The Φ-Stack for Smart Web of Things

Zhiwei Xu
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
zxu@ict.ac.cn

Xiaohui Peng
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
pengxiaohui@ict.ac.cn

Lei Zhang
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
zlei@ict.ac.cn

Dong Li
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
lidong@ict.ac.cn

Ninghui Sun
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
snh@ict.ac.cn

ABSTRACT

With the emergence of smart IoT, a key problem has becoming increasingly acute: the "Paradox of Classis Insecta", which in industry is called the fragmentation problem. We propose an architecture called Smart Web of Things (SWoT) to address this problem. The SWoT architecture extends the REST architectural style that has been validated by PC Web and mobile Web to the smart IoT arena. However, previous researches have shown that such extension is nontrivial. According to Vint Cerf, intelligent Web is heavy, difficult to be realized in resource constrained smart IoT devices. We propose a novel co-designed hardware-software stack, called Φ-Stack, to natively support Web and intelligence (e.g., pattern recognition and deep learning), with minimized cost, footprint, response time, and energy consumption. This hardware-software stack must be an open architecture, to encourage innovations and accommodate diversity inherent in the smart IoT field. Preliminary experiment results show that our approach is promising in extending the Web technology to resource constrained smart IoT environments and helpful in partially unifying APIs and interactions among smart IoT devices.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; • **Software and its engineering** → Software system structures;

KEYWORDS

Edge Computing; Internet of Things; IoT system stack; RISC-V; REST; Smart Web of Things

ACM Reference format:
Zhiwei Xu, Xiaohui Peng, Lei Zhang, Dong Li, and Ninghui Sun. 2017. The Φ-Stack for Smart Web of Things. In *Proceedings of SmartIoT'17, San Jose / Silicon Valley, CA, USA, October 14, 2017,* 6 pages.

SmartIoT'17, October 14, 2017, San Jose / Silicon Valley, CA, USA

ACM ISBN 978-1-4503-5528-5/17/10…$15.00
https://doi.org/10.1145/3132479.3132489

https://doi.org/10.1145/3132479.3132489

1 INTRODUCTION

The past decade has seen an emergence of Smart Internet of Things. BI Intelligence estimated that there will be 34 billion connected devices worldwide by 2020, of which IoT devices will accounted for 70%, or 24 billion devices lower-end than smart phones and PCs[6]. Various estimations agree that we will soon have over a trillion sensors installed [5]. However, the smart IoT field faces a serious fragmentation problem, far worse than the previous two generations of PC Web and mobile Web.

We call this problem the "Paradox of Classis Insecta", which manifests as follows. On the one hand, the overall smart IoT market is huge, and apparently should soon reach tens or even hundreds of billions of devices. On the other hand, we have not seen a single smart IoT platform which sells over 100 million devices per year. This is in sharp contrast to PC and mobile Web. In the PC Web case (desktop and laptop computers), we have two predominant platforms: the X86+Windows+REST platform and the X86+macOS+REST platform. In the mobile Web case (smart phones and smart pads), we have two predominant platforms: the ARM+iOS+REST platform and the ARM+Android+REST platform. In fact, over 99% of PC and mobile Web devices use fewer than a dozen platforms. In the case of smart IoT, we have many platforms, each selling fewer than 10 million devices a year, including the famous Amazon Echo.

Professor Ken Sakamura of Tokyo University holds the viewpoint that the IoT field is analogous to the Classis Insecta, which has about 5 million species. In contrast, the PC and mobile Web field is analogous to the Classis Mammalia, which has only 5 thousand species [9]. In other words, IoT is inherently more diverse.

The Paradox of Classis Insecta calls for the research community to come up with innovative ideas and technologies to overcome or embrace such diversity and fragmentation. In this paper, we propose a Smart Web of Things (SWoT) architecture to address the Paradox of Classis Insecta challenge. The SWoT architecture extends the REST architectural style [13], which has been so successfully used in the PC Web and mobile Web fields, to the smart IoT arena.

This idea of extending Web to smart IoT resonates with the sentiments of the IoT community. According to the 2017 IoT Developer Survey [10] co-sponsored by the Eclipse IoT Working Group, IEEE IoT, AGILE IoT and IoT Council, of the top 5 protocols used by IoT

developers, three are REST protocols (HTTP, CoAP, and HTTP-2), with HTTP being top 1.

However, previous researches have shown that extending Web to smart IoT is nontrivial. According to Vint Cerf [8], intelligent Web is heavy, difficult to be realized in resource constrained smart IoT devices. Novel hardware-software co-designed stacks are needed to address Cerf's problem.

Both industry and academia have done extensive work in this direction. In hardware, traditional MCUs have been augmented to have more computing power and integration, with related designs of board-level systems. Examples include ARM Cortex-M series, Atmel AVR, TI CC series, Arduino, Raspberry Pi. In software, there are several efforts to adapt REST to sensor networks and IoT. Examples include pico-REST [11], CoAP [4], ubiREST [7]. REST protocols are also supported in IoT operating systems [14], such as Zephyr [17], Mbed [19], Contiki [12] and RIOT [3]. In computing models, several new concepts have emerged, such as Internet of Everything (IoE) [1], edge computing [18], and cloud-sea computing [20]. However, we still lack a co-designed hardware-software stack natively supporting SWoT.

To efficiently realizing SWoT, we propose Φ-Stack, to natively support Web and intelligent computing, with minimized resource consumptions. We also present preliminary experiment results showing evidence supporting our approaches.

2 THE SWOT ARCHITECTURE

The Smart Web of Things (SWoT) is both a research goal and a unifying architecture for the desired platform stack and systems. SWoT states three simultaneous objectives:

- SWoT systems must show smartness and intelligence.
- SWoT systems must efficiently support REST Web, including REST resource commands and scripting.
- SWoT systems must effectively support physical things, especially physical world facing devices such as sensors and actuators, not only human-facing devices such as phones and PCs.

All these objectives should be realized within the resource constraints such as cost, footprint, response time, and energy consumption. Due to the inherent diversity of smart IoT, the SWoT architecture should have the property of *resource constraints scalability*, namely, the same SWoT architecture and platform should be able to both scale down and scale up, in terms of resource constraints. Our current design aims at the following set of four resource constraints:

- Processor die costs no more than a single wafer site.
- Footprints of the software stack are no more than 128 KB ROM and 128 KB RAM.
- For simple Web benchmarks (such as turning on a humidifier via an HTTP Put), each command takes no more than 100 ms latency and consumes no more than 50 mW of power.
- For intelligent Web benchmarks (such as asking SWoT to turn on the kitchen light by voice), each command takes no more than 100 ms latency and consumes no more than 200 mW of power.

The SWoT architecture is illustrated in Figure 1. The area outside the dashed box shows traditional PC Web and mobile Web. Their

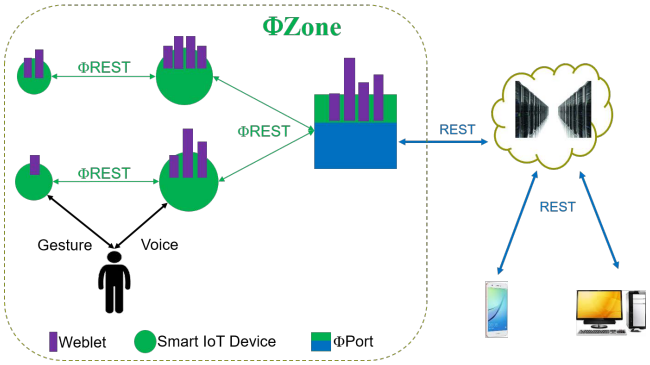


Figure 1: The Smart Web of Things architecture

thin waist is the REST architectural style [14] with associated protocols such as HTTP, HTTPS, HTTP-2. This REST commonality is a main reason why the PC Web and the mobile Web fields are much less fragmented than IoT.

Any SWoT application system is normally (but does not have to be) connected to the cloud. A key insight and assumption of SWoT is that smart IoT systems should not be chaotic or completed unorganized. They should have structures. A SWoT system is comprised of one or more zones, called Φ-zone in Figure 1. A Φ-zone is an abstraction of a structured part of the physical world, such as a smart home. A Φ-zone connects to the cloud via a gateway called a Φ-port. Within a Φ-zone, there are one or more smart IoT devices interacting by new Φ-REST protocols optimized for resource constrained IoT devices. The Φ-port automatically converts messages of Φ-REST within and REST without.

There may be multiple types of IoT devices within a Φ-zone, with different resource constraints, showing in Figure 1 as different circle sizes. However, they all share the same application abstraction called Weblet, analogous to the APP abstraction on a smart phone, but much lighter weighted. Zero, one or more Weblets may be running on an IoT device at the same time. A distinct feature is that within a Φ-zone, Φ-REST is the one and only thin-waist interaction mode for all interactions, be they device-device interactions, device-gateway interactions, or human-device interactions. All such interactions are realized via Weblets. That is, Weblets are the endpoints of interactions.

We further relate and differentiate the SWoT architecture from existing work below:

- To encourage innovation, the SWoT architecture is meant to be a clean slate design, and all existing systems aspects can be revisited, except one invariant: the REST architecture style.
- The clean slate design is a full-stack one, coordinating processor, operating system, software development kid, and application software.
- When saying inheriting REST, we mean to inherit the REST principles, not any particular protocols (HTTP or CoAP) or implementations.
- Several of the REST principles are emphasized:
  - Hypertext driven
  - Uniform interface

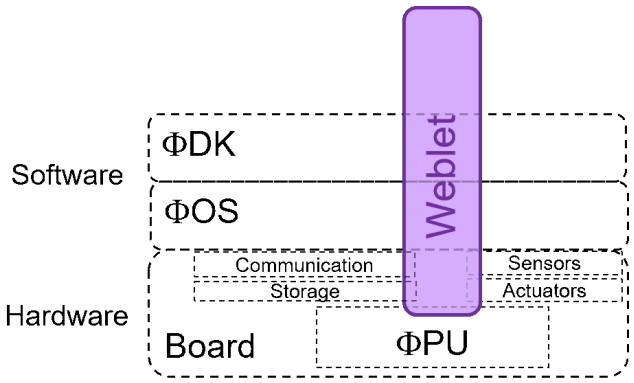


Figure 2: The Φ-Stack for Smart Web of Things

– Code on demand, such as scripting

- The concept of Φ-zone abstracts and delimitates the natural user logic boundaries, which can, but do not have to, coincide or intersect with geographic, administrative, or ownership boundaries.
- Each Φ-zone has one or more Φ-ports (gateways), which connect IoT devices within a Φ-zone to the cloud. The REST Web principles apply to both within and without a Φ-zone.
- Some work classifies devices into three groups: resource constrained devices, smart devices, and gateways. We offer an alternative classification. There are IoT devices and Φ-ports. IoT devices cannot directly talk to the cloud but have to go through a Φ-port. A Φ-port can also function as an edge server.

3 THE Φ-STACK

The Φ-Stack is a hardware-software stack with four main components, as shown in Figure 2. This is a clean-slate, holistically co-designed stack, specifically targeting the three objectives and four resource constraints of SWoT listed in Section 2.

That is why the Φ-Stack illustrated in Figure 2 needs to start from the ground up, namely, its design starts from a novel processor (ΦPU) all the way up to a novel form of application (Weblet), with novel system software in between (ΦOS and ΦDK). The Greek letter Φ represents "physical world", "physical", or "things". In other words, all innovations in processor, operating system, software development kit, and application software should focus on "physical". These innovative components should coordinate to generate the desired system effects of Smart Web of Things. We elaborate the Φ-Stack platform by discussing its design principles, implementation and evaluation below.

3.1 Principles of the Φ-Stack

The Φ-Stack follows four design principles to realize SWoT: hyper-integration, native interpretation, lightweight protection, and open architecture.

Hyper-Integration: Try to have a few hyper-integrated subsystem modules, instead of a federation of many arbitrarily loosely coupled components.

Table 1: Openness in PC Web, mobile Web, and things Web

Era	Hardware	Software
PC Web	Closed	Mostly Closed
Mobile Web	Mostly Closed	Closed + Open Source
Things Web	Mostly Closed	Closed + Open Source
Φ-Stack	**Open Source**	**Open Source**

For instance, in the case of application software (Weblet), most of the dirty work is done by reusable modules written as library codes. The Web commands (e.g., HTTP PUT and GET) and scripting codes (e.g., in Javascript or Lua [16]) glue together calls to libraries. An example of such hyper-integrated libraries is a deep learning library targeting the AI core of the processor.

In the case of hardware, we should strive for integration beyond traditional SoC practices. For desktop computing, microprocessors are introduced with only instruction pipeline and memory access interfaces. Other parts are on the motherboard. For mobile computing, SoCs are introduced to integrate processor cores and many functional IPs in a single chip. Other parts like sensors, memory and storage are on the PCB. For smart IoT systems, the physical form factors for computing are usually further shrunk, calling for chip design more integrated than traditional SoCs. Our goal is to achieve hyper-integration by putting processor, memory, RF, reconfigurable logic in a single chip. Only sensors and actuators interacting with the physical world are placed outside the chip. This could greatly ease the design of complex PCB boards.

Native Interpretation: Try to execute Weblet applications directly on the bare metal most of the time. The Weblets for things are different from smart phone APPs in that the Weblets often do not have heavy touch-screen interactions. On the other hand, Weblets normally need to interact with sensors and actuators. A Weblet program usually consists of three types of codes: Web commands for interactions, scripting codes for application logic, and library codes for low-level dirty work. All should be directly executed on the bare metal most of the time.

Lightweight Protection: Security and privacy protection is a major concern for smart IoT applications. In addition, since each IoT device can host multiple Weblets, the Φ-Stack design must make sure that they are properly isolated and do not interfere with one another.

Many techniques have been proposed for security, privacy, and isolation. The challenge is to find sufficiently light-weighted protection schemes within the budgeted resource constraints.

Open Architecture: Both the SWoT architecture and the Φ-Stack platform should be open. Our aim is to provide open architecture specifications, open interfaces and open source hardware and software implementations. This is essential to encourage innovations and to accommodate diversity inherent in the smart IoT field (cf. the Classis Insecta Paradox).

Our position is illustrated in Table 1, contrasting the PC Web, the mobile Web, and the smart things Web eras.

For the PC Web, the hardware design is closed. For example, only a few companies can design and produce X86 chips. The software is also mostly closed (Windows and macOS), except for a few Linux platforms. The mobile Web is more open. We have the open

Table 2: Main parameters of the ΦPU components

Components	Parameters		Functions
Big Core	ISA	RV32IM	Support Zephyr, FreeRTOS
	SRAM	64 KB data, 64 KB instruction	
	Pipeline	5 stages with sleep and wakeup by interrupts	
	Frequency	200 MHz	
Small Core	ISA	RV32I	Sensor Hub
	SRAM	32 KB data, 32 KB instruction	
	Pipeline	3 stages duty cycle	
	Frequency	80 MHz	
AI Core	ISA	RV32IM	Support ANN, RNN
	Processing Engines	8	
	Frequency	200 MHz	

source Android operating system, besides the closed iOS. ARM is the predominant processor architecture. The ARM company sells hard-core and soft-core licenses, enabling many companies to produce their own processor chips. The current smart IoT field has both closed and open source platforms. A goal of the Φ-Stack is to enable an open source hardware-software platform for the smart IoT field.

An encouraging development is the emergence of the RISC-V instruction set originated from University of California at Berkley, which is a free, open source instruction set for servers, PCs, smart phones, and IoT devices [2].The RISC-V community grows very quickly, and has established a RISC-V Foundation (https://riscv.org/). Our team (ICT) has joined the RISC-V Foundation as a founding member.

3.2 Implementation of the Φ-Stack

Implementing the entire Φ-Stack is a vast project. We are developing a prototype system to validate our approaches. We report four progresses implementing the Φ-Stack platform to support SWoT applications, including the ΦPU processor chip, the ΦOS software, the ΦDK tools, and application benchmarks.

ΦPU. We have taped out a ΦPU chip for smart IoT applications. The chip has three cores inside, a big core for general-purpose processing, a small core for monitoring and controlling sensors, and an AI core. All cores are based on RISC-V with extensions. Their main features are listed in Table 2.

To save energy, the three cores can be individually turned off. In many smart IoT application situations, only the Small Core needs to be kept on, to monitor sensors and to receive command signals from outside. However, even the Small Core does not have to be kept literally "always on". Instead, an operational mode of duty cycle can be used.

The chip is being manufactured at SMIC using 55 nanometer technology. The layout of the ΦPU die is shown in Figure 3. As we have a streamlined architecture, the die size is only 3μm x 4μm, which can be kept within one single site on a multi-project wafer (MPW).

ΦOS and ΦDK. We are trying two approaches to implement ΦOS and ΦDK, to test the feasibility of extending Web to resource constrained IoT devices.

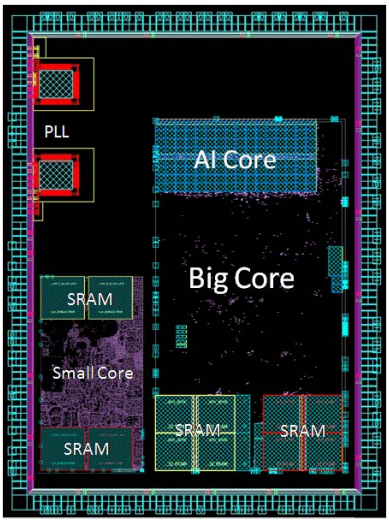


Figure 3: The ΦPU die layout

The first is an aggressive approach by adding new methods to the HTTP protocol to produce a new protocol called SeaHTTP [15], with associated scripting support called WebletScript, which is a modification of Javascript. Its implementation aggressively minimizes memory footprints.

Figure 4 illustrates the second approach, which modifies Contiki and Lua to support Weblets comprised of HTTP commands and scripting codes. The modified Lua is a new scripting language called LiteLua. We have streamlined the Lua scripting language to reduce footprint.

We are doing active conceptual research in the ΦOS area, by evaluating, modifying, and experimenting with existing operating systems such as Contiki, FreeRTOS and Zephyr. We are doing this to try out new ideas, especially in single address space isolations, labeled modules, mutually intelligible name spaces and algorithm-defined security and privacy.

Benchmarks. We are developing two types of benchmarks: (1) micro benchmarks which help pinpoint systems bottlenecks in terms of time, energy, footprint, and cost; (2) macro benchmarks

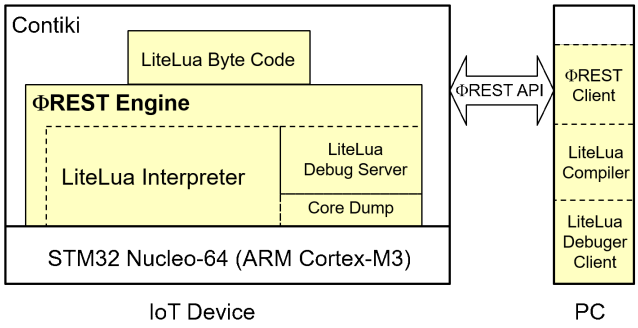


Figure 4: The ΦOS and ΦDK prototype architecture

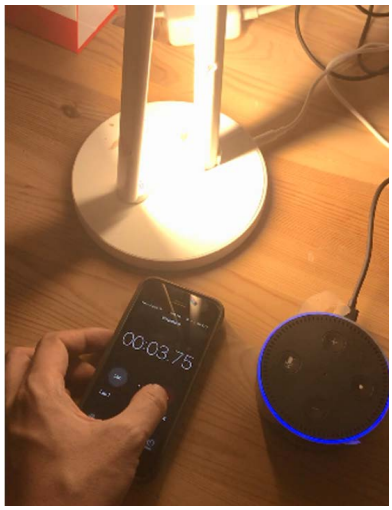


Figure 5: Measuring the delay in turning on a light by voice command

which evaluate user experience in terms of functionality, quality (e.g., user perceived accuracy and precision), and responsiveness. The macro benchmarks are further divided into two sets: simple Web requests such as automatically monitoring room temperature and humidity, and intelligent computing applications such as voice-assisted control of smart home devices.

A typical benchmark example of intelligent computing applications is turning on a light via Amazon Echo and Alexa. We did an experiment in London, as illustrated in Figure 5. The user asks by voice: "Alexa, please turn on the bedroom light". The Amazon Echo receives the voice command from the London apartment, sends a service request to the Amazon cloud in the USA for processing such as voice recognition, routes the results to another cloud in Singapore, which finally sends a command back to the London apartment to turn on the Xiaomi smart light in the bedroom.

This nontrivial process involves intelligent computing tasks such as human-machine natural interaction and far field voice recognition. The downside is that currently, this cloud-based process takes 3-5 seconds, which is not very responsive. By utilizing technology such as SWoT and Φ-Stack, we hope to reduce the latency to below

Figure 6: The FPGA evaluation board for ΦPU and Φ-Stack

0.1 second, below the psychological threshold of human perception, so that a human user will not feel any time gap.

In developing benchmarks for smart IoT, we should be aware that we cannot blindly use existing benchmarks in hot areas such as deep learning. For instance, for image recognition in smart IoT, the popular ImageNet suite may not be suitable. Most current work focuses on intelligent processing in cloud datacenters and mobile devices. The major objectives there are throughput and accuracy, such as processing more images per second, achieving higher recognition accuracy, etc.

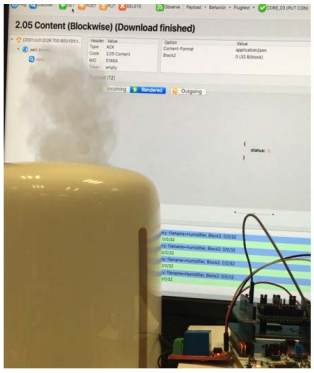
For smart IoT, the requirements of intelligent computing benchmarks are often different. A major evaluation metric in our benchmarks is low power consumption to achieve acceptable recognition accuracy.

3.3 Evaluation of the Φ-Stack

The ΦPU chip is still being manufactured. It is premature to evaluate the design and implementation of the full Φ-Stack. We evaluate the hardware and software separately.

Evaluation of ΦPU. We have built a FPGA-based system (Figure 6) to evaluate the ΦPU design and implementation.

- Low Cost. With 55 nanometer technology, it is possible to implement a ΦPU on a single MPW site, while supporting Web, intelligence, and physical things. A single-site core indicates that low cost is possible.
- Supporting Web. The FPGA-based evaluation board successfully runs the Zephyr operating system with its associated Web stack.
- Supporting Things. A distinct feature is how the ΦPU chip allocates one of its important resources: the pins. The chip employs a 256-pin BGA package, with 109 functional pins, of which 89 pins are used to support sensors and actuators.
- Supporting Intelligence. The ΦPU chip realizes a concept called "elastic AI" by implementing an AI core, which is an elastic neural network processing engine that can both scale down and scale up. The engine can do a simple character recognition task within 50 mW, and a complex face detection task within 300 mW.



LiteLua Script

```
local humidity = get_humidity()
    if ( humidity < 30 )
    then
    humidifier_on()
end
```

Figure 7: LiteLua script to turn on the humidifier when humidity is low

Evaluation of ΦOS and ΦDK. We report results of two experiments representative of the more aggressive SeaHTTP approach and the more balanced LiteLua approach. We use the experiments to test memory footprint, power consumption, and latency.

The IoT device for the SeaHTTP approach has the following configuration: an TelosB compatible platform with TI MSP430F1611 microcontroller and TI CC2420 RF module running a Contiki 2.7 operating system. We use the simple benchmark of turning on an LED on board via an HTTP PUT request issued from a laptop computer using Google Chrome explorer. The board receiving the request sends back the LED ON/OFF status as the response.

The total footprint of the SeaHTTP server and the Contiki OS is 47.44 KB ROM and 7.25 KB RAM. Excluding the Contiki with the UDP stack, SeaHTTP itself consumes only 5.12 KB ROM and 0.13 KB RAM. The average response time is 0.7 seconds and the peak power consumption is 63 mW measured by an external power meter.

The IoT device for the LiteLua approach has the following configuration: an STM32 Nucleo-64 development board equipped with 32 MHz ARM Cortex-M3 CPU, 80 KB RAM, and 512 KB flash. The communication capacity is enabled by adding ST X-NUCLEO-IDS01A5 RF module. The operating system is Contiki 3.1.

We use the simple benchmark of turning on a humidifier by the control board which locally has a power switch and a humidity sensor. The LiteLua script code is shown in Figure 7. The total footprint of this solution is 198 KB ROM and 16 KB RAM.

4 CONCLUSIONS

This paper reports progress of a clean-slate research project in smart IoT conducted at Chinese Academy of Sciences. Four results are presented: (1) the "Paradox of Classis Insecta" (i.e., the fragmentation problem) as the main research issue; (2) a Smart Web of Things (SWoT) architecture to address this problem, which extends the REST architectural style of the PC and mobile Web to the smart IoT field; (3) an co-designed hardware-software open platform, called Φ-Stack, to realize the SWoT architecture; and (4) preliminary implementation and experiment results giving evidence that our approach is promising. We are actively seeking academic criticism, feedbacks, and cooperation among the international smart IoT community, especially in the following areas:

- The SWoT architecture and the Φ-Stack platform.
- Killer examples: exemplar application scenarios revealing the challenges and advantages of smart IoT.
- Weblet: what should be the ideal form (not contents) of applications in the field of Smart Web of Things?

ACKNOWLEDGMENTS

This research is supported in part by the Ministry of Science and Technology of China (2016YFB1000200), the NSF of China (61532016) and Institute of Computing Technology, Chinese Academy of Sciences (20156010).

REFERENCES

[1] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. 2014. Enabling smart cloud services through remote sensing: An Internet of Everything enabler. *IEEE Internet of Things Journal* 1, 3 (2014), 276–288.
[2] Krste Asanović and David A Patterson. 2014. *Instruction sets should be free: The case for risc-v.* Technical Report. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146.
[3] Emmanuel Baccelli, Oliver Hahm, Mesut Gunes, Matthias Wahlisch, and Thomas C Schmidt. 2013. RIOT OS: Towards an OS for the Internet of Things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 79–80.
[4] Carsten Bormann, Angelo P Castellani, and Zach Shelby. 2012. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing* 16, 2 (2012), 62–67.
[5] Janusz Bryzek. 2014. Trillion sensors: Foundation for abundance, exponential organizations, Internet of Everything and mHealth. *Sensor Magazine* (2014).
[6] Jonathan Camhi. 2015. BI Intelligence projects 34 billion devices will be connected by 2020. http://www.businessinsider.com/bi-intelligence-34-billion-connected-devices-2020-2015-11. (2015).
[7] Mauro Caporuscio, Marco Funaro, Carlo Ghezzi, and Valérie Issarny. 2014. ubiREST: A restful service-oriented middleware for ubiquitous networking. In *Advanced Web Services*. Springer, 475–500.
[8] Vint Cerf. 2017. Private communication. (2017).
[9] Arthur D Chapman et al. 2009. Numbers of living species in Australia and the world. *Department of the Environment, Water, Heritage and the Arts Canberra* (2009).
[10] IoT developer survey. 2017. https://www.slideshare.net/IanSkerrett/iot-developer-survey-2017. (2017).
[11] Witold Drytkiewicz, Ilja Radusch, Stefan Arbanowski, and Radu Popescu-Zeletin. 2004. pREST: A REST-based protocol for pervasive systems. In *2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems*. IEEE, 340–348.
[12] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. 2004. Contiki-A lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*. IEEE, 455–462.
[13] Roy T Fielding and Richard N Taylor. 2002. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)* 2, 2 (2002), 115–150.
[14] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes. 2016. Operating systems for low-end devices in the internet of things: a survey. *IEEE Internet of Things Journal* 3, 5 (2016), 720–734.
[15] Chen-Da Hou, Dong Li, Jie-Fan Qiu, Hai-Long Shi, and Li Cui. 2014. SeaHttp: A resource-oriented protocol to extend REST style for Web of Things. *Journal of Computer Science and Technology* 29, 2 (2014), 205–215.
[16] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. 2007. The evolution of Lua. In *the third ACM SIGPLAN conference on History of programming languages*. ACM, 2–1–2–26.
[17] Zephyr project. 2017. https://www.zephyrproject.org/doc/index.html. (2017).
[18] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.
[19] Rob Toulson and Tim Wilmshurst. 2016. *Fast and effective embedded systems design: Applying the ARM mbed.* Newnes.
[20] Zhi-Wei Xu. 2014. Cloud-sea computing systems: Towards thousand-fold improvement in performance per watt for the coming zettabyte era. *Journal of Computer Science and Technology* 29, 2 (2014), 177.