud Computing using dynamic proxies. It aggregates the features of message-oriented and component-based approaches. Each dynamic sensor component has its own individual connectivity and global interoperability. As the middleware is located in the proxy, less energy is consumed. The evaluation demonstrated that the architecture proposed is suited for resource-constrained environments. The evaluation of this approach illustrates that only 0.6 ms of latency is added and the memory usage is minimised.

REFERENCES

[1] E. Deelman, G. Singh, M. Livny, B. Berriman and J. Good, "The cost of doing science on thecloud: the Montage example", *in Proceedings of theACM/IEEE Conference on High PerformanceComputing* . Austin, Texas, USA. November 15-21, 2008.

[2] A. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, G. Lee, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", California: EECS Department, Tech. Rep. UCB/EECS-2009-28, 2009.

[3] L. Liu, H. Wang, X. Liu, X. Jin, W. Bohe and Y. Chen, "GreenCloud: a new architecture for green data center", *ICAC-INDST '09 Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session,* New York, USA, 2009.

[4] K. Stanoevska-Slabeva and T.W. Ristol. *Grid and Cloud Computing: A Business Perspective on Technology and Applications*, pp.160-164, 2009.

[5] K. Lee, D. Murray, D. Hughes and W. Joosen, "Extending sensor networks into the Cloud using Amazon Web Services", In IEEE International Conference *Networked Embedded Systems for Enterprise Applications,* Suzhou, China, 2010.

[6] K. Lee and D. Hughes, " System architecture directions for tangible Cloud Computing",*International Workshop on Information Security and Applications,* Qinhuangdao, China, October 22-25, 2010.

[7] D. Hughes, K. Thoelen, W. Horré, N. Matthys, S. Michiels, C. Huygen and W. Joosen. "LooCI: A Loosely-coupled Component Infrastructure for Networked Embedded Systems", *The 7th International Conference on Advances in Mobile Computing & Multimedia*, 2009.

[8] W. Dargie, and C. Poellabauer, *Fundamentals of wireless sensor networks:theory and practice*, pp.168–183, 191–192, 2010.

[9] J. Lloret, M. Garcia, D. Bri and S. Sendra, "A Wireless Sensor Network Deployment for Rural and Forest Fire Detection and Verification", *Integrated Management Coastal Research Institute*, Spain, 2009.

[10] J. A. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru and A. Wood, "Wireless Sensor Networks for In-Home Healthcare: Potential and Challenges", *Workshop on High Confidence Medical Devices Software and Systems (HCMDSS),* 2005.

[11] W. Tsujita, A. Yoshino, H. Ishida and T. Moriizumi, "Gas sensor network for air-pollution monitoring", *Sensors andActuators B: Chemical,* vol.110, no.2, pp.304 – 311, 2005.

[12] F. N. Eduardo, A. F. L. Antonio, C. F. Alejandro, "Information fusion for wireless sensor networks:Methods, models, and classifications", *ACM Computing Surveys* , vol.39, no.3, 2007.

[13] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud Computing: Distributed Internet Computing for IT and Scientific Research", *Internet Computing, IEEE,* vol.13, no.5, 2009.

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov, "Eucalyptus : A technical report on an elastic utility computing architecture linking your programs to useful systems", *UCSB Technical Report,* 2008.

[15] Amazon. (2012)Amazon Web Services. http://aws.amazon.com

[16] P. Smith, D. Hughes, K.J. Beven, P. Cross, W. Tych, G. Coulson and G. Blair. "Towards the provision of site specific flood warnings using wireless sensor networks" *In Meteorological Applications, Special Issue: Flood Forecasting and Warning*, vol.16, no.1, pp.57–64, 2009.

[17] T. Amirhosein, R. Rouvoy and F. Eliassen, "A Component-based Approach for Service Distribution in Sensor Networks" *in Proceedings of the 5th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, 2010.

[18] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. "System Architecture Directions for Networked Sensors," *SIGPLAN Not.,*vol.35, no.11, pp.93–104, 2000.

[19] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," *In LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks,* Tampa, FL, USA, pp.455–462, 2004.

[20] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks," *The 6th ACM Conference on Embedded Network Sensor Systems,* Raleigh, NC, USA, pp.253–266, 2008.

[21] T. Luckenbach, P. Gober, K. Kotsopoulos, Andreas Kim, and S. Arbanowski, "TinyREST: a Protocol for Integrating Sensor Networks into theInternet*," In REALWSN'05: Proceedings of the Workshop on Real-World WSNs,* Stockholm, Sweden, 2005.

[22] W. Colitti, K. Steenhaut and N. D. Caro, "Integrating Wireless Sensor Networks with the Web," *In IP+SN*, 2011.

[23] M. Marin-Perianu, "Decentralized Enterprise Systems: a Multi-Platform Wireless Sensor Network Approach", Wireless Communications, IEEE,vol.14, no.6, pp.57–66, 2007.

[24] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. P. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis. "The runes middlewarefornetworked embedded systems and its application in a disaster management scenario," *The 5th IEEEInternational Conference on Computing and Communications.*pp.69–78, 2007.

[25] G. Coulson, G. Blair, P. Grace, A. Joolia, K. Lee, J. Ueyama and T. Sivaharan, "A Generic component model for building systems software", ACM Trans. Comput.Syst., vol.26, no.1 pp.1–42, 2008.

[26] Daniel Nurmi et al, The Eucalyptus Open-Source Cloud-ComputingSystem, 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[27] R. Faludi, "Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing", Sebastopol, CA: O'Reilly Media, 2010.

[28] G. Drew, "ZigBee wireless networks," Burlington, USA: Newnes, 2008, pp. 3-28.