IET Cyber-Physical Systems: Theory & Applications

Review Article

Designing CPS/IoT applications for smart buildings and cities

ISSN 2398-3396
Received on 21st October 2016
Revised on 4th November 2016
Accepted on 6th November 2016
doi: 10.1049/iet-cps.2016.0025
www.ietdl.org

Chi-Sheng Shih[1] ✉, Jyun-Jhe Chou[2], Niels Reijers[2], Tei-Wei Kuo[2,3]

[1]*Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan*
[2]*Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan*
[3]*Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan*
✉ *E-mail: cshih@csie.ntu.edu.tw*

Abstract: Internet of Things (IoT) and cyber-physical systems (CPS) technologies can be applied to many application domains. Examples include intelligent green house, intelligent transportation system, power distribution grid, smart home, smart building, and smart city. Among these application domains, some of them have been extensively studied, e.g., smart home and intelligent transportation systems. In the meantime, smart buildings and smart cities attract researchers and industries to investigate these two use scenarios. Well-designed IoT/CPS can reduce energy consumption, enhance safety in buildings and cities, or can increase the comfortability in the building. In the last few years, the research communities and industrial partners started to study and investigate these two use scenarios to develop prototype or commercial services for these two scenarios. Although many works have been conducted on these two scenarios, many challenges remain open. In this study, the authors study the development and challenges in five topics. They are middleware, computation model, fault tolerance, quality of data, and virtual run-time environment.

1 Motivation

Cyber-physical systems (CPS) target on integrating computing services and physical systems to provide coherent and intelligent services. In the meantime, CPS are distributed and hybrid real-time dynamic systems, with many loops of different degree of application criticality operating at different time and space scales. In the last decades, both the industry and research communities have been actively working on this area. In 2011, Germany announced that CPS are the major targets for its 'Industry 4.0' project for investment and industry development and wishes to the be leader on CPS before 2020 [1]. In 2013, Intel Inc. announced [2] that it will devote its resources to develop the platforms for Internet of Things (IoT) and CPS. National Institute of Standard and Technology (NIST) in the USA also published three reports in a row to address the strategic research and development opportunity for CPS [3]. In 2013 and 2014, there are more than 20 international conferences and journals targeting related research issues for CPS. Cyber-physical system week (CPSWeek) and embedded systems week (ESWeek) are two major conferences and attract more than 500 participants in the conferences. The number of researchers also published papers, reports, and books, e.g. [4–6], on trend and research opportunities on CPS.

Although there are many efforts on developing CPS, many reports expect that the CPS will not be available for general public within five or ten years. One well-known example is autonomous vehicles developed by Tesla and major car manufactures around the world. Having the similar concept of integrating computing and physical devices, IoT applications are available to general public on the market any time soon. Many market survey reports including the ones published by CISCO [7] and IDC [8] estimate that the number of IoT devices will be more than number of populations in the world in a few years. Today, many of us may need to manage three to seven digital devices and the total number of devices connected to the network will grow to 50 billion in 2020. Among the 50 billion devices, large number of the devices will be installed in the factories and commercial/residential buildings, on the vehicles, and on the street. Consequently, how to manage these devices to assure that most, if not all, of them function correctly to conduct the deployed services is a fundamental challenge for IoT applications. In particular, most of the users for these devices do not have the training and are not willing to be trained to manually configure the services on these devices. The IoT services should be managed with the least amount of user efforts. Without losing generality, we refer CPS/IoT to both of CPS and IoT systems in this proposal although there are significant differences among their system architectures. The goal of this paper is to study the new challenges, on-going researches, and open problems to resolved.

Fig. 1 shows four example CPS/IoT systems, which are experiencing new challenges. These four examples are all mission critical, where the system cannot be rolled back after the system states have been changed. Figs. 1*a* and *b* show two traffic control systems, where one is the air traffic control systems (ATCS) and the other one is the intelligent vehicular traffic control system. ATCS is mission critical in the sense if a decision has been delayed or wrong, the change to the system cannot be reversed and recovered. A wrong decision may cause either collision of airplanes or delay on the schedule. On 12 December 2014, a glitch on ATCS, which started to operate in 2012, caused 1300 flights to be cancelled and many more to be delayed at Heathrow Airport [9]. Furthermore, knock-on effect caused more delays on nearby airports. There is no report on the disruption at the time of this writing. In this example, the travellers are physical systems. Their travel schedules are changed and cannot be fully recovered. The airplanes are commanded by ATCS and always wait for the commands to continue. While the airplanes in the system can be grounded and waits for safety concerns, but the individual travel schedules cannot. An intelligent vehicular traffic control system controls the traffic lights using the traffic data collected on the roads. In such systems, the traffic lights remain unchanged when there are no new commands and are parts of a computing system. On the other hand, vehicles on the road are controlled by their drivers and are physical systems. A wrong command for traffic lights may cause collisions on intersections or delay to driver's schedules. Hence, both the ATCS and the intelligent vehicular traffic control systems are mission critical CPS/IoT.

Figs. 1*c* and *d* show two weather control systems, which maintain the temperatures within a given range. Specifically, the temperatures

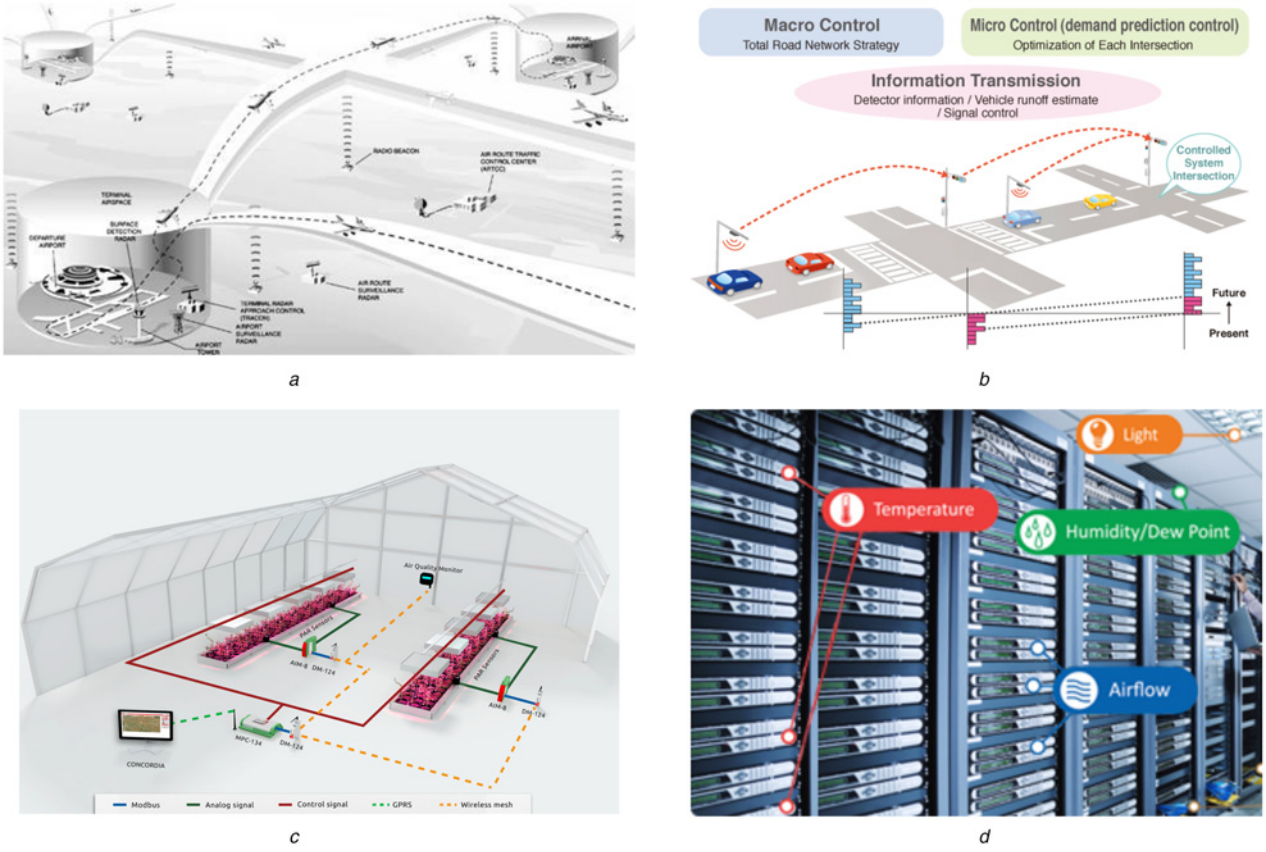

Fig. 1 *Examples for mission critical CPS and IoT*

a ATCS
b Intelligent traffic control systems
c Intelligent Greenhouse
d Temperature control in server room

in Intelligent Greenhouses shown in Fig. 1*c* have been controlled in the range of few degrees and that in Server room shown in Fig. 1*d* are limited up to 27°C or 80°F [10]. A wrong decision, caused by incorrect temperature readings, in Intelligent Greenhouse can damage the plants in greenhouses, such as orchid, and the damage to plants are usually not recoverable; similarly, a wrong decision in Server room may shut down the servers, which leads to financial loss to the data centre such as compensation requests from the customers.

Smart buildings and smart cities are two representative use scenario for IoT/CPS. Well-designed IoT/CPS can reduce energy consumption, enhance safety in buildings and city, or can increase the comfortability in the building. In the last few years, the research communities and industrial partners started to study and investigate these two use scenarios to develop prototype or commercial services [11–18] for these two scenarios. Although many works have been conducted on these two scenarios, many challenges remain open. In this paper, we study the development and challenges in five topics. They are middleware, computation model, fault tolerance, quality of data, and virtual run-time environment.

Middleware for service development and management: To eliminate the burden of service management, many of CPS/IoT service providers chose to treat CPS/IoT end devices as *dumb* devices, which have no or little computation capability, and deploy most, if not all, of the services to the servers on the Internet. Consequently, the data collected on end devices are transmitted to the servers via a communication gateway, and the control commands are downloaded from the server via the communication gateway. It is clear that there is a need for better middleware support to ease the manageability of CPS/IoT applications. One missing block is a *flexible middleware support* so that developers and users of a CPS/IoT system do not need to be constrained by which and how sensors have been deployed in a target environment. The built-in intelligence from the middleware can dynamically perform sensor detection, device selection, system configuration, software deployment, and system reconfiguration.

Quality of data: Quality of data is the second challenge for computing services in the CPS/IoT smart building and smart cities. CPS/IoT systems integrate two types of systems which have distinguished assumptions on how the system behaves. One significant difference is the lifetime of the data. Data in computing systems are available until being updated or deleted; data in physical systems are valid at sensed time and will not be stored unless specified to do so. Without proper representation of the quality of the data, including accuracy, lifetime, location, and sample time, it is very likely that the computing processes will misinterpret or misuse the data collected from physical processes. In the meantime, the computing processes may assume the received data meet the certain quality, including accuracy, freshness, correctness, and reliability. Hence, the CPS/IoT needs to assure that the collected data meet the requirements specified by computing processes if any.

Service composition and computing models: The goal of deploying sensing devices in the building and cities is to collect data and response passively, actively, or even pro-actively. Hence, how to process collected data is the fundamental design decision: either centralised or distributed model, either on edge devices or on cloud devices. SmartThings is an example of using centralised model and cloud model. This computing model has its own merits: the users do not need to manage the services, including deployment and update, and service updates are only conducted on the servers, which are exclusively managed by the service providers. Unfortunately, the approaches have several significant

IET Cyber-Phys. Syst., Theory Appl., 2016, Vol. 1, Iss. 1, pp. 3–12

disadvantages: one is the **scalability** and the other one is **responsiveness**. Hence, another approach is to use distributed computing model, in which the services can be conducted on the devices in the system and may not require Internet connection. Developing and executing distributed applications are not straightforward in all domains in computing societies, including enterprise computing, desktop computing, and embedded systems. The challenges include algorithm design, delay tolerance, and data transmission among computing parties.

Fault tolerance: Fault tolerance is the forth challenges for the aforementioned IoT application deployed in real world, because a users' life may be disrupted or even endangered due to an IoT device failure. There have been many works on fault tolerance for distributed systems. However, for IoT/CPS, the heterogeneity on computation and communication capability makes it not trivial to detect faults.

Different from wireless sensor networks [19, 20] where single-purpose devices are embedded in the physical world, IoT devices for smart buildings and smart cities are often loaded with multiple sensors and actuators to support multi-purpose applications. IoT/CPS may have heterogeneous sensors to be used by different applications at different times. Meanwhile, each IoT application may use a subset of sensor devices according to its needs. This brings innate heterogeneity and constant evolution to IoT devices and poses challenges [21] on their management. On the one hand, large streams of data collected by various sensors enable a better precision for recognising the context, and supporting real-time adaptation according to the new context. On the other hand, relatively frequent failures of individual sensors may persist and must be coped with extra care. In any fault-tolerant mechanism, efficient fault monitoring is a necessary first step. Our study presented in this paper is to design fault monitoring clusters in IoT systems, with the goal of minimising the overall monitoring traffic. In IoT, communication overhead is a critical factor due to the energy capacity constraint on many devices.

Virtual run-time environment: Heterogeneity on IoT hardware platforms provides extensive choices on selecting the suitable platform for different application domains and contexts. In the meantime, it also makes it difficult to port IoT/CPS services to different platforms. There are several advantages to running a virtual machine (VM), even on resource-limited IoT devices. One is ease of programming. Many VMs allow the developer to write programs at a higher level of abstraction than the bare-metal C programming that is still common for these devices. Second, a VM can offer a safe execution environment, preventing buggy or malicious code from disabling the whole device. A third advantage is platform independence. While early wireless sensor network applications often consisted of homogeneous devices, current IoT/CPS applications are expected to run on a range of different platforms. A VM can significantly ease the deployment of applications.

While current VMs offer an impressive set of features, almost all sacrifice performance. All interpreters for which we have found concrete performance data are between one and two orders of magnitude slower than native code. In many scenarios, this may not be acceptable for two reasons: for many tasks such as periodic sensing there is a hard limit on the amount of time that can be spent on each measurement, and an application may not be able to tolerate a slowdown of 10–100 times. Second, and perhaps more importantly, one of the main reasons for using resource-limited devices in the first place is their extremely low power consumption which allows them to last for months or even years on battery power. Often, the CPU will be in sleep mode most of the time, so little energy is being spent in the CPU compared with communication, sensors, or actuators. However, if the slowdown incurred as a result of using a VM means the CPU has to be active 10–100 times longer, this may suddenly become the dominant factor.

The reminder of this paper is organised to discuss the challenges, on-going researches, and open problem to be resolved on five topics. Sections 2 and 3 will discuss middleware and virtual run-time environment. Sections 4 and 5 will discuss quality of data and computing model. Section 6 summarises the paper.

2 Middleware for service development and management

Without loss of generality, middleware refers to the software to bridge multiple services together to conduct coherence functions. Hence, operating systems are middleware and run-time environment on computing systems are also middleware. In this paper, middleware for CPS/IoT represents the environment to develop, deploy, execute and manage services for CPS/IoT.

2.1 Middleware for CPS/IoT

Zhang *et al.* [22] proposed a *reconfigurable real-time middleware for distributed CPS with aperiodic events* in 2008. This work targeted the flexible workflows in CPS. Different distributed CPS must handle a periodic and periodic events with diverse requirements. While existing real-time middleware such as real-time CORBA has shown promise as a platform for distributed systems with time constraints, it lacks flexible configuration mechanisms needed to manage end-to-end timing easily for a wide range of different CPS with both aperiodic and periodic events. The primary contribution of this work is the design, implementation and performance evaluation of the first configurable component middleware services for admission control and load balancing of a periodic and periodic event handling in distributed CPS. Empirical results demonstrate the need for, and the effectiveness of, our configurable component middleware approach in supporting different applications with a periodic and periodic events, and providing a flexible software platform for distributed CPS with end-to-end timing constraints.

WuKong [23] is an intelligent middleware between applications and IoT devices. It supports the following features: The first is to recognise heterogeneous devices in connection, user context and needs. The second is to configure and transform devices into service components. The third is to adapt and distribute application codes to achieve the result. The final is to conduct all of the above via remote access to sensors. Another goal is to use the WuKong middleware as the foundation of next generation IoT application development for constructing high level applications. WuKong middleware includes a light weight JVM. JVM allows developers to add application specific behaviour onto the functionalities already in the sensor/actuator hardware.

WuKong middleware consists of two major components to fill the gap for developing and managing IoT applications: *intelligent middleware* and *flow-based developing framework*. WuKong middleware is regarded as a Virtual middleware (VMW). The reasons for this are two-folds. First, as sensor networks become widely available, it is very likely that applications have to use sensors developed by different manufactures and communicating with different network protocols. Having a virtual sensor will allow applications to run on heterogeneous networks of sensors. Second, when the system decides to reconfigure the network, the process of reprogramming nodes will be less expensive by using a virtual machine design so that line of codes will be less since the virtual sensor can offer higher level primitives specific for IoT applications. On top of WuKong middleware, WuKong framework can postpone binding logical components with physical devices until an application is deployed, rather than when an application is developed. With the intelligent middleware and FBP, WuKong framework enables intelligent binding for IoT systems.

2.2 Open challenges on middleware design for smart buildings and smart cities

Middleware presented above provide the development and runtime environment of number of IoT applications. There are several challenges remained open:

- *Privacy control:* IoT for smart buildings and smart cities collect users' behaviours with or without identity. How the collected data are owned, published, stored, and used are critical concerns for the

residence in the buildings and cities. The middleware itself should provide a mechanism to protect the privacy and to control the data flows to meet privacy requirements specified by developers or users. The privacy control policy defined in middleware has to be conducted on CPS/IoT systems. Hence, a machine-readable representation for privacy control is missing and should be defined to be followed by all participant devices in the system.

- *Access control:* Compared to privacy control, access control policy defines the permission for services or devices to access data. Data collected in smart buildings and smart cities have location and temporal information. Hence, access control should take into account not only the services collecting the data, but also other properties, including location and time, to authorise the access to data.
- *Scalability:* In buildings and cities, the devices and services deployed can be more than thousands or hundreds thousands in one building and one city. It is not feasible to design services in this scale to be a single service with complex communication links among service components and with complex network connectivity among devices. Hence, on middleware, it is desirable to design single service for devices of the same types and to deploy the service to all the devices in the building and city. The links among services should be connected by the middleware according to different rules. One example can be intelligent smoke detector which provides the correct evacuation direction according to the data collected by other smoke detectors in the same building. The data flow links among smoke detectors are naturally location dependent. Hence, the middleware can automatically link the devices deployed in proximity to exchange collected data.
- *Service management:* Managing services conducted on thousands or hundreds of thousands devices in a single system are a challenging job. In addition, the devices are very likely to be heterogeneous to conduct same or similar services. Many service management tools are designed to manage computing systems with full capability, including network protocol to support remote management and storage systems to log the operations and events on the managed devices. In CPS/IoT systems, not all the devices have the aforementioned capabilities. Some devices have limited communication capability; some devices can only be accessed at certain time intervals due to limited energy capacity. CPS/IoT middleware for smart building and smart city should be able to remote monitor and manage the services deployed to the devices. Hence, when there are faults, either hardware or software, the failed services can be identified and replaced in short time; when the configuration or requirement changes, the services can be reconfigured or replaced in minimum amount of management overhead.

3 Quality of data

3.1 Background and related works

One of the challenges on implementing CPS/IoT is to integrate two types of systems which have distinguished assumptions on how the system behaves. The major difference is the lifetime of the data. In computing systems, the data have no attributes to describe its temporal properties such as created time, sampled time, and expiration time. In the meantime, computing systems assume that the data are created at received times and valid until deleted or updated. However, in physical systems, the data are stochastic and changes whenever time goes by. Therefore, the data are only valid at sampling time and invalid hereafter.

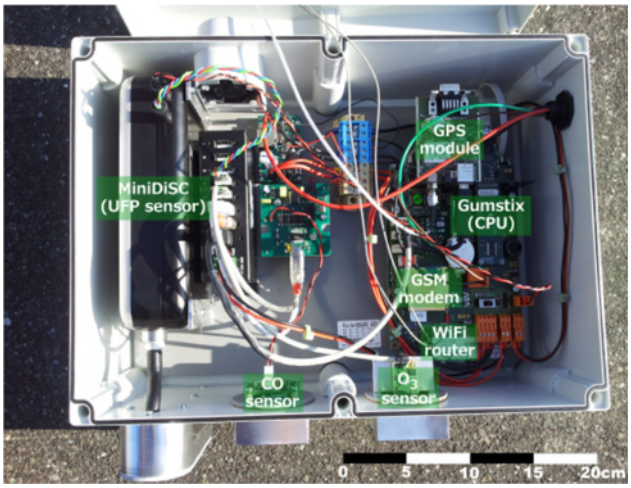

Fig. 2 *Air quality sensors installed on trolley cars in Zurich [25]*

In the four examples shown in Fig. 1, one may notice that irrecoverable damage occurs if the computing systems do not interact with physical systems properly. Use autonomous computer-controlled and human-controlled vehicles as example to illustrate the above claim. Assume that an autonomous vehicle receives the traffic control signal via visual sensors and drives during a snow storm. When the vehicle approaches the intersection, it receives a green light, but loses the vision to the traffic light afterward. Unfortunately, the light turns into the red before pre-programmed interval due to an approaching emergency vehicle in different directions. If the vehicle keeps the green signal and continues to move, it is very likely that the two vehicles will collide into each other at intersection. On the other hand, when a human, a physical system, drives the vehicle to approach the intersection in poor vision, the driver is request to slow down and check the traffic at all directions before moving on. Hence, the accident can be avoided. These examples clearly show how the two types of system handle data differently.

One naive solution for the aforementioned challenge is assigning an expiration time for each data and invalidating all expired data. Then, the computing systems can be programmed to reject expired data, which may be incorrect, and hold when any required data is not available. Although this approach assures that the systems do not make mistakes, it is very likely that the system will stop and go due to a large number of false alarms on incorrect data due to transient communication delay, or transient faults on data sampling.

In CPS/IoT applications, it is very common to use multiple sensors, either heterogeneous or homogeneous, in an application. All the applications shown in Fig. 1 have multiple sensors. Our observations show that when the correlation model among sensor readings is found, it is feasible to use the readings from different sensors to calibrate or verify the readings when any of data are missing. Below is an example of temperature readings from sensors installed on urban trolley car [24, 25]. In this example, an air pollution sensor box, as shown in Fig. 2, is installed on top of trolley cars to measure the air quality and temperature.

There are two temperature sensors in the box: one is installed outside of the box to measure outdoor temperature and the other is installed inside the box to measure CPU temperature. Usually, only the reading for outdoor temperature is reported and the CPU temperature is used internally to prevent the CPU from overheating. Fig. 3*a* shows the readings of these two temperature sensors over a short period of time. In the figure, one can find that the readings for outdoor temperatures are missing from time to time. This may be caused by communication disruptions, transient faulty sensors, or transient error readings caused by the objects covering the sensors.

When observing the readings from two sensors, one may find that there is a strong correlation between them. If the correlation model is established, the reading from CPU temperature sensors can be used to approximate outdoor temperature when the outdoor temperature sensor fails to report reading. Fig. 3*b* shows the results of the approximated temperature. The blue round points show the approximated outdoor temperatures, which mostly overlap with measured outdoor temperatures, if exists, represented by green crossing points.

Slijepcevic *et al.* [26] published the study on the impact on application outcomes while the accuracy of location data varies. This paper provides a comprehensive understanding on sensor errors. In particular, the sensing errors for location discovery and systematic-error are discussed. Systematic errors are difficult to be corrected. This paper does not intend to correct the errors, but aims on understanding the impact of faulty data to the applications. The finding from the paper includes (a) systematic

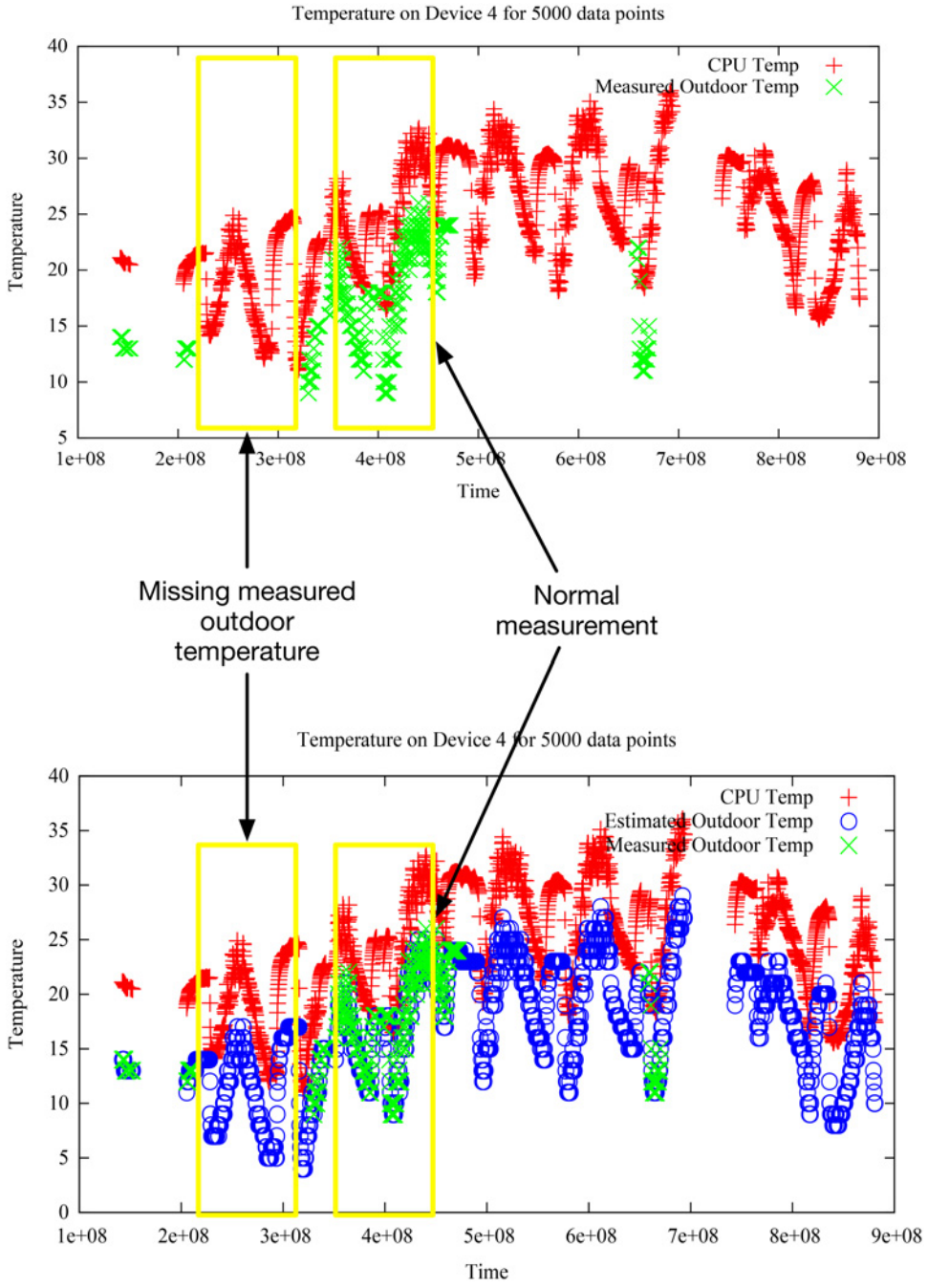

Fig. 3 *Temperature readings on trolley cars in Zurich [24]*

errors can be modelled, (b) the computation results, derived from faulty sensing data, can be very significant. This paper also raises several open issues. One issue is if different sensors deployed on different devices can be used to remove the error. One example in this paper is using camera to verify if the acoustic sensors are blocked by any obstacle. Acoustic sensors have better accuracy than radio strength sensors. However, acoustic sensors require light of sights. When the two sampled values are different from each other, camera can be used to figure out which data point has better accuracy than the other one.

Hasenfratz *et al.* [24, 25, 27] studied the method to enhance the accuracy of data, including scheduling the time instances to calibrate sensors with reference sensor station via single-hop calibration, determining the multi-hop calibration based on the routes of urban trolley car, and visualise the data. The family of works provide the solid foundation on removing non-sampling errors in CPS/IoT applications. When calibration is required and is possible, the proposed method can greatly enhance the quality of data.

Noguero *et al.* [28] proposed a *time-triggered middleware architecture for ubiquitous CPS applications*. This work focuses on CPS that requires (i) the timely execution of their activities in an accurate way and (ii) flexibility to reconfigure the systems dynamically at run-time. Frequently, these systems require synchronous or cascaded measurement and actuation of several devices, such as outdoor sensors, cameras and robots. Examples of these systems may be found in different domains that range from industrial applications to home automation. In some cases, real-time requirements are not only applicable to the application tasks themselves, but also to the messages exchanged among them, needing some kind of network management. In addition, the management of other resources, such as memory, CPU usage or memory may become a phenomenal task that augments time to market of the final application without providing any added value

in terms of functionality. More specifically, the authors propose a time-triggered middleware architecture that allows developers to focus on the functionality of the applications liberating programmers from synchronisation and resource management issues. This architecture, which is capable of addressing dynamic reconfiguration of the applications at run-time, has been specially designed for soft real-time applications at a variety of application domains from industrial applications to home automation. Different types of data traffic requiring different priorities may be involved, such as sensor data, video streams and alarm messages. Finally, the proposed middleware architecture is also capable of scheduling the execution of a set of replicated tasks and optimising the use of the resources of the distributed system in order to adapt to changes in the functionality and cope with temporary faults at run-time.

3.2 Challenges on quality of data

The above observations motivate our research on answering three fundamental scientific questions:

- **What are the causes of invalid data?** In computing systems, the value of a variable is valid until modified or deleted. For single task/process systems, it is straight forward to identify the time instances at which a variable being manipulated. For multiple task/process/thread systems, concurrent execution makes it difficult to identify the time stances at which a variable being manipulated. In CPS/IoT, a variable can be defined to describe the status of physical objects, which may change at any time without notice. As a result, the value of the variable, which is data, may become invalid at any time without notice. In this part of the research, we will study the causes of invalid data in mission critical CPS/IoT systems, categorise the causes, and define the property of data.
- **How to present the quality of data?** Due to various causes of invalid data, it is not straight forward to present the quality of data. This section aims on the presentation for the data with temporal properties in mission critical CPS/IoT. Few parameters will be added to the definition of the variables. For instance, the minimal arrival interval, valid interval, and criticality of the data, minimal accuracy requirement of the data and so on. For knowing the outdoor temperature, there is no need to use a sensor at one tenth of one hundreds degree. The representation should also be comparable [http://en.wikipedia.org/wiki/Comparability]. It will be used to check if current data meet the requirement or the data collected by other devices meet the requirement.

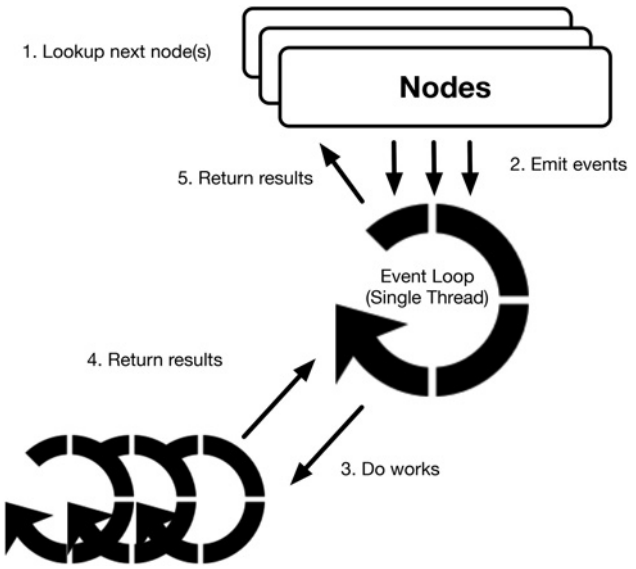

Fig. 4 *Software architecture of node-RED [30]*

- **How to assure the quality of data for computing systems?** It is discussed above that quality of sampled data can be improved in certain cases. The section aims on designing a framework on reducing sampling errors and non-sampling errors so as to assure the quality of data. There could be two approaches to prevent the failure: one is to avoid the failure and the other one is to handle the failure. In many cases, the quality of data can be improved by increasing the sampling frequencies or adding additional sensors. The issue is how to select the proper method so that the method will not over-consume the resources in the system. It also raises the questions on how to schedule the sampling or to calibrate the sensor in advance. The second approach is to handle the failure. One possible solution is to derive the data from other end devices or data. This can be done in dynamically remapping the data stream to the applications. It requires the middleware to learn the correlation among data and assign the proper one to replace the missed one. This paper will focus on the second approach.

4 Distributed computation models

4.1 Background and related works

Data collection and intelligent decision are two fundamental factors of success IoT applications or networked applications in general. Many existing IoT applications use cloud services as the main host of collected data and intelligent decision. Examples are SmartThings, IFTTT, Amazon Echo, and Google Home.

Using cloud services as the host of intelligent IoT applications have its own merits. The service providers have the full control to the collected data and decision logics. It also reduces the overhead of managing the services, including revising, updating, and revoking services. Unfortunately, the aforementioned cloud-based service model limits users' capability on configuring and controlling the services. Revolv market their Smart Home hubs, which lets home users control various appliances and home automation systems with a central application. The company was acquired by Nest in October 2014 and a subsidiary of Alphabet. In February 2016, Revolv and Nest [29] announced that all Revolv services will be terminated, starting at 15 May. Although the owners of Revolv hubs may receive refunds for their hardware devices, all the smart home services have to be migrated, if possible, or will stop functioning. The aforementioned example uses cloud-based service model and limits the users on controlling the services.

Node-RED is a visual tool using flow-based programming model to connect the number of Node-RED compatible services into a connected flow. Each flow consists of a number of nodes, each of which is a predefined service, and the nodes are connected by directed link to represent the data flow from one node to another one.

Fig. 4 shows the runtime architecture for Node-RED.

Each Node-RED flow is monitored by an event loop, which is a single thread process, to check the emitted events (Step 2) from the nodes. After receiving the emitted events, the event loop triggers the receiving node to do the works (Step 3). After accomplishing the work, the node return results back to the event loop (Step 4) and to the nodes (Step 5).

As shown in Fig. 4, Node-RED is a centralised flow-based programming framework. The flow consisting of a number of nodes has to be hosted by a single thread process. Each node emits event to and receive event from the event loop component. Hence, all the events in the flow are received and forwarded by the event loop component.

One limitation of Node-RED is the communication among nodes in different flows. The links between nodes are limited to the nodes in same flow. Hence, the nodes in different flows have to communicate with each other using external communication protocols, such as RESTFul API. When there is a need to exchange data among the nodes in different flows, the limitation introduces additional overhead on communication and makes it difficult to validate the interaction among flows.

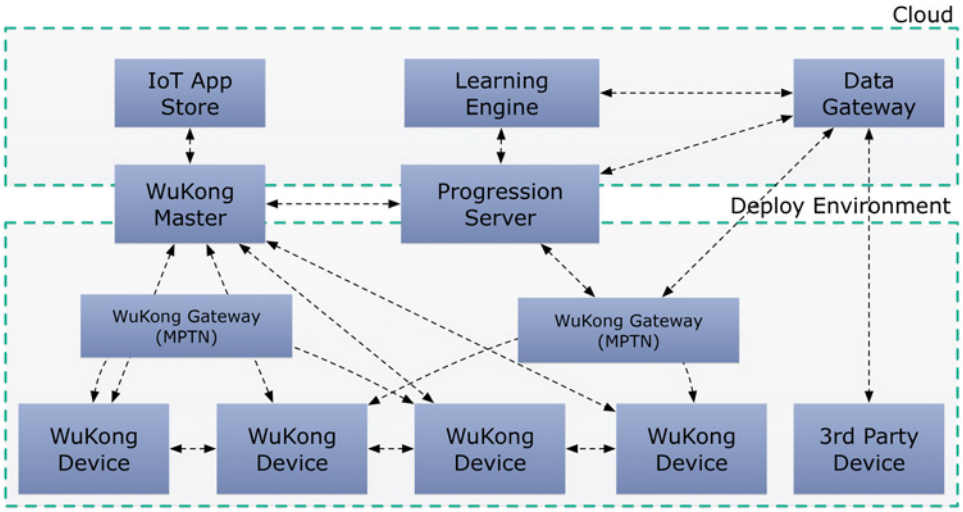

Fig. 5 *System architecture of WuKong*

Blackstock and Lea [30] extended Node-RED framework to support distributed data flow services. This work aims on automating the distribution of data flows using appropriate distribution mechanism, and optimisation heuristics based on participating resource capabilities and constraints imposed by the developer. Similar to NodeRED, the work only includes web-based or server-based services in the flow. Services deployed on resource limited devices are not supported in this work. In addition, the privacy concern is still open. Khan and Kim [31] developed a DIY interface to customise remote IoT device using CoAP interface. The interface allows remote IoT devices to be customised remotely. Although this work can federate the services on resource-limited devices and cloud-based servers, the privacy issue remains.

WuKong, presented in Section 2, is a distributed computing runtime to accomplish the requests from users and applications. Fig. 5 shows the system architecture of WuKong middleware. WuKong systems consist of the components on the cloud and that in deployment environment. On the cloud server, WuKong deploys IoT application store, learning engines to learn users' behaviours/contexts, and data gateway to archive historical data. In deployment environment, WuKong deploys one WuKong master (called WuMaster for short), several WuKong gateways, and number of WuKong devices.

4.2 Challenges on computation models for IoT

Right computation model for CPS/IoT systems highly depends the application domains and requirements. For a system in the scale of buildings and cities, it is not likely to use single computation model to conduct all the services in the system. Hence, the challenging issues are how to federate computing services in different computation models.

4.2.1 Service impedance: public and private services: IoT applications consist of sensed data, intelligent decision logic, and actuation. Sensed data and actuation are conducted on physical devices; intelligent decision logic can be conducted on either remote or local computing devices. Intelligent decision logics are the core of IoT applications and are the targets of this work. Intelligent decision logics can be classified into two types: *public services* and *private services*. Public and private services can be distinguished from their (i) *availability*, (ii) *access model*, and (iii) *data sources* for decision logic. In the following, we define the three criteria and two types of decision services.

The *availability* of decision logics refers to the time instances to access the services. Public services are hosted on computing devices available at all time and can be accessed with proper authentication protocols. Examples of public services are cloud-based services, including SmartThings and Amazon echo. On the other hand, private services are hosted on resource-limited computing devices and are not available at all time. Resource-limited computing devices are usually deployed on site and may be put into sleep or hibernation from time to time to reduce energy consumption. Hence, the services are available at all time.

Access model of decision logics refers to the openness of access interface. Public services can be accessed by open communication interface. As long as the name of protocol, address of the services, and authentication configuration are available, the public services can be accessed by any service component. On the other hand, private services can only be accessed by pre-defined and limited communication interface. In many cases, the services are accessible by wired interface.

Data sources for decision logic refer to the sources of the data. Public services make decision based on public data such as traffic flow, bus schedule, and weather forecast. On the other hand, private services make decision based on local data such as the brand and model number of installed air conditioner, user preference on public transportation, and the statistic of the occupancy of a particular parking lot.

The above three factors distinguish the services into public services and private services. It is nature to deploy public services on the cloud to provide high availability, open access, and public data services. However, it may not be intuitive to deploy private services on the cloud. In addition, public services are usually shared by a large number of users; private services are usually customised for specific users or applications. Hence, it is not feasible to deploy private services to the cloud.

4.2.2 Federating public and private services: The service model of public service and private service differ in several aspects. The framework should provide a federation interface to bridge public services and private services. The method should not only integrate these two services, but also enable local intelligence to compose IoT applications. Hence, public services are composed by centralised service composition tool and private services are composed by distributed service composition tool.

5 Device clustering for fault monitoring in IoT systems

The goal of fault monitoring in IoT is for locating faults or anomalies on devices, and identifying affected services so that these services can be replaced during fault recovery. The easiest way for fault

monitoring is to use a central controller. The controller sends polls to devices periodically to ask for their status. If any abnormal behaviour is detected, or if a device does not reply, there may be a fault present. The central controller will then consider which services may be at fault based on what services are deployed on that device.

This simple mechanism may face the scalability issue. First, as more devices join a network, the controller has to take care of more fault tolerance combinations. Second, with the increasing number of devices, the time for sending and receiving polls also increases. Finally, all monitoring related messages going to the controller may create bottlenecks at the controller and devices around it. This will not only waste the communication resource and energy, but also affect the performance of active applications running on these devices.

Therefore, instead of only one controller, we divide IoT devices into clusters for monitoring. Monitoring in conducted only by each cluster separately. Each cluster is managed by a coordinator, or cluster head (CH), for efficiency. A CH is used to keep the backup information for each device in a cluster. When a fault happens, some device in the cluster will detect the fault and will report to CH. CH will then replace a faulty service with backup ones.

In IoT, an event-trigger communication pattern may be desired in many cases to save the communication cost. Event-triggered means a message is sent only when a significant change in the sensed value is detected. However, this also means that if no message is received, a receiving device cannot tell whether nothing has happened or the sender has failed. For fault detection in event-triggered communication, periodic beacons are often used to let the beacon receivers know that the senders are still healthy.

To save communication cost and to divide the load of a CH, we use the loop topology for sending beacon messages. We show the differences between the loop topology and the commonly used star topology in Fig. 1. The star topology expects CH to send and receive all beacon or ack messages. However, for a loop topology, the work is divided by all devices in the cluster. As long as the beacon can be passed down through the loop, there is no need for reporting back. This greatly reduces the number of packets used for monitoring in a loop topology cluster. Considering CH may need more than one hop to reach other devices in a cluster, the loop topology saves more hops than the star topology.

However, one main challenge of the loop topology is that any fault in it may interrupt future beacon message propagation. To handle this, two timers are used to measure beacon period for each device: one for controlling the pace of beacon message sending, one for message receiving time out. Depending on different application requirements, different devices could have different sending and receiving periods. If no beacon message is received for a whole beacon period, the receiver timer should trigger the device to report to CH about the abnormal receiving.

Another challenge of the loop topology is that a big cluster may result in extra management overheads and scalability concerns, while a small cluster may limit the capability of CH to coordinate the message sending among nodes. Thus, for scalability and load-balancing reasons, we have a lower bound (LB) and an upper bound (UB) on the number of nodes a cluster can have. LB is dependent on the fault-tolerance requirement of all applications in the system. If there may be two nodes fail simultaneously, the size of a cluster should be at least three, and there should be two backups for CH. UB is chosen based on the maximum number of nodes in a space (e.g. a room). Putting nodes from different spaces into one cluster will bring extra monitoring cost and is undesired. Setting a UB can also help limit the search space for the monitoring solution.

Clustering is widely used in sensor network to deal with the inherently lack of structure in mesh networks, for fault-tolerance as well as other purposes like energy efficiency, communication efficiency [32]. One widely used model to solve fault-tolerant clustering problem is dominating set clustering [33–35]. This technique tries to find a dominate set within the network, thus every node in the network is within k hops to the nodes in the dominating set. One well-known work based on this model is LEACH [36]. LEACH proposes a distributed algorithm in which a cluster head is elected based on their energy level among neighbours, and nodes join different clusters according to their communication cost to those cluster heads. Cluster heads could be rotated as energy changes in order to help load balance and fault tolerance of the network. All these works care more about clustering, but not the organisation of nodes within a cluster, which means a star topology is assumed, resulting in extra burden in cluster head at run time.

Most of the work we surveyed complied with a distributed pattern. The benefit of distributed algorithm mainly resides on that it could better adapt to the mobility of nodes in wireless *ad-hoc* networks, and has a better scalability [34, 37]. However, for smart home or office scenarios, only limited mobility for devices is required. Once we deploy nodes, we rarely move them. Location tracking may have moving objects; but in most cases, sensors detecting human badges or moving objects are set at fixed locations. The benefit of a centralised algorithm is that it can get more information and be more powerful. Among the limited centralised algorithms, Gupta and Younis [38] try to achieve fault tolerance on clustering by assigning nodes to some existing gateway nodes through a heuristic algorithm. This aims more at partitioning devices. Ilker Oyman and Ersoy [39] use k-means to set the sinks in multiple-sink networks considering energy. They try to find the minimum number of sinks while maximising network lifetime. Our work aims more at finding the sinks or CHs of a cluster. We formulate a centralised clustering problem, which tries to solve both the CH location problem and the node assignment problem at the same time.

6 Virtual run-time environment

6.1 Background and related works

Since the beginning of the sensor networks field, many different VMs have been proposed that are small enough to fit on a resource-constrained sensor node. They can be roughly split into two categories: generic VMs and application-specific VMs, or ASVMs [40].

ASVMs provide specialised instructions for a specific problem domain. One of the first VMs proposed for sensor networks, Mate [41], falls in this category. It provides single instructions for common tasks on a sensor node, so programs can be very short. Unfortunately programs have to be written in a low-level assembly-like language, limiting its target audience. SwissQM [42] is a more traditional VM, based on a subset of the Java VM, but extended with a number of sensor network specific instructions to access sensors and do data aggregation. VM* [43] sits halfway between the ASVM and general approach. It is a Java VM that can be extended with specific new features according to application requirements. Unfortunately, it is closed source.

Several generic VMs have also been developed, allowing the programmer to use general purpose languages like Java, Python, or even LISP [44–47]. The smallest official Java standard is the Connected Device Limited Configuration [48], but since it targets devices with at least a 16 or 32 bit CPU and 160–512 kB of flash memory available, it is still much larger than most sensor nodes. The available sensor node Java VMs all offer some subset of the standard Java functionality, occupying different points in the trade-off between the features they provide, and the resources they require.

6.2 Challenges for virtual run-time environment

The major challenging issue for virtual run-time environment is performed. Of all the papers describing sensor node VMs, only a few contain detailed performance measurements. TinyVM [49] reports a slowdown between 14 and 72× compared with native C, for a set of 9 different benchmarks. DVM [50] has different versions of the same benchmark, where the fully interpreted version is 108 times slower than the fully native version. Ellul and

Martinez report some measurements on the TakaTuka VM [46, 51] where the VM takes 230 times longer than native code, and consumes 150 times as much energy. Finally, Darjeeling [45] reports between 30 and 113× slowdown. Since performance depends on many factors, it is hard to compare these numbers directly. However, the general picture is clear: current interpreters are one to two orders of magnitude slower than native code.

On the desktop VM performance has been studied extensively, but for sensor node VMs this aspect has been mostly ignored. To the best of our knowledge Ahead-of-Time (AOT) compilation on a sensor node has only been tried by Ellul and Martinez [51]. Their results improve performance considerably compared with the interpreters, but there is still a lot to gain. Using our benchmarks, their approach produces code that has a slowdown of 328%, and is 246% larger, compared with optimised native C. While the reduced throughput may be acceptable for some applications, there are two other reasons why it is important to improve on this result: the loss of performance results in an equivalent increase in CPU power consumption, thus limiting battery lifetime. More importantly, the increased size of the compiled code reduces the amount of code we can load onto a node. Given the fact that flash memory is already restricted, this is a major sacrifice to make when adopting AOT on sensor nodes.

7 Summary

Well-designed IoT/CPS systems can reduce energy consumption, enhance safety in buildings and city, or can increase the comfortability in the building. In the last few years, the research communities and industrial partners started to study and investigate these two use scenarios to develop prototype or commercial services for these two scenarios. Although many works have been conducted on these two applications, many challenges remain open. In this paper, we study the development and discuss the challenges in five topics.

8 References

1 *Germany -Lead Market for Cyber-Physical Systems 2020*. 2011. Available at: http://www.gtai.de/GTAI/Navigation/EN/Invest/Industries/Smarter-business/Smart-systems/cyber-physical-systems,did=626020.html (visited on 12/28/2013)

2 *The Internet of Things Starts with Intelligence Inside*. Available at: http://www.intel.com/content/www/us/en/intelligentsystems/iot/internet-of-things-starts-with-intelligence-inside.html

3 Sztipanovits, J., Ying, S., Cohen, I., *et al.*: 'Strategic R&D Opportunities for 21st Century Cyber-Physical Systems – Connecting computation and Information systems with the physical worlds'. Technical Report, National Institute of Standards and Technology (NIST), Rosement, Illionis, 2013, p. 32, Available at: http://www.nist.gov/el/upload/12-Cyber-Physical-Systems020113%5C_final.pdf

4 Sha, L., Gopalakrishnan, S., Liu, X., *et al.*: 'Cyber-physical systems: a new frontier'. Proc. 2008 IEEE Int. Conf. on Sensor Networks, UBIQUITOUS, and Trustworthy Computing, 2008

5 Liu, J., Shih, C.-S., Chu, E.: 'Cyber-physical elements of disaster prepared smart environment'. IEEE Computer Magazine, February 2013

6 Park, S.O., Park, J.H., Jeong, Y.S.: 'An efficient dynamic integration middleware for cyber-physical systems in mobile environments'. Mobile Networks and Applications, 2012, Available at: http://www.springerlink.com/index/T7M1U617U4RUT612.pdf

7 Evans, D.: 'The Internet of Things How the Next Evolution of the Internet Is Changing Everything', http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, Accessed August 30, 2014. April 2011

8 Spencer, L.: 'Internet of Things market to hit $7.1 trillion by 2020', http://www.zdnet.com/internet-of-things-marketto-hit-7-1-trillion-by-2020-idc-7000030236, Accessed August 30, 2014. June 2014

9 *Flights disrupted after computer failure at UK control centre*. 2014. Available at: http://www.bbc.com/news/uk-30454240 (visited on 12/22/2014)

10 Google Inc.: 'Efficiency: How we do it – Data Centers', Available at: http://www.google.com/about/datacenters/efficiency/internal/

11 Stankovic, J.A.: 'Research directions for the internet of things', *IEEE Internet of Things J.*, 2014, **1**, (1), pp. 3–9, doi: 10.1109/JIOT.2014.2312291

12 Kortuem, G., Kawsar, F., Sundramoorthy, V., *et al.*: 'Smart objects as building blocks for the Internet of things', *IEEE Internet Comput*, 2010, **14**, (1), pp. 44–51

13 Zanella, A., Bui, N., Castellani, A., *et al.*: 'Internet of things for smart cities', *IEEE Internet of Things J.*, 2014, **1**, (1), pp. 22–32. doi: 10.1109/JIOT.2014.2306328

14 Nam, T., Pardo, T.A.: 'Smart city as urban innovation: focusing on management, policy, and context'. Proc. of the fifth Int. Conf. on Theory and Practice of Electronic Governance. ICEGOV '11, Tallinn, Estonia, 2011, pp. 185–194. doi: 10.1145/2072069.2072100. Available at: http://doi.acm.org/10.1145/2072069.2072100

15 Nam, T., Pardo, T.A.: 'Conceptualizing smart city with dimensions of technology, people, and institutions'. Proc. 12th Annual Int. Digital Government Research Conf.: Digital Government Innovation in Challenging Times. dg.o '11, College Park, Maryland, USA, 2011, pp. 282–291. doi: 10.1145/2037556.2037602. Available at: http://doi.acm.org/10.1145/2037556.2037602

16 Su, K., Li, J., Fu, H.: 'Smart city and the applications'. Int. Conf. on Electronics, Communications and Control (ICECC), 2011, September 2011, pp. 1028–1031, doi: 10.1109/ICECC.2011.6066743

17 Bowerman, B., Braverman, J., Taylor, J., *et al.*: 'The vision of a smart city'. Second Int. Life Extension Technology Workshop, Paris, 2000, vol. 28

18 Neirotti, P., De Marco, A., Corinna Cagliano, A., *et al.*: 'Current trends in smart city initiatives: some stylised facts', *Cities*, 2014, **38**, pp. 25–36. doi: http://dx.doi.org/10.1016/j.cities.2013.12.010. Available at: http://www.sciencedirect.com/science/article/pii/S0264275113001935

19 Heidemann, J., Govindan, R.: 'Embedded sensor networks', in Hristu-Varsakelis, D., Levine, W.S. (Eds.): 'Handbook of networked and embedded control systems' (Springer, 2005), **7656**, pp. 721–738

20 Koushanfar, F., Potkonjak, M., Sangiovanni-Vincentell, A.: 'Fault tolerance techniques for wireless *ad hoc* sensor networks'. Proc. of IEEE Sensors, 2002, 2002, vol. 2, pp. 1491–1496

21 Sugihara, R., Gupta, R.K.: 'Programming models for sensor networks: a survey', *ACM Trans. Sensor Netw. (TOSN)*, 2008, **4**, (2), p. 8

22 Zhang, Y., Gill, C., Lu, C.: 'Reconfigurable real-time middleware for distributed cyber-physical systems with aperiodic events'. The 28th Int. Conf. on Distributed Computing Systems, 2008, ICDCS '08, 2008, pp. 581–588

23 Reijers, N., Lin, K.J., Wang, Y.C., *et al.*: 'Design of an intelligent middleware for flexible sensor configuration in M2M systems'. Proc. Second Int. Conf. on Sensor Networks (SENSORNETS), February 2013, pp. 1–6

24 Hasenfratz, D., Saukh, O., Walser, C., *et al.*: 'Deriving high-resolution urban air pollution maps using mobile sensor nodes', *Pervasive Mob. Comput.*, 2015, **16**, (PB), pp. 268–285. ISSN: 1574-1192

25 Hasenfratz, D., Saukh, O., Walser, C., *et al.*: 'Pushing the spatio-temporal resolution limit of urban air pollution maps'. 2014 IEEE Int. Conf. on Pervasive Computing and Communications (PerCom), March 2014, pp. 69–77

26 Slijepcevic, S., Megerian, S., Potkonjak, M.: 'Location errors in wireless embedded sensor networks', *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, 2002, **6**, (3), p. 67. ISSN: 15591662

27 Saukh, O., Hasenfratz, D., Thiele, L.: 'Route selection for mobile sensor nodes on public transport networks', *J. Ambient Intell. Humanized Comput.*, 2012, **5**, (3), pp. 307–321. doi: 10.1007/s12652-012-0170-7. Available at: http://link.springer.com/10.1007/s12652-012-0170-7

28 Noguero, A., Calvo, I., Almeida, L.: 'A time-triggered middleware architecture for ubiquitous cyber physical system applications', *Ubiquitous Comput. Ambient Intell.*, 2012, pp. 73–80, Available at: http://www.springerlink.com/index/613T770T84771364.pdf

29 Statt, N.: 'NEST is permanently disabling the Revolv smart home hub', http://www.theverge.com/2016/4/4/11362928/google-nest-revolv-shutdown-smart-home-products. Last access: June 26th, 2016. April 2016. Available at: http://www.theverge.com/2016/4/4/11362928/google-nest-revolv-shutdown-smart-home-products

30 Blackstock, M., Lea, R.: 'Toward a distributed data flow platform for the web of things (Distributed Node-RED)'. Proc. fifth Int. Workshop on Web of Things. WoT '14, Cambridge, MA, USA, 2014, pp. 34–39. doi: 10.1145/2684432.2684439. Available at: http://doi.acm.org/10.1145/2684432.2684439

31 Khan, M.S., Kim, D.: 'DIY interface for enhanced service customization of remote IoT devices: a CoAP based prototype', *Int. J. Distrib. Sen. Netw.*, 2016, **2015**, pp. 185:185–185:185. ISSN: 1550-1329

32 Boyinbode, O., Le, H., Takizawa, M.: 'A survey on clustering algorithms for wireless sensor networks', *Int. J. Space-Based Situated Comput.*, 2011, **1**, (2-3), pp. 130–136

33 Wang, J., Yonamine, Y., Kodama, E., *et al.*: 'A distributed approach to constructing k-hop connected dominating set in *ad hoc* networks'. Int. Conf. on Parallel and Distributed Systems (ICPADS), 2013, 2013, pp. 357–364

34 Kuhn, F., Moscibroda, T., Wattenhofer, R.: 'Fault-tolerant clustering in *ad hoc* and sensor networks'. 26th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'06), 2006, pp. 68–68

35 Wu, J., Li, H.: 'On calculating connected dominating set for efficient routing in *ad hoc* wireless networks'. Proc. Third Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 1999, pp. 7–14

36 Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: 'Energy-efficient communication protocol for wireless microsensor networks'. Proc. 33rd Annual Hawaii Int. Conf. on System sciences, 2000, 2000, p. 10

37 Abbasi, A.A., Younis, M.: 'A survey on clustering algorithms for wireless sensor networks', *Comput. Commun.*, 2007, **30**, (14), pp. 2826–2841

38 Gupta, G., Younis, M.: 'Fault-tolerant clustering of wireless sensor networks'. Wireless Communications and Networking, 2003, WCNC 2003, 2003, vol. 3, pp. 1579–1584

39 Ilker Oyman, E., Ersoy, C.: 'Multiple sink network design problem in large scale wireless sensor networks'. IEEE Int. Conf. on Communications, 2004, 2004, vol. 6, pp. 3663–3667

40 Levis, P., Gay, D., Culler, D.: 'Active sensor networks'. NSDI'05 Proc. second Conf. on Symp. on Networked Systems Design & Implementation, 2005, pp. 343–356

41 Levis, P., Culler, D.: 'Maté a tiny virtual machine for sensor networks'. The 10th Int. Conf. on Architectural Support for Programming Languages and Operating Systems ASPLOS X, 2002, pp. 85–95

42 Müller, R., Alonso, G., Kossmann, D.: 'A virtual machine for sensor networks'. Proc. second ACM European Conf. on Computer Systems 2007, EuroSys '07, 2007, pp. 145–158
43 Koshy, J., Pandey, R.: 'VM*: synthesizing scalable runtime environments for sensor networks'. Third ACM Conf. on Embedded Networked Sensor Systems Sensys '05, 2005, pp. 243–254
44 Harbaum, T.: 'NanoVM', Available at: http://harbaum.org/till/nanovm/index.shtml
45 Brouwers, N., Langendoen, K., Corke, P.: 'Darjeeling, a feature-rich VM for the resource poor'. Seventh ACM Conf. on Embedded Networked Sensor Systems Sensys '09, 2009, pp. 169–182
46 Aslam, F., Schindelhauer, C., Ernst, G., *et al.*: 'Introducing TakaTuka: A Java Virtual machine for Motes'. Sixth ACM Conf. on Embedded Networked Sensor Systems Sensys '08, 2008, pp. 399–400
47 Evers, L.: 'Concise and flexible programming of wireless sensor networks', 2010
48 *Connected Limited Device Configuration (CLDC)*. Available at: http://www.oracle.com/technetwork/java/cldc-141990.html
49 Hong, K., Park, J., Kim, T., *et al.*: 'TinyVM, an efficient virtual machine infrastructure for sensor networks'. Seventh ACM Conf. on Embedded Networked Sensor Systems Sensys '09, 2009, pp. 399–400
50 Balani, R., Han, C.-C., Kumar Rengaswamy, R., *et al.*: 'Multi-level software reconfiguration for sensor networks'. Proc. sixth ACM & IEEE Int. Conf. on Embedded Software. EMSOFT '06, New York, NY, USA, 2006, pp. 112–121
51 Ellul, J., Martinez, K.: 'Run-time compilation of bytecode in sensor networks'. SENSORCOMM 2010 Fourth Int. Conf. on Sensor Technologies and Applications, 2010, pp. 133–138