Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 867693, 10 pages
http://dx.doi.org/10.1155/2013/867693

Research Article

Introducing WebSocket-Based Real-Time Monitoring System for Remote Intelligent Buildings

Kun Ma and Runyuan Sun

Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250022, China

Correspondence should be addressed to Kun Ma; ise_mak@ujn.edu.cn

Received 11 July 2013; Revised 30 October 2013; Accepted 13 November 2013

Academic Editor: Joo-Ho Lee

Today, wireless sensor networks (WSNs) in electronic engineering are used in the monitoring of remote intelligent buildings, and the need for emerging Web 3.0 is becoming more and more in every aspect of electronic engineering. However, the key challenges of monitoring are the monitoring approaches and storage models of huge historical monitoring data. To address these limitations, we attempt to design a WebSocket-based real-time monitoring system for remote intelligent buildings. On one hand, we utilize the latest HTML5 WebSocket, Canvas and Chart technologies to monitor the sensor data collected in WSNs in the Web browser. The proposed monitoring system supports the latest HTML5 browsers and legacy browsers without native WebSocket capability transparently. On the other hand, we propose a storage model with lifecycle to optimize the NoSQL data warehouse. Finally, we have made the monitoring and storage experiments to illustrate the superiority of our approach. The monitoring experimental results show that the average latency time of our WebSocket monitoring is generally lower than polling, FlashSocket, and Socket solution, and the storage experimental results show that our storage model has low redundancy rate, storage space, and latency.

1. Introduction

Wireless sensor networks (WSNs) in electronic engineering are used in many industrial applications, such as industrial process, and environmental and health monitoring [1]. With the development of green computing and Internet of Things, more and more people have contributed an increased demand to reduce the energy consumption of buildings [2]. This implies the necessity of a conscious way of thinking and actions regarding efforts to monitor smart objects in remote intelligent buildings [3]. However, there are some challenges of current monitoring systems for remote intelligent buildings in two aspects. The first aspect is the monitoring approach. The intelligent buildings need a real-time monitoring approach with fast loading and low latency [4]. Moreover, high concurrency and low consumption play an important role in the monitoring because sensor node or server has limited energy and computational resources. The second aspect is the storage of the historical monitoring data. In the era of big data, the historical monitoring sensor data is so huge that the relational database will encounter the bottleneck problems in the track of historical data. In addition, there are many redundant data in the historical data warehouse.

WSNs used in intelligent building management systems consist of different types of sensor nodes measuring parameters such as temperature, humidity, light, and asphyxiating smoke. In addition, the systems may include actuators, gateways, servers, and communication and application software on different levels as well as different home appliances [5]. A large amount of research has been conducted focusing on different aspects of WSN for monitoring. There are some deficiencies of current approaches, such as the delay and concurrence of the monitoring. Some emerging techniques, such as HTML5 WebSocket [6], can assist this monitoring.

Future monitoring system will collect a large amount of real-time monitoring data. In the context of monitoring of remote intelligent buildings, eventual consistency is acceptable [7]. The features of the track of historical monitoring data determine that the monitoring system exhibits a high read

to write ratio. Therefore, we can utilize the emerging NoSQL [8] instead of Relational Database Management Systems (RDBMS).

Our motivation of this paper is trying to address these two limitations to design a WebSocket-based real-time monitoring system for remote intelligent buildings. We extend the Internet of Things paradigms to support more scalable and interoperable and monitor the sensors for intelligent buildings in real-time and bidirectional manner. The contributions of this study are divided into two aspects.

The first aspect is our WebSocket-based monitoring approach. First, this WebSocket-based monitoring approach is pervasive in other high-frequency application scenario. Second, the WebSocket-based monitoring approach is advantageous because it provides browser independence with low latency and low energy. Third, monitoring data in the transmission is updated in the browser in the high frequency, without the use of any browser plugins. Our solution also provides WebSocket emulation to support those popular legacy browsers.

The second aspect is the storage model of historical monitoring data. First, we propose the storage model with lifecycle to reduce redundant data and storage space. Second, we utilize the NoSQL data warehouse to provide the high-performance tracking of the historical monitoring sensor data.

The rest of the paper is organized as follows. Section 2 discusses the related work, and Section 3 introduces the architecture of the proposed WebSocket-based real-time monitoring system. Section 4 gives the experimental evaluation of our WebSocket-based solution. Brief conclusions are outlined in the last section.

2. Related Work

Before we introduce our WebSocket-based real-time monitoring system for remote intelligent buildings, we discuss the current networking technology of intelligent building, the classical monitoring methods, and the storage model of monitoring data.

2.1. Networking of Intelligent Building. Many efforts are currently going towards networking smart things of intelligent building (e.g., Radio Frequency Identification (RFID) [9], wireless sensor networks, and embedded devices) on a larger scale. Compared with the WSNs themselves, Internet of Things has mainly focused on establishing connectivity in a variety of challenging and constrained networking environments, and the next logical objective is to build on top of network connectivity by focusing on the application layer.

There are some key issues of the networking of intelligent buildings. First, how to utilize the emerging Web technology [10] to assist the monitoring of smart objects in remote intelligent buildings is challenging. The new sensor Web is also associated with a sensing system which heavily utilizes the World Wide Web (WWW) to define a suite of interactive interfaces and communication protocols abstracting from well-accepted and understood standards (such as Ajax, REST, and HTML5), especially well suited for environmental monitoring.

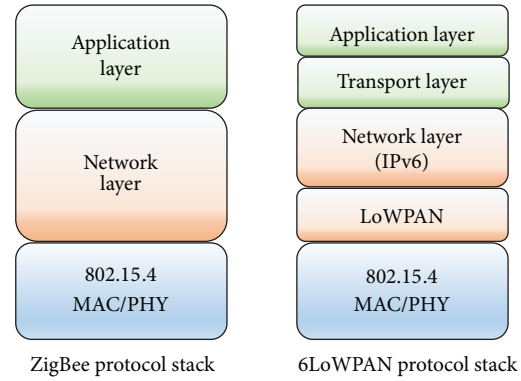

Figure 1: ZigBee/6LoWPAN Protocol Stack.

Second, how to support different heterogeneous network (such as ZigBee [11] and 6LoWPAN [12]) is challenging. Today, WSN system technologies suffer from nonflexibility and proprietary solutions. It is necessary to come to an understanding and bridging the gap of different protocol stacks. There are two popular wireless communication protocols adopted by IEEE 1451.5 standard [13]: ZigBee and 6LoWPAN. However, there are some deficiencies of these two protocol stacks. Although ZigBee has the advantage of power saving and low cost, it is rather immature compared with Internet Protocol (IP) which has been developed over the past 40 years. The main disadvantages of ZigBee include short range, low complexity, and low data speed. 6LoWPAN, an alternative to ZigBee, is an acronym of IPv6 over Low power Wireless Personal Area Networks. As shown in Figure 1, at the physical layer and the data link layer, it uses the same IEEE 802.15.4 protocol as ZigBee. 6LoWPAN has defined encapsulation and header compression mechanisms that allow IPv6 packets to be transferred over WSN. Even if there are a large number of devices deployed in a WSN, each device can also be assigned with a unique IP address. This feature makes it easy to support end-to-end communication. However, this protocol stack does not define a specification for the layers above IP. Both ZigBee and 6LoWPAN networks cannot communicate with each other without a sensor gateway. In our paper, we seek to build on the findings of the earlier work [14, 15] in this area to support the development of a WSN that goes beyond current approaches. The networking approach is the integration of wireless ethernet and WSNs (such as ZigBee and 6LoWPAN).

2.2. Classical Monitoring Methods. The most common monitoring is Web monitoring, which is suited for remote intelligent buildings. Currently, there are several categories of methods of monitoring. The first monitoring approach is HTTP long polling (hereinafter referred to as HTTP polling). HTTP polling is a variation of the traditional polling technique, but it allows emulating a push mechanism under circumstances where a real push is not possible, such as sites with security policies that require rejection of incoming

HTTP requests. However, polling has many disadvantages, including unnecessary requests and the response can be delayed as much as the polling interval [16].

The second monitoring approach is primitive Socket in ActiveX. The Socket control components provide easy-to-use Socket ActiveX control to develop applications that communicate using either TCP or UDP protocols. These have the advantage of working identically across the browsers with the appropriate plugin installed and need not rely on HTTP connections, but with the disadvantage of requiring the plugin to be installed. Besides, ActiveX runs from Windows Internet Explorer only, and embedding application in the browser would affect browser-side performance.

The third monitoring approach is FlashSocket. FlashSocket relays make use of the XMLSocket object in a single-pixel Adobe Flash movie [17]. The advantage of this approach is that it appreciates the natural read-write asymmetry that is typical of many web applications, and as a consequence, it offers high efficiency. Since it does not accept data on outgoing sockets, the relay server does not need to poll outgoing TCP connections at all, making it possible to hold open tens of thousands of concurrent connections. In this model, the limit to scale is Flash itself. Web applications that feature Flash features tend to take longer to load than those that do not.

The last monitoring approach is WebSocket. WebSocket is a web technology in the HTML5 specification providing full-duplex communications channels over a single TCP connection [18]. In this way, the WebSocket protocol makes possible more interaction between a browser and a web server. However, the legacy browsers lack the support of the emerging WebSocket technology.

There are many drawbacks of the above approaches. We will deeply analyze the monitoring approaches in Section 4. Our investigation explored the WebSocket compatible server integrated with WSNs to support every browser. Even if WebSocket is selected as the transport, WebSocket compatible server does more than WebSocket itself, including heartbeats, timeouts, and disconnection features. This networking has led to a greater awareness of the conditions of buildings. The main benefits of this are economies of scale gained from real-time monitoring, fast loading, low latency, and low cost.

2.3. Storage of Monitoring Data. Existing WSN database abstractions use the SQL query interface and RDBMS techniques. Currently, there are several traditional RDBMS database abstractions available for WSN, such as TinyDB [19] and TikiriDB [20]. Recently, several database experts argued that RDBMS will have the I/O bottleneck issues encountered by handling huge amounts of data, especially for the query operation [8]. To address this limitation, several systems have already emerged to propose an alternative NoSQL [21] database as the storage of big data. With regards to NoSQL databases, handling massive amounts of data is much easier and faster, especially for the query processing. The advantages which NoSQL databases offer over the query processing are reflected when it exhibits a high read to write ratio. Currently, there are several NoSQL databases available, such as HBase [22] and BigTable [23].

Although there are some best solutions of NoSQL, few publications have mentioned the data warehouse technologies about NoSQL. For the monitoring system, the basic business is the track of huge historical data. Therefore, we utilize MongoDB as the storage of historical monitoring data in WSNs. The motivations for this approach include high availability and query efficiency.

Current storage models of RDBMS data warehouse provide lessons on the NoSQL stores. The first historical storage model is called record with timestamp. In this solution, the historical data are saved in another separate table with the timestamp every day. However, this solution has too many disadvantages. The first one is that the data warehouse is huge enough so that the storage becomes a disaster over a long period. The second one is that it will consume too many excessive storage spaces due to the redundant data.

The second historical storage model is called model with versions. In this solution, all the historical data are tracked by creating multiple records for a given natural key in the dimensional tables with separate surrogate keys and/or different version numbers. This approach is suitable in the frequent updates scenario. However, the historical monitoring data are new data that can never be altered.

3. Architecture

3.1. Overview of System Architecture. The core idea of the proposed system is to enable wireless Web-based interactions with sensor gateways and WebSocket and Web server for monitoring the remote intelligent buildings. Figure 2 shows the proposed WebSocket-based monitoring system for WSNs of remote intelligent buildings in WSNs. This WebSocket-based monitoring system is composed of three points of view in order to reduce the complexity: WSNs of remote intelligent buildings, WebSocket-based real-time monitoring, and storage of the monitoring sensor data.

3.2. WSN of Intelligent Buildings

3.2.1. Overview. We seek to integrate WSNs with ethernet communications to support data monitoring. This system is piloted by the deployment of WSNs in a remote intelligent building using the communication standards ZigBee and 6LoWPAN. These standards are embedded into a large number of chips for building automation. In our solution, we adopt these two protocol stacks to support more sensor devices. We provide two types of sensor networks (ZigBee and 6LoWPAN) to collect the monitoring data. Although the two sensor networks do not have the interoperability to communicate with one another, they both transfer the monitoring data to the nearby sensor gateway, which makes the system function as a whole.

The WSNs consist of commercially available low cost nodes with small size and low power, which are integrated with sensing and communication capabilities. These elements

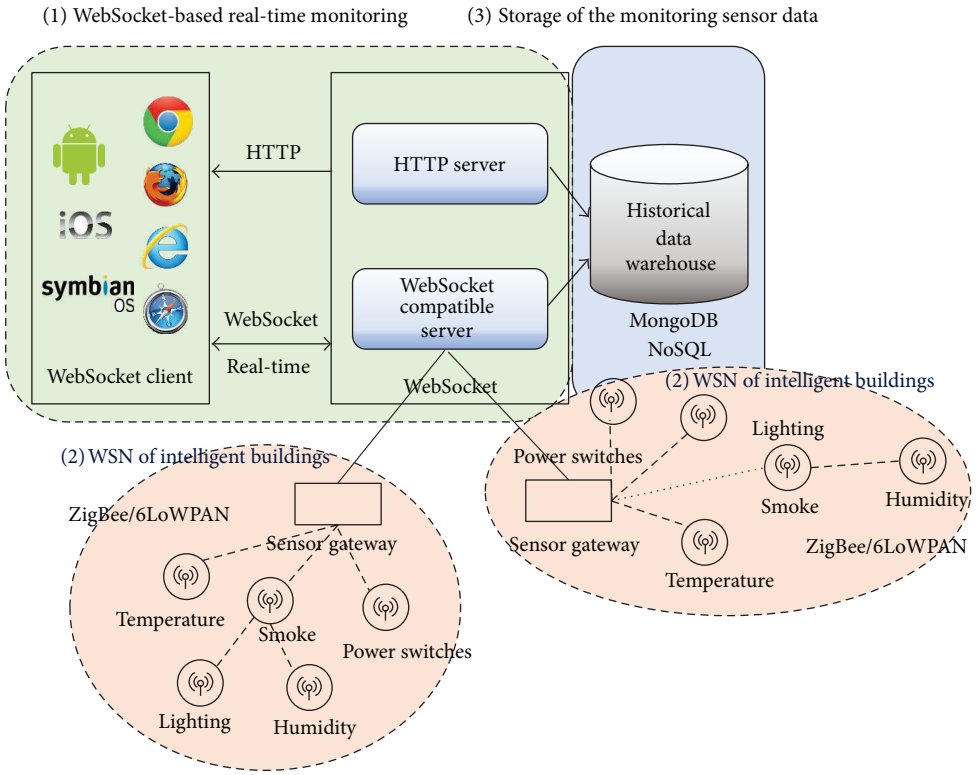

Figure 2: Monitoring system architecture.

in our proposed system include nodes with integrated sensors measuring temperature, light, humidity, and combined power switches. The latter power switches are used to control remote power switches to do more things. Historical monitoring data collected from the sensor nodes are stored in the distributed NoSQL data warehouse. The details of historical monitoring data are discussed in Section 3.4.

3.2.2. Monitoring Process. The whole process is divided into three steps. The first step is that sensor nodes collect the sensor data from devices and transfer the monitoring data to the sensor gateway using the communication standards ZigBee and 6LoWPAN. The second step is that the sensor gateway transfers the UDP monitoring data to the server using the WebSocket protocol. We also embed the WebSocket protocol into the chip of sensor gateway. The last step is that the WebSocket compatible server stores the historical monitoring data in the distributed NoSQL databases and pushes the monitoring data to the WebSocket client.

In order to actually see what is happening when controlling the switches from a remote terminal, a network camera has been connected to the ethernet. This network camera displays in the browser the live monitoring in the building.

3.2.3. ZigBee Wireless Sensor Network. As depicted in Figure 3, ZigBee Wireless Sensor Network is composed of devices from different vendors. The sensor gateway functions both as a server transferring the monitoring data to the WebSocket server and a connection point between ZigBee

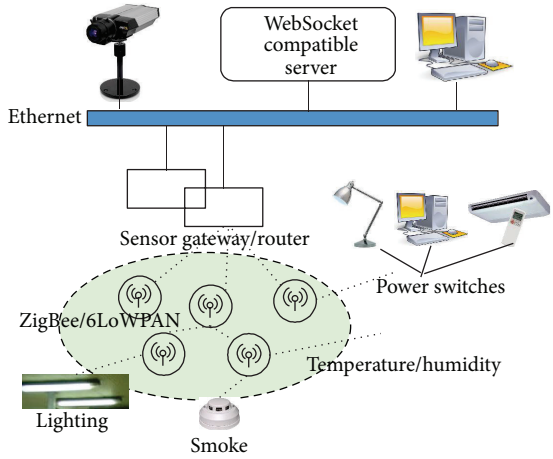

Figure 3: ZigBee/6LoWPAN wireless sensor network.

WSN and ethernet. It has the coordinating role in the ZigBee network. The script running on the gateway enables the transfer of monitoring data in the ethernet.

We deploy two batteries-driven wireless sensor nodes with internal sensor chips to measure the temperature and humidity in each room. These nodes are able to form mesh networks and can operate within an indoor range of about 20 meters. They function as end devices in the ZigBee WSN. Another sensor node is equipped with a power socket and has a gateway function in the network.

We deploy a remotely controlled ZigBee device with each switch to power the external circuit, such as desk lamps, computers, and air conditioning. They are also used as meters for measuring the load, voltage, current, and power of the attached electrical equipment.

3.2.4. 6LoWPAN Wireless Sensor Network. The 6LoWPAN WSN is formed by one WebSocket compatible server, two sensor routers, and five battery driven sensor nodes. The architecture of 6LoWPAN WSN is shown in Figure 3. The WebSocket compatible server is connected to the ethernet and provides a Web user interface, which stores monitoring sensor data. Through this WebSocket and Web user interface, the user is able to see the real-time monitoring graph and manages to control WSNs of intelligent buildings in the Web browser.

Access to the network from the server goes through the two parallel working sensor routers. These routers manage the routing between the sensor nodes and the IP ethernet. Deploying at least two routers in the same network of sensor nodes will scale the throughput of the network. Each router is able to take over the other router's tasks in cases of a nonoperational router, which increases the redundancy and reliability of the network. To utilize this range of usage of the 6LoWPAN sensors, they are connected to a device functioning as a relay controlling the circuits in the building. From the Web interface in the browser, the device was remotely controlled with the circuit switches.

3.3. WebSocket-Based Real-Time Monitoring

3.3.1. Web User Interface. Besides the monitoring based on WebSocket, the proposed system uses HTML5 Canvas API to represent data from WSNs in electronic engineering easily and efficiently. In the proposed HTML5-based monitoring system, sensor gateway can be interacted with and controlled through WebSocket API, and their status and functions can be monitored on Canvas. We can perform the deployment of sensor nodes and the configuration management of WSNs in electronic engineering.

WebSocket-based Web clients are JavaScript applications that run in a Web browser, which communicate with the Web and WebSocket server. Canvas API provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, or other visual images on the fly. We employ open-source Chart.js [24] to represent the data gathered from sensors. Chart.js is an HTML5 JavaScript Charts library, which supports over 20 different types of charts. Using the HTML5 Canvas tag, Chart.js creates these charts in the Web browser, meaning quicker pages and less Web server load. Figure 4 shows the Web user interface of temperature movements (°C) collected from one sensor of the ZigBee WSN. This Web user interface is implemented with HTML5 Canvas. This Web graph is listed with the accompanying information of the latest sensor data as well as the time of the sensor readings.

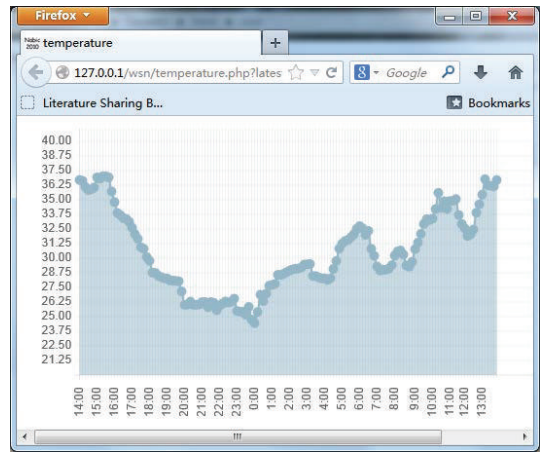

Figure 4: Temperature chart reading from ZigBee sensor nodes.

3.3.2. WebSocket Server versus WebSocket Client. WebSocket requires its own back end application to communicate with the server side. Therefore, we utilize Socket.IO [25] to develop WebSocket server. Socket.IO, plugin of Node.js, aims to easily make real-time applications possible in every browser and mobile device, blurring the differences between the different transport mechanisms, which is care-free real-time 100% in JavaScript.

On one hand, we take advantage of socket.IO [25] server API to implement a WebSocket server. While the monitoring gateway has received the sensor data, WebSocket server launches an event to notify all the online browsers. On the other hand, we take advantage of socket.IO [25] client API to intercept the message from WebSocket server in near real time. The call-back function is used to display the monitoring data.

3.3.3. WebSocket Server versus Sensor Gateway. Besides monitoring, our WebSocket-based monitoring system for remote intelligent building in WSNs can send real-time commands to the sensor gateway to control the sensors (turn on lights, turn off air condition, and change camera's sampling frequency) directly through the WebSocket API. Notifications are sent when an event occurs from sensor nodes to the WebSocket server, and then the WebSocket server sends these notifications the WebSocket clients in real time. The monitoring actions are sent promptly from the WebSocket server to gateways and/or sensor nodes to respond to the notifications. And sensor monitoring data are collected periodically and sent to store in the NoSQL databases.

3.4. Storage of Historical Monitoring Sensor Data

3.4.1. Overview. Figure 5 illustrates the overall design architecture of storage of historical monitoring data including all the main components. As displayed in Figure 5, storage architecture consists of three main components which are directly contributing to the NoSQL data warehouse. These

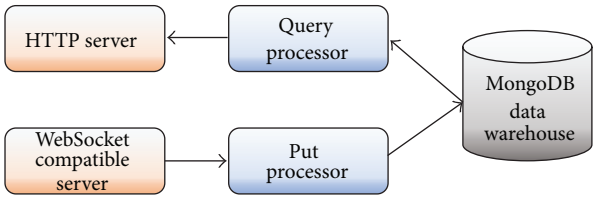

Figure 5: Storage and query processing architecture.

three main components are query processor, put processor, and MongoDB database.

Query and put processors are the read/write components of the NoSQL data warehouse. Query processor is used to track the historical monitoring data to generate reports or charts, while put processor is used to store the historical monitoring data. We adopt MongoDB as the storage of NoSQL data warehouse, because MongoDB is an open-source leading document NoSQL database. Some features of MongoDB adapt to the monitoring scene. For example, it provides fast query with full index support.

3.4.2. Storage Model with Lifecycle. In order to reduce the redundancy rate and storage space of NoSQL data warehouse, we propose a historical storage model with lifecycle. Compared with current timestamp solution introduced in Section 2.3, we use the start and end timestamp instead of the unique timestamp.

We give the definition of the lifecycle. The lifecycle tag is a 2-tuple of start and end element, which implies the lifecycle of a record in the NoSQL data warehouse. The lifecycle is denoted as $(start, end)$. The first element is the start timestamp, and the second element indicates the end timestamp.

We define the historical storage model with lifecycle in the data warehouse as a list of attributes that comprise a key, a set of regular attributes, and a lifecycle field. This definition is the metamodel of the historical storage model with lifecycle. That is to say that the record in practice is the instance of this model.

There are two characteristics of this model. First, each record has one and only one lifecycle. Second, each lifecycle belongs to a set of records. In the scenario of the collection of the historical monitoring sensor data, we utilize this storage model to compress the real-time monitoring data.

Figure 6 shows the method to get the historical snapshot over a period of time. The storage models with lifecycle that are penetrated by the two black lines formulate the full snapshot of the historical data between the date 1015 and 1017. The snapshot for some time is achieved in this way.

4. Experiment Results

The experiments were performed on an Intel(R) Core(TM) CPU I5-2300 3.0 GHz server with 8 GB RAM, a 100 M Realtek 8139 NIC in 100 M LAN. We have made two experiments to illustrate the advantages of our monitoring approach and

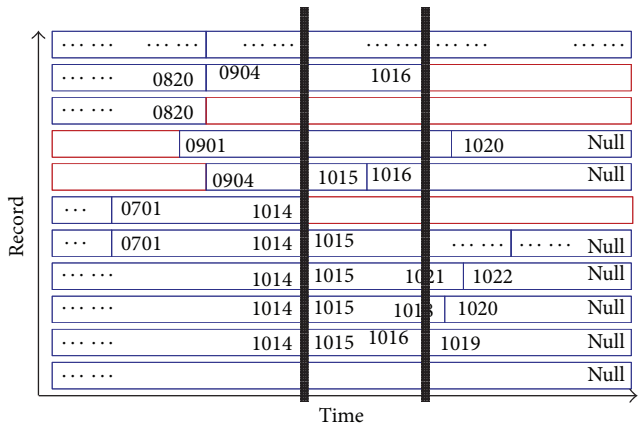

Figure 6: The method to get the historical snapshot over a period of time.

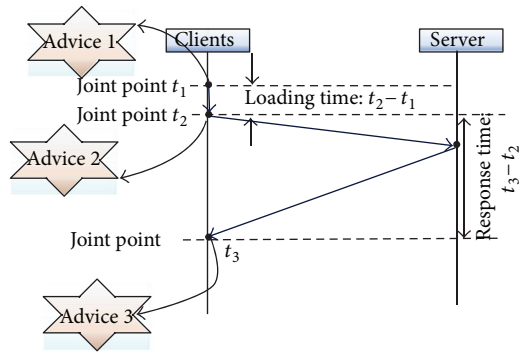

Figure 7: AOP interceptors to calculate response time.

system. The systems were configured with a Windows Server 2012 x64.

4.1. Monitoring Experiment. Our first experiment is comparing HTTP polling, Socket in ActiveX, FlashSocket, and our WebSocket monitoring approach. Our experiment are based on aspect-oriented programming (AOP) code that intercepts the methods of the client at the well defined join points to collect test data and measure the loading time and response time at important instants of time.

In order to evaluate the performance of different approaches, we are assuming that the startup and shutdown time of AOP interceptors are small enough to ignore. We just calculate the loading time and response time shown in Figure 7. The aspect code measures the timestamp by advice 1, 2, and 3. The loading time is evaluated by the time difference $t_2 - t_1$, and the response time is evaluated by the time difference $t_3 - t_2$.

We have made four experiments to illustrate the superiority of our system.

4.1.1. Fast Loading. We ran 10 tests to measure the loading time. Since the loading time is calculated by $t_2 - t_1$ and is independent of the server, we just calculate the average

Table 1: Comparison of loading time.

Solution	Loading time (ms)
HTTP polling	53
Socket in ActiveX	852
FlashSocket	796
WebSocket	121

Table 2: Standard deviation of loading time.

Solution	High speed network	Low speed network
HTTP polling	22.5740	80.4718
Socket in ActiveX	22.8496	83.1940
FlashSocket	11.083	27.9408
WebSocket	10.4430	27.4649

loading time as a reference. Table 1 shows the average loading time of different monitoring solutions. Since Socket in ActiveX and FlashSocket depend on the plugins, they have the higher loading time. Due to the similarity to the HTTP protocol, HTTP polling solution has the lowest loading time. The loading time of our WebSocket solution is the median.

4.1.2. Low Latency. We ran a total of two groups of tests for one monitoring process with different solutions. The IP of WebSocket server in the first group is in China to simulate high speed network, while the IP of WebSocket server in the second group is outside China just to simulate low speed network. We have made 5 test use cases to calculate the response time of different solutions. The sampling time is 05:00, 09:00, 13:00, 17:00, and 21:00. Figure 8 shows the results of this experiment.

When the network transport is not very satisfying, the latency of FlashSocket and WebSocket solution is similar. When the network transport goes smoothly, the latency of WebSocket solution is lower than FlashSocket solution. Since FlashSocket is embedded in the form of ActiveX or plugins in the browser, the performance of FlashSocket cannot reach the full potential, especially in the low speed network. Although the response time of WebSocket solution is close to FlashSocket solution, it should be noted that the FlashSocket solution needs to download an additional Flash object file before a connection is established. That process will cost extra time. The experimental evidence suggests that the response time of WebSocket shows a little below FlashSocket solution. For the real-time, all the tests show that the HTTP polling and Socket in ActiveX solutions cost significantly longer response time than FlashSocket and WebSocket solution.

Table 2 shows the standard deviation of the response time in Figure 8. The standard deviation of our WebScoket solution is the lowest.

4.1.3. High Concurrency. We have made the third experiment to test the availability of different monitoring approaches in different concurrency values. We use multithreads to simulate concurrent access. Figure 9 shows the average response time of different concurrencies. In the high-concurrency situation, the response time of HTTP polling and Socket in ActiveX is so high that the whole system does not work. As the concurrency counts increase, the response time widens between WebSocket and FlashSocket solution.

4.1.4. Low Consumption. We have made the fourth experiment to test the consumption of different monitoring approaches in different concurrency values. Since all the monitoring approaches are based on the browser, the CPU utilization of client is one performance criterion. We have made this experiment to test the average CPU utilization of WebSocket client accompanied by the last experiment. Figure 10 shows the average CPU utilization in different concurrency values. The average CPU utilization in WebSocket solution is almost stable. That is to say that the users do not feel the existence of extra resource consumption of WebSocket by contrast with the traditional HTTP request and response. FlashSocket and Socket in ActiveX consume a lot of resources because of the embedding plugins techniques, especially in high-concurrency situations.

4.2. Storage Experiment. We initialize the data warehouse as empty. After that, the collected historical data are stored in the data warehouse. We observe the collected experimental data of timestamp and lifecycle solutions.

4.2.1. Redundancy Rate. First, we have made the redundancy experiment of timestamp and lifecycle solutions to illustrate the superiority of our approach. Figure 11 shows the redundancy rate in 180 days. As depicted in Figure 11, the redundancy rate of the timestamp solution is down with the unchanged probability of the lifecycle solution. Our lifecycle solution is advantageous in the long run.

4.2.2. Storage Space. Next, we analyze the storage space of different solutions. We assume that the data size increases by average 5,000 documents from the sensor gateway every day. We observe the storage space of the corresponding data warehouse every day. Figure 12 shows the variation of different solutions. Scales of the data size of the corresponding data warehouse increase linearly. The fastest increasing solution is timestamp solution, since all the records need to be stored regardless of whether the documents are changed or not. However, our lifecycle solution takes remarkable superiority in the scenario of big data.

4.2.3. Query Time. In the process of the storage space experiment, we measure the query time of the historical data in the data warehouse. In order to make the comparison of different solutions, we select the historical data on the first month. We select 8 points to record the time of the same query. Figure 13 shows the query time of different solutions. The query time of the lifecycle solution is lower than the timestamp solution all the time. We can conclude that our solution is feasible in practice.

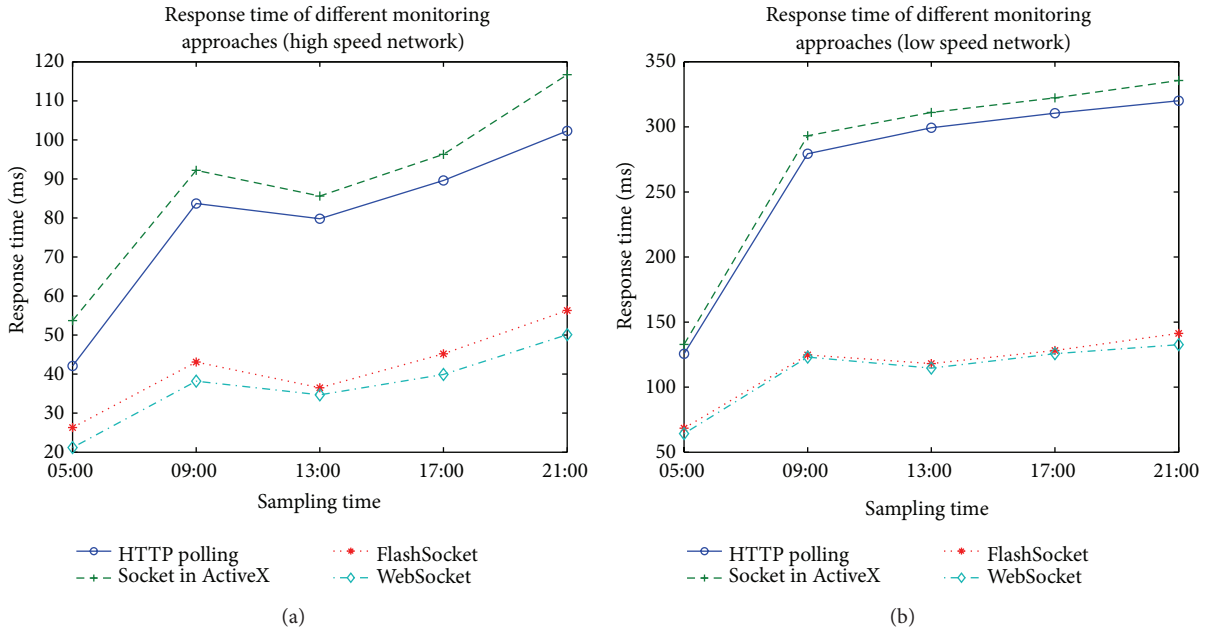

Figure 8: Response time of different monitoring approaches.

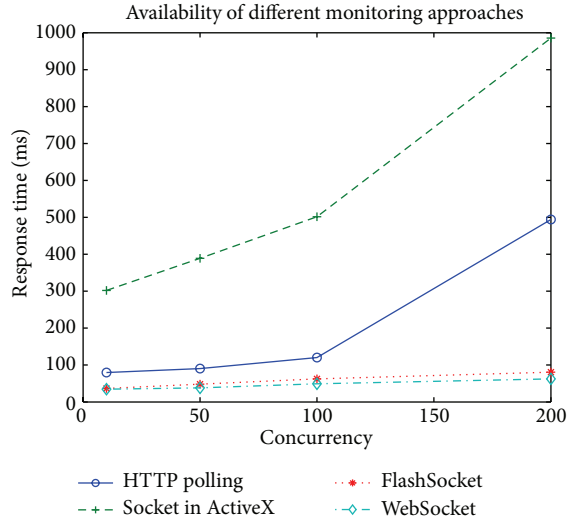

Figure 9: Concurrent test of different monitoring approaches.

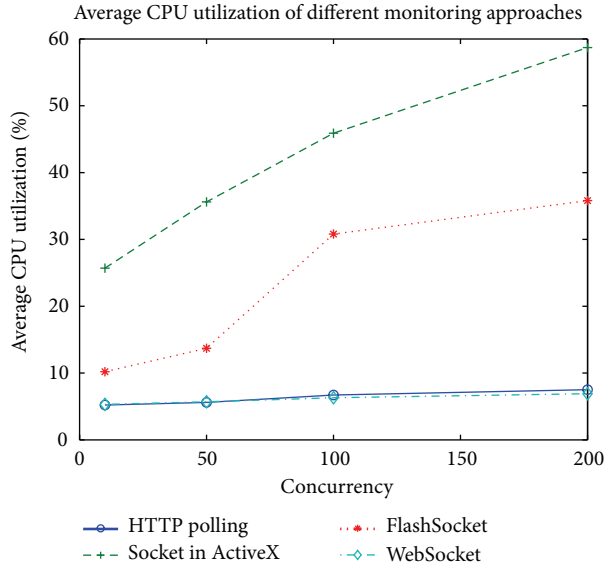

Figure 10: Average CPU utilization of different monitoring approaches.

5. Conclusions

In this paper, a WebSocket-based real-time monitoring system for remote intelligent buildings in electronic engineering has been proposed. By utilizing the WebSocket, Canvas, Chart, and NoSQL, the Web-based monitoring system can be implemented to easily control and monitor the sensor activity in real time. This demonstration illustrated that it is possible to remotely control the electrical appliances from this Web user interface of smart terminals. The open architecture of the concept allows for easy and continuous updates and unlimited expandability. This experimental work presented illustrates that a combination of available WSN and ethernet can be employed to monitor and measure real time data such as the temperature, light, humidity, and power consumption. Besides, the experimental results show the efficiency of the storage of NoSQL data warehouse, especially in a world of big data. The capabilities offered by the type of wireless sensor system presented in this paper are vast. They provide the managers and owners of buildings feedback on the energy consumption of buildings to support improved building control and inhabitant behavioral change. Improvements in the systems sensors could also be integrated into the type

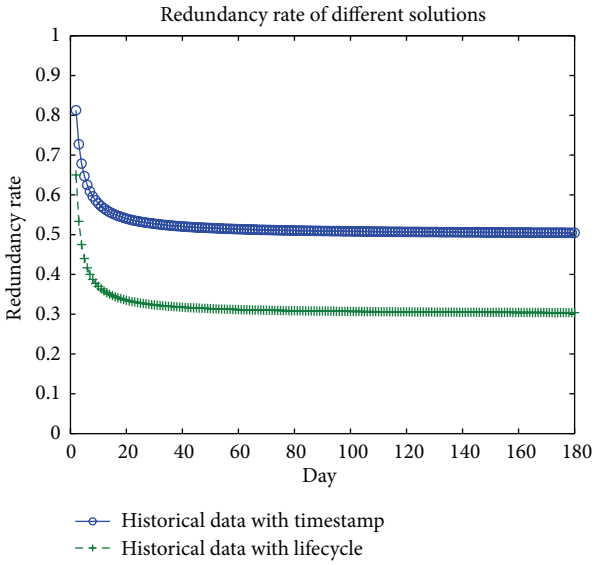

Figure 11: Redundancy rate of different storage approaches in 180 days.

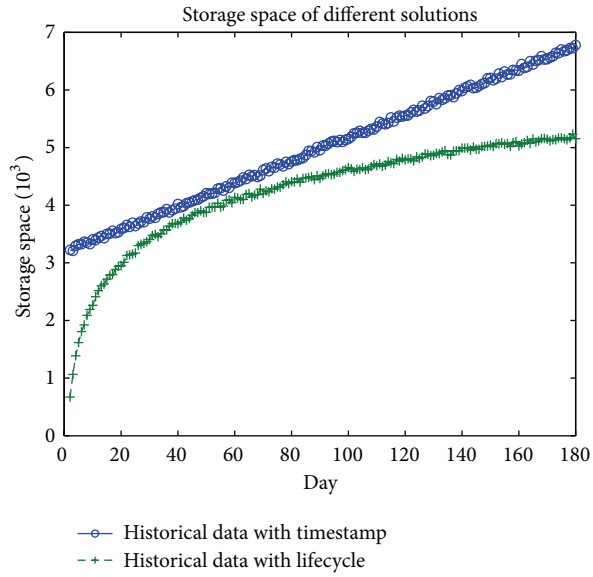

Figure 12: Storage spaces in 180 days.

of WSN discussed in this paper to supply more detailed information to building occupants.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Doctoral Fund of University of Jinan (XBS1237), the National Student Training Program of Innovation & Entrepreneurship (201210427016), the Technology development Program of Shandong Province (2011GGX10116), and the National Key Technology R&D Program (2012BAF12B07).

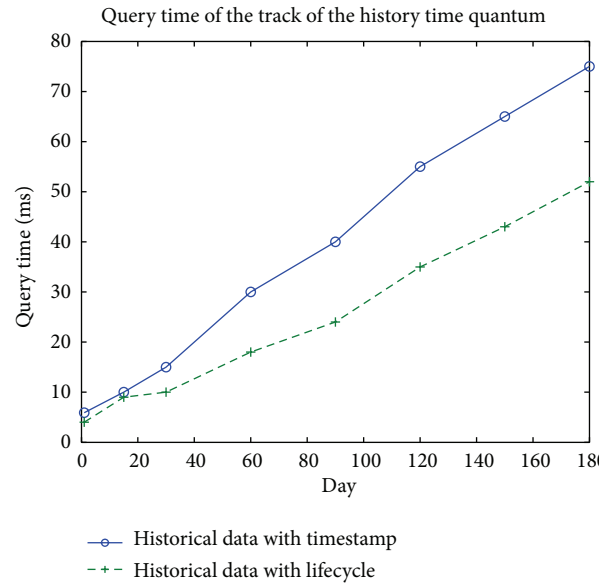

Figure 13: Query time of different storage approaches.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[2] J. K. W. Wong, H. Li, and S. W. Wang, "Intelligent building research: a review," *Automation in Construction*, vol. 14, no. 1, pp. 143–159, 2005.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[4] M. Dibley, H. Li, Y. Rezgui, and J. Miles, "Cost effective and scalable sensor network for intelligent building monitoring," *International Journal of Innovative Computing, Information and Control*, vol. 8, no. 12, pp. 8415–8433, 2012.

[5] K. Jaafar and M. K. Watfa, "Sensor networks in future smart rotating buildings," in *Proceedings of 2013 IEEE Consumer Communications and Networking Conference*, pp. 962–967, 2013.

[6] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data pimentel webSocket," *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, 2012.

[7] T. Li, Y. Liu, Y. Tian, S. Shen, and W. Mao, "A storage solution for massive IoT data based on NoSQL," in *Proceedings of 2012 IEEE International Conference on Green Computing and Communications*, pp. 50–57, 2012.

[8] R. Cattell, "Scalable SQL and NoSQL data stores," *SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2010.

[9] A. Juels, "RFID security and privacy: a research survey," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 381–394, 2006.

[10] J. Hendler, "Web 3.0 emerging," *Computer*, vol. 42, no. 1, pp. 111–113, 2009.

[11] J. Higuera, J. Polo, and M. Gasulla, "A ZigBee wireless sensor network compliant with the IEEE 1451 standard," in *Proceedings*

of the IEEE Sensors Applications Symposium (SAS '09), pp. 309–313, February 2009.

[12] V. Kumar and S. Tiwari, "Routing in IPv6 over low-power wireless personal area networks (6LoWPAN): a survey," *Journal of Computer Networks and Communications*, vol. 2012, Article ID 316839, 10 pages, 2012.

[13] E. Y. Song, K. B. Lee, S. E. Fick, and A. M. Donmez, "An IEEE 1451.5-802.11 standard-based wireless sensor network with embedded WTIM," in *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC '11)*, pp. 1201–1206, May 2011.

[14] K. Ma, R. Sun, and A. Abraham:, "Toward a lightweight framework for monitoring public clouds," in *Proceedings of the 4th International Conference on Computational Aspects of Social Networks*, pp. 361–365, 2012.

[15] K. Ma and W. Zhang, "Introducing browser-basedhigh-frequency cloud monitoring system using WebSocket Proxy," *International Journal of Grid and Utility Computing*. In press.

[16] R. Y. Sun, K. Ma, L. Z. Peng, and S. Jing, "A network utilization measurement method based on enhanced maximum traffic accumulation," *Journal of Northeastern University*, vol. 31, no. 2, pp. 381–394, 2010.

[17] D. Syme, A. Granicz, and A. Cisternino, "Building mobile web applications," in *Expert F# 3.0*, pp. 391–426, Springer, 2012.

[18] Internet Engineering Task Force, *IETF RFC 6455: The WebSocket Protocol*, IETF, Orlando, Fla, USA, 2011.

[19] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[20] N. M. Laxaman, M. D. J. S. GoonathiIake, and K. D. Zoysa, *TikiriDB: Shared Wireless Sensor Network Database for Multi-User Data Access*, CSSL, 2010.

[21] N. Leavitt, "Will NoSQL databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[22] N. Dimiduk and A. Khurana:, *HBase in Action*, O'Reilly Medi, Sebastopol, Calif, USA, 2013.

[23] F. Chang, J. Dean, S. Ghemawat et al., "Bigtable: a distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, article 4, 2008.

[24] N. Downie, "Chart.js," 2013, http://www.chartjs.org/.

[25] "Introducing Socket.IO," 2013, http://socket.io/.