A Software Framework for the Bidirectionally Controlled IoT Gateway and Server

TsuYi Peng
Department of Information Management
Shu-Te University
Kaohsiung, Taiwan
johnpeng@stu.edu.tw

***Abstract*— The control of sensors over the Internet has become a reality in the past few years in the IoT (internet of things) field. Numerous SaaS (software as a service) applications for sensors are already provided in the public cloud, but the operations of a dedicated server platform to perform a scalable and bidirectional sensor control have not been much examined, and this lack may limit the sensor application to commercial SaaS operators. In this work, the primitive components of both a sensor gateway and a server platform are designed and implemented, to develop a generic web service framework for supporting the application of bidirectional controlled sensors . This work may provide a reference design for related works.**

Keywords— IOT, Web service, Embedded gateway

I. INTRODUCTION

The IoT framework can be divided into two major parts: a gateway device that is operated at the sensor side to handle the sensor network data and to process the data exchange with a remote IoT Server ; a IoT server that is operated remotely to collect sensor data and to control its dominated sensor gateways [1].

Since the gateway device may be operated in a private network zone, a tunnel or a persistent connection should be set up between the gateway device and the IoT server to achieve transparency for bidirectional transmission. For a web based software framework, the long-polling mechanism can be applied to achieve the bidirectional message exchanging requirement.[2]

In recent years, sensors for personal use, such as pedometers and health-care devices, have become popular for use with public cloud services. A well-known commercial example is the NIKE+™ Fuelband™, which allows users to check their data on their running history via the NIKE+ service portal. These data can be shared with friends on social networks. For the framework perspective, the wearable Fuelband device is the sensor; the smartphone is the sensor gateway device, and the NIKE+ portal is the IoT server, akin to the IOT framework that was mentioned above and depicted in Fig. 1.

More advanced commercial examples of the bidirectional control of sensors are available, one of which is a “lost and found” service [3]. This service enables members of social networks to issue a “lost and found” APP run on members’ smartphones to search globally for a lost item to which Bluetooth sensors are attached. The examples presented above preceding paragraphs all operate on the commercial SaaS

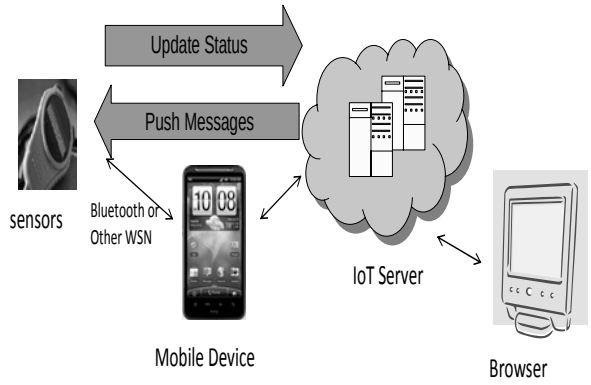

Figure 1. Gateway Device and IoT Server

platform and require the purchasing of a platform bundle sensor device. However, novel applications require a dominant and generic framework.

In this research, a software framework for the operations of the IoT gateway and IoT server is designed and implemented. Our works illustrate the design of the software components by the use case analysis and the data flow operational diagrams. The software framework is also evaluated by using an embedded hardware and a dominant server that is run on a VMware platform.

II. REQUIREMENTS

The IoT framework makes the machines or physical systems to be connected and controlled, many intelligent applications (e.g., heath care, smart home, children safety monitoring, power meter reading) can be deployed on top of the IoT devices and software components. In [4], the IoT framework has been expressed by four major components including sensing, heterogeneous access, information processing, applications and services. For a software framework perspective, these four components can be divided into two major parts: gateway software components that handle sensors interactions and data exchanges; IoT server components that handle data processes and device controls.

To operate the IoT framework, a service infrastructure has to be established to process the IoT data and control

exchange between the gateway device and the IoT server. In today's cloud computing environment, this service can be operated in SaaS (software as a service), PaaS (platform as a service), or IaaS (infrastructure as a service).

Table I compares various factors regarding the service operation such as cost for large number of gateway devices

TABLE I. COMPARISION FOR SERVICE PLATFORM

	SaaS Platform	**PaaS Platform**	**IaaS Platform**
Cost(large device numbers)	High	Mid	Low
Deployment Time	Short	Long	Long
Level of domination	Low	Mid	High
Flexibility	Low	Mid	High
Expendability	Low	Mid	High

handling, deployment time, level of domination, flexibility and expendability.

The comparison is based on surveys of the major players in the commercial market: IoT SaaS operators such as iDigi™, Axeda™, PaaS operators such as Google™ and Microsoft™, and IaaS operators such as Amazon™ and Google™.

The goal of this research is to provide a dominant framework which can be deployed with flexibility, hence, the chosen of our service operational platform based on Table I will be an IaaS environment in which a self-owned server platform can be deployed on different vendors.

III. SOFTWARE FRAMEWORKS

The following sub-sections will separately describe the use case analysis of the framework, gateway software components and IoT server interactions.

A. Use case analysis

Fig. 3 presents a use case diagram of the usage behavior analysis, the left side displays the use cases at the gateway side with the data exchange processing in the gateway device; the right side displays use cases for the IoT server with the provided web services .

The use cases in the gateway are procedures to handle the actors of user configuration, sensor I/O data, gateway agent program and time event. A gateway agent program has responsibilities to provide the network transporting procures for data and control exchanged with the IoT server.

The use cases in the IoT server are procedures to handle the actors of gateway device, query device and other long-polling clients. Notes that, besides the agent program of the gateway device , other clients such as web browser clients can also use the web services procures in the IoT server to query the status of sensor data.

B. IoT Gateway Components

Software components for gateway device must be portable for various kinds of hardware. To develop software that is portable, the software is divided into two parts - a platform-independent client library and a platform-dependent agent program. Figure 3 presents the DFD diagram of the designed modules. The agent program can poll the sensor data or be triggered by the sensors. The data exchange part and the long-polling connection part are handled in the portable library.

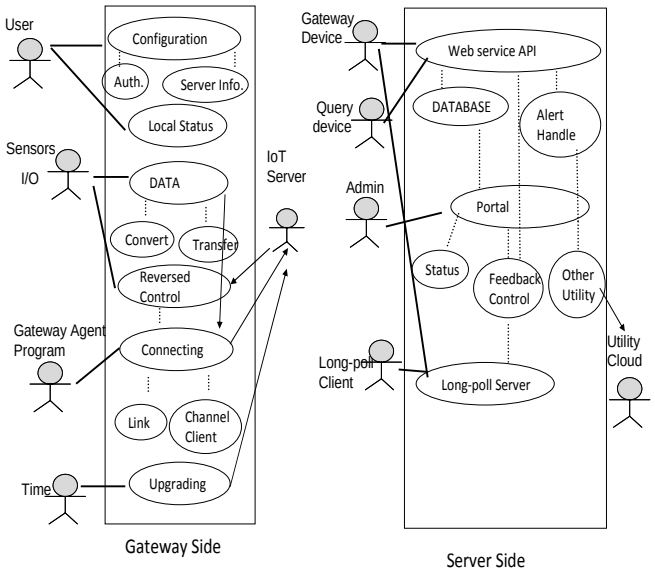

Figure 2. Use Case Diagram

C. Server Interactions

The IoT server provides the web service for client device accessing. The software framework needs to consider the scalability issues. Under an IoT service framework, many thousands of devices could request the services at the same time. Studies of dynamic web contents [5][6] show that FastCGI outperforms CGI, JSP and Serverlet technology with comparison of service performance. In our works, the FastCGI mechanism is applied to provide the scalable web service.

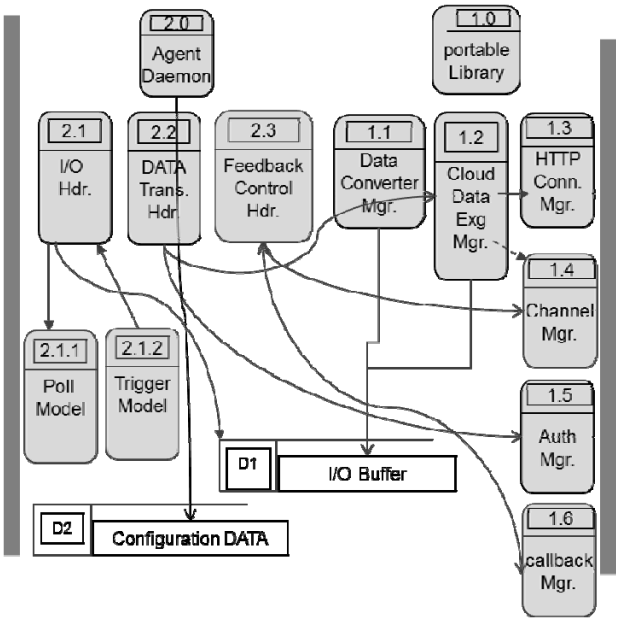

Figure 3. Software modules for the gateway device

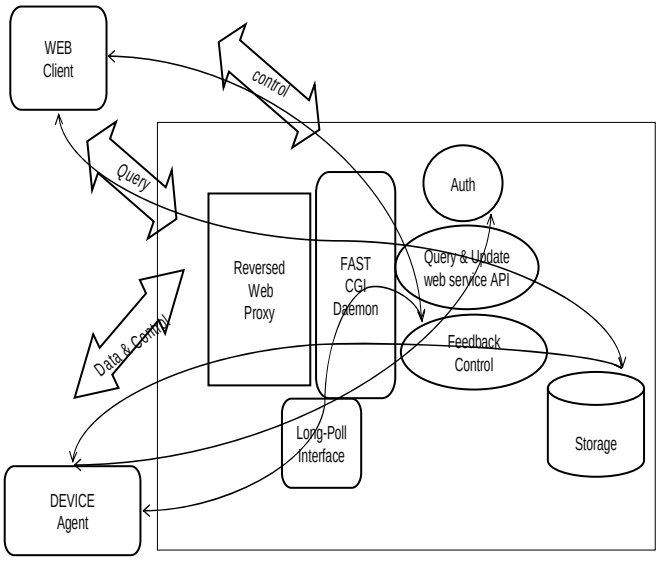

Figure 4. Server framework for client interactions

Another consideration has to be made that is the bidirectional transmission. Studies of event-based feedback control [2][7] have shown that long-polling technology can outperform push technology; it eliminates the periodic polling overhead that is associated with push technology. In our works, the long-polling mechanism is applied to provide the bidirectional transmission.

Fig. 4 shows that the design of the server framework. The web services are provided by a FastCGI daemon and the long -polling interface is used as the mechanism for feedback control. A reversed web proxy is placed ahead of the FastCGI daemon in which this proxy is used as a load balancing entry point to direct the web traffic load into different FastCGI daemons for scalability control.

IV. IMPLEMENTATION and EVALUATION

This section will first discuss the implementation issues of the software frameworks, then the evaluation of the implemented frameworks will be illustrated.

A. Implementation discussion of server

The IoT server is implemented by using a dedicated Linux server (Ubuntu 12.04) with domination of the OS control. Considering the management of more than 100k connections requires some fine-tuning of the operating system, to increase the number of file handlers, socket handlers, socket buffers, TCP queue backlogs and thread numbers. Table II summarizes the tuning parameters of the operating system and the web server. The fine-tune yields a 100K capacity as shown in Section Ⅳ.B.

As discussed in the previous section, a long-polling server is applied in the server framework to provide the bidirectional transmission control. The gateway device initials a persistent connection to the long-polling server and waits for the messages to be handled.

TABLE II. TUNING PARAMETERS

Resources	**Value**	**Config file**
File number	102400	/etc/limits,conf
TCP memory	67108864	/etc/sysctl.conf
TCP syn backlog	10240	/etc/sysctl.conf
Web process number	48	/etc/nginx/nginx.conf
Connection number Per process	100000	/etc/nginx/nginx.conf

The source code of the long-polling server is modified from that of the NGINX comet module [8], which can provide long poll interfaces for bidirectional communication. The comet module supports two kinds of service - publishing and subscription. The publisher service receives messages sending for a specific channel or a group of channels that share the same prefix. Messages are delivered to the channels accordingly. The message queue length and message timeout should also be considered. The implementation herein uses a single length queue (length =1) and short timeout period (60 second) to eliminate the system resources. The subscriber service receives the HTTP GET long-polling connection and maintains a channel ID for each connection that is used for exchanging messages

B. Implementation discussion of gateway device

To support portability, the client library incorporates the libcurl library to perform HTTP communication; the libcurl library is a cross-platform http client library. The client library also implements the feedback control mechanism to support the bidirectional control. The client library can convert the sensor I/O data to the HTTP payload, by using

the HTTP GET and HTTP POST commands to communicate to the IoT server which provides web services.

C. Framework Evalution

In our works, a “Raspberry Pi” embeded board is used for the gateway device demostration. The “Raspberry Pi” board is an ARM 11 based embeded hardware with a 700Mhz CPU. The hardware confiure is shown in Fig. 5. , a 3G / Wifi USB dongle is used as the transmission interface for the wireless usage ; a RS232 USB converter is used as an I/O interface to communicate with the sensor networks. The client library for the gateway device is ported for the linux platform on a raspberry board with the “.so” format in our demostration. An agent program is exectued in the raspberry board by relying on the portabl library to perform data and control exchange with the IoT server.

The IoT server is acted in a VMware environment to provide the web services. Fig. 6 shows the performance of the long-polling server in handling different loads of connections for the connection “Establish Time”. The long-polling connections are generated in a stable network environment with a fixed interval of 5 sec; the total establishing time is recorded for every additional 5000 connections. Fig. 6 shows that the performance is very stable (established time < 10 sec) under 60,000 connections. The limitation is reached near the point at 100,000 connections where the establish time grows exponentially. To resolve this performance issue, the reversed proxy showed in the Fig.4 can be applied to redirect the traffic load to other FastCGI daemons.

V. CONCLUSION

In this work, a software framework for providing the IoT service with a bidirectional transmission is proposed and implemented. The software components are designed for the gateway device and the IoT server. Many novel sensor applications can apply our framework to enable a flexible service deployment, running IoT services in a dominant IaaS environment.

The scalability requirement for supporting large number of connected gateway devices is also evaluated; the FastCGI and long-polling mechanisms for the IoT web service implementation are proofed to have capability of handling 100K connections.

REFERENCES

[1] Kim EJ, Youm S, Machine-to-machine platform architecture for horizontal service integration, EURASIP Journal on Wireless Communications and Networking 2013,2013(79): 1–9

[2] Eric Stratmann , John Ousterhout , Sameer Madan, Integrating long polling with an MVC framework, Proceedings of the 2nd USENIX conference on Web Web application development, 2011

[3] http://www.thetileapp.com/, accessed 25 Sep 2013.

[4] Chen M, Wan J, Li F , Machine-to-machine communications: architectures, standards, and applications. KSII T Internet Inf 6(2):471–489 , 2012

[5] Bhupesh Kothari and Mark Claypool, Performance Analysis of Dynamic Web Page Generation Technologies, International Network Conference (INC), Plymouth, United Kingdom, July 3-6, 2000

[6] V. Apte, T. Hansen, and P. Reeser, “Performance Comparison of Dynamic Web Platforms”, Computer Communications, vol. 26, no. 8, pp. 888–898, Out 2003

[7] E. Bozdag, A. Mesbah, and A. van Deursen, A comparison of push and pull techniques for Ajax. In Proceedings of the 9th IEEE International Symposium on Web Site Evolution, pages 15–22, 2007.

[8] http://pushmodule.slact.net/ , accessed 25 Sep 2013.

Figure 5. Gateway device using RasberryPi board

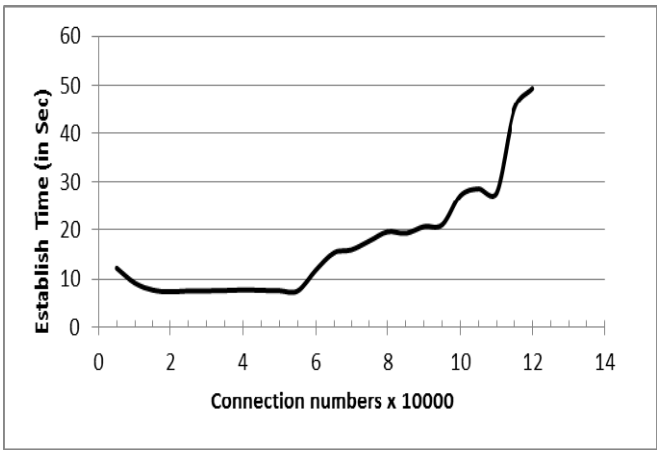

Figure 6. Performance on connection numbers