WiPi: An Extendable Edge Platform for Building Time-critical Cyber-Physical-Human Systems

MD Asif Hasan
FEECS, Ningbo University
Ningbo, Zhejiang, China

Haiming Chen*
FEECS, Ningbo University
Ningbo, Zhejiang, China
chenhaiming@nbu.edu.cn

Yimo Lin, Xiwen Liu
FEECS, Ningbo University
Ningbo, Zhejiang, China

ABSTRACT

Currently most of cyber-physical-human systems are built on the cloud centric architecture, which can not support delay-sensitive applications so well. With the development of edge computing, it is possible to construct time-critical cyber-physical-human systems based on edge platforms which can process users requests locally and provide low-latency interaction for users effectively. In this paper, we present our work on designing and implementing an edge platform, named WiPi. To verify its effectiveness in building time-critical cyber-physical-human systems, we use WiPi in building a system for human-in-the-loop control of an Unmanned Surface Vehicle (USV) for online water quality monitoring. Experimental results show that the edge-enabled system can reduce the interaction delay between the device and the user by 54% at least, as compared with the cloud-centric system.

CCS CONCEPTS

• **Computer systems organization** → **Embedded hardware**; *Embedded and cyber-physical systems*.

KEYWORDS

Cyber-Physical-Human Systems, Cloud Computing, Edge Computing, Unmanned Surface Vehicle

ACM Reference Format:
MD Asif Hasan, Haiming Chen, and Yimo Lin, Xiwen Liu. 2019. WiPi: An Extendable Edge Platform for Building Time-critical Cyber-Physical-Human Systems. In *ACM Turing Celebration Conference - China (ACM TURC 2019), May 17–19, 2019, Chengdu, China*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3321408.3321412

*Corresponding author.

ACM TURC 2019, May 17–19, 2019, Chengdu, China

ACM ISBN 978-1-4503-7158-2/19/05…$15.00
https://doi.org/10.1145/3321408.3321412

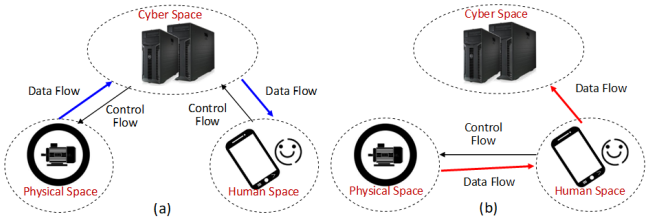

Figure 1: Two architectures of cyber-physical-human systems: (a) cloud platform centric (b)edge computing enabled.

1 INTRODUCTION

With rapid development of information technology, more and more physical objects, like household appliance (e.g. air conditioner), factory equipment (e.g. grinder mixer) and municipal infrastructure (e.g. gas meters), are digitized as smart devices. These smart devices interact with networked computers in the cyber space and involved persons in the social space, to compose a cyber-physical-human system. Currently most of cyber-physical-human systems are built with the architecture shown in Fig. 1(a), which has been widely adopted in building Internet of Things (IoT) systems, such as the smart manufacturing system built on the China-Mobile OneNET platform[1] and the smart home system built on the Haier U+ platform[2]. The advantage of building cyber-physical-human systems centered on the Cloud platform (a.k.a. IoT cloud platforms [4]) is making integration of devices in cyber-physical-human spaces easily, and management of these devices distributed in the Internet efficiently. So now many IoT cloud platforms have been provided for open access, such as Apples iCloud, Google Cloud Platform, Microsoft Azure, Amazon AWS, IBM Watson, Ali Cloud IoT Hub, and etc.

However, the cyber-physical-human systems built with the architecture centering on the IoT cloud platforms have a drawback in providing low-latency interaction between smart devices and users, which is required by time-critical applications, such as human-in-the-loop control of an Unmanned Surface Vehicle (USV), because both data flows from smart devices to users and control flows from users to devices need going through the Cloud platform in the cyber space. With the development of edge computing [6], it is possible to construct time-critical cyber-physical-human systems based on edge platforms which can process users' requests locally

[1] https://open.iot.10086.cn/ocp/welcome/cases
[2] http://www.haieruplus.com/haierzhihui.htm

and provide users with access to smart devices easily, effectively and immediately, as shown in Fig. 1(b). Now several edge platforms[1, 2, 9] have been proposed for building such systems. However, these platforms focus on solving the resource sharing problem by loading services into container, such as Docker, in the edge, which is not so applicable in some resource limited edge devices.

In this paper, we present our work on designing and implementing an extendable edge platform based on Raspberry Pi, named WiPi, for low-latency interaction between smart devices and users. The main challenges and our contributions of this work are embodied in the following aspects.

(1) How to implement edge services in the WiPi platform? The edge services can handle requests from users for reading sensor data and operating associated peripherals effectively. In this paper, we proposed a service architecture based on the raw TCP socket programming framework to implement it.
(2) How to make the WiPi platform easily used in practical cases? To make the platform easily used, it usually needs a user-friendly interface. In this paper, we implemented a user interface based on the Android APP, which is illustrated with a case study of applying the platform in building a system for human-in-the-loop control of an Unmanned Surface Vehicle (USV).

The rest of the paper is organized as follows. In section 2, some related works are briefly described. In section 3, we introduce the design of WiPi, and then elaborate on the implementation of WiPi in section 4. In section 5, we give a case study to test the performance of the platform in terms of the interaction latency between the device in the physical space and the user in the human space. At last, we make a conclusion and discuss our future work in section 6.

2 RELATED WORK

There have been some edge platforms proposed for building cyber-physical-human systems. Most of these platforms are designed to support online job dispatching[5, 7] or task scheduling[3, 10] between the cloud and the edge. Some of the latest works are briefly introduced below.

In [1], the authors considered the requirements of deploying pervasive services in open environment and designed a participatory edge computing platform based on the home gateway running Docker containers.

In [9], an edge platform using contrainer-based virtualization technology was designed to modularize each data processing instance and provide various data processing services for user requirements.

In [2], the authors considered the requirements of constructing the Web of Things systems with the mashup programming model, and developed a RESTful runtime container for dynamic installation, update, and removal of scripts.

The main concerns addressed by these platforms are how to provide support for running multiple applications in the edge, while WiPi mainly aims to provide support for low-latency interaction between the smart devices and the users.

3 DESIGN OF WIPI

From the system architecture shown in Fig. 1(b), we see that WiPi is located in the physical space and implemented in a local computer or an embedded device. This edge platform can be deployed using wireless communication technology, such as Bluetooth, ZigBee and WiFi. In this work, we use WiFi to connect all the smart devices in the cyber space and the human space to the platform. A WiFi router may be used as a gateway to establish this connection.

The reasons why we use WiFi as the communication technology of the edge platform are as follows. (1) An edge service built on TCP socket will run on it, and more than one interaction requests will be handled by the service. WiFi is qualified to support multiple connection requests. (2) WiFi provides sufficient bandwidth for enabling large volume data transmission between smart devices and users with relatively low data loss rate. (3) WiFi has relatively wide user accessibility. So connections between smart devices and users can expand in a larger area, which makes it preferable in outside environments.

When designing the platform, besides consideration of choosing a proper wireless communication technology, we need to find an embedded device which are capable of integrated with any kind of peripherals in the physical space, such as the manufacturing component or smart home appliance. To make the edge platform easily used, we also take the choice of user device and development of user interface into consideration when designing the platform. Below we will explain the design of these two parts briefly.

3.1 Embedded Device

Developing the edge platform in a tiny embedded device, firstly it needs to have a network interface to connect to the WiFi. Secondly, it requires certain hardware equipment, like some I/O (Input/Output) pins that can read digital signal and able to produce digital signal, so that it can control status of the devices and get data from the sensors. Thirdly, it should have an operating system running inside the embedded device, so that a service can be deployed inside it and perform some specific tasks such as controlling other electronic components and interacting with them.

Raspberry Pi meets all these requirements. So we build WiPi on it, and deploy a service for interacting with users directly. Considering the running cost of a service in the embedded device, we create the edge service using the raw TCP socket.

3.2 User Device and User Interface

Considering that smart phone is widely used in recent days, we choose smart phone as the user device and realize a graphic user interface to interact with the edge platform. In our work, we take the most popularly installed Android system as operating system of the user device, and develop corresponding APP to show the sensed data received from WiPi, and send control commands to WiPi.

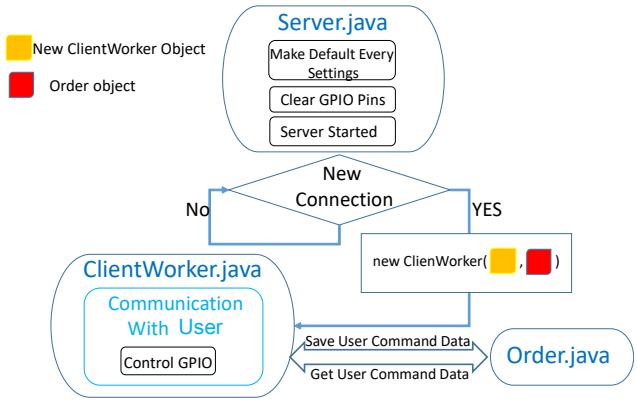

Figure 2: Software components and their relationship of the edge service running in the WiPi.

4 IMPLEMENTATION OF WIPI

According to the design of the edge platform, we implement WiPi by developing the edge service running in the Raspberry Pi and the user interface running in the smart phone.

4.1 Edge Service

We construct the edge service using the raw TCP socket programming, because of its high efficiency. In Fig. 2, we can see the way the service works inside the embedded device and the calling relationship among the classes. The edge service is implemented in JAVA, and is mainly comprised of the following three java files.

- Server.java: In Fig. 2, we can see how the program runs inside the platform. In the Server class, it has two methods, namely makeDefault() and wait(). The method makeDefault() sets the status of the platform to the default, by sending signals to the GPIO pins where sensors or other associated peripherals are connected to the platform. After it finishes setting of GPIO pins, it will shut down them. Then it will start server at port number 54321. After opening server port, it will start an infinite loop to wait for new connections. Whenever a new connection is established, it will call a class method defined in Clientworker.java and run it in a new thread. Besides, the method wait() is defined to make the platform work in an efficient way by calling the "Thread. Sleep()" method when we want to post a system delay. This wait () method takes string parameters for specifying the amount of time we want to delay in seconds, milliseconds or microseconds.
- Clientworker.java: The Clientwork class is an implementation of the Runnable class. It is called from the Server class. Its constructer receives two parameters from the Server class, one is the socket client object and the other is the object of Order class. Inside the run method of Clientwork, it has an infinite loop that continuously communicates with the user client. If communication between the user clients is somehow interrupted or disconnected, or if the error of reading timeout occurs it will break the loop by calling the endTransmission () method. In the loop, it first waits for the client to send data. It will process the data and set the GPIO pins state by making them high or low. After controlling the GPIO pins according to user data it will read data from the sensors and send them to user in the "###" string format.
- Order.java: The Order class has some variables to indicate the statuses of the associated peripherals of WiPi, which is changed by commands sent from users. It also has some methods to set and get the values of these variables. For example, if we are controlling the brightness of a LED connected to WiPi, it will have a variable to save the brightness level of the LED in this class. In the Server.java class, we create an object of the Order class and pass it along with every new objects of the ClientWorker class. Every time a new user client connects to the platform, it will create an object of the ClientWorker class, and then the Server passes the object of the Order class along with the object associated with the socket of user client. Every ClientWorker class has the same object of this class, letting users to view the changes made by other users easily.

4.2 User Interface

The user interface of WiPI is implemented by developing an APP in Android Studio. In current version, we only implement two activity layouts in the APP.

In the 1st activity, there is one button named connect. Users click on it to activate the second layout. This layout class is defined in the file MainActivity.java.

In the 2nd activity, it mainly has two threads. One thread is used to send commands and check if the end device gets the command successfully or not. The other thread is to communicate with IoT device to get sensor data. In the case study presented below, the commands are defined to change the rotating speed and direction of the propeller in a USV.

5 PERFORMANCE EVALUATION

To verify of the advantage of the proposed edge platform in providing low-latency interaction between smart devices and users for building time-critical cyber-physical-human systems, we implemented a system for human-in-the-loop control of an Unmanned Surface Vehicle (USV) for online water quality monitoring.

The architecture of the whole system is based on the edge computing enabled one, as shown in Fig. 1 (b). So in the system there is a device in the physical space, a remote server running in the cyber space, and a user in the human space.

For the part of the system in the cyber space, we have developed a tiny cloud server, named Octopus[3], running in the AliYun.

[3]http://120.27.227.135:8080

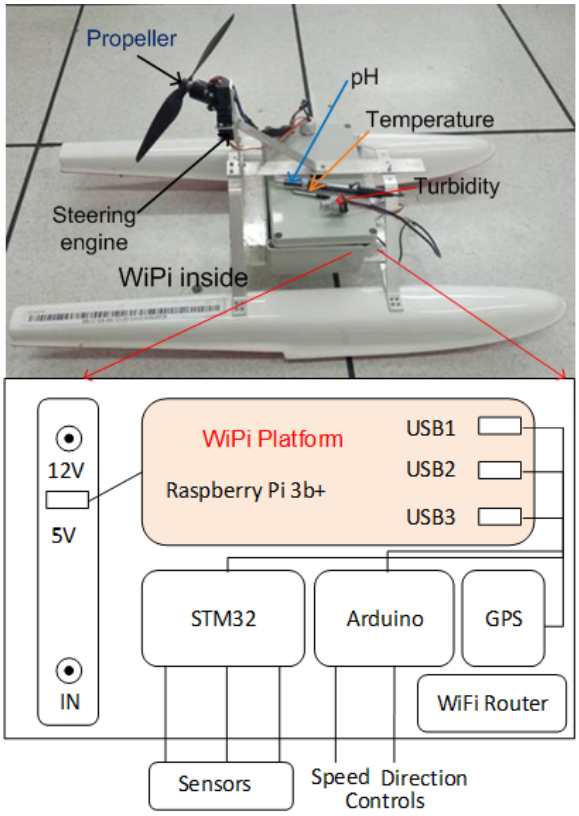

Figure 3: An Unmanned Surface Vehicle (USV) with WiPi inside.

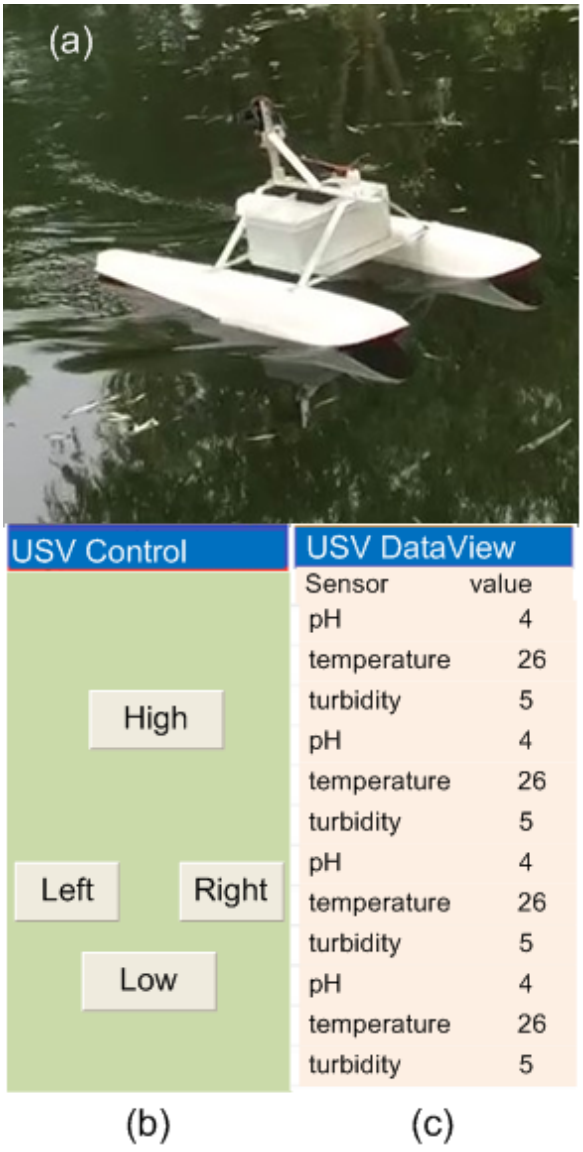

Figure 4: Interacting between WiPi and APP. (a) Moving status of the USV, (b) User interface for controlling the moving status of the USV, (c) User interface for showing the collected data.

For the part of the system in the physical space, we developed a USV based on WiPi, as shown in Fig. 3. The USV is driven by the propeller, of which rotating speed and direction can be controlled by the steering engine connected with WiPi. It carries some sensors, namely a pH sensor, a temperature sensor and a turbidity sensor, to collect some data about the water quality while it moves. Other components carried by the device are shown in detail in the right half part of Fig. 3. We can see that two embedded boards, namely STM32 and Arduino, and a GPS module are connected with the WiPi platform as associated peripherals through USB interfaces. Besides, there is a rechargeable power supplier and a mini WiFi router, which provide WiFi connection for both the platform itself and the users in human space.

For the part of the system in the human space, we have an APP running in Android smart phone, as described in the previous section.

In Fig. 4, it shows the interacting between WiPi and APP. In particular, in Fig. 4(b), it shows the user interface for controlling the moving status of the USV (i.e. the 2nd activity mentioned in the subsection 4.2), where the "Left" and "Right" buttons are for direction control, and the "High" and "Low" buttons are for speed control. In Fig. 4(c), it shows the user interface for showing the collected data, from which we can see the list of sensing data online.

Because the interaction between the USV and the user is delay sensitive, to show the effectiveness of the proposed edge platform in this kind of cyber-physical-human systems, we also implemented the system based on the cloud-centric architecture, as shown in Fig. 1(a), as the comparison object.

Fig. 5 shows the comparison results of interaction delay between devices and users with the number of devices in the system increasing. The interactive delay is defined as the time span from the time point when the user sends a

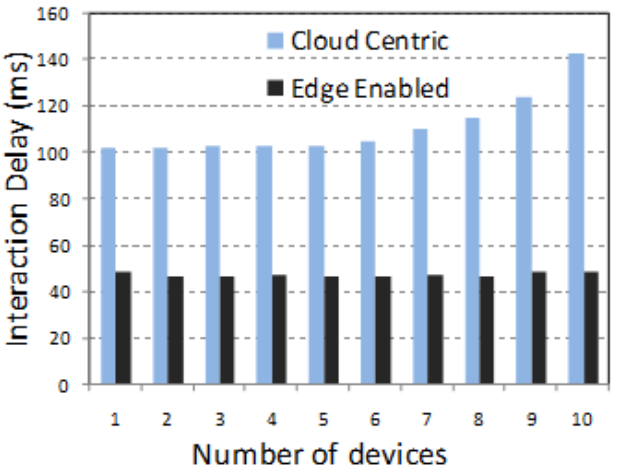

Figure 5: Interaction delay between devices and users in the system for human-in-the-loop control of a USV with the number of devices increasing.

command to the time point when he gets action response from the device. We can see that the interaction delay can be reduced by 54% at least as the number of devices increases from 1 to 10, when it is compared with that in the system based on the cloud-centric architecture. The increasing trend of the interaction delay starts from the point when there are 7 devices in the system, mainly due to that all the data flows and control flows are processed by the central cloud platform, which has limited resource. However, this limitation does not exist for the edge computing enabled system, because all the interactive requests are handled by the corresponding WiPi platform distributedly.

6 CONCLUSION AND DISCUSSION

In this paper, we present our work on designing and implementing an edge platform, named WiPi, for building time-critical cyber-physical-human systems. In current stage of our work, we mainly solve the problem of how to provide low-latency interaction service between devices in the physical space and users in the human space. We have verified the effectiveness of the developed edge platform for building time-critical cyber-physical-human systems with a case study. In the case study, we have built a system for human-in-the-loop control of an Unmanned Surface Vehicle (USV) for online water quality monitoring. Experimental results show that the edge-enabled system can reduce the interaction delay between the device and the user by 54% at least, as compared with the cloud-centric system.

It is worth noting that in this paper we focus on providing edge service for low-latency interaction between the device and the user, which is required to build time-critical cyber-physical-human systems. This is not the unique purpose of our developed edge platform. We will extend the edge service to process the sensed data locally and make on-the-spot decisions, so as to provide more support for building time-critical cyber-physical-human systems. Taking the case studied in the paper, we will make the WiPi platform process the sensed data to judge the water quality locally. Besides, because the USV can move on the river under the interactive control of human, it can collect data continuously. Therefore, we consider to make the WiPi platform deduce the source of water pollution, when it detects the water quality lower than a threshold.

It is also worth noting that WiPi can be easily extended by associating with other peripherals and sensors to be used in other applications (i.e. the meaning of "extendable" in the title of the paper), such as continuous condition monitoring of vibration for diagnosis of mechanical fault or structural health, and compare their performance with the systems for condition monitoring in the cloud[8].

ACKNOWLEDGMENTS

Partial work of this paper is supported by the Zhejiang Provincial Natural Science Foundation of China (LY18F020011), Ningbo Natural Science Foundation (2018A610154) and the National Science Foundation of China (NSFC) (61100180).

REFERENCES

[1] A.M. Khan and F. Freitag. 2017. On Participatory Service Provision at the Network Edge with Community Home Gateways. In *Proceeding of the 8th International Conference on Ambient Systems, Networks and Technologies (ANT'17)*. Madeira Portugal, 1–4.

[2] M. Kovatsch, M. Lanter, and S. Duquennoy. 2012. Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications. In *Proceeding of the 3rd International Conference on the Internet of Things (IOT)*. Wuxi, China, 135–142.

[3] Y. Li, Y. Chen, T. Lan, and G. Venkataramani. 2017. MobiQoR: Pushing the Envelope of Mobile Edge Computing via Quality-of-Result Optimization. In *Proceeding of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17)*. Atlanta, GA, 1261–1270.

[4] P. P. Ray. 2016. A survey of IoT cloud platforms. *Future Computing and Informatics Journal* 1, 1-2 (2016), 35–46.

[5] S. Sarkar, S. Chatterjee, and S. Misra. 2018. Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Transactions on Cloud Computing* 6, 5 (2018), 46–59.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[7] H. Tan, Z. Han, X. Y. Li, and F. C. M. Lau. 2017. Online Job Dispatching and Scheduling in Edge-Clouds. In *Proceeding of the IEEE Conference on Computer Communications (INFOCOM'17)*. Atlanta, GA, 1–9.

[8] E. Uhlmann, A. Laghmouchi, E. Hohwieler, and C. Geisert. 2015. Condition Monitoring in the Cloud. *Procedia CIRP* 38 (2015), 53–57.

[9] H. Watanabe, Tohru Kondo, and Toshihiro Ohigashi. 2019. On Participatory Service Provision at the Network Edge with Community Home Gateways. In *Proceeding of IEEE International Conference on Pervasive Computing and Communications Work In Progress (PerCom'19 WiP)*. Kyoto, Japan, 1–4.

[10] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu. 2016. Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software Defined Embedded System. *IEEE Trans. Comput.* 65, 12 (2016), 3702–3712.