WuKong: Secure Run-Time environment and data-driven IoT applications for Smart Cities and Smart Buildings

Chi-Sheng Shih[1]*, Jyun-Jhe Chou[1], and Kwei-Jay Lin[1,2]
[1]National Taiwan University, Taipei, Taiwan 10617
[2]University of California, Irvine, CA, US
kjlin@uci.edu

Abstract

Applying IoT technologies to cities and buildings can not only provide intelligent services to the users but also better utilize resources. While developing applications for these two domains, there are several challenges. *Manageability*, *connectivity*, and *programmability* are three of the major challenges. This paper presents the design and study of programming applications for smart cities and smart buildings. Data-driven programming model is designed to simplify the complexity on programming applications on large scale devices. To reduce the complexity of inter-connecting large numbers of devices, three message exchange models supported at WuKong, an intelligent virtual middleware for IoT applications, are studied in this paper: *cloud-based*, *NDN-based*, and *peer-to-peer* message exchange models. Cloud-based model is the most intuitive model but is only suitable for time-insensitive applications. The other two models can support time sensitive applications: NDN-based model can support large scale deployment but leads to higher hardware cost; peer-to-peer model can be employed to the applications in the scale of building without heavy hardware cost. Our evaluation results show that cloud-based model performs worse than the other two models when the number of senders/receivers in the order of magnitude. Last but not the least, security is the foundation of the smart city/building applications. CapeVM, the core of WuKong virtual middleware, takes the advantage of ahead-of-time complier to prevent illeagal memory access and execution order without excessive performance overhead.

Keywords: Data-Driven IoT applcations, Smart Cities, Named Data Network

1 Motivation

Many market survey reports including the ones published by CISCO [5] and IDC [14] estimate that the number of IoT devices will be more than number of populations in the world in few years. Today, many of us may need to manage 3 to 7 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. Smart building and smart cities are two representative use scenario for IoT/CPS. Well designed IoT/CPS systems can reduce energy consumption, enhance safety in buildings and city, or can increase the comfortability in the building. In last few years, the research communities and industrial partners started to study and investigate these two use scenarios to develop prototype or commercial services [15] to [11] for these two scenario.

Applying IoT technologies to cities and buildings can not only provide intelligent services to the users but also better utilize resources. While developing applications for these two domains, there are several challenges. *Managability*, *connectivity*, and *programmability* are three of the major challenges.

*Corresponding author: Chi-Sheng Shih, cshih@csie.ntu.edu.tw, Department of Computer and Information Sciences, National Taiwan University, 10617 Taipei, Taiwan

Managability refers to the capability of managing the services on large number of devices remotely; programmability refers to the capability of composing services onto heterogeneous IoT devices. *Connectivity* refers to the capability of exchanging data among heterogenous devices using heterogeneous network protocols. Last, *programmability* refers to the capability to compose the services for large number of devices and to be replicated to same or simialar environment.

The aforementioned four challenges are in fact interwoven. They are boiled down to how to manage these devices to assure that most, if not all, of them function correctly to conduct the deployed services in large scale deployment. In particular, most of the users and developers for these devices do not have the training to configure the services on these devices. The smart city services should be managed with least amount of user efforts.

Although many works have been conducted on these two scenarios, many challenges remain open. One major challenge for Smart City applications is to develop and manage applications on large number of devices. Figure 1 shows an IoT example inside buildings: Intelligent Active Dynamic Signage System (IADSS). In many existing building regulation, the directions for emergency evacuation point to the closest emergency exit. However, during emergency, some of the emergency exits man not be available and should not be used. In this example, there are two emergency exits in the building. When a fire breaks out in front of the emergency exit on the right, this emergency exit should be blocked. Building regulations in several countries already requires the emergency sign should be able to dynamically show the directions. In order to show correct direction, the system has to collect real-time data from the smoke detectors and computes the safe route when there is an emergency. Galea et al. [6] shows that the

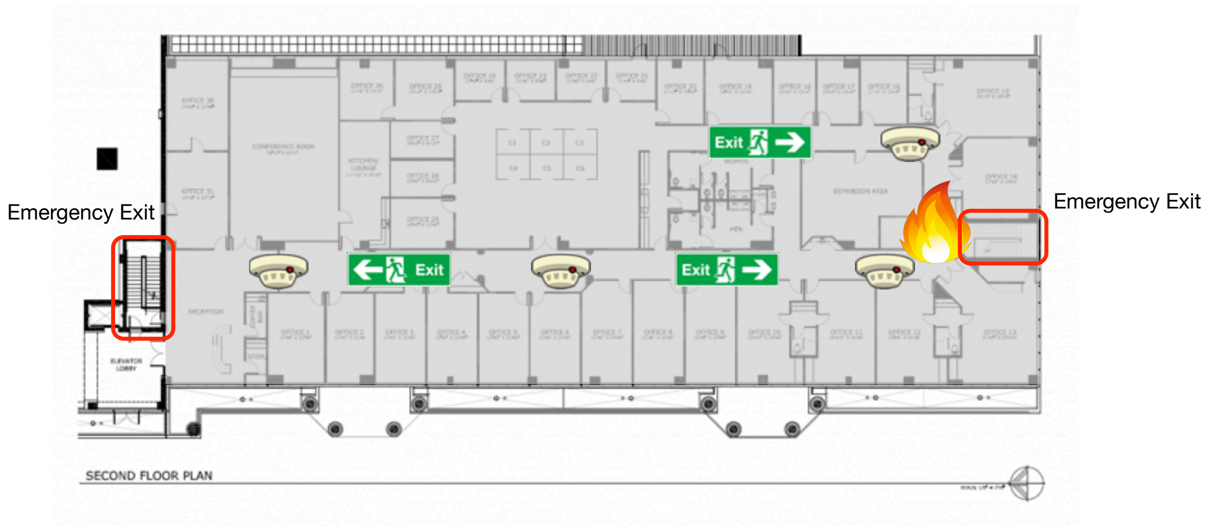


Figure 1: Intelligent Active Dynamic Signage System (IADSS)

dynamic signs can shorten the reaction time from 5.7s to 1.8s in the experiments. The example can be extended to the scale of city or airport. In order to collect information in the city and building and display information to the citizens, large number of devices and applications have to be installed physically. In Frankfurt Airport, it took three years from 2003 to 2006 to install 8,000 passive RFID tags to smoke detector and emergency lights [3]. Not mention to add intelligence to these devices and manage to put these devices to work together.

Systems in this scale are not alone. There are 250,000 street lights and 12,500 intersection in New York city [4]; there are 2,500 street intersection in Taipei city which has three millions resident in the city [2]. Figure 2 is another popular example application for smart city: intelligent parking system. Networked parking meters have be deployed in many cities to detect the occupancy of parking spaces and

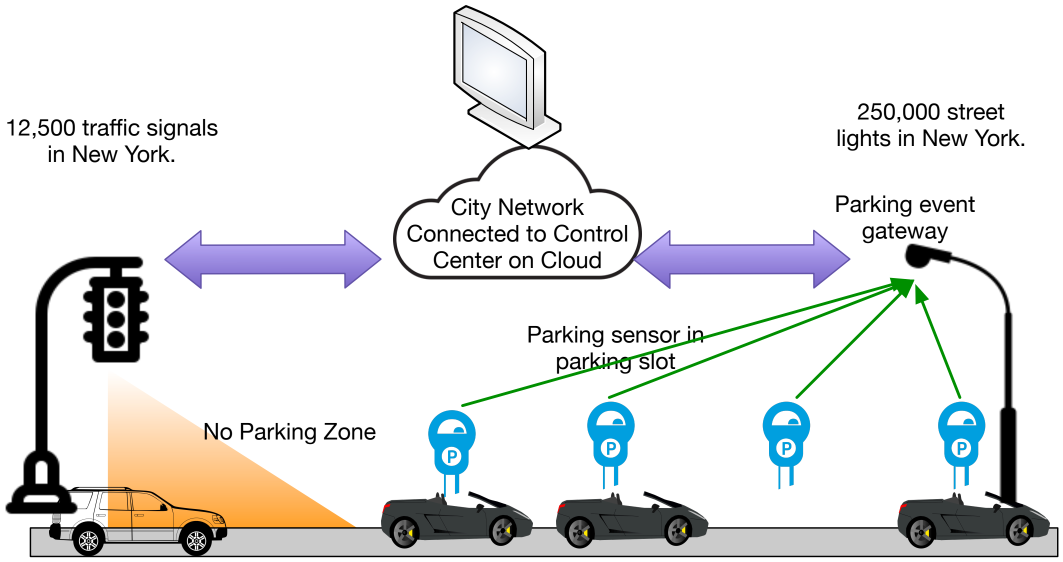

12,500 traffic signals in New York.
City Network Connected to Control Center on Cloud
250,000 street lights in New York.
Parking event gateway
Parking sensor in parking slot
No Parking Zone
P

Figure 2: Intelligent Parking Systems

to collect parking fees. To provide coherent services to the drivers, the information should be collected and shown in (near) real-time. One popular system architecture is to collect information on cloud server. This approach subjects to performance bottleneck and is difficul to maintain. In September 2016, a software glitch in public bike rental service in Taipei City took two days to reconfigure the 17,317 rental stations so as to resume the service [17].

Programming for large scale systems is not trivial. In a smart city application, the system has to coordinate millions or at least thousands of devices to work together. Among them, some are dumb devices, some can do simple computation, and some can store and process several tera-bytes of data per day. Although distributed computing and parallel programming have been studied for several decades, programming a system in city scale has different challenges. The diversity of computing and communication capability among devices in smart city application is much great than that in distributed computing and parallel programming.

To tackle managability, conectivity, and programmability, we propose an intelligent middleware for smart city applications. The middleware provides a data-driven application development environment and hardware-independent service composition method. Hence, the middleware can postpone the binding between logical services and physical devices until deployment phase. To tackle scalability challenge, the middleware provides three message exchange models to support large scale data collect and exchange.

The reminder of this paper is organized as follow. Section 2 presents the middleware for smart city applications, inculding development environment and deployment environment. Section 3 presents three message exchange models for different smart city applications. Section 4 presents the performance evaluation results of three messagin mechanism in WuKong. Finally, Section 5 concludes the paper.

2 Middleware for Service Development and Management

WuKong [12] is an intelligent middleware for designing and managing large scale IoT services. It consists of two major components to fill the gap for developing and managing IoT applications: *intelligent run-time environment* and *flow-based developing framework*. WuKong run-time environment is regarded as a Virtual Middle-Ware (VMW). The reasons for this are two-folds. First, as IoT applications are deployed at different locations and evolve over time, it is very likely that the systems use devices (including sensor, actuators, and computing platforms) developed by different manufactures and communicating via different network protocols. Having virtual devices allow applications to run on heterogeneous devices and networks. Second, when the system needs to be reconfigured, the process of reprogramming devices will be less expensive by using a virtual machine design. In addition, the line of codes will be less since the virtual devices can offer higher level primitives specific for IoT applications.

Flow-based developing framework provides a high level development environment to design hardware independent IoT applications. The goal of this framework is to allow domain experts to design their IoT applications without complete knowledge of the hardware devices and network protocols to conduct the services. In this framework, the users compose the services using data-flow model and pre-defined services. The framework then generates the executable code for selected devices using distributed computing model.

2.1 Run-time environment of WuKong devices

Most of the IoT application development environments are hardware dependent and require the developers to specify hardware properties while developing IoT/Smart city applications. For example, the operating system if any, CPU type, and port number for the sensors should be known. In order to reduce management overhead, the developed system may be required to use identical devices and platforms, or to store configuration files for each type of platforms. It is certainly impractical in large scale and evolving smart city systems. WuKong run-time virtualizes IoT devices by deploying virtual machines to these resource limited devices. This model does not only provide a hardware independent runtime environment but also enhance reliability and security for IoT devices.

Figure 3 shows the run-time environment on WuKong-enabled IoT devices. Figure 3(a) shows the

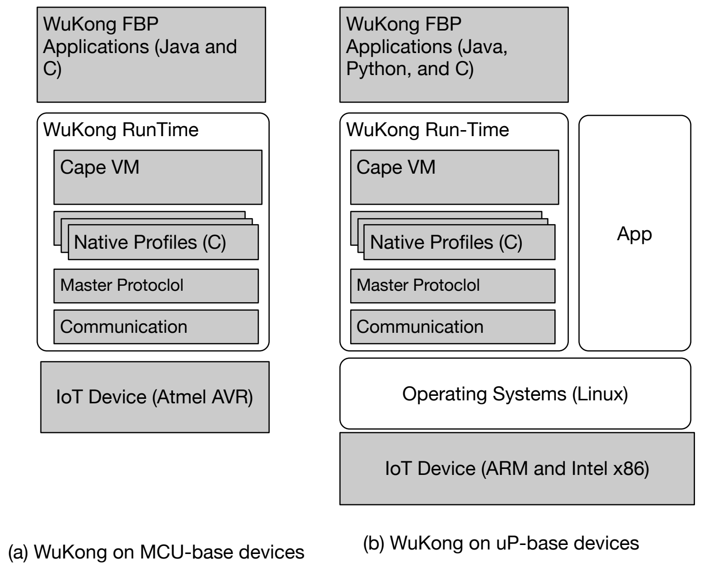

WuKong FBP Applications (Java and C)
WuKong RunTime
Cape VM
Native Profiles (C)
Master Protoclol
Communication
IoT Device (Atmel AVR)
(a) WuKong on MCU-base devices
WuKong FBP Applications (Java, Python, and C)
WuKong Run-Time
Cape VM
Native Profiles (C)
Master Protoclol
Communication
App
Operating Systems (Linux)
IoT Device (ARM and Intel x86)
(b) WuKong on uP-base devices

Figure 3: WuKong Runtime on IoT devices

runtime for micro-controller-based IoT devices such as Arduino MEGA2560, which is powered by Atmel AVR ATmega2560 micro-controller. On this type of devices, there is no operating system and WuKong run-time starts when the system is turned on. WuKong runtime consists of communication component, master protocol component, native profiles, and Darjeeling JVM. Communication component is responsible to communicate with radio interface on the device so as to send and receive messages to/from other devices; master protocol component is responsible to communicate with WuKong master, which will be discussed later, so as to provide device properties, and to download applications from WuKong master. Native profiles refer to the service adapter (or device driver) of hardware components such as sensors and actuators. Last, Darjeeling JVM is the Java Virtual Machine to execute Java applications. Darjeeling VM supports limited Java APIs for embedded devices and is a stack-based VM. Different from traditional JVM, Darjeeling VM uses AOT (Ahead-of-Time) compiler, rather than JIT compiler, to reduce the memory and storage usage requirements. The memory footprint of DarjeelingVM is less than 80KB and the optimized Java executable code are only 86% slower than optimized native C executable code [13]. (Other Java JIT compilers lead to 30x to 200x slower, compared to optimized C implementation.)

2.2 WuKong Development Environment

WuKong development environment is a graphical programming environment for flow-based programming. In WuKong development environment, the users select appropriate pre-defined service class, called *WuClass*, to compose their applications. One WuClass can represent primitive sensing services such as temperature sensing and motion sensing, primitive actuation services such as buzz and display, or programmable decision services using Python, Java, or C.

Figure 4 shows the flow-based development environment in WuKong. In the development environ-

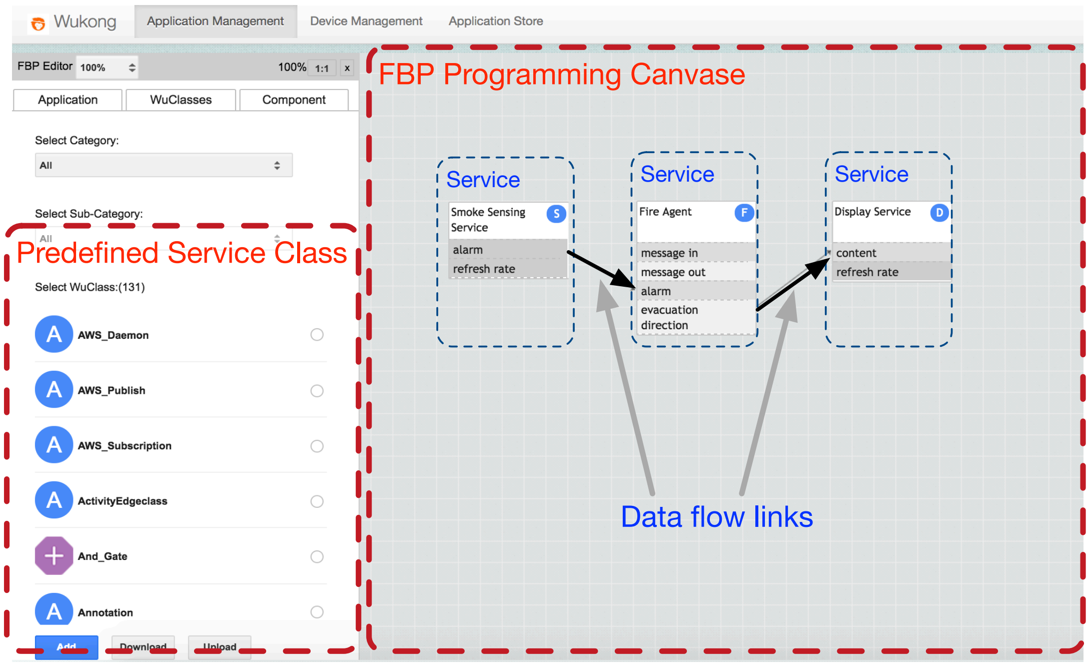


Figure 4: Example application in WuKong FBP Environment

ment, the developers drag and drop predefined service components, named *WuClass* in WuKong, and data links to FBP programming canvas shown on the right. The example shows a smoke detector and evacuation sign application, which detects smoke event using 'Smoke Sensing Services' shown on the left and displays evacuation route using 'Display Services' shown on the right. The service in the center represents a fire agent to intelligently find the safe evacuation route. Each WuClass has predefined properties, which can be read-only, write-only, or read/write. In this example, the alarm property in Smoke Sensing Service WuClass is a read-only property; the content property in Display Service WuClass is a write-only property. The directed lines between WuClasses represent directed data flows from one WuClass to another one.

Application developed in WuKong FBP environment only defines the services and logical flows for the application. The service class can specify the minimum requirements of hardware devices to conduct the service. The application shown in Figure 4 can be deployed to one edge device or multiple devices connected by computer networks or wires. We discuss the deployment process in next subsection.

2.3 Deployment-Time Service Mapping

In WuKong middleware, WuMaster is responsible for managing the services and devices in an area, similar to a wireless access point for a wireless network. When a new device starts, it looks for WuMaster in the network and registers itself to the WuMaster. The WuMaster then starts the discovery process to collect hardware properties from the device. These properties will be stored on WuMaster for service management.

WuKong middleware explicitly separates the deployment phase from development phase. Figure 5 shows the process of mapping an FBP application to hardware devices and that of deploying applications to the devices. When the application is ready to be deployed, the application is downloaded to WuMaster

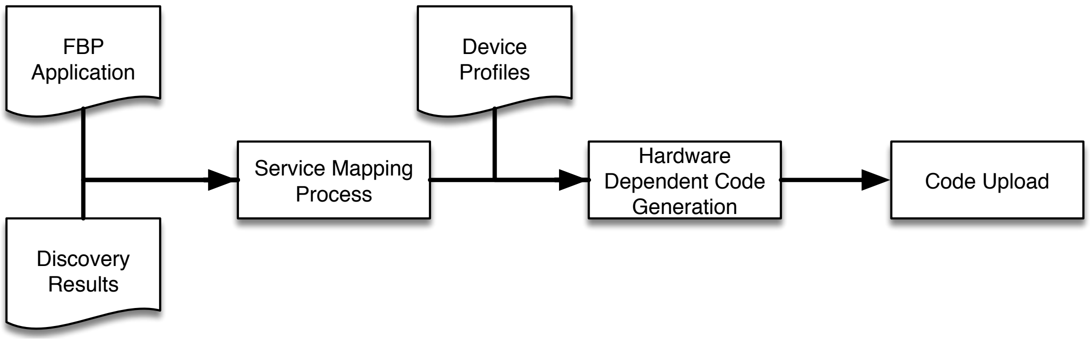


Figure 5: Flow for Service Mapping and Application Deployment on WuKong Master

from application store on the cloud. The first step for application deployment is to map logical services, i.e., WuClass, to physical devices. The servier installer or developers can choose different mapping algorithm to map WuClass in FBP applications to meet different QoS requirements. For example, one may ask to use minimal number of devices; the other may ask to minimize the network traffic in the system. WuMaster will map services to physical devices based on the selected mapping policies. The discovery results are used to search for capable hardware devices to conduct WuClass. Moreover, the mapping service also creates messaging links from the sending device to the receiving device for each data link defined in FBP application.

The second step generates the executable code for the devices. (Many of the edge devices are not able to generate executable code from the source codes of high-level programming languages.) Each IoT device may have different physical sensing and actuation devices, and supports different software-enabled services. These capabilities are specified in device profiles. WuMaster generates the code based on the device profiles. The last step is to upload the code to the devices using computer networks. WuMaster communicates with the Master Protocol component, shown in Figure 3, to upload the code. The uploaded code will be executed on top of WuKong Run-Time environment. Note that WuClass implemented in Python can only be uploaded to microprocessor-based devices.

3 Data Exchange Models for Smart City Application

To develop applications for large scale deployment, one has to coordinate the data exchange among the devices. As mentioned earlier, the diversity of devices in smart city systems is much greater than that in traditional distributed systems. In smart city systems, there are edge devices which have sensors installed or can execute actuation commands; there are gateway devices which can aggregate and process data, or can make intelligent decision. It is not scalable to configure the communication links on each of the devices. In this section, we study three different communication models for smart city applications. They are *cloud-based* message exchange model, *named-data* message exchange model, and *dynamic data link* message exchange model.

3.1 Cloud-based Message Exchange Model

Cloud-based message exchange model uses the servers on the cloud to exchange messages among services. Despite the communication delay between edge devices and cloud server, the model is well accepted due to its simplicity.

In this model, each device sends its message along with data properties to the cloud server and requests information/data from the cloud server by specifying appropriate data properties. IoT services provided by Amazon Web Services is one example of this model.

Figure 6 shows an example of cloud-based messaging FBP application in WuKong. This example

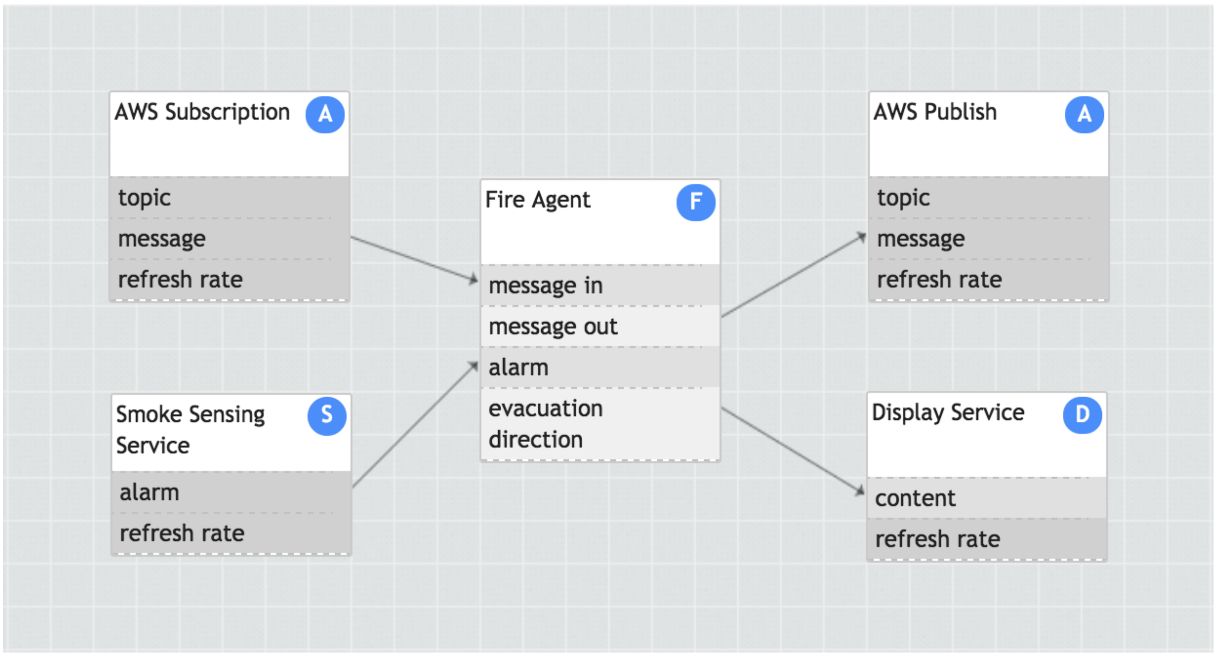


Figure 6: Example WuKong Application Using Cloud-Based Message Exchange Model

is an extension of the single device FBP application shown in Figure 4. Rather than only using the data collected by local smoke sensing service, the Fire Agent WuClass also subscribes the smoke sensing services nearby and publishes its own smoke events to the cloud. As the results, the Fire Agent can not only sound alerts according to local smoke event but also compute the safe evacuation route based on the information collected by other smoke sensing services in the building.

The cloud service is usually known at development phase. Hence, the developers can configure the cloud service before deployment. As a result, all the edge devices running the deployed application send data the pre-configured cloud services. There is no deployment-time or run-time configuration in this model.

While publishing data to the cloud server, the Fire Agent WuClass specifies its location information and time-stamp of the published data. The location information can be a property of the edge devices, which can either be configured by WuMaster or acquired from localization services such as LBeacon [8]. While subscribing data from the cloud server, the Fire Agent WuClass specifies the information of interests using location query.

This model is suitable for sharing sensing data for large number of data subscribers. One example is air quality monitoring. Air quality sensors can be installed in a city or state to monitor air quality. The users can use smartphones or other networked devices to monitor air quality. In this scenario, the number of sensors are limited but the number of users are not. It also is suitable for the applications requiring *global data* to make decision. Route planning for vehicle navigation requires the traffic condition from starting location to destination location to plan the best route.

3.2 NDN-Based Message Exchange

Named-data network [19] takes advantage of the storage services in the network to store data having short life-time and shared among nearby or limited number of devices. As a result, the data are not exchanged via cloud server so as to avoid traffic congestion on server sides and to reduce the amount of data to be stored.

Figure 7 shows an example of NDN-based messaging FBP in WuKong. Compared to the one shown

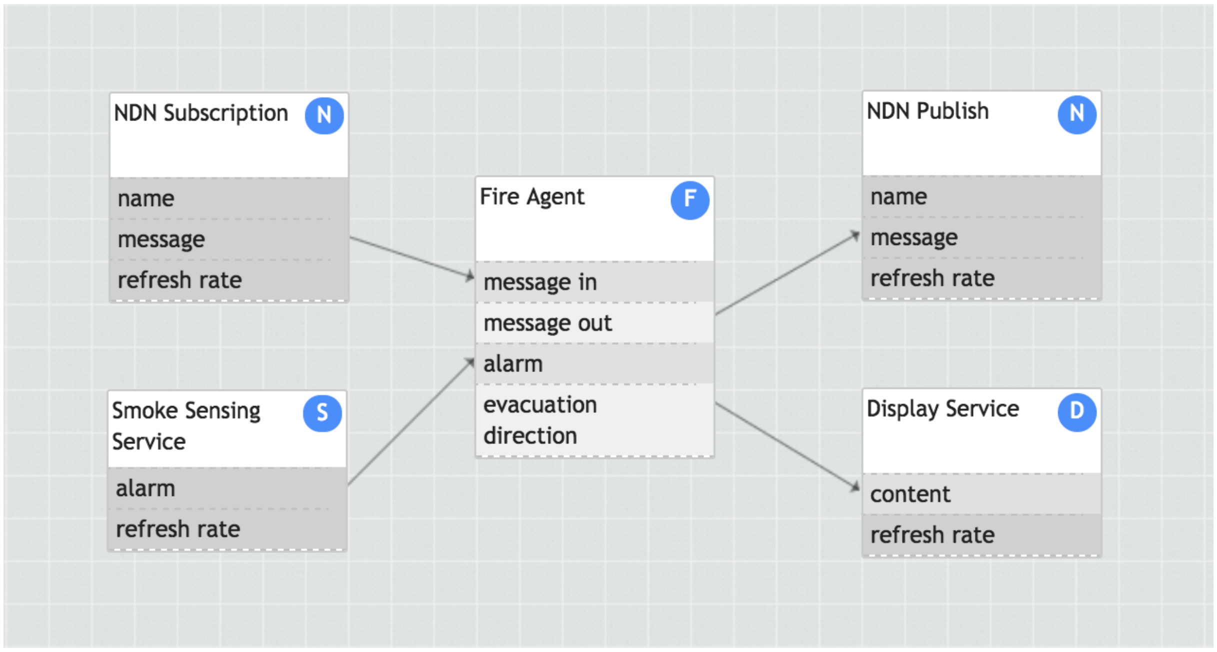


Figure 7: Example WuKong Application Using NDN-Based Message Exchange Model

in Figure 6, the NDN Publish and Subscription WuClasses are used to publish and subscribe smoking events. In named-data network, the gateways to publish and subscribe data are not identical for all edge devices in the network. Hence, when the application is started on an edge device, NDN Publish and Subscription WuClass in the application will first discover NDN services on the network. Then, the discovered NDN gateways will be stored as a property of NDN publisher and subscription WuClass.

NDN is mainly designed to search and cache data having short life time or shared by a group of nearby devices. Hence, it adapts a pull model to request data from publishers and NDN gateway, which is different from traditional publish/subscription model. In many smart city applications such as intelligent parking and fire evacuation shown above, the subscribers are interested in the events at certain location whenever occurs.

To support traditional publish/subscription model, WuKong designs a reverse NDN mechanism. Figure 8 shows the flow of reverse NDN to register the interest and to publish updated data. In reverse

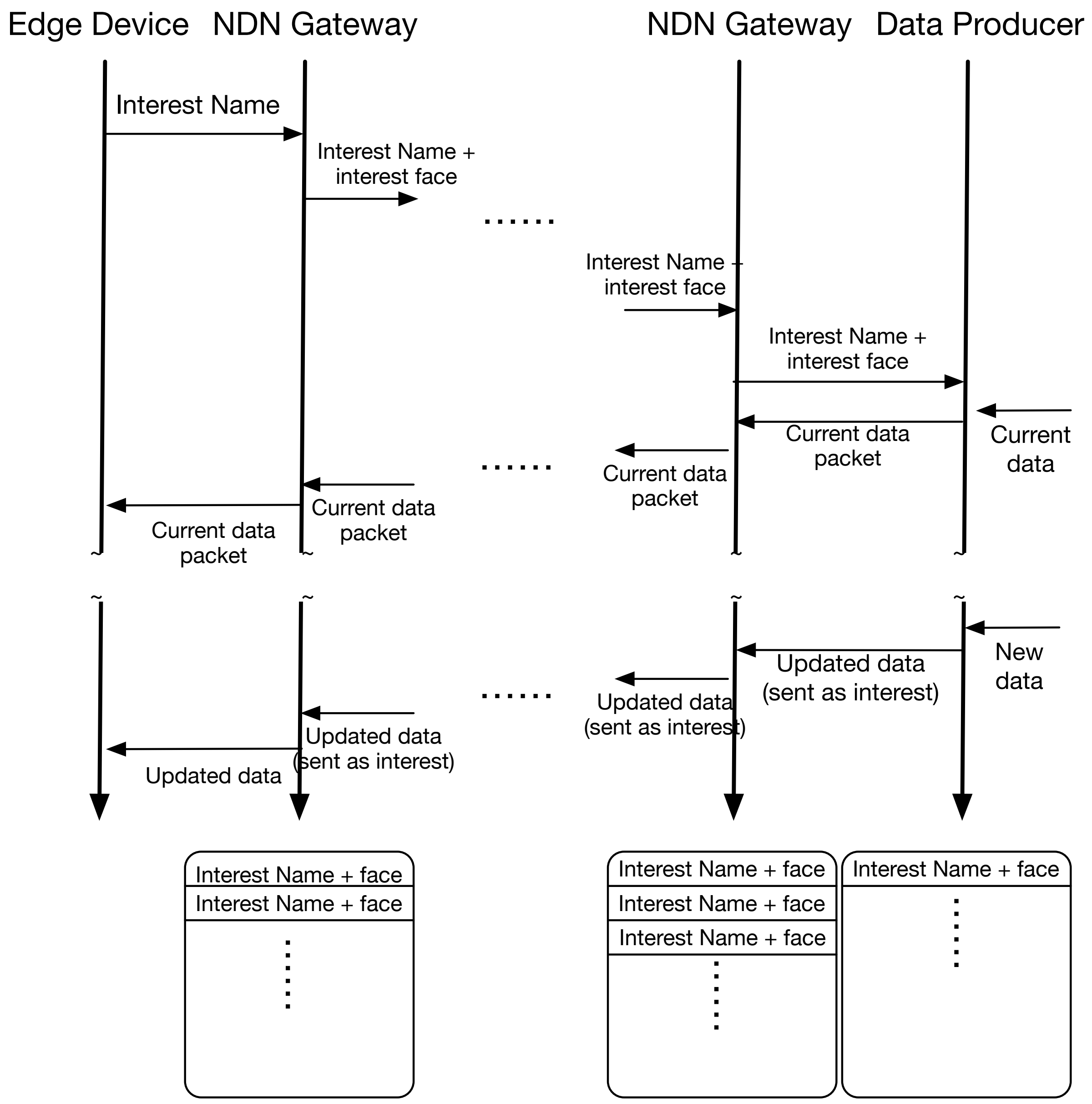


Figure 8: Data flow for reverse NDN

NDN, the subscribers on edge devices publish the interests to their upstream gateways. When an NDN gateway receives an interest request, it checks if any cached data matches the received interest to be returned. If not, the interest will not only be forwarded to its upstream gateway but also be stored in the gateway. While forwarding the interest, the gateway also includes its own incoming interest face, i.e., the interface to send/receive interest, to receive interest from the upstream gateway when available. The process repeats until named data is founded or the interest reaches the data producer on edge devices. The data producer of interest will store the interest face received from NDN gateway as so to publish new data when available. Then, the requested data will be sent back to the subscribers along the same path. Normal NDN protocol ends after the data of interest is received on the subscriber.

In reverse NDN, the registration table allows the data producers to push data when there is any change or update interval expires. To push data, the data producer embeds the data value in interests and sends the interest to the face stored in registration table.

NDN-based messaging service is suitable for the applications only requiring local, rather than global, information to make decision. For example, an intelligent parking system showing the suggested direction to go only needs the number of available parking spaces nearby. It is *not* necessary to know that parking spaces are available few mile/kilometers away.

3.3 Peer-to-Peer Message Exchange

The third model is to create peer-to-peer data links among selected devices. This approach eliminates the needs of messaging brokers, either cloud server or NDN gateway, which require complete IP protocols to be deployed and full-brew operating systems, in the system. When the edge devices in the network have limited resources, it may not be practical to use the two aforementioned messaging models.

Figure 9 shows the intelligent smoke detector using peer-to-peer message exchanging model. This

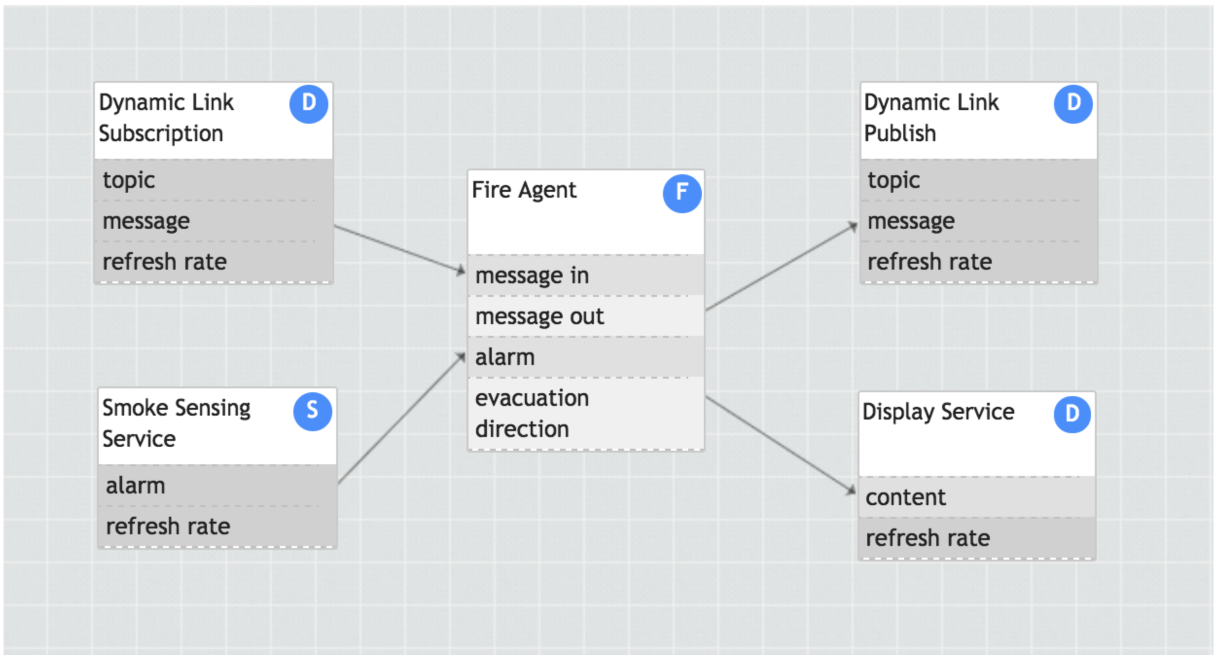


Figure 9: Example WuKong Application Using Peer-to-Peer Message Exchange

model also uses publish/subscription model for message exchange. However, there is no (physical and logical) messaging brokers in the network. Hence, the peer-to-peer data links are created at service mapping phase based on the publish and subscription topics specified in the WuClass. The location framework in WuKong allows the developers to specify the location using wildcard expression and range function. For example, one may subscribe to the information published by the services deployed on the same floor or in the same room. Based on the publish/subscription requirements, the service mapper on WuMaster generates location-dependent data link tables for each device. This model is suitable for the system in which the mobility of the devices is very limited. Every change on available edge devices will trigger WuMaster to regenerate parts of the link tables.

In this model, WuMaster may create one communication link between two edge devices on two ends of one data link. Hence, each of the communication link is a peer-to-peer communication link. Consequently, one property update can be duplicated to multiple communication links when there are multiple receivers for the properties. Hence, the challenge of this model is how to minimize the network traffic in the networks, which are closely related to energy consumption on the edge devices. The problem can be formulated as follow.

Suppose that data link $l_{i,j}$ for $0 < i, j < N$ where N is the number of WuClass in the FBP represents the data link from WuObject o_i, which is an instance of a WuClass C_i, to WuObject o_j and the minimal length of data link $l_{i,j}$ is $m_{i,j}$ where $m_{i,j} > 0$ and $m_{i,j} = \infty$ if a data link is not defined from WuClass C_i to WuClass C_j. *Link table* on edge device D_k, denoted as T_k, stores the communication link from WuObject deployed on device D_k to its destinations to transmit the property values. Entry $e^k_{i,j}$ in link table T_k represents the data link from WuObject o_i to WuObject o_j.

Given a set of data links $\mathbf{L} = \{l_{i,j} \mid m_{i,j} \neq \infty \text{ for } 0 < i, j < N \text{ and } i \neq j.\}$ in an FBP, the challenge is to generate a set of link tables $\{T_k \mid T_k \text{ for } 0 \leq k \leq M\}$ where M is the number of devices selected by service mappers such that the total length of communication links are minimized subject to the following constraints.

- if $l_{i,j}$ exists, there must exist a path from WuObject o_i to WuObject o_j.

The above problem can be solved by minimal spanning tree algorithm to find an optimal results. The link tables for each edge device will be generated based on the solution to minimize the redundant message transmission.

4 Performance Evaluation

To study the performance of these three message exchange models, we evaluate these three models using simulations.

4.1 Experimental Settings

The experiments are designed to simulate a system having 10,000 edge devices as either sensors or actuators. Edge devices are micro-controller-based devices such as Arduino MEGA2560 or similar devices. Intermediate devices serve communicaiton gateways or network access points. They can function as NDN gateway to cache data and store interest table. Hence, they are usually micro-processor-based devices such as Intel Edison, Raspberry Pi, or similar platforms.

To extend the coverage, the experiments simulate two different use scenarios. The first use scenario has only one sender to send messages to multiple receivers and the number of receivers range from 10 to 10,000. This scenario serves as the base to understand the performance of three data exchange models. The second scenario simulates the applications in smart cities and smart buildings. It has multiple senders, each of which sends messages to multiple receivers in the network. In the second scenario, one receiver may receive messages from multiple senders.

Without loss of generality, the star topology is used to connect all the edge devices to the Internet. In the star topology, each intermediate device or network gateway has 10 downstream devices participating the systems. (It may connect to more than 10 devices but only 10 of them participating the system.) Below are other parameters used to simulate the system.

Table 1: Parameters for Experimental Settings

Parameters	Values
Number of edge devices	10,000
Bandwidth of Wireless networks	100 mbps
Average propagation Delay on gateways and networks	1ms, 0.8 ms, 0.64 ms, 0.51 ms, and 0.41 ms
Ratio of sender/receiver	1/10, 1/100, 1/1000, 1/10,000
Number of hops between sender/receivers	1, 3, 5, 7
Date size	1Kb

Two performance metrics are measured in the simulation. *Average end-to-end delay* measures the time delay to exchange one message from data source to destination; *Total message count* measures the number of messages sent during the experiments.

4.2 Evaluation Results

The first scenario has single sender and multiple receivers. Figure 10 shows the average end-to-end delay in log scale from the sender to multiple receivers when cloud-based message exchange model is used. The horizontal axis represents the number of hops between sender and receiver. Where there is only one hop, the sender and receiver are located in the same subnetwork; when there are seven hops, the sender and receiver can be located in different countries/states. In cloud-based model, the sender sends the data to the server which is located on the top of the star topology; the receivers receive messages from the server. The results show that the end-to-end delay is not sensitive to the number of hops. However, it

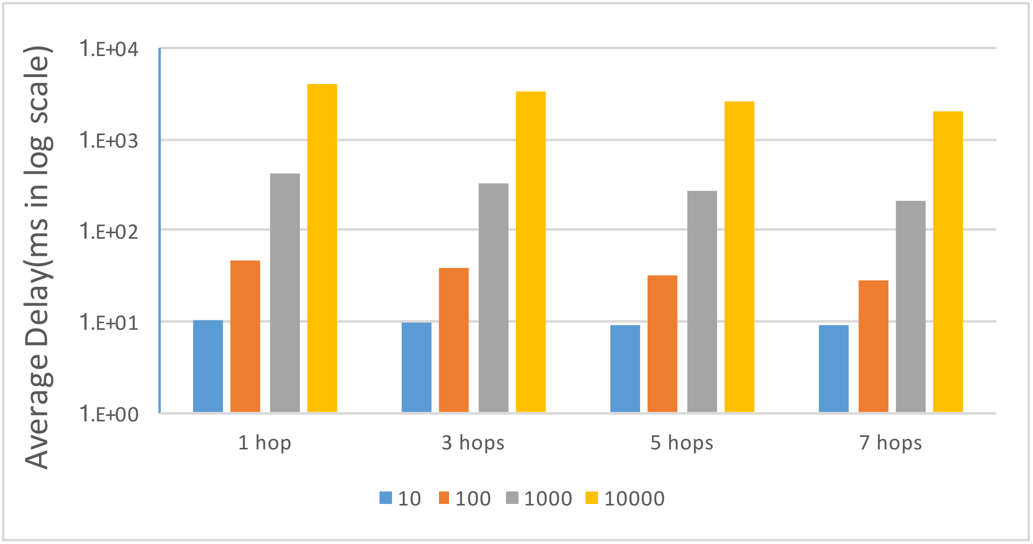


Figure 10: Average end-to-end delay for cloud-based model (single sender)

is very sensitive to the number of receivers in the applications. When the number of receivers increase from 10 to 10,000, the average end-to-end delay increases increased from 10ms to 4,000ms in average. It shows that when the receivers are designed to receive the sensed data simultaneously, the delay will be significant. However, the delay can be shorten if the receivers receive the change asynchronously.

In addition, the delays are shorter when the number of hops increase, i.e., the receivers are not located in same subnetwork. This is because the network traffic are scattered in the Internet. In the air quality monitor application, the users, using smartphone or web browsers, check the latest monitor results at different time instants and may not experience significant delays. Figure 11 shows the results for NDN-based model. When NDN-based model is used, the average end-to-end delays are significantly shorter

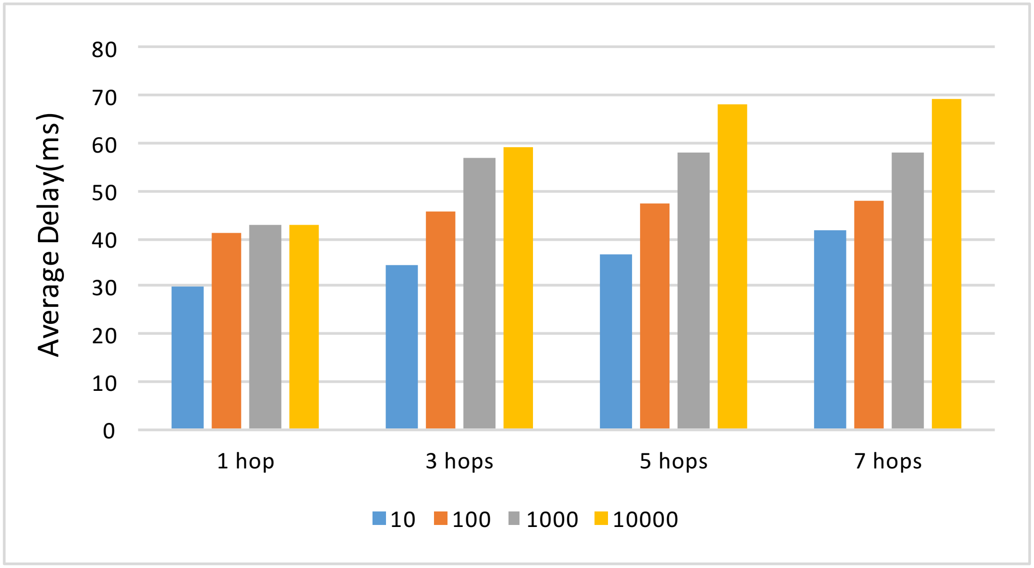


Figure 11: Average end-to-end delay for NDN-based model

than that in cloud-based model. The cached mechanism can greatly shorter the path from the sender to the receiver. When the sender and receivers are located in same subnetwork, the average end-to-end delays are less than 50ms for 10,000 receivers. When the number of hops between sender and receiver increase, the average end-to-end delays do increase. This is because the data have to be transmitted to the NDN gateways, which are multiple hops away from the sender. However, the increase is less than 50%.

Figure 11 shows the results for peer-to-peer model. The results show that the average end-to-end in this model are sensitive to the number of receivers, similar to the cloud-based model. This is because the peer-to-peer model has to conduct one peer-to-peer transmission for each receiver. When the number of hops increase, the delays remain the same. This is because each transmission always starts from the sender to the receiver. The peer-to-peer model does outperform the other two models when the number of receivers are small. The results suggest that the peer-to-peer model is suitable for the scenario that each sender sends data to small number of nearby receivers.

Figure 13 shows the average end-to-end delay when there are 100 receivers under three different models. The results show that cloud-based message exchange model leads to long delay time due to the processing and transmission bottleneck on the server. However, the delays become shorter when the receivers are scattered in the Internet. The other two models have very stable and similar performance for 100 receivers.

The second scenario is multiple senders for multiple receivers. Figure 14 shows the results for this scenario. The horizontal axis presents the ratio of overlapped receivers for each sender. When there is no overlap, each receiver only requests data from one sender. Overlap ratio refers to the percentage of receivers for one sender also request data from another sender. When the ratio is greater than 50%, one receiver may request data from more than three senders. In the simulation, there are 1,000 senders and 9,000 receivers. When there is no overlap, each sender have nine receivers. The results show that all

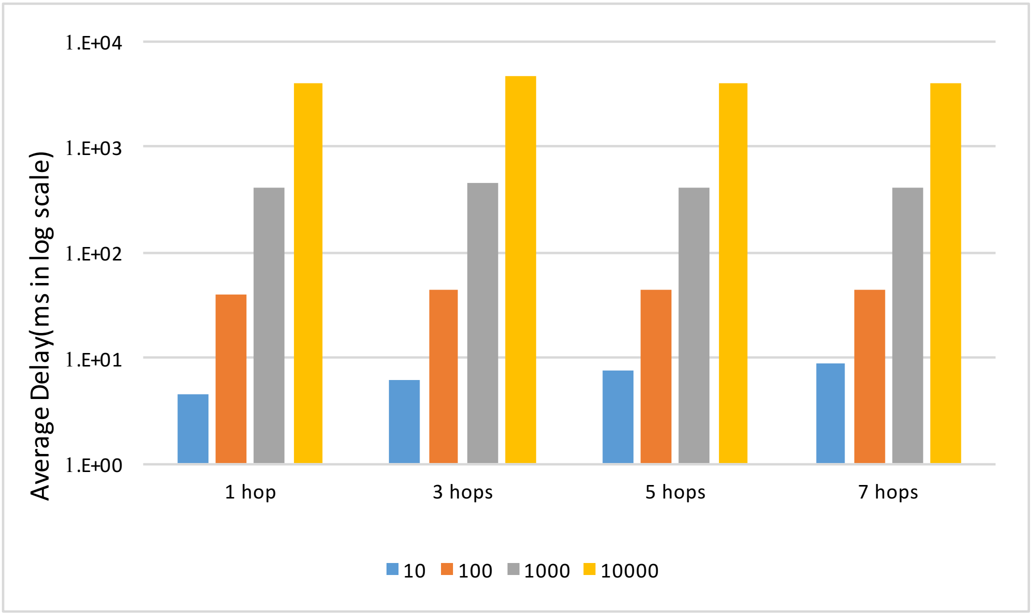


Figure 12: Average end-to-end delay for peer-to-peer model

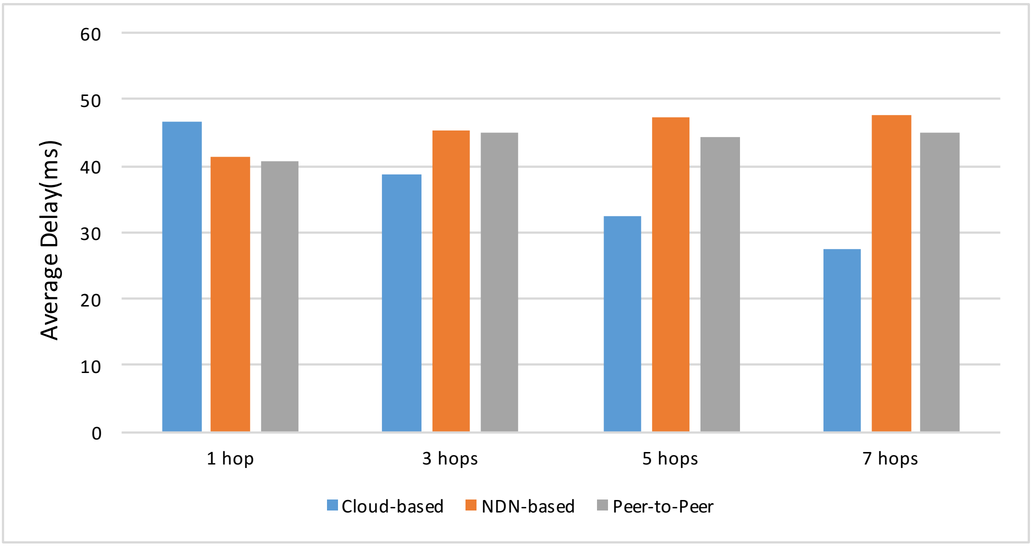


Figure 13: Average end-to-end delay for 100 receivers

the models lead to longer delay when the overlap ratio increases. NDN-based model takes advantage of the cached data to reduce the amount of message transmission and shorten the end-to-end delay when messages have to replicated to many receivers; however, peer-to-peer model has to transmit the messages for each receiver.

The results show that NDN-based model outperforms the other two models to provide short average end-to-end delay for majorities of the scenario. However, the model requries the gateway to cache the data and store interests. In other words, the network infrastructure support is required. The other two models do not require infrastructure support. When there is no need timing requirement, i.e., synchronous messages, the cloud-based model has reasonable end-to-end delays across the network. Peer-to-peer model is suitable for the applications requiring synchronous messages among senders and

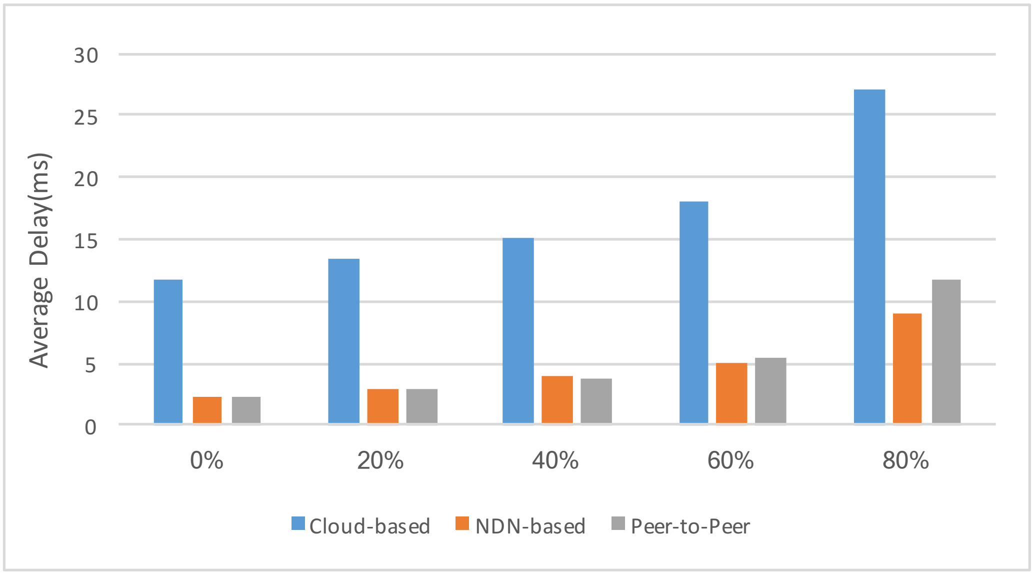


Figure 14: Average end-to-end delay for multiple senders and receivers

receivers.

5 Summary

This paper presents the messaging mechanisms and safety mechanism developed in WuKong middleware so as to tackle four challenges of deploying IoT applications for smart cities and buildings. The three messaging mechanisms are *cloud-based*, *NDN-based*, and *peer-to-peer* message exchange models. Cloud-based messaging mechanism has low implementation overhead but greatest transmission delay. It is not recommended for time-sensitive IoT applications. On the other hands, the other two mechanisms are not sensitive to the numbers of devices in the networks.

Acknowledgment

This research was supported in part by the Ministry of Science and Technology of Taiwan (MOST 106-2633-E-002-001), National Taiwan University (NTU-106R104045), Intel Corporation, Delta Electronics, and Advantech.

References

[1] B. Bowerman, J. Braverman, J. Taylor, H. Todosow, and U. Von Wimmersperg. The vision of a smart city. In *Proc. of the 2nd International Life Extension Technology Workshop, Paris, France*, volume 28, September 2000.

[2] Y.-Y. Chien. Traffic signals in taipei (in chinese). `http://www.yuyen.tw/2013/05/13.html` [Online; accessed on December 20, 2016].

[3] J. Collins. RFID lands at frankfurt airport. `http://www.rfidjournal.com/articles/view?2121/4` [Online; accessed on December 20, 2016].

[4] Department of Transportation, New York City. Infrastructure - traffic signals. `http://www.nyc.gov/html/dot/html/infrastructure/signals.shtml` [Online; accessed on December 20, 2016].

[5] D. Evans. 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` [Online; accessed on August 30, 2014], April 2011.

[6] E. R. Galea, H. Xie, and P. Lawrence. Intelligent active dynamic signage system: bringing the humble emergency exit sign into the 21st century. `https://www.sfpe.org/page/Issue3Feature1` [Online; accessed on May 15, 2018].

[7] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy. Smart objects as building blocks for the Internet of things. 14(1):44–51, January 2010.

[8] F. Lin, J. W.-S. Liu, E. T.-H. Chu, and C.-S. Shih. Data requirements of intelligent indoor emergency evacuation systems. Technical report, Institute of Information Science, 2016.

[9] T. Nam and T. A. Pardo. Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proc. of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times (dg.o'11), College Park, Maryland, USA*, pages 282–291. ACM, June 2011.

[10] T. Nam and T. A. Pardo. Smart city as urban innovation: Focusing on management, policy, and context. In *Proc. of the 5th International Conference on Theory and Practice of Electronic Governance (ICEGOV'11), Tallinn, Estonia*, pages 185–194. ACM, September 2011.

[11] P. Neirotti, A. D. Marco, A. C. Cagliano, G. Mangano, and F. Scorrano. Current trends in smart city initiatives: Some stylised facts. *Cities*, 38:25–36, June 2014.

[12] N. Reijers, K.-J. Lin, Y.-C. Wang, C.-S. Shih, and J. Y. Hsu. Design of an intelligent middleware for flexible sensor configuration in m2m systems. In *Proc. of the 2nd International Conference on Sensor Networks (SENSORNETS'13), Barcelona, Spain*, pages 1–6, February 2013.

[13] N. Reijers and C.-S. Shih. Ahead-of-time compilation of stack-based JVM bytecode on resource-constrained devices. In *Proc. of 2017 International Conference on Embedded Wireless Systems and Networks (EWSN'17), Uppsala, Sweden*, pages 84–95. ACM, February 2017.

[14] L. Spencer. Internet of things market to hit $7.1 trillion by 2020. `http://www.zdnet.com/internet-of-things-market-to-hit-7-1-trillion-by-2020-idc-7000030236` [Online; accessed on August 30, 2014], June 2014.

[15] J. A. Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3–9, February 2014.

[16] K. Su, J. Li, and H. Fu. Smart city and the applications. In *Proc. of the 2011 International Conference on Electronics, Communications and Control (ICECC'11), Ningbo, China*, pages 1028–1031. IEEE, September 2011.

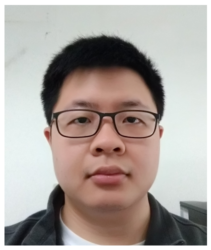
[17] Taipei Times. Youbike offers free rental after operational failure. `http://news.ltn.com.tw/news/focus/breakingnews/1813602` [Online; accessed on May 15, 2018], September 2016.

[18] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, February 2014.

[19] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named data networking. *SIGCOMM Computer Communication Review*, 44(3):66–73, July 2014.

Author Biography

Chi-Sheng Shih received the B.S. in Engineering Science and M.S. in Computer Science from National Cheng Kung University in 1993 and 1995, respectively. He joined National Taiwan University in 2004 and has been a Professor since 2018. His main research interests include embedded systems, hardware/software codesign, real-time systems, and database systems. Specifically, his main research interests focus on real-time operating systems, real-time scheduling theory, embedded software, and software/hardware co-design for system-on-a-chip. His research results won several awards including the Best Paper Award, 2011 ACM Research in Applied Computation Symposium (RACS 2011), the Best Paper Award, IEEE RTCSA 2005, the Best Student Paper Award, IEEE RTSS 2004, and Best Paper Award, IEEE International Symposium of Service and Cloud Computing (SC2), 2016. He also serve the steering committee for several international conferences including IEEE Cyber-Physical Systems, Network and Applications (CPSNA) and editoral board of international journals such as Journal of Service-Oriented-Computing Architectures.



Jyun-Jhe Chou received his bachelor degree in Computer Science and Information Engineering at National Taiwan University in 2014, and is now a PhD candidate. His research interests include Internet of Things, Quality of Data, and Cyber-Physical Systems.

Kwei-Jay Lin received the MS and PhD degrees in Computer Science from the University of Maryland, College Park. He is a Professor in the EECS Department at the University of California, Irvine. He is an Adjunct Professor at the National Taiwan University, Zhejiang University, China, and Nagoya Institute of Technology in Japan. He is an IEEE Fellow and Editor-In-Chief of the Springer Journal on Service-Oriented Computing and Applications (SOCA). He was an Associate Editor of IEEE Transactions on Parallel and Distributed Systems, 2002-2006, and an Associate Editor of IEEE Transactions on Computers, 1996-2000. He has served on many international conferences as conference and program chairs. His research interest includes IoT application and middleware, service-oriented computing, real-time systems, and distributed computing.