國立清華大學電機資訊學院資訊工程研究所

博士論文

Department of Computer Science

College of Electrical Engineering and Computer Science

National Tsing Hua University

PhD Thesis

針對雲到物之連續性平台進行分析與多媒體應用之資源分配最佳化

Optimal Resource Allocation for Analytics and Multimedia Applications in Cloud-to-Things Continuum Platforms

洪華駿

Hua-Jun Hong

學號：103062808

Student ID:103062808

指導教授：徐正炘 博士

Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 107 年 10 月

October, 2018

國立清華大學
資訊工程研究所

博士論文

針對雲到物之連續性平台進行分析與多媒體應用之資源分配最佳化

洪華駿 撰

107
10

Acknowledgments

I would like to express my gratitude to all the people who help me to finish my thesis in my PhD career. First, the most important person is professor Cheng-Hsin Hsu, who is the best, humorous and critical advisor. Throughout his guidance, Based on the leading by him, I can connect with foreign researchers, make understandable slides, have good presentation, lead a team to cooperate with a company, and finally, complete my thesis. Second, I would like to thank professor Nalini Venkatasubramanian, professor Chun-Ying Huang, professor Kuan-Ta Chen, Yusuf Sarwar Uddin, Qiuxi Zhu, Fangqi Liu, and Guoxi Wang from UC Irvine, National Chiao Tung University, and Academia Sinica, respectively. It is appreciated to have your suggestions and contributions in our joint work. Third, I would like to thank my proposal commitees, who provide me many precious comments to polish my thesis. Fourth, I have to thank all the labmates. It is lucky to work with you. We stay overnight in the laboratory together, cellerbrate together, help one another when submission deadline is approaching, and so on. Moreover, I must to thank Ching-Ling Fan, Yi-Ying Huang, Pin-Chun Wang, Zou-You Hou and other friends who are always with me when I am depressed. Finally, I would like to thank my parents and sister who always support and help me so I can complete my works perfectly.

致謝

我要感謝所有在我念博士期間幫助我的人。首先我要謝謝我的指導教授，徐正炘教授，雖然他很嚴格不過他也很有趣，真的是很棒的教授。 徐老師給了我很多機會，讓我能夠在博士期間能夠學會怎麼和外國人溝通合作、 怎麼做出對的讓人看得懂的投影片、怎麼樣有流暢的報告技巧、怎麼樣帶領學弟妹完成一個工作、最終讓我完成了 我的畢業論文，無庸置疑徐老師是我最感謝的對象。接著，我也要謝謝在這個期間與我合作的教授朋友們， Prof. Nalini Venkatasubramanian, Prof. Chun-Ying Huang, Prof. Kuan-Ta Chen, Yusuf Sarwar Uddin, Qiuxi Zhu, Fangqi Liu, and Guoxi Wang，他們來自加利福尼亞大學爾灣分校，國立交通大學，以及台灣中央研究院， 謝謝各位合作者的建議和貢獻，讓我們可以產出許多有用的工作。然後，我要謝謝實驗室遇到的所有朋友， 我很幸運在我念博士期間有你們的陪伴，我們常一起在實驗室熬夜趕工，一起慶祝各種節日，我們互相幫助對方 當有人被論文期限追殺的時候。此外我尤其要感謝樊慶玲、黃宜瑩、王品淳、侯宗佑以及其他朋友們在我人生最低潮的時候 陪伴我度過，沒有你們我可能不會在這裡寫著感謝函。最後我要謝謝我的家人，總是在背後默默支持我， 雖然我總是在做著一些你們可能不太喜歡的決定，但是最後還是尊重著我的決定也一直關心著我，謝謝！

中文摘要

由於物聯網、人工智慧、巨量資料以及雲端運算的快速發展，促使了新興的應用出現，像是自駕車、擴增實境、智慧監控等等。 這些應用有著嚴格的延遲限制，需要大量的運算資源，以及不同的資源需求，像是感測、運算、網路、地點等等需求， 因此，開發者開始試著將各種不同的設備進行整合。這些設備可能來自雲、基地台、網路設備、以及各種終端設施， 而像這樣從雲到終端設施的整合平台，我們將他稱之為雲到物之連續性平台（cloud-to-things continuum platform）。 這樣的平台包含了兩個角色，一個是提供者一個是使用者，提供者負責建立平台並且提供應用給予使用者購買與使用， 在這篇論文裡面，我們根據這兩個角色設計了兩個聰明的部件，分別是全域優化部件（global optimizer）以及應用優化部件（application-specific optimizer）。全域優化部件是設計給提供者的，用來優化整個平台，這個部件考慮了使用者的服務品質需求、平台可用的資源量、應用所要的感測器等條件進行優化， 進而最大化整個平台能夠服務的使用者數量。而應用優化部件則是設計給使用者的，來優化正在運行的應用。 為了實現這兩個優化部件，我們解決了三個資源分配的問題，分別是應用佈建問題（application deployment problem）、 對延遲敏感之應用優化問題（delay-sensitive application optimization problem）、 以及對延遲不敏感之應用優化問題（delay-insensitive application optimization problem）。 對於應用佈建問題，我們設計了一套演算法，他可以在多項式時間（polynomial time）內解決問題， 並且有一數學證明其近似指標（approximation factor）。對於延遲敏感之應用優化問題， 我們的演算法能在多項式時間解出最佳解並有數學證明佐證其最佳性。對於延遲不敏感之應用優化問題， 我們則有一動態規劃算法解出最佳解，並有另一高效率啓發式算法來快速解決問題，雖然啓發式算法沒有數學佐證， 但與現存最新的其他算法比較可以有最少20%的優化。提供這樣雲到物的連續性、複雜且大規模的平台顯然是未來的趨勢， 但實現這樣的平台是昂貴又困難的，這也正是這篇論文的價值所在。有了我們的算法， 提供者可以更好地去預測可能需要的花費以及制定價格，而使用者們也因為有了針對應用優化的演算法們， 可以獲得更好的服務。除此之外我們也利用智慧路燈建立了一真實平台， 這樣的一

個真實的平台未來除了對學術研究以及業界測試有幫助，能夠進行更多更真實的實驗以外， 更重要的是能夠加速這樣一個複雜的雲到物的連續性平台開發。

Abstract

The rapid growth of Internet-of-Things (IoT), Artificial Intelligence (AI), big data, and cloud computing allow developments of next-generation applications, such as self-driving cars, wearable Augmented Reality (AR), and intelligent surveillance cameras. These applications have strict delay, large computing power, and diverse resources requirements, which make developers start to integrate geographically scattered resources. The resources have diverse types, including computation, networks, storage, and sensing. They are gathered from the cloud data centers to the end devices, which is referred to as cloud-to-things continuum. In this thesis, we propose an intelligent framework for the cloud-to-things continuum platform. The platform consists of two actors: a provider and users. The provider is responsible for building the platform and preparing applications, which are requested by the users. According to different actors in the platform, we design two intelligent components: global optimizer and application specific optimizer. They are designed for the platform's provider and users, respectively. The global optimizer is used to optimize the platform while considering different factors, such as users' QoS requirements, platform's available resources, and applications' sensor requirements. The application specific optimizer is designed for ongoing applications running on our platform to adapt to system dynamics. In order to realize the optimizers, we solve three resource allocation problems, including application deployment problem, delay-sensitive application optimization problem, and delay-insensitive optimization problem on top of the cloud-to-things continuum platforms. For the application deployment problem, our solution runs in polynomial time with a mathematically proved approximation factor. For the delay-sensitive optimization problem, our solutions run in polynomial time and result in optimal solutions. For the delay-insensitive problem, our dynamic programming based solution results in optimal results under delay-tolerable environment and our efficient solution outperforms state-of-the-art algorithms by at least 20%. Providing such complicated and large-scale cloud-to-things continuum platform is a clear trend, but it is expensive and challenging. With our optimization algorithms, the provider can readily estimate costs and decide prices with the given approximation factor, and the users can expect an optimal user experience with our application specific optimizers. In addition, we are building a real testbed by installing various resources on novel smart poles. The testbed allows researchers from academia and industries to test, develop, and evaluate latest technologies and algorithms of cloud-to-things continuum platforms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Internet-of-Things (IoT) and multimedia applications are popular all over the world. The number of IoT devices is expected to reach 50.1 billion by 2020 [24]. The IoT devices come with several sensors and actuators. The sensor data are collected through the Internet and the actuators perform some actions after receiving commands over the Internet, which makes our life easier. For example, we can connect our air conditioners to the Internet and ask the air conditioner to cool our rooms on our way home. Recently, researchers advocate combining IoT and analytics, such as the complicated Artificial Intelligence (AI) to make the IoT devices more intelligent [3, 9, 19]. For example, if cars become IoT devices with AI analytics, they can communicate among one another to realize self-driving and avoid traffic congestions. Moreover, nowadays, people use multimedia applications anytime and anywhere. We use multimedia applications when cooking, driving, working, and so on. Cisco's report predicts that video traffic alone will represent more than 80% of the Internet traffic by 2020 [8]. Currently, these IoT and multimedia applications are often served by cloud data centers. However, the huge amount of data generated from the applications congests the network traffics, overloads the data centers, and leads to very high communication costs. In addition, the cloud data centers are too far to satisfy delay-sensitive applications, such as remote rendering and augmented reality ones. Hence, we leverage a concept of fog computing to solve these problems. In this thesis, we adopt the *cloud-to-things continuum* proposed by OpenFog [39] as the definition of the fog computing. The details of the fog computing background is given in Chapter 2.

The cloud-to-things continuum integrates devices in the continuum from cloud to things. The devices, such as data centers, routers, cellular network base stations, WiFi access points, and personal computers are used to perform tasks in a distributed way. In this thesis, these devices are referred to as *fog devices*. The heterogeneous fog devices are placed at different locations with different capacities, e.g., end devices are closer to end users without powerful computing power; while data centers are powerful, but far away from end users. The cloud-to-things-continuum platforms allow us to run applications on suitable fog devices. For example, for delay-sensitive multimedia applications, we can run the applications closer to end users, such as on cellular base stations. In contrast, for complicated or delay-insensitive applications, we can run them in powerful data centers or distributedly run them on multiple nearby fog devices.

The cloud-to-things continuum platform has many benefits, such as delay-sensitive application support, privacy-sensitive data protection, mobility support, and network cost reduction. Due to these reasons, the market of fog computing is increasing. It was 20.28 million dollars in 2017 and is expected to have a ten fold increase in 2022 [14]. Many famous companies, such as Amazon and Microsoft start to produce products based on the concept of fog computing. For example, Amazon develops a device called AWS Snowball Edge [5], which is meant to be placed next to its end users. The users store their data on the device and install another product called AWS Greengrass [4] to analyze the data before sending them to the Amazon cloud. Microsoft's fog computing product, named Azure IoT Edge [33], is a platform used to virtualize IoT applications and move those virtualized applications from cloud to IoT devices. These products expand their services by moving the existing cloud services to end devices. Moreover, in a comprehensive fog computing survey [139], the authors report that many studies have proved that fog computing significantly reduces latency, network traffics, and energy consumptions. *Nonetheless, these commercial products and academic studies do not consider designing a comprehensive framework and optimizing cloud-to-things continuum platforms, which horizontally and vertically integrate heterogeneous fog devices.*

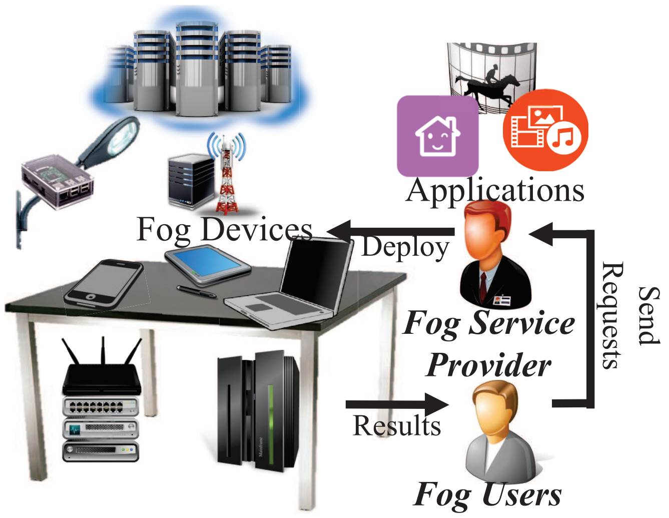


Figure 1.1: The overview of our cloud-to-things continuum platform.

In this thesis, we design an intelligent cloud-to-things continuum framework, which is inspired by OpenFog. From this, we solve three key resource allocation problems: (i) the application deployment problem [105], (ii) the delay-sensitive application optimization problem [104], and (iii) the delay-insensitive optimization [102] problem. These resource optimization problems are formulated, solved, and evaluated in this thesis. Our cloud-to-things continuum platforms contain two actors: providers and users, as illustrated in Fig. 1.1. The provider manages the heterogeneous fog devices, which offer diverse resources, including computation, networks, storage, and sensing to serve multiple users. The users send requests to the fog provider to buy some applications with various QoS targets. However, managing such a complicated platform with heterogeneous devices, diverse resources, and different applications used by multiple users is challenging. Hence, in this thesis, we aim to optimally allocate the resources to serve as many users as possible while user experiences are maximized.

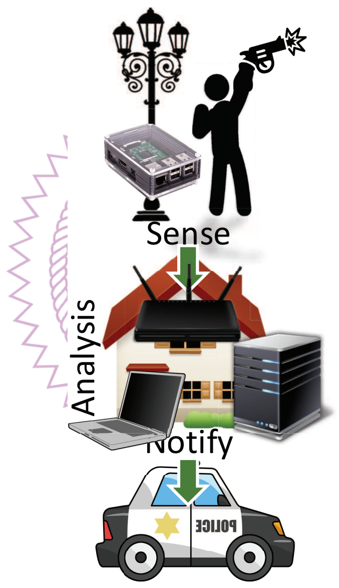


Figure 1.2: A sample usage scenario of our cloud-to-things continuum platform.

This cloud-to-things continuum platform can be used in various scenarios supporting diverse applications. Fig. 1.2 uses a robber as an illustrated example. The robber lives in a city with fog devices installed everywhere, such as on street lights. Assuming the robber tends to commit a crime at midnight (time). The police department (a user) therefore sends a request to the provider to request for a sound detection application between 22:00 to 4:00. One day, the robber fires a gun shot on a street. The fog devices with microphones on street lights detect the unusually large volume. The sound detection application is then deployed on nearby fog devices, which collaborate with one another to immediately analyze the sound and notify the police as soon as possible. This example shows a next-generation city security system, which improve citizens' safety with the cloud-to-things continuum platform. The platform can also allow many novel applications to be developed, such as wearable augmented reality, self-driving cars, and intelligent surveillance cameras. These applications require various resources and fast computations/communications, which can be provided by the heterogeneous fog devices.

1.1 Contributions

The contributions of this thesis are listed below:

- **Framework.** We extend the framework proposed by OpenFog to achieve optimizations from two points of view: the provider and users.
- **Global Optimization.** We design two application deployment algorithms to solve the deployment problem in the ideal and general cloud-to-things continuum. In the ideal setup, our algorithm gives an approximation factor; while in the general setup, our algorithm outperforms the state-of-the-art algorithms.
- **Delay-Sensitive Optimization.** We design an optimal bitrate adaptation algorithm to optimize a delay-sensitive application. We mathematically prove the optimality and evaluate the realtimeness of the algorithm.
- **Delay-Insensitive Optimization.** We design an efficient and an optimal dynamic programming based content delivery planning algorithm to optimize a delay-insensitive application. We prove the optimality and efficiency of our algorithms by comparing them with existing algorithms and some optimization solvers.
- **Outlook.** We envision and present a comprehensive ecosystem of the cloud-to-things continuum platforms.

1.2 Organization

The organization of this thesis is given below. We report background of fog computing, similar concepts, and a timeline of the concepts in Chapter 2. In Chapter 3, we show the proposed intelligent framework and differentiate it from the existing framework proposed by OpenFog. After describing the framework, we solve the application deployment problem in Chapter 4, the delay-sensitive optimization problem in Chapter 5, and the delay-insensitive optimization problem in Chapter 6. Finally, we discuss future work and conclusion in Chapters 7 and 8, respectively.

Chapter 2

Background

In this chapter, we introduce the background of this research area, including cloud computing, wireless sensor networks, IoTs, and fog computing. We also present a timeline at the end of this chapter to connect these similar concepts.

2.1 Cloud Computing

The concept of cloud computing is proposed by Compaq in 1996 [25] and the first large-scale commercial cloud computing service is created by Amazon in 2006 [50]. Cloud computing is used to efficiently utilize large-scale, integrated resources, mostly in a few data centers [154]. More specifically, cloud computing provides on-demand resources to multiple users, referred to as tenants. That is, the users can rent and use the resources from cloud service providers anytime through the Internet, in order to avoid the burden of maintaining and upgrading the hardware and software. For example, a start-up company requires some servers to launch their services. Buying the resources from the cloud, the start-up company does not need to: (i) buy the hardware devices, (ii) rent a space for the devices, (iii) build a cooling system, and (iv) upgrade the devices when the devices are too old or the size of the start-up company expands. Another example is that gamers do not need to worry about their GPU, RAM, CPU, and disk when you want to play the latest, and resource-intensive games, if the games run in the cloud.

Several services are provided by cloud computing, as illustrated in Fig. 2.1, which are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) targeting different users, such as students, engineers, and companies. The IaaS service, such as Amazon's EC2 [2] sells their hardware resources to their customers. The customers have to install the operating systems, build their platforms, and implement their applications themselves. The PaaS service does not directly sell the hardware resources. The cloud provider creates platforms upon the hardware resources and sells the platforms, such as Google App Engine [17], to the customers. Doing so, the customers are free from building the platform, so that the customers can focus on creating their applications. About the SaaS service , it means that the cloud provider implements some cloud applications and sell these applications, such as Google drive to the customers.

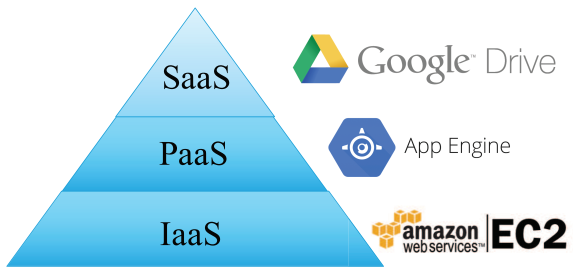


Figure 2.1: The services provided by cloud computing.

However, sharing the resources among multiple tenants may harm the user experience. Due to the interference of usage patterns among different users. In order to solve this problem, the cloud providers have to isolate the users using *virtualization* technologies, which is the key technology used in cloud computing. There are many virtualization technologies, such as VMware [48], KVM [29], Xen [52], and Docker [10]. They virtualize a single hardware resource to multiple virtualized resources for individual users. For example, when a virtualization technology is used on a server with a 8-core CPU, 16 GB RAM, and 1TB disk, we can virtualize the server into 8 virtualized devices with a 1-core CPU, 2 GB RAM, and 128 GB disk. Hence, the server can transparently serve 8 users while these users feel that they are served by isolated and dedicated devices. We note that the resources of the virtualized devices are allocated according to the requirements from users. In summary, the virtualization technologies result in many benefits, including on-demand resource provisioning, isolation, and multi-tenant supports.

Fig. 2.2 shows different approaches, including traditional servers, virtual machines, and containers, to share the resources. In Fig. 2.2(a), the traditional server has four components, including infrastructure, operating system, binaries/libraries, and applications. The infrastructure is the hardware components, such as CPU and RAM, which are managed by the operating system for the applications that requires some binaries/libraries running on the server. This approach is generally used by our personal computers and laptops. However, it is not suitable to the cloud because of the inflexibility due to lack of vitualization. In contrast, Fig. 2.2(b) shows the structure of virtual machines. There is a unique component, called hypervisor, for virtualizing resources and isolating applications. As shown in this figure, on top of the hypervisor, every application (user) has its own operating system and binaries/libraries, so that the users are isolated and protected.

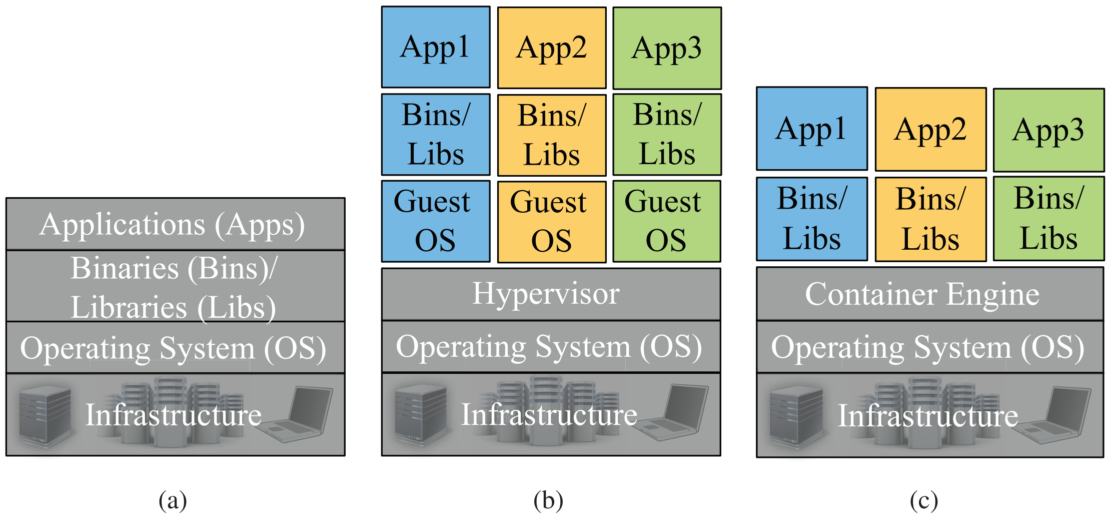


Figure 2.2: Various resource utilizing approaches: (a) traditional servers, (b) virtual machines, and (c) containers.

Comparing to virtual machines, Fig. 2.2(c) shows the containerization approach, which replaces the hypervisor with a container engine, which is also used to virtualize resources and isolate applications. Upon the container engine, every application has its own binaries/libraries, while sharing the same operating system. It trades some isolation abilities for lower virtualization overhead. As a approximation, every container requires less than 1 GB storage , while a virtual machine requires more than 10 GB storage. Moreover, we can start a container in a few seconds, while starting a virtual machine needs few minutes. Thus, we also refer to containerization as light-weight virtualization.

Leveraging the virtualization technologies, the cloud computing is mature now. Nowadays, there are quite a few cloud data centers places at different locations, e.g., in 2016, Amazon has 13 data centers located in 10 countries [18]. The number of Amazon users is 300 million [1]. These worldwide users can rent the cloud services from the nearest data centers to reduce the network latency and have better user experience. However, the users may not always stay at the same place. He/she may travel around the world. In order to keep the user experience, researchers start to study *distributed cloud* [82, 190]. The concept of distributed cloud is to collaborate the geographically distributed resources to improve the cloud services. For example,if a user living in Taiwan travels to America, we can leverage the power of distributed cloud to migrate the cloud services and corresponding data to the cloud data center located in America. This migration reduces the network latency and traffic, but the cloud data centers may be owned by different cloud providers, such as Google and Microsoft. Hence, in order to aggregate the cloud resources among different providers, the concept of *cloud federation* [123] is proposed. The goal of cloud federation is to generalize the distributed cloud to the cloud data centers owned by different cloud providers. Doing so, we can put all the cloud data centers into a single resource pool to have better cloud service quality.

Comparing our cloud-to-things continuum platforms with cloud computing platforms, cloud computing is a more centralized concept. The cloud resources are placed at the same place, which makes the cloud providers' life easier to manage the resources. However, it fails to active location awareness and mobility support. Although the distributed cloud and federate cloud partially cope with such limitations, they are still not close enough to the users for some applications, such as game streaming and connected cars. Moreover, cloud computing platforms do not have sensors and actuators, and thus are less suitable for some types of applications. We note that the detailed comparisons among other similar concepts will be summarized at the end of this chapter.

2.2 Wireless Sensor Networks and Internet of Things

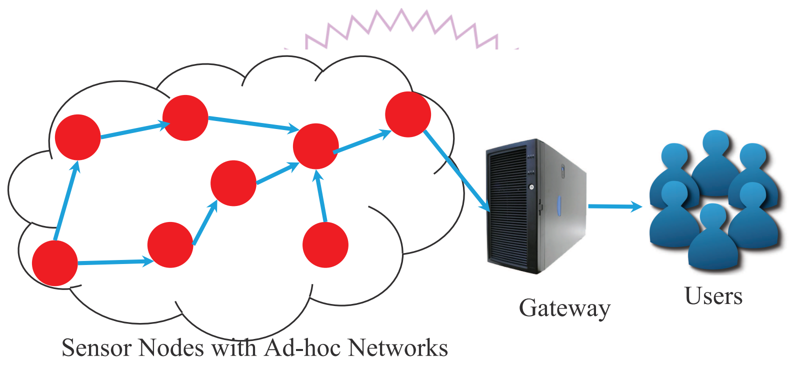


Figure 2.3: A basic WSN scenario.

Leveraging sensors to build services is not a new thing. During Cold War, militaries use acoustic sensors to build a sound surveillance system to detect underwater and surface enemies. After Cold War, the system is used to detect some events in oceans. However, the sensors used in the system are expensive and huge, so it is difficult to build a large-scale system with many sensors. In order to bridge the real world and the digital world to enrich our daily life, the Wireless Sensor Networks (WSN) is proposed [13,59] in 1980's. In WSN, we install multiple sensors on each node, which are small, inexpensive, and battery equipped. Because of that, WSN nodes can be readily, randomly, and massively distributedly anywhere to serve various applications.

These sensor nodes are usually deployed at difficult-to-access places, say in a desert, to monitor the environments. In order to make it sustainable at a lower maintenance cost, some sensor nodes harvest natural energy, such as solar power to charge the batteries. In addition, the sensor nodes have wireless communication abilities to send the collected data back to users. Fig. 2.3 shows a basic usage scenario of WSN, which consists of sensor nodes, a gateway, and users [184]. The data measured by the sensor nodes are aggregated by the gateway, which then sends the data to the users. In order to aggregate the data, the sensor nodes form an ad-hoc network to forward the measured data among the sensor nodes and towards the gateway. Moreover, in order to reduce the transmission data size, the sensor nodes also have data processing power to clean up useless data or extract useful data. After aggregating the data at the gateway, the data is further interpreted into meaningful information. This is because gateways usually have more computing power than sensor nodes. Additionally, the gateway has stable connections to the Internet, so the users can access the data from the gateway.

WSN realizes high scalability, high adaptability, and low cost ad-hoc networks [117]. We can easily add new or remove outdated sensor nodes. Ideally, we want to distribute many small and cheap sensor nodes anywhere around the world to achieve high scalability. The WSNs may adapt to network topologies changes , e.g., when we add new or remove outdated sensor nodes. For example, we can densely distribute enough sensor nodes installed air pollution sensors in our city to plot a detailed pollution map. This map allows us to figure out the sources of pollution and clean them up. When some of the sensor nodes are broken or new sensor nodes are added for a more detailed pollution map, the sensors nodes can automatically adapt to the new network topology and form a new ad-hoc networks to aggregate data. However, integrating different WSNs with different proprietary and non-proprietary approaches turns out to be challenging [134]. For example, in a city with three different WSNs using different protocols, including Zigbee [54], Bluetooth, and Z-Wave [53] will need to communicate over diverse wireless protocols. This process is tedious to integrate the sensor nodes for efficient collaboration.

To cope with the limitations, the concept of IoT [60, 156] appears. In IoT environments, the nodes are communicated through homogeneous IP networks to the Internet. We refer these nodes as IoT devices. Because these IoT devices are connected to the Internet, we can readily access the data collected by the IoT devices and cross reference them among one another. The IoT devices consist of various objects, such as sensors, actuators, and embedded devices. Generally speaking, any object connected to the Internet, such as a refrigerator is an IoT device. In addition to the networking side of difference, another important difference between IoT and WSN is the interaction between the users and sensor nodes/IoT devices. While the sensor nodes are responsible to collect sensor data, the IoT devices collect the sensor data and receive commands from users or service providers to perform some actions. For example, we can remotely ask our air conditioner that is connected to the Internet to cool off our house before we arrive home. These IoT devices influence our daily life. The IoT devices can collect our daily profiles and interact with us to improve our living quality. There are many IoT applications, such as smart grid, smart healthcare, and smart connected cars. The data collected from the IoT devices can be leveraged to create new applications or improve traditional applications. For example, if cars are all connected to the Internet, drivers can share the road conditions, so that drivers can be informed when an emergency event occurs.

The number of IoT devices is increasing and expected to be at 50.1 billion by 2020. Such magnanimous number of IoT devices means that it generates a huge amount of data. The data are critical and helpful to the most popular topic: Artificial Intelligence (AI). The AI market is 7345 million dollars in 2018 and expected to be 12.5 times by 2025 [41]. Therefore, more and more researchers and companies start to bind IoT and AI together to make the IoT applications more intelligent. Take connected cars as an example, we not only share the road condition data, but also analyze the car trajectories in a city. These data are sent to a remote powerful cloud data center for analysis. Based on the data, some AI algorithms can help the government to mitigate the traffic congestion in the city. However, the existing IoT applications collaborating with cloud data centers also suffer from some limitations. Because the cloud data center is far away from IoT devices, it limits us to provide delay-sensitive IoT applications. For example, we cannot send sensor data from cars to cloud data centers to provide a collision avoidance application. Otherwise, the drivers are bound to car accidents due to the excessive delay. This is the reason for Cisco to propose the concept of fog computing, which will be introduced in next section.

2.3 Fog Computing

Fog computing is a new concept with a set of diverse definitions. At first, Cisco proposes to leverage resources in between cloud and end devices. More specifically, Cisco [61] says: *"fog computing is a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional cloud computing data centers, typically, but not exclusively located at the edge of network."* After the term got coined, some academic studies [101, 107] generalize fog computing to include the resources from end devices, such as personal computers and Raspberry Pis. In 2018, OpenFog [39] consortium proposes the first fog computing standard [23] and defines fog computing as *"system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from cloud to things"*. In brief, Cisco defines the fog as a cloud close to the ground, which uses resources between end devices and cloud. After that, some studies integrate end devices into fog. Eventually, OpenFog defines fog as a link in the cloud-to-things continuum, which vertically and horizontally collaborates all devices in the cloud-to-things continuum.

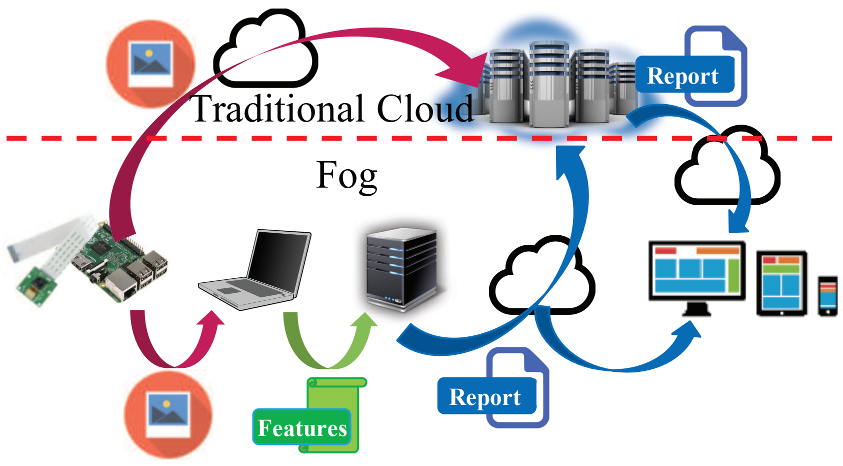


Figure 2.4: Benefits of fog computing.

Fog computing results in several benefits, such as reducing network latency, saving network traffic cost, protecting privacy, alleviating load of data centers, and efficiently leveraging heterogeneous devices. Taking an intelligent surveillance camera as an example, the camera captures images and analyzes it to detect emergent events. As show in Fig. 2.4, there are two approaches to realize it: (i) traditional cloud and (ii) fog computing approaches. In the traditional way, the captured images are sent to data centers for analysis. However, the image size is large, which consumes nontrivial network resources. The analysis is complicated, so sending cameras images to the data center may overload the data center. The data center is far away from end users, and is not suitable for delay-sensitive analysis to report emergent events. Moreover, your private images will be sent to the public cloud. These drawbacks can be solved by fog computing. Fog computing capitalizes nearby heterogeneous devices owned by the end users to pre-process the captured images. It first extracts features from of the images and sends the features to other devices for further analysis. The final reports and then sent to the end users. Because the image is pre-processed by the device owned by the end user, the original images are never sent to the data center. Therefore, fog computing reduces the network traffics, alleviates the load of data centers, and protects privacy. Besides, the final report is produced by some nearby devices, so the network latency is also shortened.

Fog computing has been studied for a few years, the benefits attract many researchers with thousands of published papers. There are six representative survey papers on fog computing listed in Table 2.1. These surveys are classified into three categories: (i) overall survey, (ii) specific target surveys, and (iii) challenge and future work surveys. For the overall survey, Perera et al. [148] introduce fog computing, ten usage scenarios, and required functionalities of an ideal fog computing scenario. Based on the functionalities,

Table 2.1: Fog Computing Survey

Year	Authors	Publications	Main Focus	
2017	Perera et al. [148]	ACM Computing Surveys	Overall Survey	
2017	Ni et al. [143]	IEEE Communications Surveys & Tutorials	Specific Target	Security & Privacy
2017	Mukherjee et al. [140]	IEEE Access		
2018	Mouradian et al. [139]	IEEE Communications Surveys & Tutorials		Algorithm
2018	Mukherjee et al. [141]	IEEE Communications Surveys & Tutorials		Network
2018	Mahmud et al. [133]	Internet of Everything (Book Chapter)	Challenges and Future Work	

the authors create a table to show the functionalities offered by individual papers. The specific target surveys can be further classified into: security & privacy [140, 143], algorithm [139], and network [141]. For the security & privacy [140, 143], the authors list the security & privacy issues that threat the fog computing platforms. The authors survey existing solutions to solve these issues. However, these solutions are still in early-stage, so that the authors list open challenges of the fog computing security & privacy protection. About the algorithms, Mouradian et al. [139] survey optimization algorithms designed for fog computing. The authors classify the algorithms based on different considered resource types and applications. Moreover, the authors propose six criteria to evaluate the fog-computing literatures. To design the fog networks, Mukherjee et al. [141] classify the literatures into three kinds of fog networking architectures. Moreover, the authors also survey mathematical models, such as latency models designed for fog computing. For the challenges of fog computing, Mahmud et al. [133] propose six representative challenges to classify the literatures. Their survey highlight the major challenges, and they also point out multiple future research directions.

Table 2.2 compares many concepts similar to fog computing, including cloud [25], distributed cloud [82], Cyber Foraging [157, 161], Cloudlet [170], and Mobile Edge Computing (MEC) [34]. We use four features: low latency, location awareness & mobility support, virtualization support, and high heterogeneity to differentiate them. Starting from the cloud computing, the cloud is composed by many centralized servers for virtualized resources. These servers however are far away from users, which lead to long network latency. Moreover, because the cloud places servers together, it is not location and mobility friendly. Besides, the devices located at different places may not be readily added to a cloud data center. Therefore, the heterogeneity of cloud is low. The cloud does not offer many types of resources. In summary, cloud only supports virtualization while other features are missing. Some researchers start to decentralize cloud to create distributed cloud. The distributed cloud data centers are placed at different locations, which collaborate with one another to reduce the latency, increase location awareness, and support mobility. However, it is still not close enough to the users to perfectly achieve low la-

Table 2.2: Fog Computing and Similar Concepts

	Low Latency	Location Awareness and Mobility Support	Virtualization Support	High Heterogeneity
Cloud [154]	X	X	√	X
Distributed Cloud [82]	△	△	√	X
Cyber Foraging [161]	√	△	X	△
Cloudlet [170] and MEC [34]	√	√	√	△
Fog [61]	√	√	√	√

tency and provide location and mobility support. In order to solve the problems, Cyber Foraging is proposed to offload resource-limited mobile devices' applications to nearby servers. However, Cyber Foraging does not consider virtualization, which increase difficulty to realize location awareness and mobility support. On the other hand, Cloudlet and Mobile Edge Computing (MEC) generalize the concept of Cyber Foraging and support virtualization. The difference between Cloudlet and MEC are the target devices running the mobile applications. Cloudlet runs the mobile applications on servers nearby end users over WiFi, while MEC runs the mobile applications on the servers at cellular base stations over cellular networks. For fog computing, it integrates all the devices in cloud-to-things continuum over diverse communication technologies. It is feasible and allows heterogeneous devices to join the cloud-to-things continuum platforms. This leads to higher flexibility and scalability to improve many existing applications and develop new applications. Moreover, some researchers, such as Mahmud et al. [133] classify Cloudlet and MEC to be subsets of fog computing.

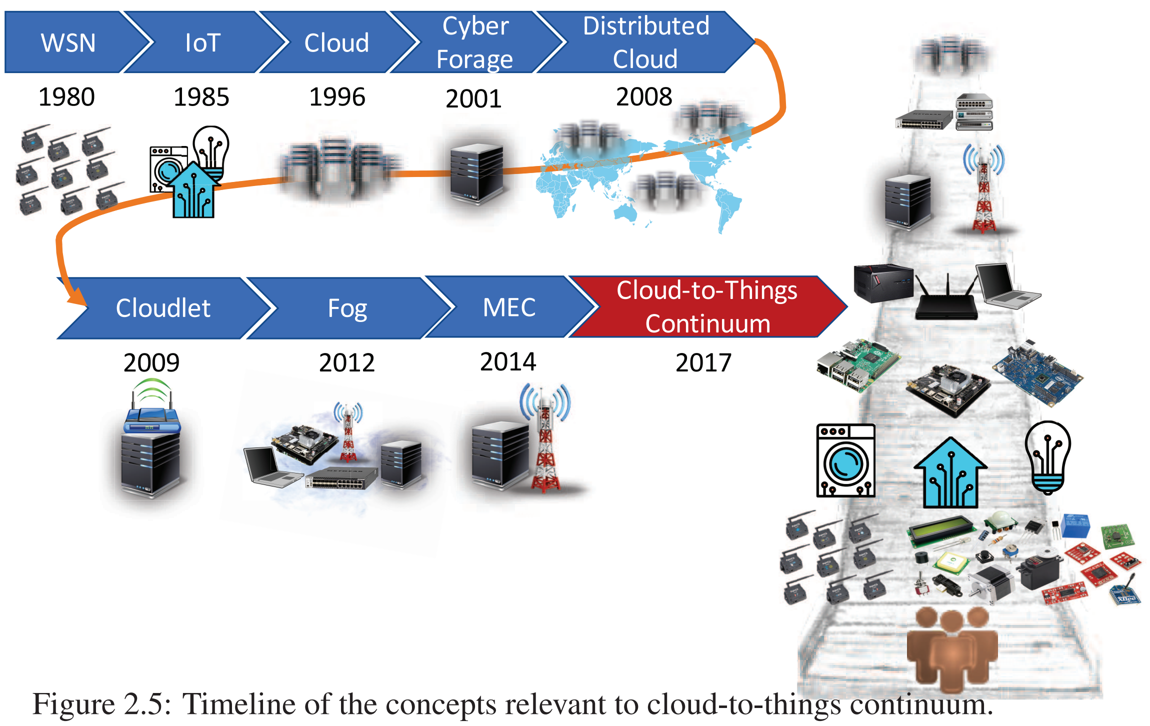


Figure 2.5: Timeline of the concepts relevant to cloud-to-things continuum.

2.4 Timeline

Fig. 2.5 shows a timeline of the aforementioned concepts. The WSN is proposed in 1980, which deploys many sensor nodes communicating via ad-hoc wireless networks to monitor our environments. In 1985, the IoT is proposed to connect various kinds of objects to the Internet to enable novel applications for enriching our daily life. In 1996, the cloud computing is proposed to aggregate resources from many servers to provide services through the Internet, e.g., offloading sensor data from IoT devices to the cloud for further analysis. In 2001, in order to solve the high network latency caused by far away cloud data centers, the Cyber Foraging is proposed to offload mobile applications to nearby servers. In 2008, cloud researchers propose the concept of distributed cloud for lower network latency. In 2009, the Cloudlet is proposed to generalize Cyber Foraging and serve mobile users as multiple tenants using virtualization technologies. The mobile users offload their applications to one-hop away Cloudlet servers through WiFi. In 2012, the fog computing is proposed to reduce the network latency of IoT applications by leveraging the edge devices. In 2014, MEC is proposed to utilize the servers placed at cellular base stations to run the mobile applications and improve mobile user experience. In 2017, the cloud-to-things continuum is proposed to bridge all the devices from cloud to things to make a more generalized platform supporting vertically and horizontally collaborations. In this thesis, we adopt, optimize, and realize the concept of cloud-to-things continuum.

Chapter 3

Cloud-to-Things Continuum Framework

In this chapter, we introduce an existing framework designed for cloud-to-things continuum platforms, analyze its limitations, propose a more intelligent framework and describe the optimization problems that need to be solved in it.

3.1 Potential and Limitations of Existing Framework

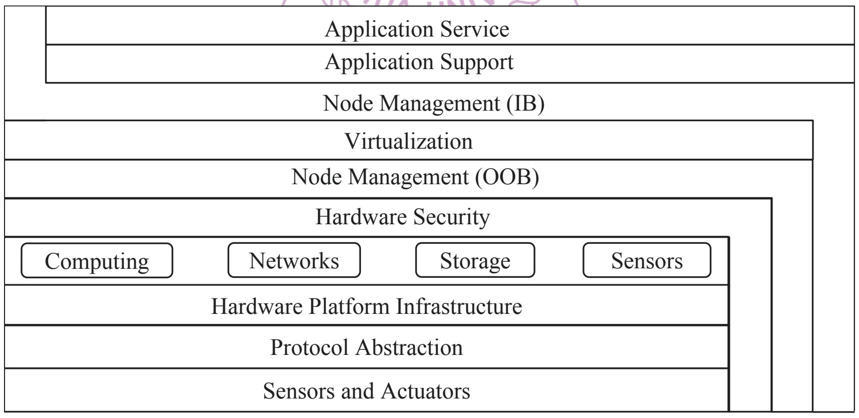


Figure 3.1: A cloud-to-things continuum framework proposed by OpenFog.

Fig. 3.1 shows a cloud-to-things continuum framework [40] proposed by OpenFog. This framework is the first fog computing standard accepted by IEEE Standards Association [23]. There are ten layers in the framework. To introduce the framework, let us imagine that we would like to add a fog device to a cloud-to-things continuum platform. From the bottom layer to the top layer, the fog device is installed with some sensors, such as PM 2.5 sensors and actuators. These sensors and actuators need protocols to facilitate communications. The device and the installed sensors/actuators are put in a hardware platform infrastructure, such as a chassis to protect them. Moreover, the device provides diverse resources, such as computing, networking, and storage. Before adding the devices to the platform, we need a hardware security module to authorize the device. After adding the device to our platform, we virtualize the resources, keep track of the device health, and monitor the resource usages through the management layers, including the In-Band (IB) node management and the Out-of-Band (OOB) node management layers. We then leverage the application support layer, which provides some libraries, such as OpenCV [38] and TensorFlow [45] to implement applications. The applications are then virtualized for being deployed to the fog device. *Although OpenFog is the first fog standard, it does not contain intelligent components for optimization.*

3.2 Proposed Intelligent Framework

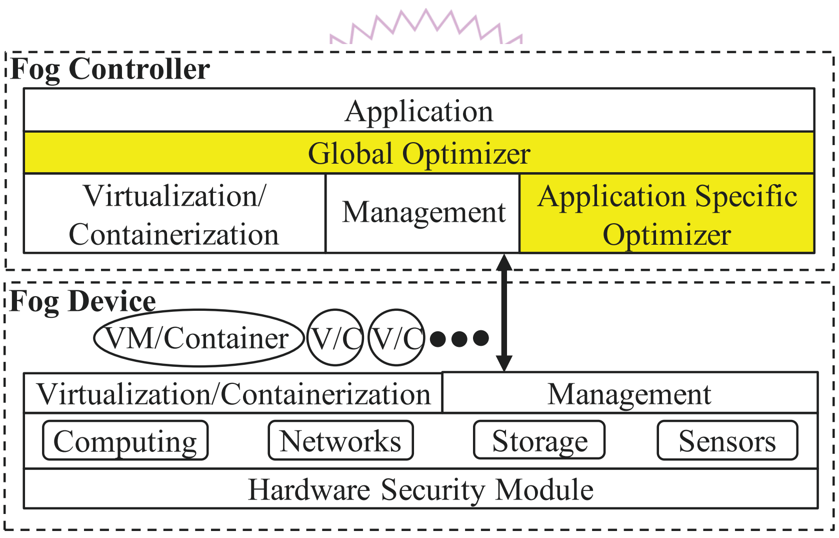


Figure 3.2: An intelligent cloud-to-thing continuum framework.

We enhance the Openfog's framework shown in Fig. 3.1 into the one shown in Fig. 3.2. There are two enhancements in the new framework: (i) a client-server architecture and (ii) intelligent optimization components (the yellow components). In the client-server architecture, we add a *fog controller* to manage fog devices that are identified by hardware security modules and responsible for running applications. We note that the fog controller is selected from fog devices when the scale of our platform goes larger, we can select multiple fog controllers to manage our platform with a hierarchical architecture. At the fog controller side, the fog providers prepare various applications that can be requested by the users. These applications are virtualized by the virtualization/containerization technologies, such as Docker [10], KVM [29], and VMware [48]. The virtualization/containerization technologies help us reserve resources and isolate different running applications, in order to achieve multi-tenancy support. When users request for applications, the fog controller deploys the applications through the management components installed on the fog devices and controllers. The virtualization/containerization component virtualizes the hardware resources, including computing, networking, storage, and sensing to serve the applications. The applications virtualized by the fog controller are launched as VMs or containers at the fog devices.

So far, we know how to request for an application, store the applications, deploy the applications, support multi-tenancy, and protect the security. In order to optimize the cloud-to-things continuum platforms, we add two components: global optimizer and application specific optimizer. The global optimizer is triggered before deploying the received requests to make deployment decisions based on required resources of the requests and available resources of the clients. The deployment decisions help the provider to serve as many users as possible for higher profits. After deploying the applications, in order to adapt to system dynamics and meet Quality-of-Service (QoS) requirements, the application specific optimizer optimizes the running applications by the perspectives of individual users.

3.3 Key Resource Allocation Problems

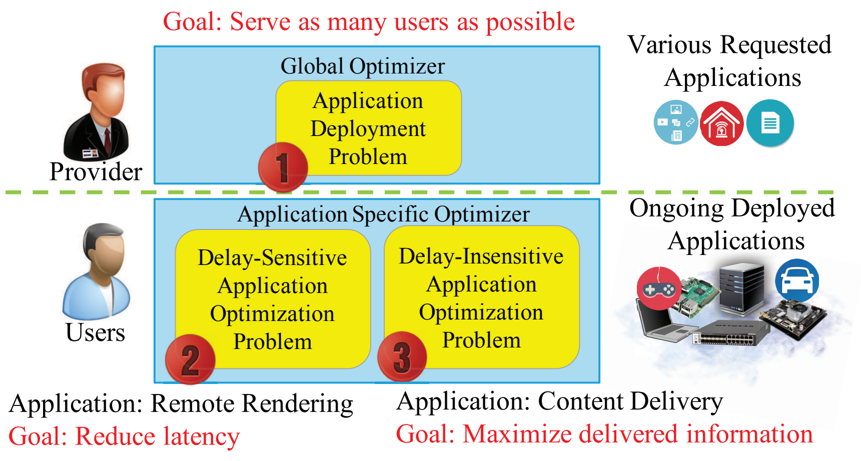

Goal: Serve as many users as possible
Global Optimizer
Application Deployment Problem
1
Provider
Various Requested Applications
Application Specific Optimizer
Delay-Sensitive Application Optimization Problem
2
Delay-Insensitive Application Optimization Problem
3
Users
Ongoing Deployed Applications
Application: Remote Rendering
Goal: Reduce latency
Application: Content Delivery
Goal: Maximize delivered information

Figure 3.3: Three critical research problems solved in this thesis for optimizing resource allocation in cloud-to-things continuum platforms.

Fig. 3.3, In this thesis, we solve three resource allocation problems to realize the global optimizer (designed for providers) and application specific optimizer (designed for users) to complete our intelligent cloud-to-things continuum framework. Fig. 3.3 illustrates the three resource allocation problems. In particular, we design an application deployment algorithm in Chapter 4, which solves the application deployment problem to help the service provide serve as many users as possible. The algorithm takes available resources of fog devices, QoS requirements of fog users, resource requirements of requested applications, and location requirements into considerations to carefully make the deployment decisions. After deploying the applications requested by the users, to adapt to system dynamics, we design an application specific optimizer to optimize the running applications and maximize the user experience. We classify the applications into two categories: (i) delay-sensitive and (ii) delay-insensitive applications. In this thesis, we select game streaming and multimedia content delivery as the representative applications for delay-sensitive and delay-insensitive ones, respectively, and optimize them. The details usage scenarios of these applications running on our cloud-to-things continuum platform and the solutions of the optimization problems are given in Chapters 5 and 6.

Chapter 4

Global Optimization

Our cloud-to-things continuum platform integrates heterogeneous fog devices located at different places to serve various applications, such as complicated wearable augmented reality, delay-sensitive game streaming, and delay-insensitive content delivery applications. However, such complicated platform also increases the difficulty to globally optimize our platform. Especially for the provider, it is challenging to make application deployment decisions while considering so many factors, such as different application resource requirements, user-specified QoS targets, heterogeneous resources, and so on. In this chapter, our goal is to help the provider serve as many users as possible by optimizing application deployment decisions. We note that in this chapter, we take *IoT analytics* as our representative applications because deploying IoT analytics is more difficult comparing to other applications. Deploying IoT analytics needs to consider many requirements, such as locations, sensors, and large amount of resources while serving other applications, we only need to take few factors into considerations. *Hence, in this chapter we refer to application deployments as analytics deployments.* Moreover, in order to deploy the complicated analytics on some resource-limited fog devices, we split the analytics into smaller *operators* and distributedly deploy the analytics on multiple fog devices.

Fig. 4.1 illustrates an usage scenario of deploying various applications. In the figure, the fog devices are distributed everywhere in cloud data centers, cars, home, and street light poles. The provider use four resource types, including computing, networking, storage, and sensing to serve three applications: (i) congestion prevention, (ii) game streaming, and (iii) home security. The congestion prevention is split into three operators and deployed in cars and poles to collaborate with one another to prevent traffic jams. The game streaming application is deployed on a cellular base station to allow gamer to play complicated games using low-end devices with very low delay. The home security application analyzes images captured by the surveillance camera installed on street lights to protect homes. This application is split into two operators. One recursively updates home

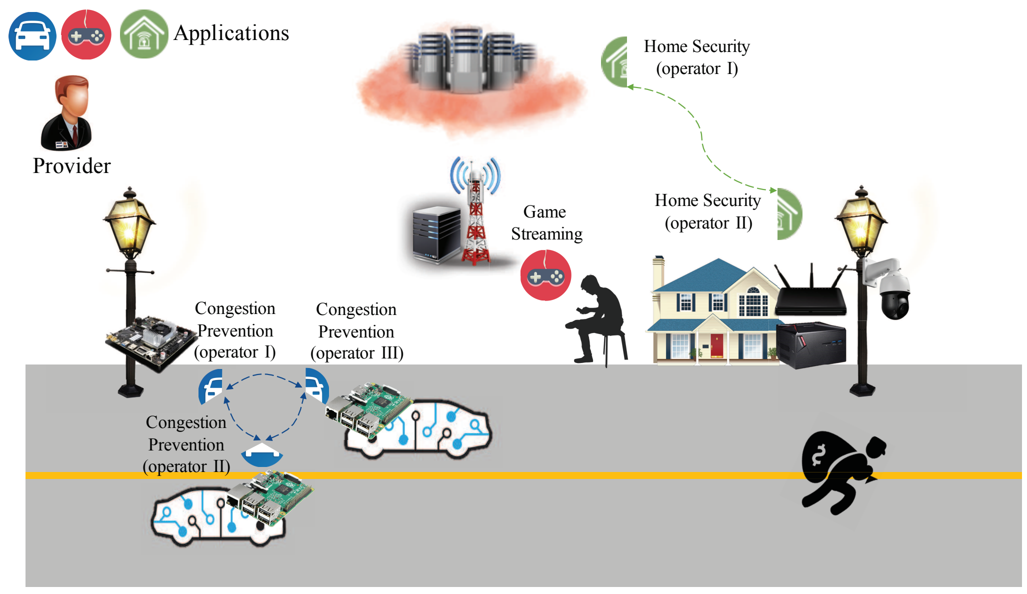

Applications
Provider
Home Security (operator I)
Home Security (operator II)
Game Streaming
Congestion Prevention (operator I)
Congestion Prevention (operator III)
Congestion Prevention (operator II)

Figure 4.1: A usage scenario of deploying various applications.

security analytics models (on the cloud) and another uses the models to detect emergency events.

The organization of this chapter is given below: we survey existing application (analytics) deployment and other similar deployment algorithms in Sec. 4.1, describe critical components that are required to solve the application deployment problem in Sec. 4.2, write problem statements of the application deployment problem and design our solutions in Sec. 4.3, evaluate our proposed algorithms in Secs. 4.4 and 4.5, and report discussions in Sec. 4.6.

4.1 Related Work

In this section, we introduce similar distributed deployment problems at different areas, including Cloud, Fog, Virtual Network Functions (VNF), and network embedding.

4.1.1 Distributed Deployment Problems

Our application (analytics) deployment problem is similar to several distributed deployment problems, such as virtual machine placement problems [151], network embedding problems [84], and Virtual Network Function (VNF) placement problems [96,127], which have been considered in different research areas. Based on how stringent the constraints are on nodes (devices) and links (networks), we place the four problem categories on the same axis in Fig. 4.2. The *virtual machine placement* problem in the cloud focuses

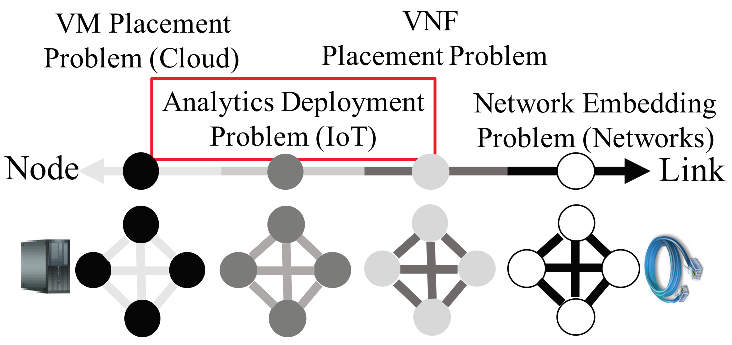

VM Placement Problem (Cloud)
VNF Placement Problem
Analytics Deployment Problem (IoT)
Network Embedding Problem (Networks)
Node
Link

Figure 4.2: Different distributed deployment problems.

on the node constraints, such as CPU power, while assuming links are oversubscribed. The *network embedding* problem focuses on link constraints, because network embedding virtualizes network resources for different customers. Between nodes and links, the *VNF placement* problem considers some node constraints, but pays more attention on link constraints. This is because the network functions have strict network requirements. Different from theses three problems [84, 96, 127, 151], our analytics deployment problem considers some link constraints, but emphasizes more on node constraints. This is because processing power of resource-limited fog devices close to data sources allows us to process the data sooner and reduce the network traffic more. Because of such difference, in the next section, we only survey the literature related to analytics deployment problems.

4.1.2 Application (Analytics) Deployment Problems

Sensor Constraints. When solving the analytics deployment problem, the sensor availability of fog devices has to be considered. There are Wireless Sensor Networks (WSNs) studies [58, 71, 130, 131, 147, 155, 185] that focus on sensor availability without link and node constraints. More specifically, most authors deploy operators on tree-structured graphs with different objectives. Tziritas et al. [168] start to consider node constraints in WSNs. The authors write the problem as an Integer Linear Programing (ILP) problem, in order to minimize network overhead. The problem is solved by a migration-based heuristic algorithm.

Node Constraints. There are several studies [68, 72, 114, 142, 158, 159, 165, 166, 186] that only take node constraints into considerations, when solving the analytics deployment problem. Huang et al. [114] and Chatzistergiou et al. [72] only consider CPU constraints. Huang et al. [114] propose a heuristic algorithm, which keeps finding local optimal solutions to make deployment decisions with minimum network usage. Chatzistergiou et al. [72] formulate an ILP problem, and solve it using a heuristic algorithm. Their algorithm sorts the devices by the communication overhead divided by processing power to minimize the network communication cost. Schilling et al. [159], Cardellini et al. [68], Skarlat et al. [166], Saurez et al. [158], and Skarlat et al. [165] consider CPU and RAM constraints. Schilling et al. [159] propose a migration-based heuristic algorithm to minimize the network usages. Cardellini et al. [68] propose a 4-step heuristic algorithm to minimize cost. Their 4-dimensional model considers network latencies, physical distances, device utilizations, and device availability. Skarlat et al. [166] propose a heuristic algorithm, which sorts the devices by latency and deploys each request to minimize network latency. Saurez et al. [158] propose a heuristic algorithm, which first checks the RAM and CPU constraints for feasible solutions and then sorts the devices by distance to make deployment decisions for short network latency. Skarlat et al. [165] also write the analytics deployment problem as an ILP problem. The authors use CPLEX [21] to optimally solve the problem for maximal total utilization. Zeng et al. [186] and Naas et al. [142] consider storage constraints of the problem. Zeng et al. [186] write the problem as a Mixed Integer Non-Linear Programming (MINLP) problem. The authors propose a three-stage algorithm, which optimizes I/O and computation time separately using linear relaxation in the first two stages. The third stage integrates the results from the first two stages for the final solutions to minimize I/O transmission and computation time. Naas et al. [142] write the problem as an ILP problem. The authors use CPLEX [21] to optimally solve the problem for minimal network latency.

Node and Link Constraints. There are a few papers [67, 69, 167] considering analytics deployment problems with node and link constraints at the same time. These studies consider CPU, RAM, and bandwidth constraints when formulating the problem. Taneja et al. [167] try to minimize network latency. The authors propose a heuristic algorithm, which sorts the devices by the device constraints and checks the device constraints to create feasible deployment decisions. Cardellini et al. [67, 69] write the problem as an ILP problem, which has a weighted sum objective function to capture multiple objectives at the same time. The authors use CPLEX [21] with 500 second limited running time to solve the problem.

Summary. In summary, the work in the literature usually formulates similar deployments problems as an ILP problem and designs heuristic algorithms or use commercial solvers, such as CPLEX [21] to solve it. In this thesis, we propose an algorithm to solve the problem in the ideal cloud-to-things continuum, with a performance guarantee in the format of an approximation factor. Moreover, our formulation and solution for the generalized cloud-to-things continuum carefully considers both node and link constraints. Compared to our work, only a couple of previous studies [67, 69, 167] simultaneously take node and link constraints into considerations and split complicated analytics into smaller operators for deployment. Hence, we chose these studies as baseline algorithms

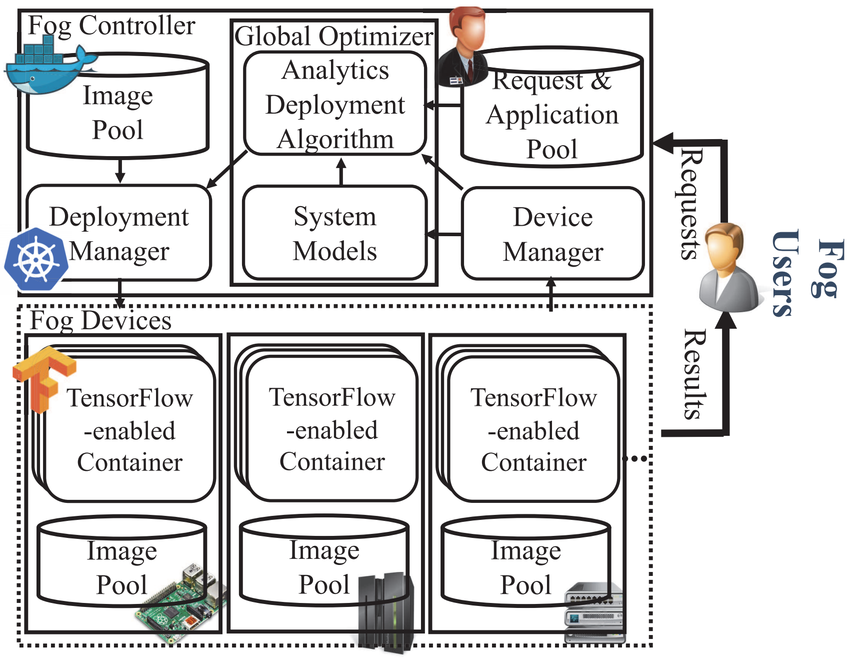


Figure 4.3: The global optimizer implementation in our proposed cloud-to-things continuum platform.

when evaluating our proposed algorithms in Sec. 4.5. Last, these studies make a strong assumption that the required resources of operators are known without conducting detailed measurements nor modeling resource consumptions. In contrast, we model the required resources of operators and the overhead caused by virtualization.

4.2 System Overview

In this section, we describe the implementation of our global optimizer in the cloud-to-things continuum platform, as illustrated in Fig. 4.3, and report details of major components bellow.

- **Device manager.** We implement a device manager to collect crucial device status, including: (i) the resource utilizations, e.g., CPU, RAM, and storage usages, (ii) the device locations, e.g., GPS readings, (iii) sensor availability and capability, e.g., existence of depth cameras at VGA resolution, and (iv) running analytics. While monitoring the running analytics, the device manager also collects the consumed resources, such as CPU and RAM of the analytics.

- **System models.** The system models are derived through real experiments, which use the device manager to collect required resources of analytics under different QoS levels (e.g., sampling rate and recognition accuracy). According to different analytics and required QoS levels as inputs, the derived models can predict the expected resources. The details of our system models are reported in Sec. 4.4.

- **Analytics deployment algorithms.** The algorithms solves the core research problem of this chapter. They consider the required resources from the system models and fog device status, such as resource capacities from the device manager to make decisions to deploy which analytics on which fog devices for all requests. The details of the analytics deployment problem, our proposed algorithms, and proofs are reported in Sec. 4.3.

- **TensorFlow-enabled container.** Because of the deployment decisions, the analytics will be dynamically deployed to fog devices. Hence, we containerize the analytics and store them as images in image pools. These images are initially created and stored in fog controller side's image pool. The images can be dynamically pushed to any fog devices and launched as running containerized analytics. We then store the images in the corresponding fog devices' image pool.

  Containers are light-weight virtual machines suitable to run on heterogeneous platform because the required resources to launch a container is much smaller compared to traditional virtual machines, such as KVM [29]. Moreover, in such heterogeneous cloud-to-things platform, some of fog devices are resource limited. The light-weight containers have very low overhead compared to traditional virtual machine and it can quickly (in few seconds) launch a container even on a low-end embedded device, such as Raspberry Pi. There are different kinds of container technologies, such as Docker [10] and LXC [32]. In this thesis, we adopt Docker as our container technology.

  We include TensorFlow [45] and its analytics libraries in Docker containers [10]. The TensorFlow provides many analytic libraries, which makes our life easier to implement some complicated analytics. Besides, TensorFlow has a graph-based programming style, which helps us easily split the complicated analytics into smaller operators.

- **Deployment manager** We build a deployment manager reading the deployment decisions to remotely launch specific images on chosen fog devices. If the fog devices' image pool have specific images, our deployment manager only sends launching command without pushing images from server to fog devices and supports other management tasks, such as killing, restarting, and migrating containers. There are many existing tools to help us manage Dockers, such as Kubernetes [28], SaltStack [42], and Docker Swarm [11]. In this thesis, we extend Kubernetes to

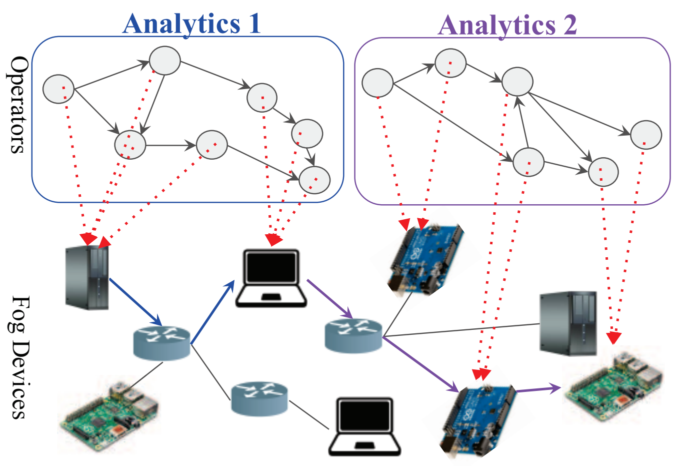


Figure 4.4: A sample analytics deployment in the cloud-to-things continuum platform.

implement our deployment manager.

4.3 Application (Analytics) Deployment Problem

In this section, we introduce our analytics deployment problem. We formulate the problem into two problem formulations, design two algorithms to solve them, and analyze their complexity and optimality. We note that in this chapter, we focus on complicated intelligent IoT analytics as target applications as mentioned in Chapter 4.

4.3.1 Problem-1: Comprehensive Formulation

Fig. 4.4 illustrates a sample analytics deployment in the cloud-to-things continuum platform. As shown in this figure, we split each IoT analytics into a few operators and create a Directed Acyclic Graph (DAG) of operators. When deploying operators, we consider the required edge resources in operator graphs and the link capacities between devices. Furthermore, we allow users to request for any sensors at any locations, and the fog service provider deploys the analytics across all devices that meet such requirements. In particular, the goal is to maximize the number of the admitted requests that satisfy the three requirements: (i) target QoS levels, (ii) required sensors and locations, and (iii) available resources.

Notations. Table 6.2 summarizes the symbols used in this chapter. Let $\mathbf{Q}$ be the set of requests. We write the target QoS levels of request q as s_q, for any $q \in \mathbf{Q}$. We let $\mathbf{V}$ be the set of fog devices and $\mathbf{U}$ be the set of resource types. The resource capacity of resource type u is represented as $R_{k,u}$ for any device $k \in \mathbf{V}$ and resource type $u \in \mathbf{U}$.

We let $G = <\mathbf{V}, \mathbf{E}>$ be the *device graph*, where $\mathbf{E}$ is a set of the links. Each physical

Table 4.1: Symbols Used Throughout this Chapter

Sym.	Description
$\mathbf{Q}$	Set of all requests
s_q	Target QoS of request q
$\mathbf{U}$	Set of considered resource types, such $\{$CPU, RAM, $\ldots\}$
$\mathbf{V}$	Set of fog devices
$\mathbf{V}_q$	Set of fog devices that can host request q
$R_{k,u}$	Capacity of resource u of device k
G	Device graph $<\mathbf{V}, \mathbf{E}>$
$\mathbf{V}_{q,i}$	Set of fog devices that can host operator i of request q
$\mathbf{E}$	Set of links
$\hat{G}_q$	Operator graph $<\hat{\mathbf{V}}_q, \hat{\mathbf{E}}_q>$ of IoT analytics of request q
$\hat{\mathbf{V}}_q$	Set of operators
$\hat{\mathbf{E}}_q$	Set of edges
B_l	Capacity of link l
$T_{(k,k'),l}$	Indicator of link l in on path from devices k to k'
$J_{q,i,i'}$	Indicator of operator i is parent of operator i'
$F(\cdot)$	Required resources of operators
$\bar{F}(\cdot)$	Required resources of edges among operators

link l has its capacity B_l. We use $T_{(k,k'),l}$ to describe whether link l is on the path from devices k to k'. To capture the sensor and location requirements for each operator, we write the available devices set as $\mathbf{V}_{q,i}$ for deploying operator i of request q. The path between two fog devices can be computed by any existing routing algorithms. We adopt the shortest paths if not otherwise specified.

We let $\hat{G}_q =<\hat{\mathbf{V}}_q, \hat{\mathbf{E}}_q>$ be the operator graph of the IoT analytics requested by q, for all $q \in \mathbf{Q}$. $\hat{\mathbf{V}}_q$ is the operator set and $\hat{\mathbf{E}}_q$ is the edge set. $F(\cdot)$ and $\bar{F}(\cdot)$ denote system models for predicting the required resources of the operators and edges among operators, respectively. We concertize the system models using our testbed in Sec. 4.4. We note that values of $F(\cdot)$, $\bar{F}(\cdot)$ and $R_{k,u}$ are normalized to total resources of our fog computing platform and thus are between 0% and 100%. For example, if a fog device has 8 GB RAM and the total amount of RAM of our fog computing platform is 80 GB, the RAM resource of the fog device is 10%. In order to represent the parent-child relationship between any two operators i, i', we use $J_{q,i,i'}$. More specifically, $J_{q,i,i'} = 1$ iff operator i is the parent operator of i'.

Lemma 1 (Hardness)**.** *The Analytics Deployment Problem-1 is NP-hard.*

Proof. The Analytics Deployment Problem-1 can be reduced from the NP-hard Multiple Knapsack Problem (MKP). The MKP problem puts as many objects as possible into multiple knapsacks with diverse capacities. If we let $|\hat{\mathbf{V}}_q| = 1$, $|\mathbf{U}| = 1$, and $B_l = \infty$, we can map knapsacks to fog devices and objects to requests without any network constraints.

Moreover, the value of each request is 1 and the weight is R_k. In this way, we reduce the MKP problem to our analytics deployment Problem-1 in polynomial time. The rest of the proof is straightforward. $\square$

Formulation. We formulate Problem-1 into an ILP problem as follows:

$$\max \sum_{q \in \mathbf{Q}} p_q \tag{4.1a}$$

$$st: z_q = \sum_{i \in \hat{\mathbf{V}}_\mathbf{q}} \sum_{k \in \mathbf{V}_{q,i}} x_{q,i,k} / |\hat{\mathbf{V}}_\mathbf{q}| \;\; \forall q \in \mathbf{Q}; \tag{4.1b}$$

$$z_q - 1 < p_q \leq z_q \;\; \forall q \in \mathbf{Q}; \tag{4.1c}$$

$$\sum_{k \in \mathbf{V}_{q,i}} x_{q,i,k} \leq 1 \;\; \forall q \in \mathbf{Q}, i \in \hat{\mathbf{V}}_\mathbf{q}; \tag{4.1d}$$

$$\sum_{q \in \mathbf{Q}} \sum_{i \in \hat{\mathbf{V}}_q} \dot{F}(\cdot) x_{q,i,k} \leq R_{k,u} \quad \forall k \in \mathbf{V}_i, u \in \mathbf{U}; \tag{4.1e}$$

$$\sum_{q \in \mathbf{Q}} \sum_{i \in \hat{\mathbf{V}}_q} \sum_{i' \in \hat{\mathbf{V}}_q} \sum_{k \in \mathbf{V}_{q,i}} \sum_{k' \in \mathbf{V}_{q,i'}} y_{q,(i,i'),(k,k')} J_{q,i,i'} T_{(k,k'),l} \dot{F}(\cdot) \leq B_l \;\; \forall l \in \mathbf{E}; \tag{4.1f}$$

$$x_{q,i,k} = \sum_{k' \in \mathbf{V}_{q,i'}} y_{q,(i,i'),(k,k')} \quad \forall q \in \mathbf{Q}, i \in \hat{\mathbf{V}}_q, i' \in \hat{\mathbf{V}}_q, k \in \mathbf{V}_{q,i}; \tag{4.1g}$$

$$x_{q,i',k'} = \sum_{k \in \mathbf{V}_{q,i}} y_{q,(i,i'),(k,k')} \quad \forall q \in \mathbf{Q}, i \in \hat{\mathbf{V}}_q, i' \in \hat{\mathbf{V}}_q k' \in \mathbf{V}_{q,i}; \tag{4.1h}$$

$$p_q, x_{q,i,k} \in \{0,1\} \;\; \forall q \in \mathbf{Q}, i \in \hat{\mathbf{V}}_\mathbf{q}, k \in \mathbf{V}_{q,i}. \tag{4.1i}$$

In this formulation, $x_{q,i,k}$ is the decision variable, where $x_{q,i,k} = 1$ iff operator i of request q is deployed on device k. The objective function in Eq. (4.1a) counts the number of satisfied requests, where an intermediate variable $p_q = 1$ iff request q is satisfied. Eqs. (4.1b) and (4.1c) make sure that when request q is satisfied, all the operators of request q are deployed. Eq. (4.1d) makes sure that each operator is only deployed once. Eq. (4.1e) ensures that the required resources of the deployed operators do not exceed the resource capacities of individual devices. Eq. (4.1f) makes sure that the required bandwidth of the deployed operators does not exceed the link capacity. Eqs. (4.1g) and (4.1h) define an intermediate variable $y_{q,(i,i'),(k,k')}$, which is 1 iff edge (i, i') of request q is mapped to physical path (k, k').

Eq. (4.1f) is the most critical part in this formulation. Different from state-of-the-art articles [67, 69, 167], which also formulate the analytics deployment problem with splitable analytics, our formulation carefully considers network constraints of every link. More specifically, Taneja et al. [167] considers network constraints of each network interfaces installed on fog devices and Cardellini et al. [67, 69] considers network constraints of each paths. These approaches may waste network resources or overload some network links.

Inputs: Request info, such as target QoS s_q, analytics info, such as operator graphs $\hat{G}_a$, device info, such as device resource capacity $R_{k,u}$, and link capacity B_l.
Output: The deployment decision $x_{q,i,k}$.

```
1:  let $\hat{C}_q$ be the scarcest resource $u'$ required by $q$
2:  sort Q by $\hat{C}_q$ in asc. order
3:  for $q \in \mathbf{Q}$ do
4:      let M to store pairs of devices, including source device $\hat{k}$ and destination device $\check{k}$, which
            are feasible to launch source $\hat{i}$ and destination operators $\check{i}$ of request $q$, respectively
5:      sort M on number of hops from $\hat{k}$ to $\check{k}$ in asc. order
6:      sort Pairs in M with the same number of hops on network latency in asc. order
7:      for $(\hat{k}, \check{k}) \in \mathbf{M}$ do
8:          let $x_{q,\hat{i},\hat{k}} = 1$
9:          let $x_{q,\check{i},\check{k}} = 1$
10:         let W be all the deployed operators
11:         add $\hat{i}$ and $\check{i}$ to W
12:         while W is not empty do
13:             pop an operator $i'$ from W
14:             let $\mathbf{N}'_i$ be the child operators of $i'$
15:             let $H$ be hop limit
16:             let $\mathbf{D}_h$ be devices that are $h$ hops to the device running operator $i'$
17:             sort $\mathbf{D}_h$ on the number of hops to $\check{k}$
18:             for $h = 0, 1, 2, \dots, H$ do
19:                 for $i \in \mathbf{N}_{i'}$ do
20:                     for $k \in \mathbf{D}_h$ do
21:                         if Eqs. (4.1d) to (4.1f) are satisfied then
22:                             let $x_{q,i,k} = 1$
23:                             add $i$ to W
24:         if operators of request $q$ are not all deployed then
25:             clear all deployed operator of request $q$
```

Figure 4.5: The pseudocode of our proposed SSE algorithm.

Our Proposed Algorithm and Analysis. Since Problem-1 is NP-hard, we propose an efficient greedy algorithm based on three intuitions (steps):

- **Scarcest resource first.** Because requests demand for different resource types at the same time, the scarcest resource becomes the limiting factor. To satisfy as many requests as possible, we identify the scarcest resource (after normalization), sort requests in the ascending order on the scarcest resource, and then consider the requests consuming less scarcest resource earlier. The result of this step is a sorted set of undeployed requests.

- **Shortest path first.** For each request, we consider all pairs of feasible fog devices for the source and destination operators. Because shorter paths generally lead to lower overall network loads, shorter end-to-end latencies, and less processing/forwarding overhead, we select the pair of fog devices with the least number of hops between the source and destination operators. To break ties, we select the pair with shortest network latency. The result of this step is the deployment decisions of the source and destination operators.

- **Early feature extraction.** Upon the source and destination operators being deployed, we start to deploy the remaining operators. Since the operators in typical IoT analytics extract increasingly higher-level features that are generally more compact, deploying the operators closer to the parent operator reduces the network workload and the transfer time. We employ a system parameter H and only consider deploying a child operator on fog devices within H hops from its parent operator. H is typically a small constant, which allows us to reduce our time complexity. The results of this step are the deployment decisions of operators other than the source and destination ones.

We refer to our proposed algorithm as SSE algorithm, after the initials of the three intuitions. Fig. 6.5 gives its pseudocode. Lines 1–2 implement the first intuition, i.e., sorting the requests in $\mathbf{Q}$ by the scarcest resource type. Lines 4–14 realize the second intuition, where we sort the pairs of source and destination devices by their path lengths. Lines 15–20 implement the third intuition to consider the fog devices that are closer to source device first. Lines 21–22 check the constraints and make the final decisions.

Lemma 2 (Correctness and Time Complexity)**.** *SSE algorithm gives a feasible solution in polynomial time.*

Proof. The correctness is ensured by lines 20–21. For complexity, we first sort the requests $\mathbf{Q}$ by the scarcest resource type in lines 1–2, which has a complexity of $O(|\mathbf{Q}| \log |\mathbf{Q}|)$. The for-loop starting from lines 3 to 19 goes through all the requests. For each request q, we go through all possible pairs of devices sorted by the number of hops for deploying the source and destination operators in lines 4–6. We then create $\mathbf{W}$ to store and go through all the operators $i' \in \mathbf{W}$ until $\mathbf{W}$ is empty in lines 9–11. In lines 13–22, We create $\mathbf{D}_h$ to store fog devices and $\mathbf{N}'_i$ to store child operators of i'. We then sort $\mathbf{D}_h$ by the number of hops to their destination devices and go through the sorted devices and child operators to deploy the operators without violating Eqs. (4.1d) to (4.1f). We let $K = |\mathbf{V}|$, $I = \max_{a \in \mathbf{A}}(|\hat{\mathbf{V}}_a|)$, and $P = |\mathbf{Q}|$. The complexity of lines 3–19 is $O(P(K^2 \log K^2 + K^2 I(K \log K + HIK)))$, which dominates the complexity of our algorithm and is polynomial. We notice that I (the maximal number of operators in each request) and H (the hop limit) are typically bounded integers. Under this assumption, the time complexity can be written as $O(PK^3 \log K + PK^2 \log K^2)$. $\square$

4.3.2 Problem-2: Tractable Formulation

In Sec. 4.3.1, we propose an efficient SSE algorithm to solve the Analytics Deployment Problem-1. However, Problem-1 is too detailed for any performance guarantee. In this section, we consider a more tractable problem, which is referred to as Analytics Deployment Problem-2. This problem employs the following four approaches: (i) grid maps, (ii) link constraint transformation, (iii) proportional resource levels, and (iv) fine-grained operators to gain more tractability. The grid maps ask each fog user to select a grid that contains his/her requested locations and sensors when submitting every request. By doing so, the location/sensor constraints are replaced by the grid map. The link constraint transformation associates the link constraints with the node constraints; in particular, each node gets an actual network constraint from the adjacent link with the least network resource. The proportional resource levels refer to fog devices scale proportionally in all resource types. For example, a lower-end fog device may have a 2-core CPU with 2 GB RAM, and a higher-end one may have a 4-core CPU with 4 GB RAM. Last, the fine-grained operators means that each IoT analytics can be split into many operators, and thus can be split into arbitrary small subsets of operators. We note that there exist several technologies enabling fine-grained decomposition, including distributed TensorFlow [55], method-level offloading [90, 122], and stream processing [97].

Lemma 3 (Hardness)**.** *The Analytics Deployment Problem-2 is NP-hard.*

Proof. We reduce the NP-hard Multi-Dimensional Knapsack Problem (MDKP) to our Analytics Deployment Problem-2. The MDKP problem puts as many objects as possible into a single knapsack with multiple types of constraints, such as weights and capacities. If we put a single fog device in each grid, the MDKP problem can be readily reduced to our Analytics Deployment Problem-2 in polynomial time. $\square$

Inputs: Request info, such as target QoS s_q, and device info, such as resource capacity $R_{k,u}$.

Output: The deployment decision $x_{q,k}$.

```
1:  let $M_q$ is the total required resources of request $q$
2:  sort Q by $M_q$ in asc. order
3:  for $q \in \mathbf{Q}$ do
4:      let $c_u$ as total required resources $u$ of deployed operators of $q$
5:      for $k \in \mathbf{V}_q$ do
6:          let Remaining $u$ resource of device $k$ is $R'_{k,u}$
7:          for $u \in \mathbf{U}$ do
8:              if $\dot{F}(q, u, \cdots) - c_u$ is more than $R'_{k,u}$ then
9:                  let $x_{q,k} = R'_{k,u}/(\dot{F}(q, u, \cdots))$
10:                 let $c_u = c_u + \dot{F}(q, u, \cdots)$
11:                 go to Line 5
12:             else
13:                 let $x_{q,k} = (\dot{F}(q, u, \cdots) - c_u)/\dot{F}(q, u, \cdots)$
14:     for $u \in \mathbf{U}$ do
15:         if $c_u \neq \dot{F}(q, u, \cdots)$ then
16:             remove all the deployed operator of $q$
```

Figure 4.6: The pseudocode of our APX algorithm.

Formulation. We formulate our Problem-2 as the following Integer Linear Programming (ILP) problem:

$$\max \sum_{q \in \mathbf{Q}} p_q \tag{4.2a}$$

$$st: z_q = \sum_{k \in \mathbf{V}_q} x_{q,k} \ \ \forall q \in \mathbf{Q}; \tag{4.2b}$$

$$z_q - 1 < p_q \leq z_q \ \ \forall q \in \mathbf{Q}; \tag{4.2c}$$

$$0 \leq z_q \leq 1 \ \ \forall q \in \mathbf{Q}; \tag{4.2d}$$

$$\sum_{q \in \mathbf{Q}} F(q, u, \cdots) x_{q,k} \leq R_{k,u} \ \ \forall k \in \mathbf{V}_q, u \in \mathbf{U}; \tag{4.2e}$$

$$0 \leq x_{q,k} \leq 1 \ \ \forall q \in \mathbf{Q}, k \in \mathbf{V}_q; \tag{4.2f}$$

$$p_q \in \mathbb{Z} \ \ \forall q \in \mathbf{Q}. \tag{4.2g}$$

In this formulation, $x_{q,k}$ is our decision variable between 0 and 1. Say, if $x_{q,1} = 0.4$ and $x_{q,2} = 0.6$, it means that we split request q into two operators, which contain 40% and 60% of its analytics, respectively. Moreover, we run them on devices 1 and 2. The objective function in Eq. (6.1a) counts the number of satisfied requests, where the intermediate variable $p_q = 1$ iff request q is satisfied. A request is satisfied iff: (i) the decomposed request are fully deployed, which is checked by Eqs. (4.2b)–(4.2d) and (ii) the required resources of the request is reserved and the required resources of the deployed requests do not exceed the available resources of the fog devices, which is ensured by Eq. (4.2e).

Approximation Algorithm. We propose an APproXimation (APX) algorithm with a $\frac{1}{|\mathbf{U}|}$ approximation factor in the following. Its pseudocode and complexity analysis are given in Fig. 4.6 and Lemma 4, respectively. Our APX algorithm iteratively performs 3 steps: (i) request selection, (ii) fog device selection, and (iii) request decomposition. In the first step, we select a request q consuming the least total resources, i.e., $\min_{q\in\mathbf{Q}}\sum_{u\in\mathbf{U}}F(q,u,\cdots)$. We then select a fog device by round robin at the location of the request, to run this IoT analytics. Finally, when the IoT analytics is not satisfied by a single fog device, we decompose the IoT analytics into two operators, where one of them is the largest possible operator. That operator is then run on the selected device. The remaining portion of the IoT analytics is regarded as a new request. We repeat these three steps until all the requests are deployed or the remaining resources of fog devices cannot satisfy any additional request.

Lemma 4 (Time Complexity)**.** *Our APX algorithm runs in polynomial time.*

Proof. We first sort the requests by line 2, which has complexity $O(|\mathbf{Q}|\log|\mathbf{Q}|)$. The for-loop starting from lines 3, 5, and 7 go through all requests, devices, and types of resources, and have a total complexity $O(|\mathbf{Q}||\mathbf{V}||\mathbf{U}|)$. The complexity is therefore $O(|\mathbf{Q}|(|\mathbf{V}||\mathbf{U}|+\log|\mathbf{Q}|))$. Hence, the APX algorithm runs in polynomial time. □

Approximation Factor. To set up the stage, we first design another algorithm that results in no fewer deployed requests than the optimal results, but may violate some constraints (and thus are infeasible). We refer to this algorithm as *OPT'*. This algorithm shares the same intuitions with our APX algorithm, but has two major differences. First OPT' sums up each resource type of all the fog devices in the same grid. Second, OPT' further sums up all the resource types of a grid into a single number. For example, a request requires 30% CPU and 10% RAM, which results in 40% resources in total. Therefore with OPT', the request will be served by the merged device of a grid with 40% total resource. Our goal is to show that OPT' gives an *upper bound* on the number of satisfied requests. Concretely, the OPT' algorithm deploys the requests to the merged device of each grid in the ascending order of requests' total resource needs. In the following, we formally prove that the OPT' algorithm gives an upper bound in Lemma 5. We then give the approximation factor of our APX algorithm in Lemma 6.

Lemma 5 (Upper Bound)**.** *The OPT' algorithm results in an upper bound on the number of satisfied requests, which may be infeasible in the original problem.*

Proof. Let OPT be the optimal algorithm, which results in $\dot{H}$ satisfied requests $(\dot{r}_1, \dot{r}_2, \cdots \dot{r}_{\dot{H}})$. In contrast, OPT' deploy H' requests $(r'_1, r'_2, \cdots r'_{H'})$. Without loss of generality, the resulting solutions are enumerated by the total required resources. We have the following inequalities:

$$\sum_{h=1}^{\dot{H}} \sum_{u \in \mathbf{U}} F(\dot{r}_h, u, \cdots) \leq \sum_{k \in \mathbf{V}} \sum_{u \in \mathbf{U}} R_{k,u}; \tag{4.3}$$

$$\sum_{h=1}^{H'+1} \sum_{u \in \mathbf{U}} F(r'_h, u, \cdots) > \sum_{k \in \mathbf{V}} \sum_{u \in \mathbf{U}} R_{k,u}; \tag{4.4}$$

$$\sum_{u \in \mathbf{U}} F(r'_h, u, \cdots) \leq \sum_{u \in \mathbf{U}} F(\dot{r}_h, u, \cdots) \ \forall 1 \leq h \leq \dot{H}. \tag{4.5}$$

Eq. (4.3) holds because the total resources used by OPT are not greater than the total resources of the fog devices. Eq. (4.4) is true because if OPT' deploys one more requests, it will certainly overload the devices. Eq. (4.5) shows that when the deployed requests are sorted in the ascending order on the total resources, the pre-request resources consumed by OPT' does not exceed that of that by OPT in the same position. Assuming $\dot{H} > H'$, adding Eqs. (4.4) and (4.5), we have:

$$\sum_{h=1}^{\dot{H}} \sum_{u \in \mathbf{U}} F(\dot{r}_h, u, \cdots) > \sum_{k \in \mathbf{V}} \sum_{u \in \mathbf{U}} R_{k,u}, \tag{4.6}$$

which contradicts to Eq. (4.3). Hence, we have $\dot{H} \leq H'$. □

Lemma 6 (Approximation Factor)**.** *Our APX algorithm results in a $\frac{1}{|\mathbf{U}|}$ approximation factor.*

Proof. Our APX algorithm satisfies H requests $(r_1, r_2, \cdots r_H)$. Since the resource capacities of devices are proportional, we have $R_{k,u} = R_{k,u'} \ \ \forall u, u' \in \mathbf{U}$. Moreover, each device stops serving operators only when at least one of resource types is used up, and hence we have:

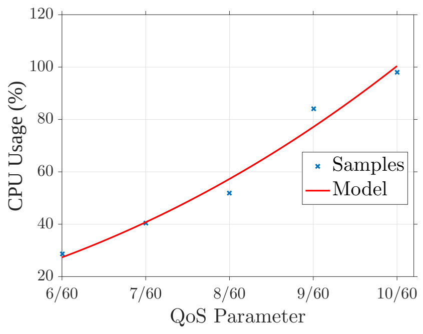
$$\sum_{h=1}^{H} \sum_{u \in \mathbf{U}} F(r_h, u, \cdots) \geq \sum_{k \in \mathbf{V}} \sum_{u \in \mathbf{U}} R_{k,u}/|\mathbf{U}|. \tag{4.7}$$

Besides, the total resources consumed by the OPT' algorithm do not exceed the total resources of fog devices. That is:

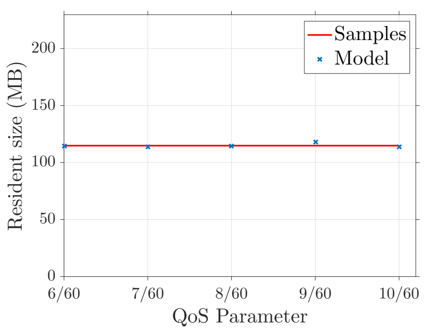
$$\sum_{h=1}^{H'} \sum_{u \in \mathbf{U}} F(r'_h, u, \cdots) \leq \sum_{k \in \mathbf{V}} \sum_{u \in \mathbf{U}} R_{k,u}. \tag{4.8}$$

Based on Eq. (4.7), we have:

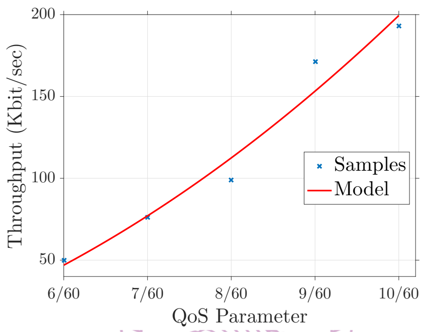
$$H \frac{\sum_{h=1}^{H} \sum_{u \in \mathbf{U}} F(r_h, u, \cdots)}{H} \geq \sum_{k \in \mathbf{V}} \sum_{u \in \mathbf{U}} R_{k,u}/|\mathbf{U}|. \tag{4.9}$$



(a)



(b)



(c)

Figure 4.7: The system models of (a) CPU load, (b) RAM usage, and (c) network throughput. Sample results from an operator and an edge of the object recognizer.

Based on Eq. (4.8), we have:

$$H' \frac{\sum_{h=1}^{H'} \sum_{u \in \mathbf{U}} F(r'_h, u, \cdots)}{H'} \leq \sum_{k \in \mathbf{V}} \sum_{u \in \mathbf{U}} R_{k,u}. \tag{4.10}$$

In addition, because both APX and OPT' algorithms enumerate requests by the total required resources and $H' \geq H$, we have:

$$\frac{\sum_{h=1}^{H'} \sum_{u \in \mathbf{U}} F(r'_h, u, \cdots)}{H'} \geq \frac{\sum_{h=1}^{H} \sum_{u \in \mathbf{U}} F(r_h, u, \cdots)}{H}, \tag{4.11}$$

This means that the average total required resources of requests deployed by OPT' is larger then APX. Based on Eqs. (4.9–4.11), we have:

$$\frac{H}{H'} \geq \frac{1}{|\mathbf{U}|}. \tag{4.12}$$

Because $H' \geq \dot{H}$, the approximation factor is $\frac{1}{|\mathbf{U}|}$. This yields the proof. □

We notice that $|\mathbf{U}|$ is typically a small integer (say 2 for CPU and RAM, for instance), and in the extreme case, where there is a single resource type ($|\mathbf{U}| = 1$), the APX algorithm gives optimal solutions.

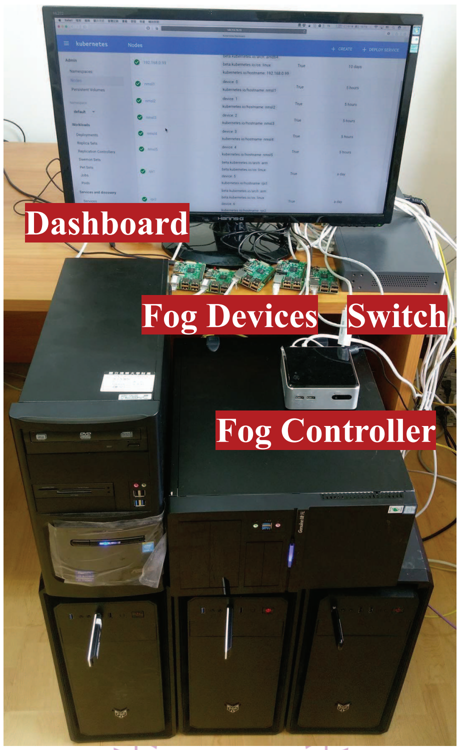


Figure 4.8: Testbed of our cloud-to-things continuum platform.

4.4 Testbed and Experiments

In this section, we realize a testbed of our cloud-to-things continuum platform for: (i) conducting a measurement study to derive our system models and (ii) evaluating the derived models and the performance of our cloud-to-things continuum platform.

4.4.1 Testbed Implementations

Fig. 4.8 illustrates our testbed which consists of: (i) an Intel i5 workstation running Ubuntu, Docker [10], and Kubernetes [28] as the server and (ii) 5 Intel PCs (1.8 GHz 8-core i7 CPUs) and 5 Raspberry Pi 3 (1.2 GHz 4-core ARM CPUs) as the fog devices running Ubuntu and HypriotOS, respectively. We implement the software components presented in Fig. 4.3 on the server and fog devices. The server and devices are connected with an Ethernet switch in a private network. To emulate WiFi-based networks, we adopt a traffic shaper [51] to throttle the bandwidth at 8 Mbps [85]. To avoid overloading the fog devices, we reserve 20% of resources for operations other than IoT analytics, e.g., OS scheduling and resource monitoring.

We implement the multi-operator IoT analytics using TensorFlow [45], in which every IoT analytics is represented as a graph consisting of *tensors* and *flows*. The tensors can be simple tasks, such as additions and subtractions, or very complex machine learning tasks.

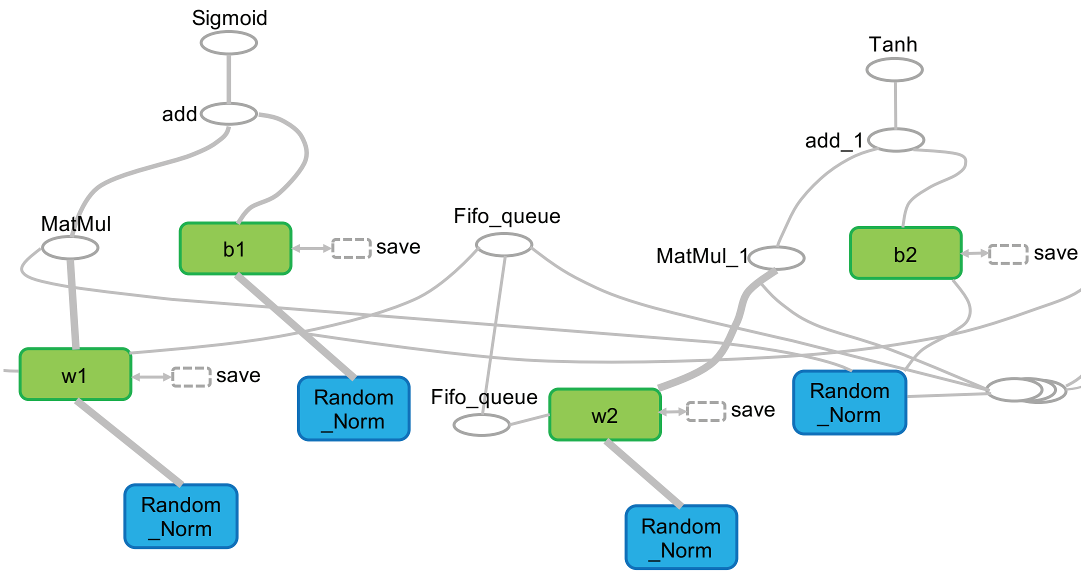


Figure 4.9: A sample TensorFlow graph (partial) of our sound classifier.

The flows transmit data among tensors. We use TensorFlow *tags* to split every graph into multiple smaller subgraphs, where each subgraph contains one or multiple tensors and associated flows. These subgraphs are essentially the *operators* in our cloud-to-things continuum platform, which are the units of deployments.

However, IoT analytics may require some features that are not available in TensorFlow. For example, OpenCV [38] is needed for processing images and librosa [31] is needed for analyzing audios. These additional libraries are quite large and take hours to compile. Therefore, installing these libraries on-the-fly is infeasible. On the other hand, reimplementing the features in TensorFlow incurs high engineering overhead, which may not be worth. Fortunately, our cloud-to-things continuum platform achieves short deployment time using Docker images. TensorFlow does not natively support stream processing, and we add a queue between any two adjacent operators to increase the overall performance.

To show the practicality of our platform, we develop three sample multi-operator IoT analytics : (i) air pollution monitor, (ii) sound classifier, and (iii) object recognizer. Each of them contains a large number of tensors, e.g., the object recognizer has 348 tensors. We split each TensorFlow graph into a few operators (subgraphs) as follows.

- **The air quality monitor** receives four gas sensor readings in JSON format through the MQTT protocol [35]. It then parses the JSON objects and calculates the moving averages (potentially, other statistics too) of individual gas sensors. The results can be used to detect emergency events, such as explosive gas leaking. We split this IoT analytics into 5 operators. The first operator subscribes to all gas sensor data

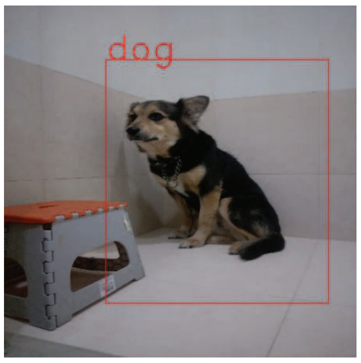
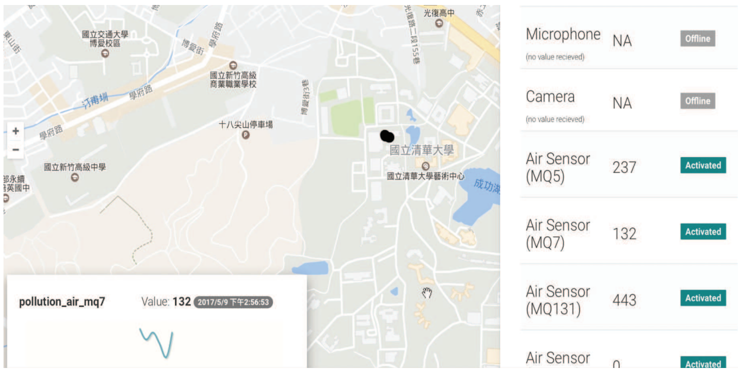


Figure 4.10: A sample result of our object recognizer.



Microphone (no value recieved)	NA	Offline
Camera (no value recieved)	NA	Offline
Air Sensor (MQ5)	237	Activated
Air Sensor (MQ7)	132	Activated
Air Sensor (MQ131)	443	Activated
Air Sensor	0	Activated

Figure 4.11: A sample web interface of our air quality monitor.

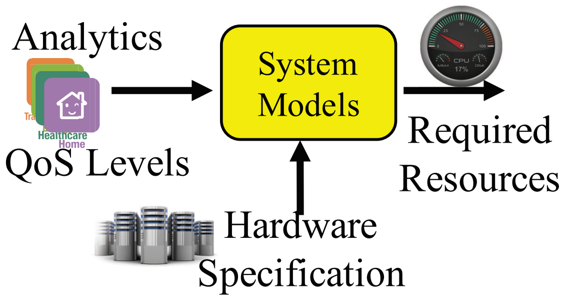


Figure 4.12: Our proposed system models.

via MQTT. The other four operators are responsible for calculating the moving averages of the four gas sensors, respectively.

- **The sound classifier** detects different types of sound for community safety. For example, this IoT analytics may detect a gun shot and automatically notify the police. It adopts a pre-trained four-layer neural network for analyzing the audio inputs. This IoT analytics is split into 5 linear operators. The first operator gathers the audio data, while the other four operators execute the four layers of the neural network, respectively.

- **The object recognizer** collects and analyzes camera images for recognizing objects in them. It employs a pre-trained neural network with more than 10 layers. We split this IoT analytics into 3 operators for input, hidden, and output layers, respectively. The first operator also captures and resizes the camera images using OpenCV [38].

Figs. 4.9, 4.10, and 4.11 show a sample TensorFlow graph of our sound classifier, recognized result of our object recognizer, and web interface of our air quality monitor.

4.4.2 System Model Derivation

The system models map the target QoS levels of different IoT analytics to the required resources on heterogeneous hardware specifications, as illustrated in Fig. 4.12. The parameters of operator model $\dot{F}(q, s_q, i, u, k)$ and edge model $\bar{F}(q, s_q, (i, i'))$ are derived as follows. First, we conduct an offline measurement study to derive the system models of Raspberry Pis (the lowest-end devices). For Intel PCs (more powerful devices without existing models), we use the models of less powerful devices for *bootstrapping*. We then update the system models online to iteratively derive the customized system models. Using system models of less powerful devices for bootstrapping prevents us from overloading the fog devices. Although we only have two types of fog devices in the testbed, the same procedure is applicable to more heterogeneous devices.

We model the following three resources:

- **CPU load:** the physical CPU load in percentage reported by Docker stats API.
- **RAM usage:** the total physical memory usage reported by Docker stats API.
- **Network throughput:** total traffic amount (in both directions) reported by Tcpdump.

We adopt different invocation frequencies as the QoS parameters; other QoS parameters may also be adopted if needed. Particularly, in our experiments, we choose the following sample parameters: (i) $\{0.25, 0.5, 1, 2, 4\}$ Hz in the air quality monitor, (ii) $\{5/60, 6/60, 7/60, 8/60, 9/60\}$ Hz in the sound classifier, and (iii) $\{6/60, 7/60, 8/60, 9/60, 10/60\}$ Hz in the object recognizer. Each QoS level of every IoT analytics is executed 5 times with 60-sec durations. We report the average results, after filtering out the results that deviate from the average for more than 1.7 times of the standard deviation.

Table 4.2: The Mean (Standard Deviation) R-square Values and P-values Among All Operators and Edges.

IoT Analytics	CPU Load		Network Throughput	
	R-Square	**P-Value**	**R-Square**	**P-Value**
Air Quality Monitor	0.99 (0.000023)	<0.001	0.99 (0.000001)	<0.001
Sound Classifier	0.85 (0.023263)	<0.010	0.84 (0.069494)	<0.010
Object Recognizer	0.99 (0.012540)	<0.010	0.97 (0.004982)	<0.010

Fig. 4.7 shows the measured results and system models of a sample operator (CPU and RAM) and a sample edge (network) from the object recognizer. The sample operator subscribes the data from all four sensors and the edge carries the flow of a gas sensor. We make a few observations on the sample results and adopt the following regression

models. First, we find that the RAM usage (Fig. 4.7(b)) remains the same under different QoS parameters. Therefore, we model it as a constant value, which gives us an average error of 0.4% and a maximum error of 1%. For the CPU load (Fig. 4.7(a)) and network throughput (Fig. 4.7(c)), we adopt power models (after considering several common functions), which give 0.99 R-square scores in both cases. The operator and edge models are summarized as:

$$\dot{F}(\mathscr{A}_q, s_q, i, u, k) = \begin{cases} d_{\mathscr{A}_q, s_q, i, k} & u = \text{RAM}; \\ \dot{a}_{\mathscr{A}_q, i, k} s_q^{\dot{b}_{\mathscr{A}_q, i, k}} + \dot{c}_{\mathscr{A}_q, i, k} & u = \text{CPU}; \end{cases} \tag{4.13}$$

$$\bar{F}(\mathscr{A}_q, s_q, (i, i')) = \bar{a}_{\mathscr{A}_q, (i,i')} s_q^{\bar{b}_{\mathscr{A}_q, (i,i')}} + \bar{c}_{\mathscr{A}_q, (i,i')}, \tag{4.14}$$

where $\dot{a}, \dot{b}, \dot{c}, d, \bar{a}, \bar{b}, \bar{c}$ are model parameters.

Similar observations can be made on other operators and edges, and model parameters can be derived by regression. We report the mean R-square scores of the CPU and network models with their standard deviations in Table 4.2. This table shows that our system models are fairly accurate. Moreover, the same table also reports p-values, which are < 0.01, showing the regression is statistically significant.

4.4.3 Validation

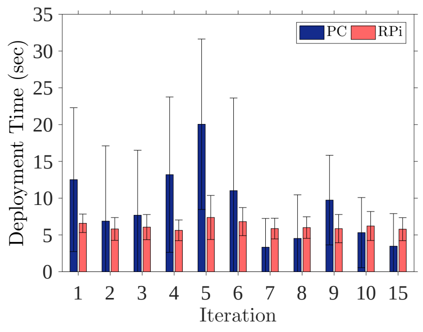


Figure 4.13: The average deployment time of operators in each iteration.

We run the following experiments for 15 iterations, where the offline system models are initially used for both Raspberry Pis and Intel PCs to drive our operator deployment algorithm. Based on the actual resource consumptions, we update our system models after each iteration that lasts for about 3 minutes. In each iteration, we randomly generate requests of three sample IoT analytics with random QoS levels chosen from the sample parameters of each IoT analytics. We up-sample the QoS levels by 10% as a small buffer for workload surges. Specifically, we generate a large number of requests, and execute

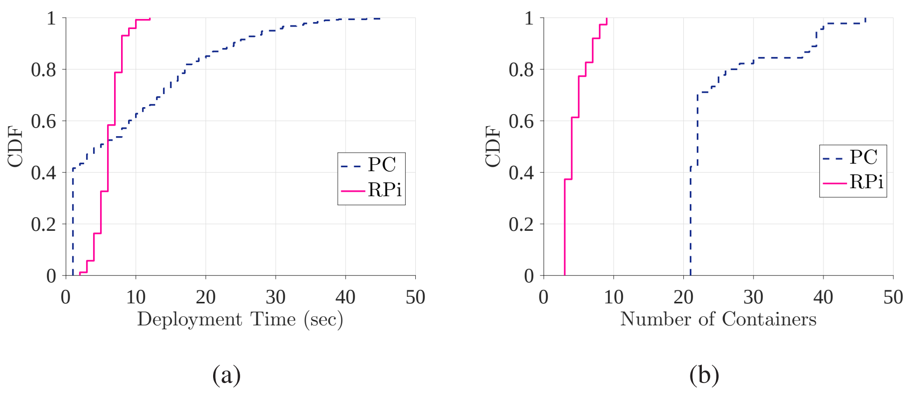


Figure 4.14: (a) Most operators deployed on Intel PCs take less time than on Raspberry Pis and (b) the number of containers created on Intel PCs are far more than on Raspberry Pis.

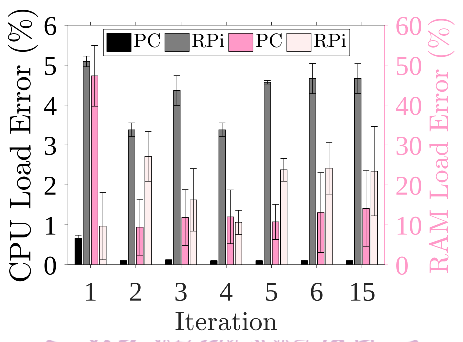


Figure 4.15: Online updates of our system models are effective. Iterations 7–14 are omitted due to the space limitations.

our operator deployment algorithm to deploy as many requests as possible. Although network conditions affect the download time of Docker images, disseminating Docker images efficiently is out of the scope of this chapter. Therefore, we assume all the Docker images are already cached in the image pools on fog devices. Sample experiment results with 95% confidence intervals are given below.

Operators are deployed almost instantly in our platform. Fig. 7.4(a) reports the average deployment time of operators in each iteration. The figure reveals that the operators take less than 20 seconds on average to be deployed, which is sufficient for most IoT analytics that need to have even shorter response time can be *pre-deployed*. As Fig. 4.14(a) shows, over 50% of operators deployed on Intel PCs take less time than on Raspberry Pis because Intel PCs have higher computational power. The rest of deployments take more time on Intel PCs than on Raspberry Pis because there are considerably more operators running on PC than Raspberry Pis, as reported in Fig. 4.14(b). When a large number of containers are simultaneously created on the same device, the I/O overhead is significantly increased.

Our system models are reasonably accurate and support heterogeneous fog devices. We report the system model errors of two sample models: (i) power models of CPU loads and (ii) constant models of RAM usage in Fig. 4.15. This figure shows that our system models become increasingly more accurate over iterations. More specifically, the CPU load errors are no worse than 5% and the RAM usage errors are no worse than 25% on average after 5 iterations.

Our operator deployment algorithm satisfies most of the requested QoS levels. Table 4.3 shows the *QoS satisfaction rate* and *the number of rejected requests* under different numbers of requests received from fog users. The QoS satisfaction rate is the fraction of deployed requests that meet the QoS targets, and the rejected requests are not deployed by our algorithm due to lack of resources. The table leads to three observations: (i) the QoS satisfaction rate is getting higher over iterations. At the fifth iteration, the QoS satisfaction rate reaches to 100%, (ii) more received requests require more iterations to satisfy all the requests, and (iii) once our platform is fully loaded, some requests are rejected to maintain the QoS levels of deployed requests.

Our experiments demonstrate the practicality and efficiency of our models, algorithm, and platform. For larger scale evaluations, we perform detailed simulations in the next section.

Table 4.3: QoS Satisfaction Rate and Number of Rejected Requests

	Iter. 1		**Iter. 2**		**Iter. 3**		**Iter. 4**		**Iter. 5**	
Recv. Req.	**Rej. Req.**	**Satis. Rate**	**Rej. Req.**	**Satis. Rate**	**Rej. Req.**	**Satis. Rate**	**Rej. Req.**	**Satis. Rate**	**Rej. Req.**	**Satis. Rate**
4	0	100%	0	100%	0	100%	0	100%	0	100%
8	0	100%	0	100%	0	100%	0	100%	0	100%
16	0	79%	0	100%	0	100%	0	100%	0	100%
32	0	89%	10	95%	10	100%	10	100%	10	100%
64	22	71%	42	95%	42	95%	42	100%	42	100%

4.5 Simulations

In this section, we consider a wider spectrum and compare our analytics deployment algorithms against the state-of-the-art ones in a simulator.

4.5.1 Setup

We build a detailed simulator in Python. In this simulator, we implement our APX and SSE algorithms. We also realize four baseline algorithms: Random, Linear, Fog and

Cloud Placement (FCP) [167] and Optimal Data Stream Processing Placement (ODP) [67, 69]. The Random algorithm mimics a fog computing platform without a central controller, and deploys operators on random devices. The Linear algorithm greedily deploys operators from the source fog devices to adjacent devices, while taking the resource, sensor, and location constraints into considerations. The FCP and ODP algorithms are state-of-the-art analytics deployment algorithms, which consider both node and link constraints as we discussed in Sec. 4.1. The FCP algorithm aims to support latency sensitive IoT analytics with high resource efficiency. It is a heuristic algorithm, which finds the best-fit device for every request. The ODP algorithm has a flexible objective function. We adapt the objective function to maximize the number of satisfied requests for fair comparisons. The ODP algorithm employs a commercial ILP solver: CPLEX [21], which may take prohibitively long time to terminate. Therefore, it is recommend to set a 500-sec cap on the execution time [67, 69]. We run simulations of each test scenario with all the algorithms for comparisons.

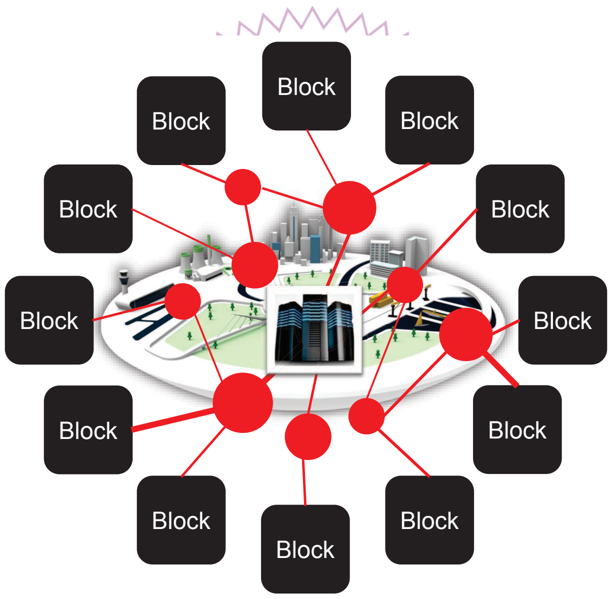


Figure 4.16: The considered network topology.

In order to simulate a large-scale and realistic network topology, we extend the network topology of the smart pole block shown in Sec. 7.4 with a mesh network as the network backbone. As illustrated in Fig. 4.16, the topology has twelve blocks and each block has eight fog devices installed in the device boxes. Half of the fog devices are connected to local Ethernet (1 Gbps) and others are connected to WiFi mesh (32 Mbps) [63]. These fog devices access the Internet through the access switch of each block. The access switches are connected to a backbone network generated by BRITE's [7] mesh topology. The link capacities from the access switches to the backbone network are randomly chosen from 3G (9.61 Mbps) [63], WiFi (32 Mbps) [63], and 4G (22.67 Mbps) [63]. The link capacities of the backbone network are randomly chosen from VDSL (60 Mbps) [63] and Fiber (1 Gbps) [63]. The actual bandwidth constraints are draw from a gaussian distribution with a mean value shown in the parentheses above, where the standard deviation is 10% of the mean value. We configure the fog device's geolocation coordinates using actual GPS coordinates of real street lamps next to 12 main buildings at NTHU campus. We also place the backbone routers among these buildings, in terms of geographical locations, following the Euclidean distances and link latencies generated by BRITE. Each fog device has a random CPU capacity between 100% and 400% (corresponding to number of cores) and a random RAM capacity between 1 and 8 GB.

Poisson processes with 1-min average arrival time and 10-min average departure time are used for generating requests. In this way, the number of active requests steadily increases over time. Each request randomly picks one of the three IoT analytics (detailed in Sec. 7.4) with random target QoS levels between 0.1 and 4 Hz. The system models (of PCs and Raspberry Pis) derived in our experiments are used in the simulations. Each requested source and destination operator randomly occurs in one of the 12 blocks. We consider random sensors, including cameras, microphones, and (4 types of) gas sensors. The IoT analytics are associated with their requested sensors.

We consider the following performance metrics:

- **Number of deployed requests** that are deployed with deployment decisions made by the algorithms.

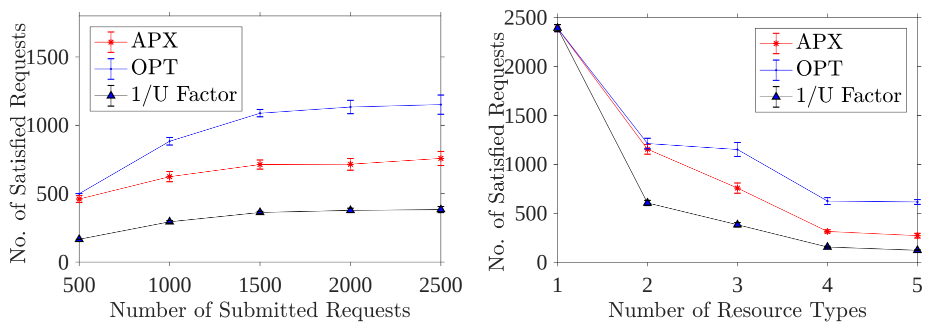
- **Number of satisfied requests** that comply with the constraints in Eqs. (4.1d) and (4.1f).

- **Number of overloaded links** that fail some deployed requests

- **Resource efficiency** that is a ratio of total satisfied requests and total resource consumptions.

- **Running time** of the analytics deployment algorithms.

We run 16-hour simulations on an AMD 64-core workstation. Our analytics deployment algorithm is invoked once every 30 mins. We consider the number of blocks $\in \{3, 6, 9, 12\}$, which means that the considered number of fog devices $|\mathbf{V}| \in \{24, 48, 72, 96\}$. Moreover, for our SSE algorithm, we consider maximal number of hop $H \in \{2, 4, 6, 8, 10\}$. We let 12 as default value of number of blocks ($|\mathbf{V}| = 96$) and the maximal number of hop $H = 10$ if not otherwise specified. We repeat each simulation 5 times and report means with 95% confidence intervals if applicable.



(a) (b)

Figure 4.17: Satisfied number of requests of our APX algorithm under different number of: (a) requests and (b) resource types.

Table 4.4: Empirical Based Approximation Factors

Req. / U	500		1000		1500		2000		2500	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	1.000	0.000	1.00	0.000	1.00	0.000	1.00	0.000	1.00	0.000
2	1.000	0.000	1.00	0.000	0.957	0.006	0.938	0.028	0.952	0.003
3	0.921	0.057	0.706	0.035	0.658	0.039	0.430	0.024	0.658	0.037
4	0.603	0.061	0.499	0.033	0.544	0.017	0.502	0.026	0.504	0.024
5	0.536	0.045	0.438	0.031	0.442	0.028	0.441	0.025	0.441	0.038

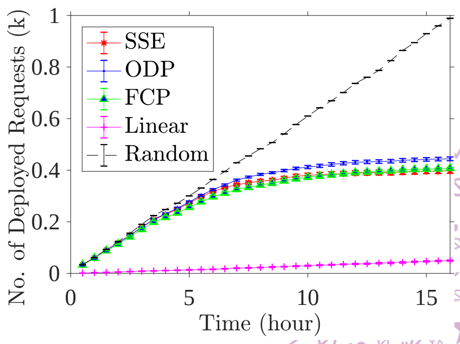
4.5.2 Results

Performance of our APX algorithm. We implement an optimal solution (OPT) using CPLEX [21] as a benchmark to evaluate our APX algorithm. Because the running time of OPT is high, we run a set of simulations with smaller topology as follows. We generate $|\mathbf{Q}|$ requests, which require $|\mathbf{U}|$ kinds of resources allocated from 100 fog devices to conduct the evaluations. We vary $|\mathbf{Q}| = \{500, 1000, 1500, 2000, \mathbf{2500}\}$ and $|\mathbf{U}| = \{1, 2, \mathbf{3}, 4, 5\}$, where the bold font indicates the default values. The required resources of the first two resource types are from our derived CPU and RAM system models. Other kinds of resources are randomly assigned in a range between 0 to 100. The fog devices' resource capacity of CPU are random values between 100% to 800% and RAM are random values between 1 to 16 GB. Other kinds of resource capacities are random values between 100 to 1000. We run the evaluations 10 times and report means with 95%

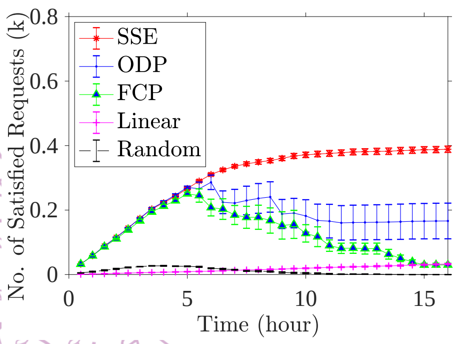
Table 4.5: Maximal Running Time (s) of APX and OPT Algorithms

Req. / U	500		1000		1500		2000		2500	
	APX	OPT	APX	OPT	APX	OPT	APX	OPT	APX	OPT
1	0.25	1.76	0.71	3.35	1.32	5.25	1.92	7.27	3.46	12.67
2	0.42	2.14	1.06	4.87	2.49	9.79	3.69	16.43	5.41	>500
3	1.01	2.64	2.73	7.86	4.44	203.65	6.81	>500	8.74	>500
4	1.94	3.27	4.61	>500	7.57	>500	10.16	>500	12.52	>500
5	2.63	5.33	5.76	>500	8.88	>500	11.88	>500	15.21	>500

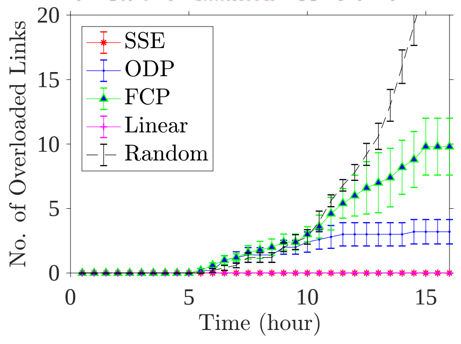
confidence intervals. Fig. 4.17 shows that when the number of requests is small or the numbers of resources types is 1, our APX algorithm indeed achieves the optimal results. Moreover, the resulting numbers of satisfied requests of our APX algorithm are always above the theoretical $1/|\mathbf{U}|$ approximation factor curve. We summarize all the empirical approximation factors, including average and standard deviation values in Table 4.4. This table shows that in the experiments, our APX algorithm achieves at least 2 times of *theoretical approximation factor*. Moreover, we report running time of APX and OPT algorithms in Table 4.5, which shows that our APX algorithm runs in real-time while the OPT algorithm has an exponentially growing running time. *APX is not considered in the rest of evaluations, because it's designed for fine-grained operators. The IoT analytics models used to evaluate SSE is not fine-grained enough.*



(a)



(b)



(c)

Figure 4.18: QoS improvement of our SSE algorithm in: (a) the number of deployed requests, (b) the number of satisfied requests, and (c) the number of overloaded links.

Our proposed SSE algorithm results in QoS improvements. Fig. 6.7 reports the overall QoS achieved by different algorithms. Fig. 4.18(a) shows that the gap of number of deployed requests among SSE, ODP, and FCP is smaller than 10%. Random deploys up to 148% more requests compared to our SSE algorithm while our SSE algorithm deploys 699% more requests compared to Linear. However, the larger numbers of deployed requests do not directly result in better performance, because many deployed requests fail to satisfy the constraints. Fig. 4.18(b) plots the number of satisfied requests. Indeed, we observe that our SSE algorithm satisfies by up to 134%, 1200%, and 1233% more requests than ODP, FCP, and Linear, respectively. On the other hand, Random satisfies zero request after running for 15 hours. The inferior performance of ODP, FCP, and Random due to too aggressive leading to an overloaded fog computing platform. To show this, we plot the number of overloaded links in Fig. 4.18(c), where ODP, FCP, and Random result in as many as 3.2, 9.8, and 24.8 overloaded links, while our SSE algorithm leads to none. Last, we note that the Linear algorithm limits its analytics deployment decisions to adjacent devices, and thus is too conservative to satisfy more requests. That explains why our SSE algorithm satisfies 1233% more requests that the Linear algorithm. *Since Random and Linear suffer from significantly fewer number of satisfied requests, we no longer consider them in the rest of this paper.*

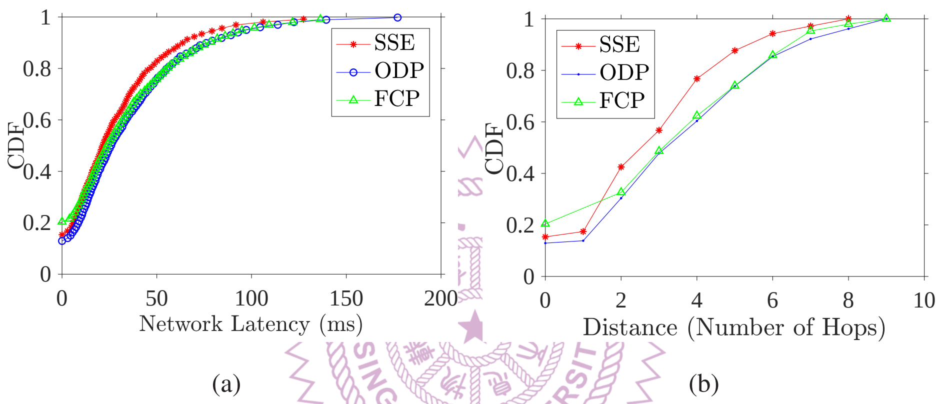


Figure 4.19: Shorter network latency of our SSE algorithm: (a) end-to-end network latencies and (b) number of hops distribution of satisfied requests.

Our proposed SSE algorithm reduces network latencies. Fig. 4.19(a) reports the end-to-end latency distribution of satisfied requests. It clearly shows that our SSE algorithm achieves lower latencies. More specifically, our SSE algorithm results in 46.5 ms for 80% of requests; while ODP results in 55.3 ms and FCP results in 55.5 ms. This can be attributed to the second intuition of our SSE algorithm, which considers shortest paths first. To validate this, Fig. 4.19(b) reports the distribution of number of hops of satisfied requests. It clearly shows that our the distance (number of hops) between source operator and destination operator of a request deployed by our SSE algorithm is shorter than others, which contributes to the low latency. Because of its low latency, our SSE algorithm is more suitable for real-time IoT analytics.

Our proposed SSE algorithm is resource efficient. Fig. 6.8 shows distributions of resource consumptions and resource efficiency under peak resource consumptions. Actually, the CPU, RAM, and network resource consumptions shown in Figs. 4.20(a)–4.20(c) are similar among different algorithms. However, our SSE algorithm results in higher resource efficiency, as illustrated in Fig. 4.20(d). In particular, Fig. 4.20(d) shows that

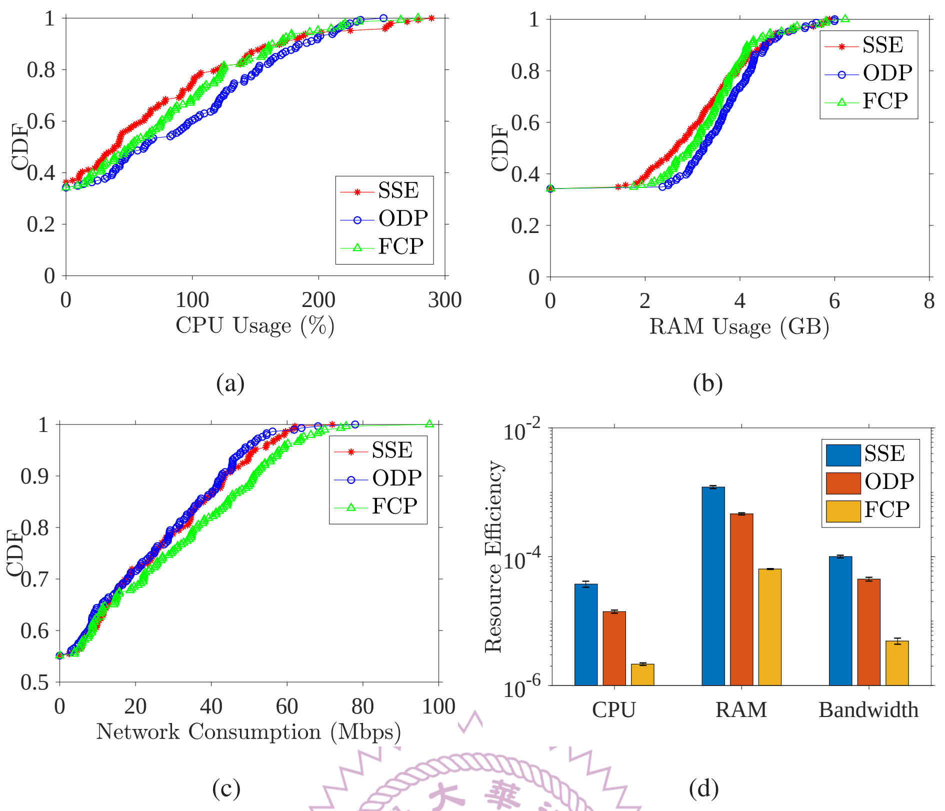
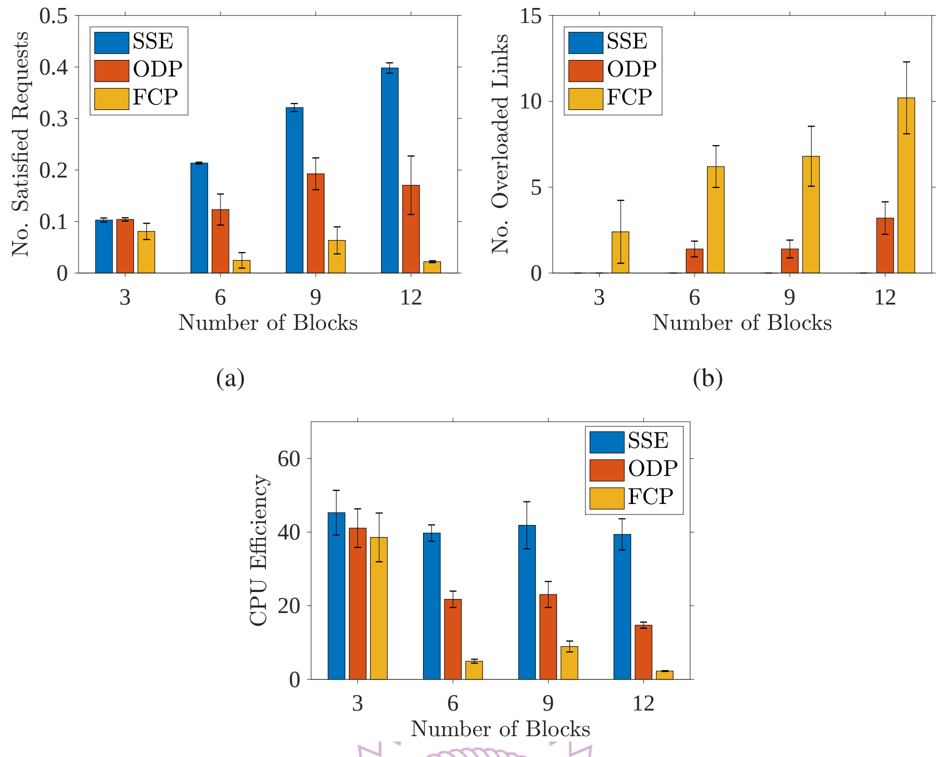


Figure 4.20: Better resource efficiency of our SSE algorithm: (a) peak resource consumption, (b) CPU load, (c) RAM usage, and (d) network throughput distributions of fog devices.

our SSE algorithm outperforms 167%, 161%, and 124% resources efficiency in terms of CPU, RAM, and network, compared to the ODP algorithm. Moreover, our SSE algorithm outperforms 1649%, 1774%, and 1951% resources efficiency in terms of CPU, RAM, and network, compared to the FCP algorithm.

Impacts of different number of blocks. We vary number of blocks from 3 to 12 (number of fog devices from 24 to 96), and repeat the same simulations. We plot the key performance metric results in Fig. 4.21. Fig. 4.21(a) reports that our SSE algorithm outperforms ODP and FCP by 65.3% and 204.1%, on average, in terms of number of satisfied requests on average. This is because our SSE algorithm does not overload any links as confirmed by Fig. 4.21(b). Fig. 4.21(c) reports that our SSE algorithm, on average, improves CPU resource efficiency by 75.4% and 442.7% compared to the ODP and FCP algorithms, respectively Fig. 4.21 demonstrates that our SSE algorithm scales well. In fact, our SSE algorithm outperforms other algorithms by larger margins when the network is larger.

Our proposed SSE algorithm is scalable. Table 4.6 reports the maximal running time of the analytics deployment algorithms throughout 16-hour simulations under different number of fog devices. This table shows that the running time of ODP exceeds



(c)

Figure 4.21: Impacts of number of blocks on: (a) number of satisfied requests, (b) number of overloaded links, and (c) CPU utilizations.

its 500-sec threshold with merely 9 blocks (72 fog devices). ODP's running time grows exponentially. While, our SSE algorithm only takes 34 sec to make the deployment decisions with 9 blocks, and its running grows linearly. In addition, the running time of FCP algorithm also grows linearly and is shorter than our SSE algorithm. However, the FCP algorithm suffers from fewer satisfied requests, inefficient resource usage, and longer end-to-end latency, as shown above.

Table 4.6: Maximal Running Time (s) of Analytics Deployment Algorithms

No. Blocks	**3**	**6**	**9**	**12**
SSE	3	12	34	132
ODP	165	336	>500	>500
FCP	1	2	3.5	6

4.6 Discussion

In this chapter, we focus on solving application deployment problem, where we take IoT analytics as representative applications. We formulate this problem into two problem formulations under different cloud-to-things continuum conditions. We design an approximation algorithm (APX) and an efficient algorithm (SSE) to maximize number of satisfied requests with diverse QoS requirements. The QoS requirements are carefully modeled by conducting measurement studies. We derive system models to predict required resources of IoT analytics with diverse QoS values. To conduct the measurement studies, evaluate our algorithms, and realize our cloud-to-things continuum, we implement our platform with : (i) Kubernetes, (ii) Docker, and (iii) TensorFlow. Our platform provides: (i) novel deployment algorithms for the provider to make deployment decisions and (ii) lightweight virtualization for users to dynamically request for on-demand applications with resource guarantees. To evaluate our algorithms, we conduct extensive trace-driven simulations with two baseline (Random and Linear) and two state-of-the-art analytics deployment algorithms: FCP and ODP. The baseline algorithms suffer inferior number of requests, and our algorithms outperform the state-of-the-art algorithms by at least 134%, 167%, 161%, and 124% in terms of number of satisfied requests, CPU resource efficiency RAM resource efficiency, and network resource efficiency, respectively. Moreover, our algorithms run in polynomial time while optimal solutions need exponential time to make deployment decisions.

Although our analytics deployment algorithms have outperformed state-of-art algorithms, there are some limitations that are not considered in this chapter: (i) priority of the requests, (ii) time sequence of the requests, and (iii) dynamicity of the requests. About the first limitation, in this chapter, we assume that requests have the same priority. Therefore, our algorithms will not tend to deploy emergency request first, e.g., the requests from police office about a gun shot will not be considered in advance. To solve this limitation, we can classify the requests into multiple categories based on various priorities. We then make deployment decisions from the most critical category to the least one. Doing so, we can support the prioritized requests without modifying our algorithms. For the second limitation, in this chapter, we batch the requests and solve the deployment problem for every batch. Therefore, we do not immediately make deployment decisions once every request is received. One possible approach is to propose a request prediction algorithm to predict coming requests in the future to take time sequence into our consideration. Hence, we can use our deployment algorithms to make decisions for the requests in advance and deploy the requests once we receive them. For the third limitation, in this chapter, we optimize the deployment for every batch. However, requests come and go, so that the deployments do not always keep optimized state. In order to solve this limitation, one possible approach is to model migration costs into our formulation. The migration cost will be affected by cache of container images. Hence, we also need to propose a caching algorithm to efficiently use the storage spaces. With the migration cost and caching algorithm, we can dynamically migrate the requests to keep our platform in the optimized state.

Chapter 5

Application Specific Optimization: Delay-Sensitive Applications

After deploying the applications with the algorithms proposed in Chapter 4, we start to optimize the delay-sensitive applications which are already running on our cloud-to-things continuum platform in this chapter. We select one representative delay-sensitive application, which will be introduced in Sec. 5.1. Moreover, we survey the related studies in Sec. 5.2, extend and measure existing open source application in Sec. 5.3 write problem statements for optimizing the applications and design our solutions in Sec. 5.4, evaluate them in Sec. 5.5, and report a summary in Sec. 5.6.

5.1 Game Streaming Applications

Our selected delay-sensitive application is *game streaming applications*, which require strict low delay and high computation power. There are two existing gaming models requiring remote servers: (i) file streaming and (ii) video streaming. The first, more conservative, model leverages the remote servers to distribute computer games, because modern computer games often take more than one DVD to distribute. File streaming allows gamers to start playing games after a small subset of files has been downloaded, while the remaining files are streamed in the background. File streaming allows the game developers, such as Blizzard, to reduce their costs on manufacturing the physical media, and delivering patches through the Internet. However, all these computer games still run on gamers' computers, and thus gamers: (i) need high-end computers for good graphics effects and (ii) can only play games on the installed computers.

The second, more aggressive, model offloads intensive computations, such as rendering, physics, and artificial intelligence, to the remote servers (fog devices), in order to deliver more visually-appealing effects on less powerful computers anywhere, anytime over the Internet. Using video streaming, game service providers, such as OnLive [37], GaiKai [15], and Ubitus [47], deliver instantaneous gaming experience to gamers using different client computers (including mobile devices). This is done by implementing and installing a thin client on PCs, laptops, tablets, smartphones, and set-top boxes. In this chapter, We only consider the more challenging video streaming model to achieve the game streaming applications.

5.1.1 Usage Scenarios

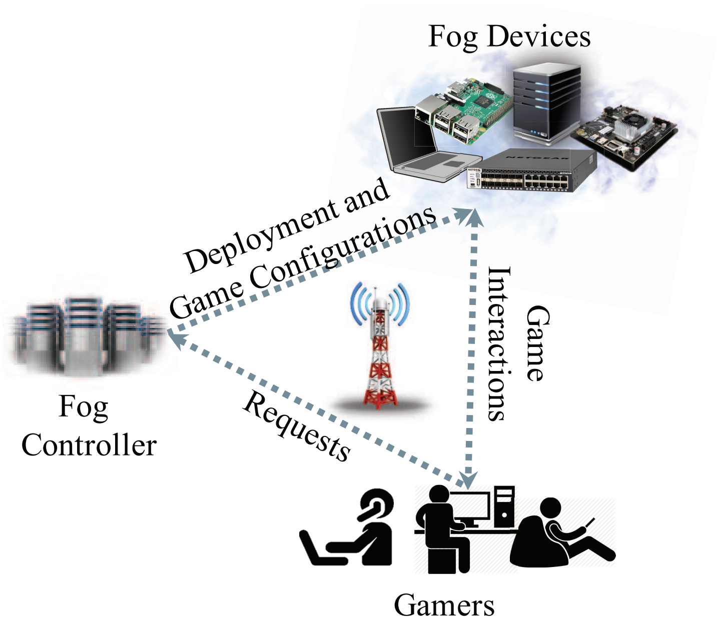


Figure 5.1: An usage scenario of the game streaming application running on our cloud-to-things continuum platform.

Figure 5.1 gives a scenario of running game streaming applications on our cloud-to-things continuum platform. We first present the life cycle of a game session. A gamer first selects a game and sends a request to the fog controller to play a game. After the gamer selects a game to play, the global optimizer (Chpater 4) installed on the fog controller selects a fog device to serve this gamer. After that, the application specific optimizer also installed on the fog controller sends game configurations, e.g., the encoding bitrate and frame rate, and 3D effect level, to the game streaming application running on the selected fog device. The application runs a piece of software to capture, compresses, and sends the audio and video, rendered by the game, over the Internet to the gamer's device. The audio and video are decompressed and played by the gamer's device to the gamer, who uses keyboard, mouse, or other input devices to interact with the game. Because the game is running on a remote fog device, the inputs from gamers are sent to the game streaming application running on the fog device for the interactions. The game session is between a fog device and gamer's device, and ends when the gamer quits the game.

5.1.2 Challenges

Delivering high-quality gaming experience is challenging, mainly because: (i) modern computer games are mostly resource hungry, (ii) the real-time nature of games imposes stringent deadline, and (iii) gamers have high expectations on different aspects of gaming experience [124]. More specifically, gamers ask for both low response delay and high-quality game scenes, where the: (i) the response delay refers to the time difference between the time when a gamer triggers an input and the time when the client renders the corresponding effect, and (ii) quality of game scenes is measured by metrics like resolutions, frame rates, fidelities, and 3D effect levels. Concurrently achieving both fast responses and high-quality scenes consumes a huge amount of computation and network resources. In Chapter 4, we cope with the problem by the global optimizer to select the most suitable fog devices to run the game streaming applications for incoming gamers to maximize the number of served gamers while satisfying QoS requirements in large time scales (in the order of tenths of minutes). However, the network resources change in small time scales (in the order of seconds), which cannot be compensated by the global optimizer *alone*, and thus a finer-grained adaptation mechanism is required for each game session.

In this chapter, we study the problem of *adapting* ongoing streaming game sessions to maximize the gamer experience in dynamic networks. Without adaptations, the cloud-to-things continuum platforms continuously deliver the streaming games to gamers at the highest possible quality even when the resources are insufficient. This may overload the network because a large part of the network resources is consumed by the downstream traffic (game videos) [77]. Thus, the video bitrate of an ongoing streaming game session should be reduced if the end-to-end bandwidth is insufficient. Moreover, when the bandwidth is significantly reduced, the video frame rate may have to be reduced to maintain a satisfying graphics quality. Otherwise, gamers would suffer from degraded gaming experience due to late and lost frames, and may quit the games prematurely.

Achieving optimal adaptations is no easy task because gamers are picky, and love to have both high graphic quality and short interaction delay. However, concurrently optimizing *both* gamers' demands is impossible. Therefore, in order to solve the bitrate adaptation problem, three main challenges are carefully studied in the following:

- **Quantifying gamer experience.** The metrics of higher-level gamer experience and

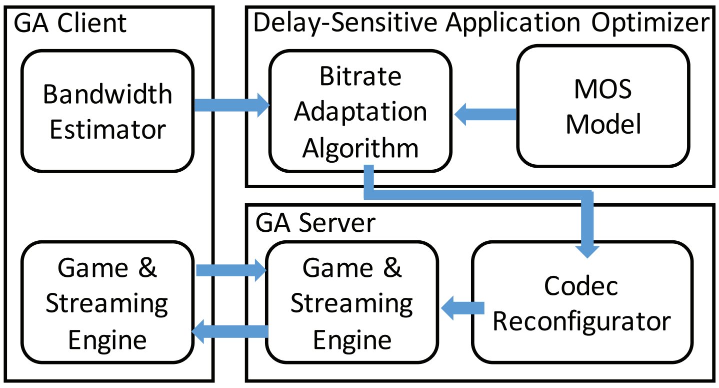


Figure 5.2: The interactions between the streaming game applications and the delay-sensitive application optimizer.

lower-level system performance are quite different, and the mapping between them is affected by many factors. Hence, deriving empirical gamer experience, or Mean Opinion Score (MOS) model is important and challenging.

- **Reconfiguring video codecs.** Compared to audio streams, encoding and transmitting video streams consume much more computation/network resources, and thus video codec is the main control knob for adapting an ongoing streaming game session to the available resources. Changing the codec configuration on-the-fly is critical to help us achieve optimal bitrate adaptations.

- **Adapting videos in dynamic networks.** With the gamer experience model and real-time codec reconfiguration mechanism, We develop a suite of techniques to quickly adapt the video codec configurations to dynamic networks. The techniques range from optimal resource allocation algorithms to real-time heuristics to maximize the gamer experience without excessive resource consumption. These techniques are presented in Sec. 5.4.

We note that this thesis focuses on solving resource allocation problems. We will only briefly describe the solutions of first and second challenges in this chapter and focus on solving the adaptation problem in Sec. 5.4. Please find the details of the first and second solutions from Hong et al. [104].

5.1.3 Software Components

The game streaming applications is built upon GamingAnywhere (GA) [16, 110], which is an open-source game streaming application designed for researchers, engineers, and gamers. The design philosophy of GA includes extensibility, portability, configurability, and openness, and thus very suitable to game streaming research. Basically, the GA application consists of a *GA server* and a *GA client*. The GA client is installed on gamer's device, such as a mobile phone to send requests to our cloud-to-things continuum platform to play a game. The global optimizer introduced in Chapter 4 then selects a fog device to run a GA server to serve the gamer. The GA server and GA client communicate through the *game & streaming engine* to allow gamers send control messages (e.g., key strokes and mouse clicks) to the GA server and receives rendered game scenes from the GA server. More details of the original GA are given in [110].

As illustrated in Figure 5.2, we add four software components, including bandwidth estimator, MOS model, codec reconfigurator, and bitrate adaptation algorithm into the GA application, to solve the challenges and optimize the game streaming applications. The *bandwidth estimator* monitors the sending/receiving timestamps of video packets at GA client, so as to estimate the effective bandwidth[1]. GA clients send the estimated effective bandwidth to the *bitrate adaptation algorithm* to determine the optimal encoding bitrate and frame rate to maximize the gaming experience. Such decisions are made based on a *MOS model*, which converts each pair of bitrate and frame rate into a game-dependent MOS score. The optimal encoding bitrate and frame rate are sent into *codec reconfigurator* for on-the-fly video adaptation. Finally, the game & streaming engine at GA server side renders and streams the game scenes based on the decisions to the game & stream engine at GA client side, which receives the stream and send control messages to the GA server. In summary, these four components are core of this chapter, that will be introduced in Secs. 5.3 and 5.4.

5.2 Related Work

5.2.1 Game Streaming Platforms

In game streaming applications, there are several approaches to divide the tasks between the remote servers and clients. With *graphics streaming* [81,118], the remote servers send all graphics commands to the clients, which then render the graphics commands using the clients' GPUs. This approach dictates more powerful GPUs to render high-quality game scenes in real time, which is less applicable for resource-limited mobile devices and set-top boxes. With *post-rendering operations* [88, 164], the 3D rendering is done at the remote servers, and part of the post-rendering operations are done on clients. Such post-rendering operations on the clients include augmenting motions, lights, and texture [74]. This approach complicates the game development and increases the development cost,

[1]Packet losses and delays are considered when estimating the *effective bandwidth.*

which may drive the game developers away from the game streaming applications. With *video streaming* [98, 179], the remote servers render the game scenes and stream videos to the clients. The clients decode and display videos, which is a much lighter weight operation compared to the other two approaches. Comparatively, the video streaming approach: (i) demands the least resources at the clients, (ii) is easier to implement, (iii) is easier to port to heterogeneous clients, and (iv) requires the minimum augmentations on game code.

Hence, the mainstream commercial game streaming applications, such as OnLive [37], GaiKai [15], and Ubitus [47], all adopt the video streaming approach. Similar approaches are taken by several studies [98, 112, 179] in the literature. Among these applications, GA [112] is open, modularized, and efficient. It is the first complete open-source game streaming application of its kind. GA has been leveraged by several research projects on game streaming, such as mobile cloud gaming [111, 113], e-learning applications [87], GPU consolidation [103], and cloud resource allocation [99, 100].

5.2.2 Resource Allocation for Game Streaming Applications

The resource allocation problem for the game streaming applications has been studied for some game genres. For example, Lee and Chen [125] study the resource allocation problem for Massively Multiplayer Online Role-Playing Game (MMORPG), and propose a zone-base algorithm to reduce the hardware requirements of the remote servers. However, these game servers handle short state update messages, and thus are different from servers that stream high-quality real-time videos to the clients. Duong et al. [80] and Wu et al. [180] focus on the admission control problem, to minimize the queueing delay for game streaming applications. In particular, Duong et al. [80] develop algorithms to selectively admit incoming users for the highest profit, and Wu et al. [180] propose a similar algorithm to quickly serve users in the waiting queue. Wang and Dey [173] propose an adaptive algorithm to dynamically adjust the rendering parameters, such as lighting modes and texture details, to adaptively allocate resources. Cai et al. [66] study the resource allocation problem between a remote server and a client computer. They divide a game into several software components and intelligently dispatch the components among multiple remote servers and client computers. Our earlier works [99, 100] consider a different problem of maximizing the gaming Quality of Experience (QoE) of all admitted users, by placing virtual machines in the best data centers under diverse CPU, RAM, and storage capacities. This chapter, in contrast, addresses the problem of dynamic adaptations in ongoing game sessions.

5.3 Reconfiguring Video Codecs in Run Time

Hong et al. [104] enhance GA to support dynamic video codec reconfigurations. In this chapter, we implement a bandwidth estimator to estimate effective bandwidth of GA client. The estimated bandwidth is sent to the dynamic video codec reconfigurator to change the streaming configurations. We then conduct experiments to validate the real-timeness of the dynamic video codec reconfigurator.

The major modification by Hong et al. [104] is to migrate *ffmpeg* to *live 555* because three major reasons: (i) *ffmpeg* is too complicated to add a customized codec, (ii) *ffmpeg* does not support dynamic reconfigurations (we cannot use it to perform bitrate adaptation for ongoing game sessions in this chapter), and (iii) The RTP feature provided by *ffmpeg* is tightly coupled with its codec implementation, which prevents us to make a customized codec to work with the RTP stack. Therefore, Hong et al. [104] revise the module design used in GA to work with *live555*. Please find more details of the implementation from Hong et al. [104].

5.3.1 Effective Bandwidth Estimation

We implement a bandwidth estimator inspired by WBest [126] to estimate the effective bandwidth. Different from other proposals [120, 126, 153], We leverage existing video packets for effective bandwidth estimation, without incurring network overhead. We can do this for two reasons: (i) the workload of sending video packets is high (for 30-fps videos, we send a frame every 33 ms) and (ii) the size of a video frame is typically higher than MSS (Maximum Segment Size), so each video frame is split into multiple back-to-back packets. Our bandwidth estimator keeps track of the dispersion time of the video packets. To cope with fluctuations, we select the median value as the estimated capacity from every W packets. We then send video packets at the estimated capacity to measure the effective bandwidth with W video packets. The value of W is empirically selected by experiments, and we consider $W \in \{100, 200, 300, 400, 500\}$. The estimation accuracy with $W = 100$ is 50% in the worst case. In other cases $(200 \leq W \leq 500)$, the accuracy is as high as 80+% and the differences among different W values are smaller than 5%. Hence, we let $W = 200$ for shorter reaction time.

5.3.2 Experiments on a Real Testbed

We conducts real experiments to show the effectiveness of the reconfiguration feature. We set up a testbed with a GA server and a GA client, and connect them via a Dummynet box. We find that the dynamic reconfiguration of video *bitrate* and *frame rate* are

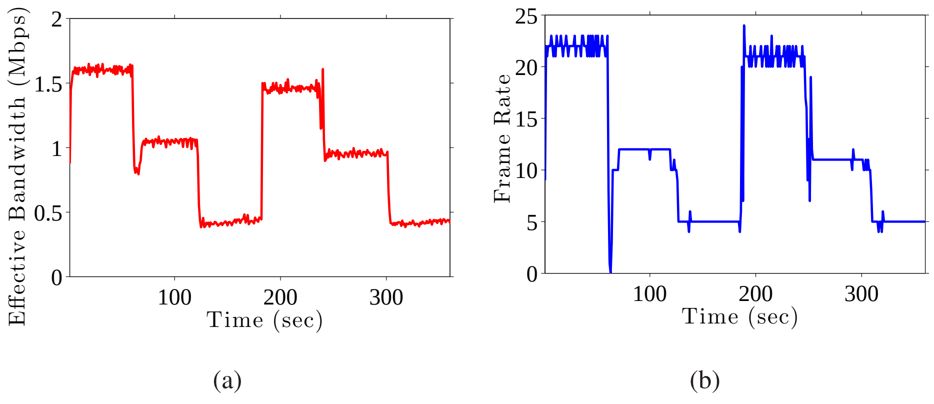


Figure 5.3: The measured: (a) effective bandwidth and (b) frame rate.

widely supported by most off-the-shelf video codecs. This allows us to leverage these two parameters in the adaptation algorithm presented in Sec. 5.4.

Next, we evaluate the adaptive GA platform as follows. We write a script on the Dummynet box to repeatedly switch the bandwidth among 2048, 1024, and 512 kbps once every 60 secs. We then adaptively reconfigure the video bitrate and frame rate using the codec parameter selector (Eq. (5.4) in Sec. 5.4), based on feedback from the bandwidth estimator (Sec. 5.3.1). Figure 5.3(a) reveals that our bandwidth estimator successfully detects the bandwidth changes. Figure 5.3(b) shows that the codec parameter selector quickly recovers from sudden frame rate drops due to throttled bandwidth, by adjusting the video coding parameters. The experiments demonstrate how to adapt to network dynamics for a single gamer. We consider the adaptation problem across multiple gamers sharing a bottleneck link in Sec. 5.4.

5.4 Adapting Videos of Multiple Gamers in Dynamic Networks

In this section, we solve the resource allocation problem among multiple gamers. Our proposed algorithm runs on the fog controller in cloud-to-things continuum.

5.4.1 Notations and Model

Table 5.1 summarizes the symbols used in this chapter. We consider a data center serving U gamers as illustrated in Fig. 5.4, where each gamer is connected to a GA server. All U GA servers share an outgoing link through a cellular base station, and we let R be the current available link bandwidth of this link. The value of R is fluctuating due to the background traffic. We let f $(1 \leq f \leq F)$ and b $(1 \leq b \leq B)$ be frame rate and bitrate,

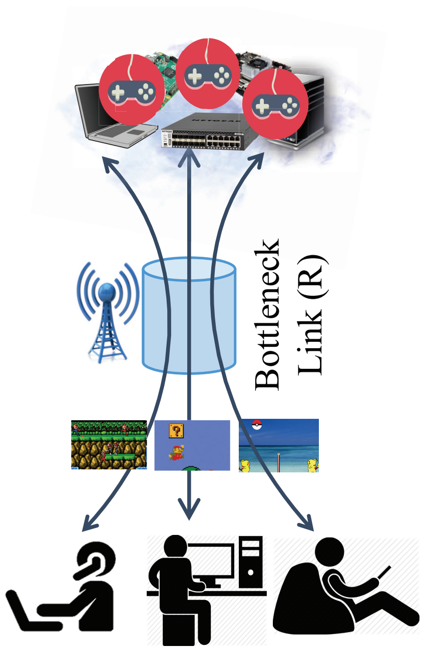


Figure 5.4: Adapting videos of GA servers deployed in cloud-to-things continuum platforms.

Table 5.1: Symbols Used Throughout this Chapter

Sym.	**Description**
U	Number of gamers
R	Available bandwidth over a shared link
F	Maximal frame rate
B	Maximal bitrate
G	Number of game types
p_u	A game played by gamer u
$m_{g,f,b}$	MOS score of playing game g at frame rate f and bitrate b

respectively. The frame rates and bitrates are specified by the system administrators. We let g ($1 \leq g \leq G$) be the game types supported by the our platform, and p_u be the game played by gamer u, where $1 \leq u \leq U$ and $1 \leq p_u \leq G$. We let $m_{g,f,b}$ be the MOS score of playing game g at frame rate f and bitrate b, which can be derived via off-line user studies or online regression.

For concrete discussions, we adopt the user study reported in the study from Hong et al. [104]. Hong et al. [104] implement a crowdsourcing online platform, which recruit 101 subjects to conduct a user study and create MOS models of gaming experience. In the user study, Hong et al. [104] vary bitrate, frame rate, and game in different configurations. We note that the MOS model is designed to drive the codec reconfiguration decisions. The MOS model implicitly considers the available bandwidth (bitrate), and ignores the network latency. This is because the network latency is not controllable by our cloud-to-things continuum platforms during ongoing gaming sessions. When a game session is affected by higher packet loss rate or longer end-to-end delay, our bandwidth estimator (Sec. 5.3.1) reports lower effective bandwidth. Then, the bitrate adaptation algorithm picks a new target bitrate (as a function of effective bandwidth), using the MOS model that converts the bitrate and frame rate into the expected MOS score. Different from the MOS model, there are full-fledged QoE models in the literature for precise user experience. For example, Game Mean Opinion Score (GMOS) [172, 174, 183] is an objective quality metric, which is a function of codec, frame rate, resolution, PSNR (Peak Signal-to-Noise Ratio), network latency, and packet loss rate. Compared to my selected relatively simple MOS model, GMOS is too complicated for video codec reconfiguration purpose. More importantly, the additional complexity comes from factors that are not directly controllable by our cloud-to-things continuum platforms. The quadratic MOS model is given below:

$$m(g, f, b) = \alpha_{g,1} f + \alpha_{g,2} b + \alpha_{g,3} f^2 + \alpha_{g,4} b^2 + \alpha_{g,5} f b + \alpha_{g,6}, \tag{5.1}$$

where $\alpha_{g,1}$–$\alpha_{g,6}$ are game-specific model parameters. The details of the model are summarized in Table 5.2.

Table 5.2: The MOS Models for Individual Games

	Batman	FGPX	CoD
	(1)	(2)	(3)
fps, $\alpha_{g,1}$	0.010 (0.036)	0.038 (0.041)	−0.011 (0.037)
rate (in Mbps), $\alpha_{g,2}$	2.940*** (0.484)	2.297** (0.553)	2.529** (0.491)
I(fpsˆ2), $\alpha_{g,3}$	−0.001 (0.001)	−0.002 (0.001)	−0.001 (0.001)
I(rateˆ2), $\alpha_{g,4}$	−1.150*** (0.176)	−0.868** (0.201)	−0.939** (0.178)
fps:rate, $\alpha_{g,5}$	0.043** (0.008)	0.036** (0.009)	0.037** (0.008)
Constant, $\alpha_{g,6}$	2.248** (0.428)	2.369** (0.490)	2.621*** (0.434)
R^2	0.988	0.981	0.987
Adjusted R^2	0.968	0.949	0.966
F Statistic (df = 5; 3)	49.518***	31.040***	46.130***

Note: $^*p<0.1$; $^{**}p<0.05$; $^{***}p<0.01$

Next, we let $\hat{k}_u$ and $\check{k}_u$ be the maximal and minimal bitrates of gamer u, respectively. The minimal $\check{k}_u$ is assigned by the system administrator based on some practical concerns. For example, a rule-of-thumb may indicate any bitrates less than 50 kbps do not produce meaningful reconstructed videos, leading to $\check{k}_u = 50$ kbps. $\hat{k}_u$, on the other hand, comes from a common property of hybrid video coders: the video quality saturates when the bitrate is increased [178]. Hence, it yields less significant improvements while the bitrate is increased beyond, e.g., $\hat{k}_u = 5$ Mbps. Moreover, the gamer's access link may be narrow and shared by multiple users/applications, leading to a bandwidth limitation on u's access link. $\hat{k}_u$ is also used to accommodate this bandwidth limit: the administrator can set $\hat{k}_u$ to be the minimum between the link bandwidth limitation and the quality-saturating bitrate, so that the allocated bitrate to u will not be wasted.

5.4.2 Problem Statement

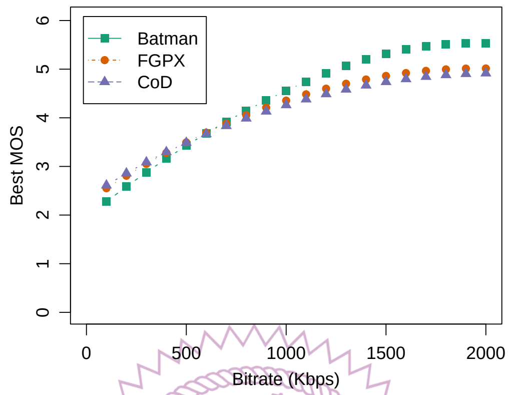


Figure 5.5: The best MOS scores under different bitrates.

Our adaptation problem is to select the best frame rate and bitrate for each gamer, in order to maximize the overall gaming quality under the bandwidth constraints. We consider two optimization criteria: (i) maximizing the average MOS score and (ii) maximizing the minimum MOS score across all gamers. We refer to these two optimization problems as: (i) quality-maximization and (ii) quality-fairness problems, respectively throughout this chapter. We let x_u and y_u ($\forall\ 1 \le u \le U$) be the decision variables, where $1 \le x_u \le F$ (frame rate) and $1 \le y_u \le B$ (bitrate). With the defined notations, we write our problem with the quality-maximization objective as:

$$maximize \sum_{u=1}^{U} m_{p_u,x_u,y_u} \tag{5.2a}$$

$$s.t.: \sum_{u=1}^{U} y_u \le R; \tag{5.2b}$$

$$\check{k}_{u'} \le y_{u'} \le \hat{k}_{u'},\ \forall\ 1 \le u' \le U; \tag{5.2c}$$

$$1 \le x_u \le F, 1 \le y_u \le B,\ \forall\ 1 \le u \le U. \tag{5.2d}$$

For the quality-fairness objective, we replace Eq. (5.2a) with:

$$maximize \min_{u=1}^{U} m_{p_u,x_u,y_u}. \tag{5.3}$$

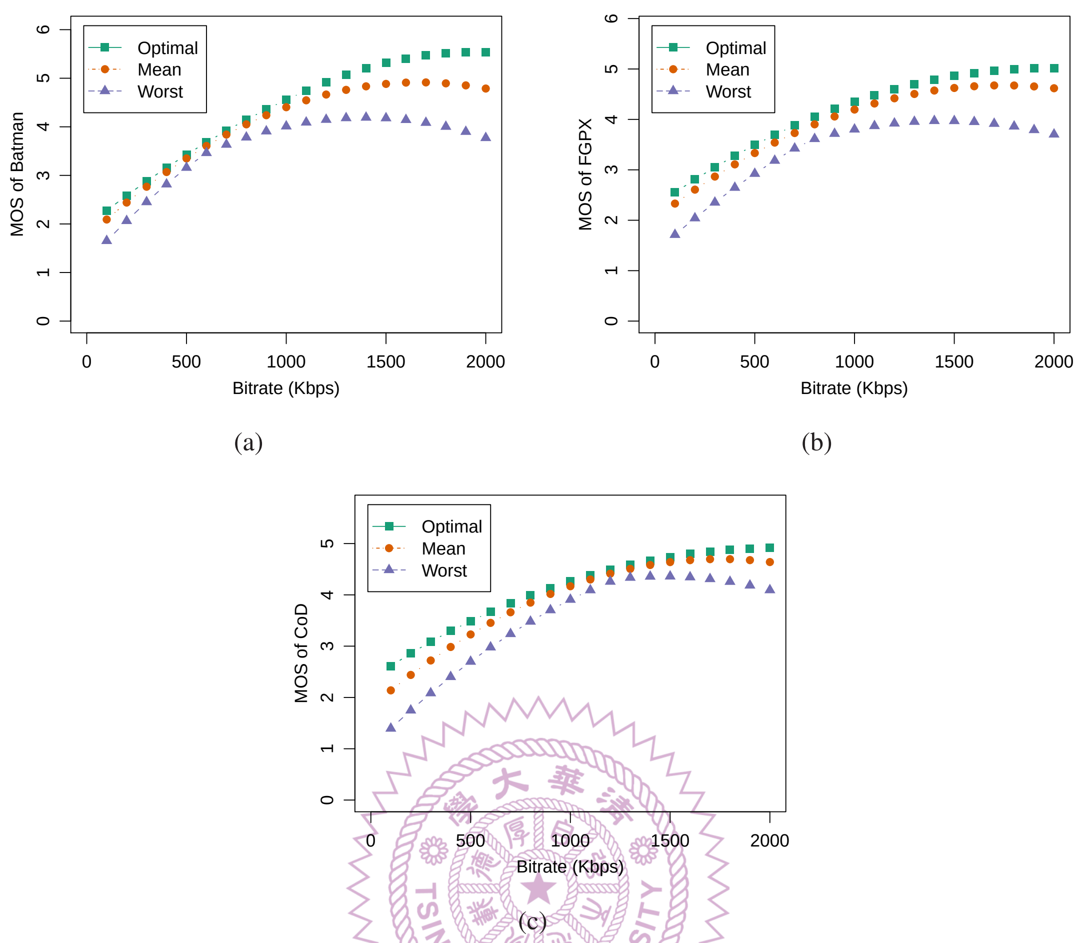


Figure 5.6: Benefits of choosing optimal frame rate of different games: (a) Batman, (b) FGPX, and (c) CoD.

The optimization problems are solved periodically, say once every T seconds in order to accommodate to the system and network dynamics. In the next two sections, we present optimal and efficient algorithms to solve the problems.

5.4.3 Optimal Algorithms

Solving the quality-maximization and quality-fairness problems is challenging due to the complex relations among the frame rates, bitrates, games, and MOS scores. The problems can be solved using commercial problem solvers. We use CPLEX [21] to solve our problems in Eqs. (5.2a)–(5.2d) and (5.3). The CPLEX comes with two solvers: CPLEX and CP solvers. In our problems, because of the *min(imum)* operation in Eq. (5.3), we have to use the CP solver. We refer to the CPLEX based algorithms for the quality-maximization and quality-fairness problems as OPT$_{avg}$ and OPT$_{mm}$, respectively.

5.4.4 Efficient Algorithms

While the OPT$_{avg}$ and OPT$_{mm}$ algorithms give the optimal bitrate allocation, they suffer from exponential running time, and are not suitable to delay-sensitive applications like game streaming applications. Next, we develop two efficient algorithms. The intuition behind the efficient algorithms is iteratively allocating some bandwidth to the gamer that can boost the objective function value the most. Given that we iteratively *add* more bitrate to each gamer, the bitrate of any gamer is known at each step. With the MOS model in Eq. (5.1), we can derive the optimal frame rate $f^*(b, g)$ for a gamer playing game g at the accumulated bitrate b. More specifically, by taking the partial derivative of Eq. (5.1) with respect to f, we have the optimal frame rate as:

$$f^*(b, g) = \frac{-(\alpha_{g,1} + \alpha_{g,5}b)}{2\alpha_{g,3}}, \tag{5.4}$$

which selects the optimal coding parameter for a single gamer. Next, we write the MOS scores under $f^*(b, g)$ as $mos^*(g, b)$, where:

$$\begin{aligned} mos^*(g, b) =& \alpha_{g,1} f^*(g, b) + \alpha_{g,2} b + \alpha_{g,3} (f^*(g, b))^2 \\ &+ \alpha_{g,4} b^2 + \alpha_{g,5} f^*(g, b) b + \alpha_{g,6}. \end{aligned} \tag{5.5}$$

We plot the best MOS scores $mos^*(g, b)$ derived from the model parameters (given in Table 5.2) in Fig. 5.5. This figure shows that $mos^*(g, b)$ curves are monotonically increasing and saturate at ~ 2 Mbps. We also plot the optimal, mean, and worst MOS scores of different games in Fig. 5.6. This figure reports how much benefit we can get when choosing the bitrates using Eq. (5.4). The gaps between the optimal and the worst MOS scores in Batman, FGPX, and CoD are up to 1.885, 1.410, and 1.122, respectively. Such gaps are nontrivial, and thus show the effectiveness of our approach.

Using Eqs. (5.4) and (5.5), we essentially transform the problem of choosing the best x_u (frame rate) and y_u (bitrate) into selecting the best y_u. For the efficient algorithms, we start from setting $\hat{y_u} = \check{k}_u$ for all $1 \leq u \leq U$, and iteratively add bitrate to the gamer that needs additional bitrate the most. We let w be the unit of allocating additional bitrate, and set $w = 1$ kbps if not otherwise specified. At each step, we define an MOS score improvement function c_u as:

$$c_u = mos^*(p_u, \hat{y_u} + w) - mos^*(p_u, \hat{y_u}), \tag{5.6}$$

which quantifies the benefits of investigating bandwidth w to gamer u. We also use $\hat{R}$ to denote the residue bandwidth on the bottleneck link.

For the quality-maximization problem, we iteratively allocate w to the gamer with the highest MOS improvement c_u that has not exceeded the bandwidth limitation and quality-saturating bitrate (Eq. (5.2c)). More specifically, we first put all U gamers in a heap sorted on their c_u in the descending order. We then follow this order to allocate the residue bandwidth until we reach the limitation of the constraint $\sum_{u=1}^{U} \hat{y_u} \leq R$. The pseudocode of our proposed EFF$_{avg}$ is given in Fig. 5.7. For the quality-fairness problem, we iteratively allocate w to the gamer playing with the lowest MOS score. More specifically, we first put all the gamers in a heap sorted on their current MOS scores $mos^*(p_u, \hat{y_u})$ in the ascending order. We then follow the order to allocate w to gamer u, as long as the allocation does not violate the limitation of the constraint $\sum_{u=1}^{U} \hat{y_u} \leq R$. The pseudocode of our proposed EFF$_{mm}$ is given in Fig. 5.8.

```
 1: let $\hat{R} = R$
 2: store gamers in a heap on quality improvement $c_u(\cdot)$ in the dsc. order
 3: while $\hat{R} > 0$ do
 4:     pop and remove the gamer $u$ with the maximal $c_u(\cdot)$ from the heap
 5:     if allocating $\hat{y_u} + w$ on $u$ satisfies (5.2c) then
 6:         let $\hat{y_u} = \hat{y_u} + w$
 7:         let $\hat{R} = \hat{R} - w$
 8:         insert the gamer $u$ to the heap
 9:     else
10:         remove gamer $u$ from the heap
11: let $y_u^* = \hat{y_u}\ \forall\ 1 \leq u \leq U$
12: return all $y_u^*$
```

Figure 5.7: The pseudocode of the EFF$_{avg}$ algorithm.

```
 1: let $\hat{R} = R$
 2: store gamers in heap on MOS scores $mos^*(\cdot)$ in the asc. order
 3: while $\hat{R} > 0$ do
 4:     pop and remove the gamer $u$ with the minimal $mos^*(p_u, \hat{y_u})$ from the heap
 5:     if allocating $\hat{y_u} + w$ on $u$ satisfies (5.2c) then
 6:         let $\hat{y_u} = \hat{y_u} + w$
 7:         let $\hat{R} = \hat{R} - w$
 8:         insert the gamer $u$ to the heap
 9:     else
10:         remove gamer $u$ from the heap
11: let $y_u^* = \hat{y_u}\ \forall\ 1 \leq u \leq U$
12: return all $y_u^*$
```

Figure 5.8: The pseudocode of the EFF$_{mm}$ algorithm.

Lemma 7 (Optimality of EFF$_{mm}$)**.** *The EFF$_{mm}$ algorithm produces optimal bitrate allocation.*

Proof. The EFF$_{mm}$ algorithm always allocates residue bandwidth w to gamer u with the lowest MOS score at each step. Consider any alternative allocation of w to a gamer $u' \neq u$. We note that adding w to $y_{u'}$ does not improve the objective function value in Eq. (5.3). Compared to allocating w to u, this alternative allocation must be followed by allocating another w bandwidth to u to reach the same objective function value, as gamer u is the gamer with the lowest MOS score. Hence, no alternative allocation can lead to better solution when $\hat{R} = 0$. This yields the lemma. □

Lemma 8 (Optimality of EFF$_{avg}$)**.** *The EFF$_{avg}$ algorithm produces optimal bitrate allocation if the slope of $mos^*(g, b)$ monotonically decreases when the bitrate is increased.*

Proof. The MOS score improvement c_u defined in Eq. (5.6) is proportional to the slope. Since EFF$_{avg}$ allocates residue bandwidth to the gamer with the highest c_u, the algorithm always finds the steepest slope at the current step (locally) and across all future steps (globally). This yields the lemma. □

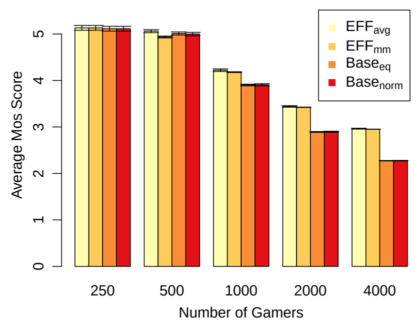
Next, we check whether the slope of $mos^*(g, b)$ is monotonically decreasing under the empirically-derived model parameters. First, we take double derivative of $mos^*(g, b)$ in Eq. (5.5), which leads to:

$$mos^*(g, b)'' = 2\alpha_{g,4} - \alpha_{g,5}^2/2\alpha_{g,3}. \quad (5.7)$$

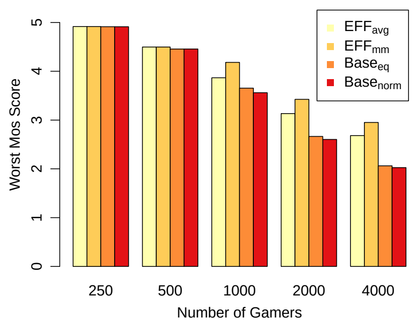
$mos^*(g, b)$ is monotonically decreasing if the value of this equation is negative. The model parameters in Table 5.2 satisfy the condition. Hence EFF$_{avg}$ is optimal.

Lemma 9 (Time Complexity)**.** *The EFF$_{avg}$ and EFF$_{mm}$ algorithms terminate in polynomial time.*

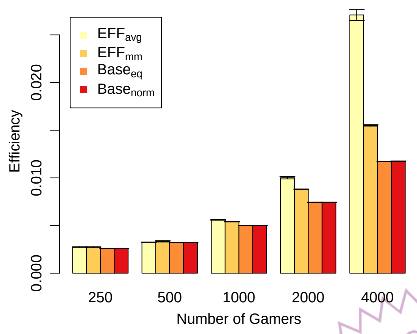
Proof. We first create the max heap and min heap for EFF$_{avg}$ and EFF$_{mm}$, respectively. For the outer loop, we go through all residue bandwidth $\hat{R}$. We also make sure that in each iteration, the value of $\hat{R}$ is decreased at least w kbps, which is a small integer. Hence, the complexity of the outer loop is $O(R)$. Inside the loop, we first pop and remove the first gamer. In this step, the time complexity is $O(log(U))$. Next, we check the condition and try to modify the bitrate of the gamer u. If the bitrate is changed, we insert u to the heap and decrease the value of $\hat{R}$, which also costs $O(log(U))$. Hence, the complexity of the two efficient algorithms is $O(Rlog(U))$. □



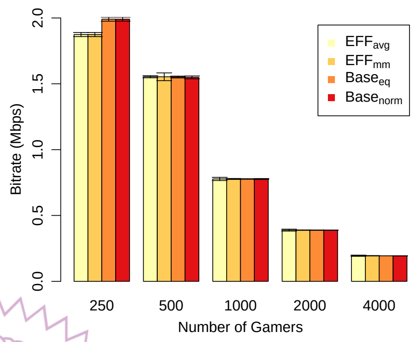
(a)



(b)



(c)



(d)

Figure 5.9: Performance of different algorithms: (a) mean MOS scores, (b) worst MOS scores, (c) overall efficiency, and (d) allocated average bitrate per gamer.

5.5 Evaluations

5.5.1 Setup

We build our simulator using Java and implement the proposed efficient algorithms in the simulator. For comparisons, we also implement two baseline adaptation algorithms called Base$_{eq}$ and Base$_{norm}$. The Base$_{eq}$ algorithm equally allocates the available network resources to the gamers. Base$_{norm}$ algorithm allocates the available network resources to gamers proportional to the average MOS scores of the games played by individual gamers. For example, the overall MOS score of Batman is higher, and Base$_{norm}$ allocates more network resources to Batman for better overall MOS scores. For brevity, Base$_{eq}$ and Base$_{norm}$ set the frame rate to be 30 fps if not otherwise specified.

For realistic simulations, we drive the simulator using real traces. Each new gamer u is randomly assigned a bandwidth $\hat{k}_u$ based on the worldwide bandwidth dataset collected between January and October 2014 [36]. Each gamer also randomly selects a game from Batman, FGPX, and CoD. We use Amazon EC2 and PlanetLab nodes to collect traces of data center bandwidth R. We create a virtual machine in Amazon's data center in Virginia, and select four PlanetLab nodes at the US East Coast. We use `iperf` to measure the bandwidth from the EC2 to each PlanetLab node once every 15 minutes for a whole day on October 12, 2014. Then, for each sample, we pick the maximal bandwidth across all four nodes, and use it as the data center bandwidth R. The average R over the 96 samples is 760 Mbps.

Table 5.3: MOS Scores of the Efficient and Optimal Algorithms

# of Gamers	**OPT**$_{avg}$	**EFF**$_{avg}$	# of Gamers	**OPT**$_{mm}$	**EFF**$_{mm}$
	Mean	Mean		Worst	Worst
100	5.16	5.16	1	5.53	5.53
200	5.15	5.15	2	5.01	5.01
400	5.15	5.15	4	4.91	4.91
800	4.49	4.49	8	4.91	4.91

Table 5.4: Running Time of the Efficient and Optimal Algorithms (s)

# of Gamers	**OPT**$_{avg}$		**EFF**$_{avg}$		# of Gamers	**OPT**$_{mm}$		**EFF**$_{mm}$	
	Mean	Max	Mean	Max		Mean	Max	Mean	Max
100	0.02	0.03	0.090	0.097	1	0.02	0.01	0.009	0.009
200	0.15	0.16	0.102	0.103	2	0.36	0.37	0.015	0.016
400	0.95	0.96	0.153	0.157	4	4.47	4.48	0.022	0.023
800	5.01	5.02	0.237	0.248	8	125.8	127.1	0.029	0.029

Using the traces, we run multiple 1-day simulations. We configure the simulator to solve the adaptation problem once every $T = 60$ seconds if not otherwise specified. We vary the number of gamers $U \in \{250, 500, 1000, 2000, 4000\}$. We run the simulations on a PC with a 2.8 GHz i7 processor and 16 GB memory. The considered metrics are:

- *MOS:* the overall MOS score. We report both average MOS scores across all gamers, and the worst MOS scores among them. They align with objective functions of the quality-maximization and quality-fairness problems.
- *Bitrate:* the bitrate consumed by each gaming session.
- *Efficiency:* the ratio between the MOS score over the consumed bandwidth in Mbps.
- *Running time:* the execution time of each run of the adaptation algorithms.

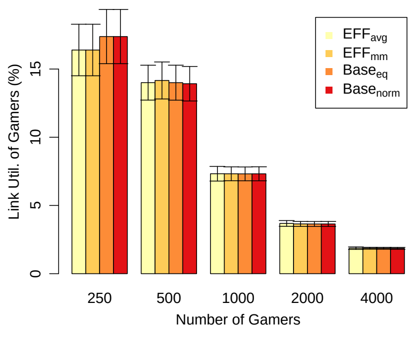
In the next section, we report average results with 95% confidence intervals whenever applicable.

5.5.2 Results

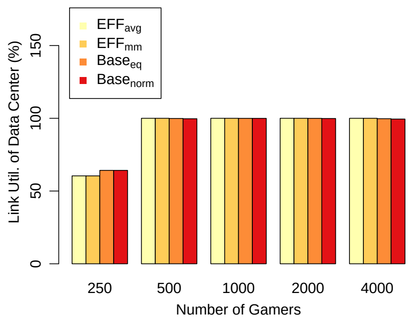
Comparisons between the optimal and efficient algorithms. We first compare the efficient algorithms against the optimal ones. Invoking OPT$_{avg}$ and OPT$_{mm}$ potentially takes prohibitively long time, and thus we set $U \in \{100, 200, 400, 800\}$ and $U \in \{1, 2, 4, 8\}$ for

OPT$_{avg}$ and OPT$_{mm}$, respectively. We run each setting for a day and report the average MOS scores and running time in Tables 5.3 and 5.4. Table 5.3 shows that our proposed efficient algorithms indeed produce the optimal adaptations, as proved in Lemmas 10 and 8. Table 5.4 reveals that the efficient algorithms run much faster than the optimal ones: more than 21 and 4337 times of running time reductions are observed for quality-maximization and quality-fairness problems. The running time gap is going to be even bigger with more gamers, as EFF$_{avg}$ and EFF$_{mm}$ run in polynomial time, but OPT$_{avg}$ and OPT$_{mm}$ do not. Given that the efficient algorithms achieve the optimal adaptations in polynomial time, we no longer consider the optimal algorithms in the rest of this section.

Comparisons between the efficient and baseline algorithms. We first report the performance results of the efficient and baseline algorithms in Fig. 5.9. Fig. 5.9(a) and 5.9(b) show that the proposed EFF$_{avg}$ and EFF$_{mm}$ achieve the design goals: (i) EFF$_{avg}$ leads to the highest average MOS scores and (ii) EFF$_{mm}$ results in the highest worst MOS scores. Moreover, they outperform the two baseline algorithms by up to 30% and 46%. Fig. 5.9(c) reveals that EFF$_{avg}$ leads to higher efficiency than EFF$_{mm}$, but both of them outperform the baseline algorithms. Last, Fig. 5.9(d) shows that when $U \geq 500$, all four algorithms allocate roughly equal bitrate to gamers. This indicates that the superior performance of our efficient algorithms is not built upon excessive resource consumption. We take a closer look by reporting the link utilization in Fig. 5.10. Fig. 5.10(a) reveals that our efficient algorithms lead to gamer link utilization no larger than those of the baseline algorithms. Furthermore, more gamers result in lower gamer link utilization, indicating that the bottleneck is at the data center. This is confirmed by Fig. 5.10(b): the data center link utilization reaches 100% when there are more than 500 gamers. Another observation on Fig. 5.10 is that no algorithm overloads the links, which illustrates the correctness of our algorithm design and simulator implementation.

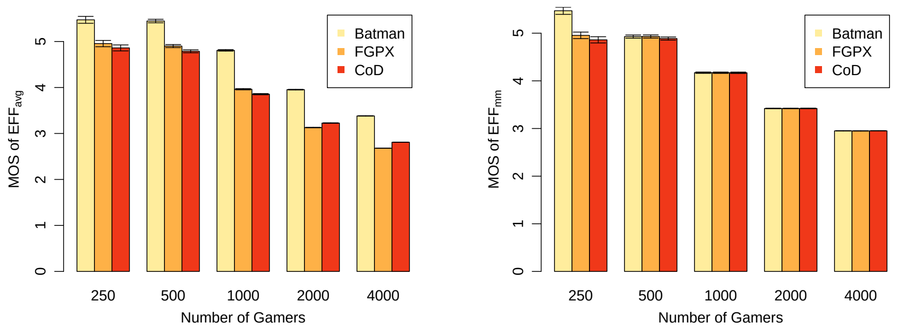


(a)



(b)

Figure 5.10: Link utilization: (a) across all gamers and (b) of the data center under different adaptation algorithms.



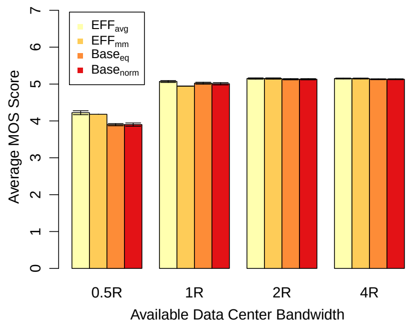
(a) (b)

Figure 5.11: Mean MOS scores under different numbers of gamers using two adaptation algorithms: (a) EFF$_{avg}$ and (b) EFF$_{mm}$.

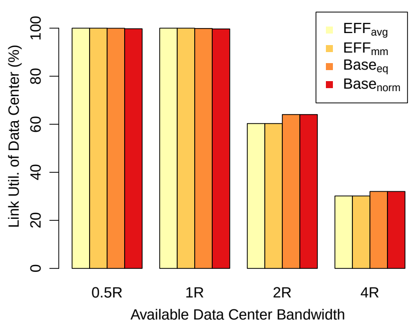
Difference between EFF$_{avg}$ and EFF$_{mm}$. The two efficient algorithms target different optimization criteria, and are optimal in terms of mean MOS scores and minimum MOS scores, respectively. We plot the average MOS scores of individual games in Fig. 5.11. Fig. 5.11(a) shows that with EFF$_{avg}$, the gamers playing Batman achieve higher MOS scores, which can be attributed to the optimization criterion: investing bandwidth on Batman leads to higher improvement on the MOS scores. In contrast, Fig. 5.11(b) demonstrates that EFF$_{mm}$ indeed achieves fairness on the MOS scores: gamers playing all three games achieve the same MOS score except when $U = 250$. A closer look (cf. Fig. 5.10(b)) indicates that when there are fewer gamers, there are actually more than enough bandwidth for all gamers to achieve the highest MOS scores. This demonstrates that our EFF$_{mm}$ algorithm is designed in a way that it does not only maintain fairness, but also capitalize all the available resources.

Implication of different data center available bandwidth. We report the MOS scores under different values of $R \in \{0.5R, 1R, 2R, 4R\}$ in Fig. 5.12. It shows that when the available bandwidth becomes higher, the gap between our efficient algorithms and the baseline algorithms becomes smaller. This can be explained by Fig. 5.12(b), which shows that higher available bandwidth results in low data center link utilization. More specifically, when the data center bandwidth increases, gamers' links become the bottlenecks. This in turn reduces the optimization room of our efficient algorithms, and results in smaller gaps.

Our efficient algorithms scale to large problems. To verify the scalability of our efficient algorithms, we measure the running time of the EFF$_{avg}$ and EFF$_{mm}$ algorithms under various number of gamers. We report the average running time in Table 5.5, which shows that it takes at most ~ 1.7 seconds to solve an adaptation problem with more than 8000 gamers, showing that the efficient algorithms scale to large scale cloud-to-things



(a)



(b)

Figure 5.12: Impacts of different available bandwidth R: (a) average MOS scores and (b) data center link utilization.

continuum platforms running with game streaming applications.

Table 5.5: Running Time in Seconds

# of Gamers	**EFF**$_{avg}$		**EFF**$_{mm}$	
	Mean	Max	Mean	Max
500	0.181	0.183	0.179	0.184
1000	0.296	0.299	0.287	0.290
2000	0.523	0.531	0.520	0.533
4000	1.000	1.104	1.060	1.066
8000	1.677	1.681	1.654	1.661

5.6 Discussion

In this chapter, we studied the problem of adapting game streaming sessions to maximize the gamer experience in dynamic environments in three steps. We extend and modify gamer experience model and implement adaptation mechanism from Hong et al. [104]. With the gamer experience model and adaptation mechanism, we presented two formulations with different optimization criteria of: (i) maximizing the average MOS scores across all gamers and (ii) maximizing the minimum MOS scores among all gamers. We then proposed two optimal and two efficient algorithms to solve these two adaptation problems. We analytically showed that the proposed efficient algorithms run in polynomial time, yet achieve optimal adaptations. We carried out extensive trace-driven simulations, and the simulation results comply with our analysis. In addition, the proposed efficient algorithms: (i) outperform the baseline algorithms by up to 46% and 30%, (ii) run fast and scale to large ($\geq$ 8000 gamers) problems, and (iii) indeed achieve the user-specified optimization criteria.

Chapter 6

Application Specific Optimization: Delay-Insensitive Application

After deploying the applications with the algorithms proposed in Chapter 4, we start to optimize the running delay-insensitive applications in this chapter. We select one representative delay-insensitive application, which will be introduced in Sec. 6.1. We will survey the related studies in Sec. 6.2, conduct a user study to know user experience in Sec. 6.3, write problem statements for optimizing the applications and design my solutions in Sec. 6.4, evaluate them in Sec. 6.5, show a real deployed testbed in Sec. 6.6, and report a summary in Sec. 6.7.

6.1 Multimedia Content Delivery Applications

Our selected delay-insensitive application is multimedia content delivery applications over challenged networks. Multimedia content has dominated the global Internet traffic and shows no trend of slowing down, e.g., Cisco's report predicts that video traffic alone will take more than 80% of the Internet traffic by 2020 [8]. While such reports demonstrate the importance of disseminating multimedia content, not all *world citizens* have the luxury of high-speed Internet access. In fact, International Telecommunications Union (ITU) reports that only 7% of households in the least developed countries have the Internet access compared with the world average of 46% [27]. In addition, although global mobile subscriptions are excepted to grow to more than 7.2 billion by the end of 2016, a large fraction of these subscriptions do not have access to the Internet, especially in developing countries and rural areas. For example, even including major cities like Cairo, Mumbai, and Shanghai, only about 15%, 21%, and 30% of mobile users in Africa, India, and China have cellular data plans [20, 26, 43]. The above statistics reveal the so-called *digital divide*, which refers to the inequality among world citizens in accessing and using information and communication technologies. Furthermore, given the increasing trend of disseminating information over the Internet in recent years, instead of traditional media outlets (e.g., radio, TV, and newspapers), the digital divide may have detrimental social and economic impacts in developing countries and rural areas.

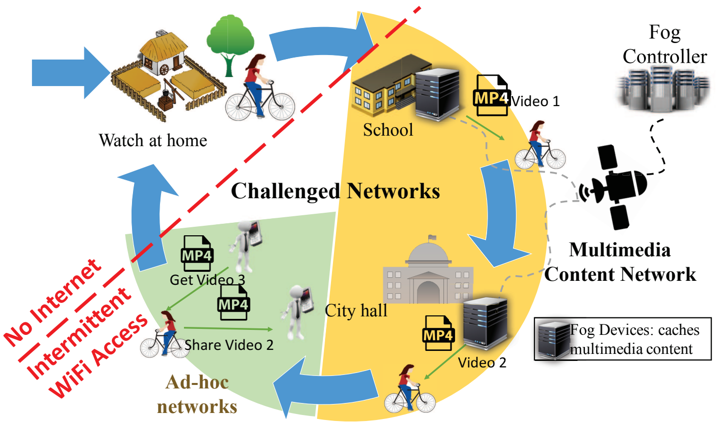


Figure 6.1: High-level operations of the proposed content delivery application running on our cloud-to-things continuum platform for distributing multi-layer multimedia content over challenged networks.

In this chapter, we implement a multimedia content delivery application running on our cloud-to-things continuum platform to reduce the digital divide gap by disseminating multimedia content to users with limited or no Internet access. Specifically, we consider mobile users that may not have cellular data plans, but they are interested in receiving various types of multimedia content such as news reports, notification messages, targeted advertisements, movie trailers, and TV shows. We assume that content providers, such as news agencies, advertisers, and government authorities, are interested in reaching such users for gaining more revenues (through for example ads) or impacting/informing them (through disseminating important messages and news). To do so, content providers push their content to fog devices in various places, such as coffee shops, city halls, public markets, and schools. These fog devices have Internet access through which they can receive multimedia objects from content providers. The fog devices are also connected to local WiFi access points, which can be used by mobile users to connect to the fog devices when they are within the communication ranges.

6.1.1 Usage Scenarios

In Fig. 6.1, we illustrate the high-level operations of the application by considering one day of a user, who owns a mobile device but without a data plan. The user is interested in specific types of multimedia content, such as news and cooking TV shows. The user takes his/her kids to the school, works at the city hall, and buys food at the market, where WiFi access is available and fog devices are installed. When the user is in the range of a WiFi access point, the associated fog device will transmit various multimedia objects to the user's smartphone. These multimedia objects mostly contain what the user is interested in, but they may also contain other objects to be relayed to other mobile users that the user will likely come across during the day. As the user moves throughout the day and runs into other mobile users, the user will exchange different multimedia content with others over an ad-hoc network. By the time the user gets back home, his/her mobile device would have collected (from proxies and other mobiles) most of the multimedia content the user is interested in watching, as well as helped other users in getting the content they are interested in. In addition, based on the routes the user takes, and thus the distributed fog devices and the mobile user comes across and their capacities, the user may obtain the desired multimedia objects at different levels of quality.

We notice that the user in the above example has, in fact, an intermittent network access via a non-traditional network called a *challenged network* [138], which suffers from frequent link downs, long queueing delays, high dynamics, and scarce resources [83]. Disseminating multimedia content over the conventional Internet has been studied in the literature [73, 89, 93, 109]. In particular, Chen et al. [73] propose to save bandwidth of streaming services by predicting the viewers' departure patterns. Hu et al. [109] build a relay network with multiple servers to reduce the streaming end-to-end latency, and solve a server selection problem. The studies in [89, 93] solve the energy-efficient video delivery problems by rate adaptation mechanism and multiple network interfaces, respectively. Because these studies [73, 89, 93, 109] assume always-on Internet access, their solutions do not work in challenged networks. A few other studies consider data communications in challenged networks, e.g., Mota et al. [138] and Ntareme et al. [145] distribute short messages, such as emails and hazard/criminal alarms, whereas Gao et al. [86] propose a solution not designed for multimedia content and heterogeneous user interests. In fact, to our best knowledge, the serving the multimedia content delivery application in a cloud-to-things continuum platform is one of the first of its kind: *it intelligently disseminates multimedia content among users with limited or no Internet access*.

6.1.2 Challenges

The crux of optimizing the application is to create a personalized *distribution plan* for each user, in order to intelligently distribute multimedia content: (i) at the best time, (ii) to the right mobile users, and (iii) at the highest possible quality. The best time refers to the contact with the best channel condition, which in turn results in faster data transfer and lower energy consumption. The right mobile users are the ones who are likely to be interested in the given multimedia content; otherwise, the transfer energy would simply be wasted. The highest possible quality is measured as the average user experience across all watched multimedia content among all mobile users. Achieving all these complex conditions related to multimedia content, mobile users, and intermittent networks makes computing the best personalized distribution plan complicated and challenging. There are three main challenges are listed below:

- **Quantifying user experience** In this chapter, we aim to optimize the multimedia content delivery application to maximize user experience. However, the application is used to solve digital divide. Hence, we cannot use some classic metrics, such as PSNR and number of received contents to model the user experience. We have to design and conduct a user study to model the user experience. Modeling such user experience is challenging because it requires many participants to derive the model. Moreover, conducting the user study and recruiting such large number of participants in a lab is tedious and expensive. In this chapter, we leverage a crowdsourcing platform to conduct the user study (Sec. 6.3).

- **Estimating user behaviors** To deliver the multimedia content to our users, we need to know the behaviors, such as trajectories and interests of the users. However, it is challenging to know these information because users have their own daily life with different trajectories and different users are interested in different multimedia content.

- **Planning content distributions** After solving the first and second challenges, we have user experience and behaviors as inputs to make distribution plans. However, the distribution planning problem is a complicated (actually a NP-hard problem) because it is difficult to optimally make a detailed personalized plans while considering various factors, such as network resources, disk resources, energy resources, user experiences, user interests, and user trajectories.

6.1.3 Software Components

Fig. 6.2 shows three main components for solving the challenges and optimizing the application : (i) Content Matcher, (ii) Contact Predictor, and (iii) Distribution Planning Algorithm. The first two components leverage multiple off-the-shelf machine learning tools and we briefly describe them in the following. The third component contains our novel algorithm for managing the distribution of multi-layer multimedia content to mobile users over challenged networks; it is described in details in Sec. 6.4.

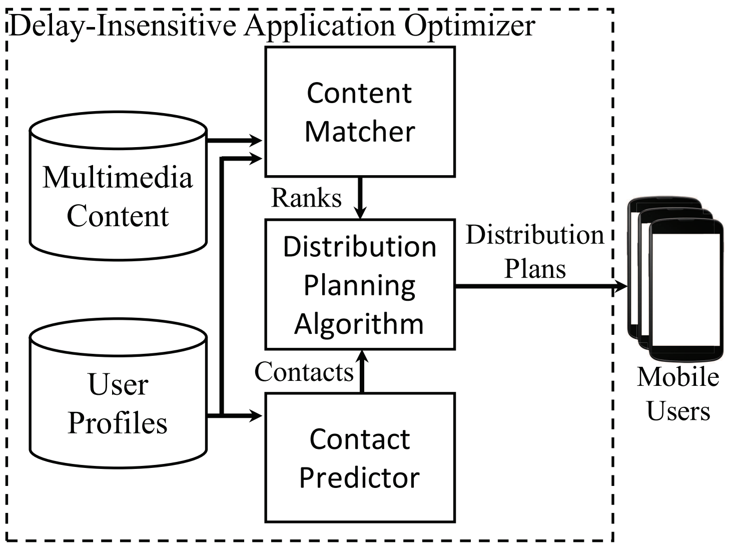


Figure 6.2: The interactions between the multimedia content delivery applications and the delay-insensitive optimizer.

The Content Matcher collects user interests, which may be manually specified by mobile users, or derived from collecting recently watched multimedia content. It then determines a ranked-list of multimedia content that matches users interests. This is done by extracting keywords that represent each multimedia object.

The Contact Predictor estimates the future contact locations of each user using, e.g., trajectory patterns [137], social networks [76], or a frequency-based approach. Upon the contact locations are determined, the contact durations can be predicted using the techniques proposed in [128]. We note that human mobility is highly predictable, and 85% of the time a mobile user stays at his/her top 5 favorite locations [91].

The Distribution Planning Algorithm computes the distribution plans for all known users. The plans are then pushed to each mobile user along with the user's profile to the fog devices that are on the user's contacts. The mobile user fetches the distribution plan when being within proximity of any of these fog devices. Some mobile users may fail to find their own distribution plans, because they are new to the system or dramatically change their daily trajectories. In this case, the nearest or the current fog device to which the user is attached to will assign that user the plan of the closest mobile user profile. Therefore, even if the exact distribution plan is not available at a fog device, the most suitable one can be sent to the mobile user.

We consider different types of multimedia content, including news reports, video clips, notification messages, targeted advertisements, movie trailers, and TV shows. Each multimedia content has different representations that are suitable under different circumstances. For example, for mobile users with a few short contacts, distributing videos to them may not always be possible. In contrast, a well-connected tablet computer user may allocate more energy and disk budgets for high-resolution videos. In fact, each multimedia content can be rendered in the following representations: text (if applicable, e.g., for news), images, left-channel audio, stereo audio, and low-, medium-, and high-resolution videos. Advanced codecs may be used to exploit the redundancy across adjacent representations, e.g., Scalable Video Coding (SVC) [62,160] allows us to *incrementally* encode videos in different resolutions (and other scalability modes), in order to reduce the bandwidth consumption. With these scalable codecs, higher layers are not decodable without lower layers; for example, the high-resolution video representation contains medium- and low-resolution video layers. Hence, there exists a linear dependency among the layers. This is because lower layers typically have much smaller sizes, e.g., if we already request for the image layer, the bandwidth consumed by the text layer is relatively negligible. Last, different user experience is observed when watching multimedia content in different representations. In Sec. 6.3, our user study indicates that the user experience improvement follows a saturated function, when the received data amount increases. For example, moving from nothing to the text of a news report is a huge jump, while moving from medium- to high-resolution videos is less dramatic. Such observation also motivates our multiple representation approach.

6.2 Related Work

We deliver multimedia content over opportunistic networks and we deploy fog devices to cache and disseminate multimedia content. The most important and unique design of our content delivery application is the personalized *distribution plan* which has not been considered by prior studies that only consider: (i) delivering short messages over opportunistic networks, (ii) disseminating multimedia content over wired/wireless networks, and (iii) caching Web content among mobile devices. In the following, we survey the related work in these three categories.

6.2.1 Opportunistic Networks

Opportunistic networks include delay-tolerant networks and challenged networks [138]. Delay-tolerant networks have been studied in the literature [12, 22, 83]. For example, Fall [83] proposes a network architecture composed of resource-constrained mobile devices, which is essentially an overlay network above the transport layer. Challenged networks have also been studied, such as flooding [169], message ferrying [192], and social-based forwarding [115]. Prior studies on forwarding messages in opportunistic networks can be categorized based on assumptions, such as controlled devices, unlimited resources, and predictable contacts. The naive flooding [169] and controlled flooding [95] are less efficient forwarding protocols in practical settings. Message ferrying [192] improves the efficiency based on the knowledge of ferry routes and contact predictability. Device mobility [75], controlled mobility of some nodes [65], and contact history [64] are also used for more efficient opportunistic networks. Capturing intrinsic behavior on social networks, researchers are able to design social-aware forwarding algorithms [115] that consider ranking or centrality information of mobile devices. However, the aforementioned studies focus on short messages instead of multimedia content.

Vahdat and Becker propose Epidemic [169] routing to deliver messages over ad-hoc networks. Epidemic routing transmits all cached content at a node to any other node that gets in contact with it. Epidemic routing results in optimal performance in terms of delivery delay and delivery ratio, but it imposes the highest delivery overhead. Hence, researchers, such as [57], in opportunistic networks area often use Epidemic as a baseline for the performance bound under the assumption of unlimited resources. CSI [108] disseminates messages among mobile devices, and leverages the user behavior to improve the dissemination efficiency. In particular, CSI collects mobility data of mobile users to compute their similarity and disseminates the requested messages accordingly. CSI and its variants [146] are probably the closest work to ours, although they only consider short messages. Since Epidemic [169] and CSI [108] are representative schemes in the literature, we adopt them as the baseline algorithms in the evaluations (Secs. 6.5 and 6.6).

6.2.2 Multimedia Dissemination

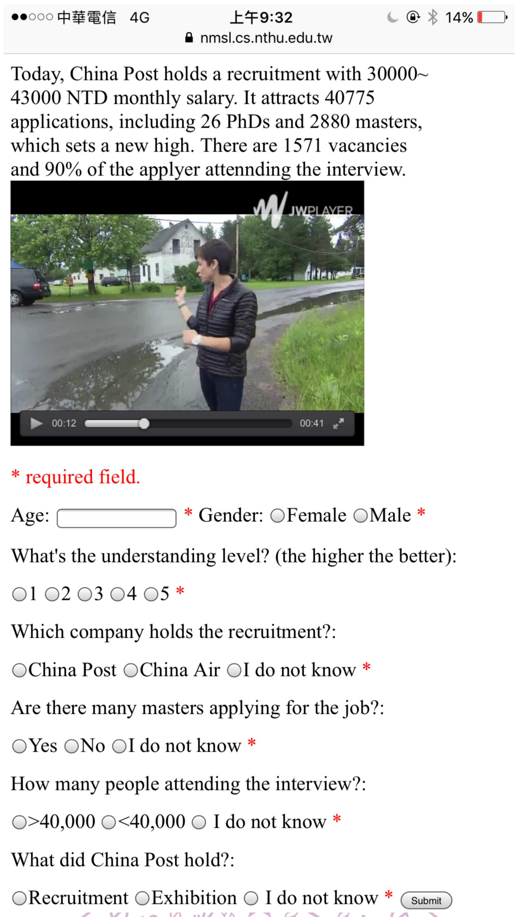
Changuel et al. [70] focus on streaming videos to a large number of users. Kang and Mutka [119] use P2P networks to reduce the cost of disseminating multimedia content using cellular networks. Xiang et al. [182] design a P2P topology overlay based on clustering mechanisms to improve the availability and Quality of Service (QoS). Mokhtarian and Hefeeda [136] study the resource allocation problem in P2P streaming system with multi-layer scalable videos. In Device-to-Device (D2D) networks, Zhou [194] and Zhang et al. [187] optimize for delivery delay and user experience when disseminating multimedia content, respectively. Zhang et al. [188] adopt smartphones in cellular networks as helpers to disseminate multimedia content. Unlike our work, D2D studies [187, 188, 194] assume that cellular infrastructure is available, and smartphone users run into one another very often. Zhang et al. [189] propose a hybrid approach of CDN and P2P networks to disseminate multimedia content. MicroCast [121] proposes to group multiple mobile users, to share their cellular connectivities over short-range auxiliary networks for better video streaming quality. Hanano et al. [94] utilize both WiFi and cellular networks to disseminate video advertisements. Do et al. [79] and HybCast [78] concurrently leverage cellular and ad-hoc networks for (video) file dissemination. Other multimedia dissemination studies [135, 144, 177, 181] focus on multiple representations of multimedia content for dissemination to clients with heterogeneous resources, such as network, energy, and computing power. Different from us, these multimedia studies are not customized for challenged networks, where clients are often without the Internet access.

6.2.3 Caching

Qian et al. [152] show that caching can eliminate redundant network traffic while disseminating Web content. Users' browsing behaviors and machine learning algorithms [132, 191] are adopted by proactive Web content caching. Cooperative caching improves performance of Web applications in opportunistic networks. The technique proposed in [86] caches data in a set of easily accessible mobile devices and exercises the tradeoff between data accessibility and caching overhead. Wang et al. [176] leverage the popularity ranks to support cooperative caching under opportunistic networks via BlueTooth or WiFi. Besides, a cooperative caching system [116] is proposed for interactive Web applications over challenged networks. Unlike our distribution planning algorithm that jointly considers multimedia content, mobile users, and intermittent networks, prior studies on caching only consider limited aspects, e.g., Isaacman and Martonosi [116] do not consider the detailed distribution plans.

6.3 Crowdsourced User Experience of Different Representations

We carefully design a user study by using a crowdsourcing platform [6] to quantify the user experience of different representations of multimedia content. The user experience of different media types may be quantified in different metrics, such as latency (ms), energy (J), and resolution (pixels). For example, latency is the most important user experience metric for interactive multimedia applications, like teleconferencing calls and online games. In this chapter, we focus on disseminating news reports to eliminate digital divide in challenged networks, and thus we adopt *understanding level* as the user experience metric. The understanding level is quantified through questionnaires in Mean Opinion Score (MOS) between 1 and 5. We note that the understanding levels may be affected by not only media types and audio/video quality, but also news structure and complexity. To avoid biased results, we retrieve news reports from a reputable news agency (Apple Daily in Taiwan, as an example), and thus the news reports have comparable structure


●●○○○ 中華電信 4G 上午9:32 14%
nmsl.cs.nthu.edu.tw

Today, China Post holds a recruitment with 30000~43000 NTD monthly salary. It attracts 40775 applications, including 26 PhDs and 2880 masters, which sets a new high. There are 1571 vacancies and 90% of the applyer attennding the interview.

* required field.

Age: * Gender: ○Female ○Male *

What's the understanding level? (the higher the better):

○1 ○2 ○3 ○4 ○5 *

Which company holds the recruitment?:

○China Post ○China Air ○I do not know *

Are there many masters applying for the job?:

○Yes ○No ○I do not know *

How many people attending the interview?:

○>40,000 ○<40,000 ○ I do not know *

What did China Post hold?:

○Recruitment ○Exhibition ○ I do not know * Submit

Figure 6.3: A sample questionnaire used in our user study.

and complexity. We present the detailed user study design below. We emphasize that understanding level is just a sample user experience metric; other user experience metrics should be adopted for other multimedia applications, while our user study procedure is also applicable.

In the user study, each participant watches several random news reports from Apple Daily using a mobile device. After watching each news report, participants fill out their ages and genders. They then answer 5 questions related to the news report. The first question asks the participant to input their understanding level in MOS scores. The scores are normalized to between 0 and 100% in our analysis. The next 4 challenging questions are multiple-choice questions on the news report for checking if a participant really comprehends the news report. We keep track of the viewing time of each news report and the number of correctly-answered challenging questions for filtering purpose detailed below. We also allow each participant to skip a news report anytime. A sample questionnaire and our Web interface opened with a smartphone are shown in Fig. 6.3.

We recruit 182 participants throughout our user study. There are 24 news reports chosen from the following news categories: sports, society, health, finance, politics, history, nature, and life. We generate 5 representations: text, audio, 240p, 360p, and 480p videos, for each news report. For each participant, we play a random news report at a random representation, and a participant may opt to watch more news reports. In total, we gather 2108 user experience scores, and we filter out the scores: (i) with zero correctly-answered challenging questions, (ii) with inconsistent answers in (intentionally) duplicated challenging questions, and (iii) with viewing time shorter than the audio/video length. Eventually, we get 587 valid user experience scores from 120 participants. The average, minimum, and maximum ages of these participants are 29, 16, and 63 years old, respectively. Additionally, 45% of the participants are male. We report the average user experience scores in Table 6.1, and through it we make two observations. First, more layers lead to better user experience. Second, the improvement on user experience is diminishing, e.g., articles give 55% improvement, while 480p videos give 77%-74% = 3% improvement.

The user study results inspire our solution presented in Sec. 6.4, and are used in our evaluations. We note that some studies [162, 163, 171] optimize multimedia disseminating system in different QoS metrics, including delivery delay and energy consumption. Compared to these studies, in this chapter, we aim to eliminate digital divide. Although delivery delay and energy consumption are not our user experience metrics, we carefully design our algorithms and systems to achieve reasonable delivery delay and energy consumption.

Table 6.1: Crowdsourced User Experience Scores

	Article	**Audio**	**240p Video**	**360p**	**480p**
Average	55%	68%	71%	74%	77%
No. Samples	112	112	134	104	125

6.4 Distribution Planning Problem and Solution

6.4.1 Notations

Table 6.2 lists all symbols used in this article. We consider a distributed system that delivers multimedia content to U mobile users. Mobile users communicate with S fog devices. Let N be the total number of multimedia objects and L be the number of layers of each multimedia content. Layer l ($1 \leq l \leq L$) is only decodable/meaningful if all layers $l' \leq l$ have also been received. We define the delivery *unit* as a layer of a multimedia content, and unit $i = nL + l$ is a unique identifier pointing to layer l of multimedia content n. We let ρ_i ($1 \leq i \leq NL$) be the user experience improvement when receiving unit i in addition to all layers beneath it. We let b_i be the size of unit i, and $\psi_{u,n}$ be the

Table 6.2: Symbols Used Throughout this Chapter

Sym.	**Description**
S	Number of fog devices
U	Number of mobile users
N	Number of multimedia content
L	Number of layers of each multimedia content
C_u	Number of contacts of user u
b_i	Size of unit i
$\psi_{u,n}$	Viewing probability on multimedia content n of a mobile user u
ρ_i	User experience improvement of unit i
$\bar{\psi}$	Minimal viewing probability
$p_{u,c}$	Contacted party of user u in contact c
$\kappa_{u,c}$	Contact duration of user u in contact c
$r_{u,c}$	Throughput of user u in contact c
$\hat{e}_{u,c}$	Per-byte transmitting energy consumption of user u in contact c
$\check{e}_{u,c}$	Per-byte receiving energy consumption of user u in contact c
q_u	Energy budget of user u
d_u	Disk budget of user u
$r_{u,c}\kappa_{u,c}$	Network budget of user u in contact c
$\hat{q}'_{u,c}$	Upload energy budget of user u in contact c
$\check{q}'_{u,c}$	Download energy budget of user u in contact c
$d'_{u,c}$	Disk budget of user u in contact c
a	Number of days of historical data for training
τ_u	Contribution potential of user u
ι_u	Number of unit be selected of user u in each round with our algorithm
Z_u	A list to store possible units can be downloaded by user u
F	Maximal segment size

probability of mobile user u to watch multimedia content n. We let $\bar{\psi}$ be the minimal viewing probability: a mobile user would not request a multimedia content from another mobile user who is unlikely to watch it.

We let T be the number of time slots that are considered in our formulation, and $t = 0$ be the starting time slot. We assume that mobile users' trajectories are given, i.e., each user's location at every time slot is provided by some localization and prediction techniques. With mobile users' trajectories and fog devices' locations, the sequence of contacts during time $[0, T]$ is determined. Each contact happens between two parties, which can be either mobile users or fog devices. We let C_u be the number of contacts for user u, and $C = \max_{u=1}^{U} C_u$. A mobile user can have multiple concurrent contacts. In this case, it equally divides the contacts into disjoint contacts along the time domain, where each contact has exactly two parties. That is, simple time-division multiplexing is done to avoid interference due to concurrent transfers. We let $p_{u,c}$ be the other party of contact

c $(1 \leq c \leq C_u)$ of user u $(1 \leq u \leq U)$, where $1 \leq p_{u,c} \leq U + S$. When $p_{u,c} \leq U$, it points to mobile user $p_{u,c}$, while $p_{u,c} > U$, it points to fog device $p_{u,c} - U$. Last, we write the duration of contact c of user u as $\kappa_{u,c}$.

Combining the contacts with trajectories, we can estimate the throughput and energy consumption of each contact. In particular, we write the receiving throughput of contact c of user u as $r_{u,c}$, the transmitting per-byte energy consumption as $\hat{e}_{u,c}$, and the receiving per-byte energy consumption as $\check{e}_{u,c}$. Last, we use q_u and d_u to represent the energy and disk budgets of mobile user u during $t \in [1, T]$. q_u and d_u are user-specified parameters.

6.4.2 Problem Formulation

We first write the distribution plans as $x_{u,n,l,c}$, where $1 \leq u \leq U, 1 \leq n \leq N, 1 \leq l \leq L$, and $1 \leq c \leq C$. $x_{u,n,l,c} = 1$ if mobile user u requests unit $nL+l$ during contact c; $x_{u,n,l,c} = 0$ otherwise. Then, we need to keep track of various types of resources: disk space, battery level, and network traffic. We make an important observation: the amounts of resource consumptions are proportional to unit sizes. Hence, we derive a unified budget $R_{u,c}$ for each contact c of user u, which is the resource cap imposed by the rarest resource among disk, battery, and network. In particular, we define $R_{u,c} = \min\left(\check{q}'_{u,c}, \hat{q}'_{u,c}, d'_{u,c}, r_{u,c}\kappa_{u,c}\right)$, where $\check{q}'_{u,c}$ is the download energy budget, $\hat{q}'_{u,c}$ is the upload energy budget, $d'_{u,c}$ is the disk budget, and $r_{u,c}\kappa_{u,c}$ is the network budget.

The precise derivation of the resource budgets is as follows. We first divide the energy budget q_u into $\check{q}'_u$ and $\hat{q}'_u$ based on the number of contacted fog devices. Specifically, we let $\check{q}_u = \frac{q_u}{2}\left(1 + \frac{\sum_{c=1}^{C_u} \max(\min(p_{u,c}-U,1),0)}{C_u}\right)$, where the term in parentheses is the weight on download energy: running into more fog devices means this user has more chances to download than upload content. We then have $\hat{q}_u = q_u - \check{q}_u$. Next, we allocate the energy budgets to individual contacts, by setting $\check{q}'_{u,c} = \check{q}_u \frac{r_{u,c}\kappa_{u,c}}{\sum_{c=1}^{C_u} r_{u,c}\kappa_{u,c}}$ and $\hat{q}'_{u,c} = \hat{q}_u \frac{r_{u,c}\kappa_{u,c} \max(\min(U-p_{u,c}+1,1),0)}{\sum_{c=1}^{C_u} r_{u,c}\kappa_{u,c} \max(\min(U-p_{u,c}+1,1),0)}$. We notice that we do not allocate upload energy for users who run into fog devices since mobile users never send any multimedia content to fog devices. Last, we allocate the disk budget by setting $d'_{u,c} = d_u \frac{r_{u,c}\kappa_{u,c}}{\sum_{c=1}^{C_u} r_{u,c}\kappa_{u,c}}$, which is also proportional to the network budget normalized across all contacts.

With all the notations developed above, we write our distribution planning problem as:

$$\max \sum_{u=1}^{U}\sum_{n=1}^{N}\sum_{l=1}^{L}\sum_{c=1}^{C_{u'}} x_{u,n,l,c}\, \rho_{nL+l}\, \psi_{u,n} \tag{6.1a}$$

$$st : \psi_{p_{u',c'},n'} \geq \bar{\psi} x_{u',n',l',c'}; \tag{6.1b}$$

$$\sum_{n=1}^{N}\sum_{l=1}^{L} b_{nL+l} x_{u',n,l,c'} \leq R_{u',c'}; \tag{6.1c}$$

$$\sum_{c=1}^{C_{u'}} x_{u',n',l',c} \geq \sum_{c=1}^{C_{u'}} x_{u',n',l'',c}; \tag{6.1d}$$

$$\sum_{c=1}^{C_{u'}} x_{u',n',l',c} \leq 1; \tag{6.1e}$$

$$\forall u' \in [1,U], n' \in [1,N], (l' < l'') \in [1,L], c' \in [1, C_{u'}].$$

$$x_{u,n,l,c} \in \{0,1\}\ \forall u,n,l,c.$$

The objective function in Eq. (6.1a) maximizes the expected overall user experience with viewing probabilities across all mobile users. Eq. (6.1b) makes sure that mobile users never request multimedia content from someone who is unlikely to watch it. Eq. (6.1c) ensures that for each contact: (i) the contact duration is long enough to complete the planned unit transfer under the given transmission throughput, (ii) the total size of planned transmission does not exceed the user's disk budget, and (iii) the total energy does not exceed the energy budget. Eq. (6.1d) captures the layer dependency, i.e., higher layer l'' is only decodable/meaningful when all its lower layers l' are received. Eq. (6.1e) ensures that users do not receive the same unit multiple times, which results in wasted resources.

Lemma 10 (Hardness)**.** *The considered distribution planning problem (Problem 6.1) is NP-Complete.*

Proof. We reduce the Multiple Knapsack Problem (MKP) to our Problem 6.1. The MKP problem is written as:

$$\max \sum_{j=1}^{J}\sum_{k=1}^{K} v_j y_{j,k} \tag{6.2a}$$

$$st : \sum_{j=1}^{J} w_j y_{j,k} \leq O_k\ \forall k = 1,2,\ldots,K \tag{6.2b}$$

$$\sum_{k=1}^{K} y_{j,k} \leq 1; \forall j = 1,2,\ldots,J \tag{6.2c}$$

$$y_{j,k} \in \{0,1\}\ \forall j = 1,2,\ldots J, k = 1,2,\ldots,K. \tag{6.2d}$$

In the MKP problem, we consider J objects and K knapsacks. The boolean decision variable $y_{j,k}$ indicates whether we want to put object j into knapsack k, while v_j represents the profit of having object j. Each knapsack k has its capacity O_k, which is a resource limit and each object j consumes a given amount of resource $w_{j,k}$. The MKP problem strives to pick a subset of objects, so that the total profit is maximized, while none of the constraints are violated.

Given an MKP problem, we generate a corresponding Problem 6.1, as follows. First, we let $U = 1$, $L = 1$, and $\psi_{u,n} = 1$, which means only one user would like to receive multimedia content from fog devices. Moreover, each content has only one layer and the viewing probability is 100%. Second, because we let $U = 1$, we write the number of contacts C_u of user u as C and the resource constraint $R_{u,c}$ as R_c. We then let $C = K$ and $R_c = O_k$. Third, because we let $L = 1$, we write the size of units b_{nL+l} as b_n and user experience improvement ρ_{nL+l} as ρ_n. We then let $b_n = w_j$ and $\rho_n = v_j$. This results in a proper instance of Problem 6.1 in polynomial time. In addition, a solution of Problem 6.1 can be verified in polynomial time. Because we let $U = 1$ and $L = 1$, the decision variable $x_{u,n,l,c}$ can be written as to $x_{n,c}$. Hence, we let $x_{n,c} = y_{j,k}$, which yields a solution of the MKP problem. Since MKP problem is NP-Complete, Problem 6.1 is also NP-Complete. □

6.4.3 Optimal Algorithm: DP

We solve the formulation in Eq. (6.1) using a Dynamic Programming (DP) algorithm, which systematically skips redundant computations. DP memorizes the computed user experience of a user u while downloading unit i during contact c with remaining unified budget $R'_{u,c}$, where $0 \leq R'_{u,c} \leq R_{u,c}$. Fig. 6.4 shows the pseudocode of our algorithm, which recursively decides whether user u downloads unit i from one of the contacts or does not download the unit, until the unified budget is used up or all units have been considered. The proposed DP algorithm essentially conducts a grid search, and goes through all possible solutions. Hence, the DP algorithm gives optimal solution at an expense of high computational complexity, which is discussed in Lemma 11.

Lemma 11 (Time Complexity). *The DP algorithm has an exponential time complexity.*

Proof. The dynamic programming algorithm recursively solves the problem. Each recursive function creates other $C + 1$ recursive functions as its children. Thus, the worst-case time complexity of creating a plan for a user is $O((C+1)^{NL})$. We need to create U plans, thus $O(U(C + 1)^{NL})$ is the time complexity of our DP algorithm. This exponential time complexity may render the DP algorithm not feasible for some large problems. Hence, we develop a heuristic algorithm in the following section. □

Input: User profiles, such as viewing probability $\psi_{u,n}$, multimedia content information, such as size of a unit b_i, and resource budgets $R_{u,c}$
Output: Distribution plan M

```
1:  let $M$ as the table to memorize the computed values
2:  for all user $u$, contact $c$, and unit $i$ do
3:      Call Procedure DP $(i, c, R_u, R_{u,c})$
4:  procedure DP$(i, c, R_u, R'_{u,c})$ $R_{u,c} = R'_{u,c}$
5:      if $R'_{u,c} \leq 0$ then //remaining resource is not enough
6:          return $-\infty$ //return negative user experience
7:      if $i = 0$ then //all the units have been put into the plan
8:          return 0
9:      if $M[i][R_{u,1}][R_{u,2}]...[R_{u,C_u}]$ exists then
10:         return $M[i][R_{u,1}][R_{u,2}]...[R_{u,C_u}]$
11:     let $M[i][R_{u,1}][R_{u,2}]...[R_{u,C_u}] =$
12:     //download from contact $c$
13:     $\max(DP(i-1, c, R_u, R_{u,c} - b[i]) + \rho_i \psi_{u,i/L})\ \forall c \in [1, C_u]$
14:     let $M[i][R_{u,1}][R_{u,2}]...[R_{u,C_u}] =$
15:     // do not download
16:     $\max(DP(i-1, 0, R_u, R_{u,c}), M[i][R_{u,1}][R_{u,2}]...[R_{u,C_u}])$
```

Figure 6.4: The pseudocode of our DP algorithm for solving the distribution planning problem.

6.4.4 Efficient Algorithm: CDRR

We propose an efficient heuristic algorithm, called Contact-Driven Round Robin (CDRR), for larger distribution planning problems. CDRR is based on *three major intuitions*:

1. Deliver higher user experience improvement using less resources (energy, disk, and network budgets).

2. Send multimedia content to mobile users who have more chances to relay the content to others.

3. Download multimedia content from mobile users who have fewer chances to send content to others.

In particular, we first define the resource efficiency as $\nu_{u,nL+l} = \psi_{u,n} \times \rho_{nL+l}/b_{nL+l}$. For each user u, we sort all units that exist on any contact user $p_{u,c}$ in the descending order on resource efficiency. To avoid allocating too much resource to a single user, users take turns to choose units from the sorted lists. In addition, we calculate the number of contacts C_u and the contact duration $\kappa_{u,c}$, and we define the *contribution potential* of mobile user u as $\tau_u = C_u \times \sum_{c=1}^{C_u} \kappa_{u,c}$. In each round, user u chooses ι_u units. We let $\iota_u = \lceil \frac{\tau_u}{\hat{\tau}_u} \bar{C} \rceil$, where $\hat{\tau}_u$ is the highest contribution potential and $\bar{C} = \frac{1}{U} \sum_{u=1}^{U} C_u$. In this way, users with higher contribution potential choose more units in each round. Then, for each mobile user, we know the units to be downloaded. Last, we determine which contact to download each unit. For each unit, user u selects the contact user $p_{u,c}$ with the smallest contribution potential $\tau_{p_{u,c}}$, as long as the resources of users u and $p_{u,c}$ are enough for transferring the unit. Receiving the unit from this contact user reduces negative impacts on other users.

Fig. 6.5 gives the pseudocode of the CDRR algorithm. Lines 1–4 implement the first intuition, and save all the units that may be downloaded by mobile user u ($1 \leq u \leq U$) in a list $\mathbf{Z}_u$, which is sorted on resource efficiency. Lines 6–9 realize the second intuition, where users choose units in a round robin fashion. Each user u gets to select up to ι_u units in each round. Lines 10–19 implement the third intuition to select the contact users to download individual units. Lines 20–21 check if there are residue resources, and terminate once resources are saturated.

Lemma 12 (Time Complexity). *The CDRR algorithm terminates in polynomial time.*

Proof. We first define $\mathbf{H}$ as the index set of unfinished users. Next, we create the sorted units list $\mathbf{Z}_u$ in lines 2–4, which has a complexity of $O(UCNL \log(CNL))$ since the maximal number of units of any contact user is NL. The for-loops starting from lines 6 and 7 both go through user $u \in \mathbf{H}$ until $\mathbf{H}$ is empty. In each iteration, we take turns to check the units in $\mathbf{Z}_u$. We also make sure that at least one unit is removed and the user u will be removed from $\mathbf{H}$ once $\mathbf{Z}_u$ is empty. Since the maximal number of units in $\mathbf{Z}_u$ is CNL, the complexity of the loop is $O(UCNL)$. Hence, $O(UCNL \log(CNL))$ dominates and thus is the complexity of our CDRR algorithm. □

Table 6.3: Sample Running Time and Total User Experience, GeoLife

No. Content	**Running Time (sec)**					**No. Content**	**Total User Experience**		
	DP		**CDRR**				**DP**	**CDRR**	**Perf. Gap**
	Mean	Max	Mean	Max					
1	0.08	0.08	0.05	0.05		**1**	0.43	0.42	3%
2	0.11	0.11	0.06	0.07		**2**	0.36	0.34	6%
3	0.19	0.19	0.63	0.66		**3**	0.33	0.31	6%
4	0.6	0.61	0.07	0.07		**4**	0.31	0.29	6%
5	1.8	1.8	0.07	0.08		**5**	0.27	0.25	7%
6	110.2	111.2	0.08	0.08		**6**	0.29	0.27	7%
7	269.4	457.2	0.08	0.09		**7**	0.29	0.27	7%

Input: User profiles, such as contact duration $\kappa_{u,c}$, multimedia content information, such as size of a unit b_i, and resource budgets, such as the disk budget d_u

Output: The distribution plan $x_{u,n,l,c}$

```
1: let $\mathbf{H} = \{1, 2, 3, \ldots, U\}$ // index set of unfinished users
2: for $u = 1$ to $U$ do
3:     store units existing on any $p_{u,c}$ in a list $\mathbf{Z}_u$
4:     sort $\mathbf{Z}_u$ by resource efficiency in the dsc. order
5: let $B_{u,c} = \gamma_{u,c}\kappa_{u,c}$ $\forall 1 \le u \le U, 1 \le c \le C_u$ // network budget
6: while $\mathbf{H}$ is not empty do
7:     for $u \in \mathbf{H}$ do // iterate in round robin fashion
8:         let $\mathbf{R} = \iota_u$ units removed from the head of $\mathbf{Z}_u$
9:         for $r \in \mathbf{R}$ do
10:            for $k = 1$ to $C_u$ do
11:                if $d_u \ge b_r$, $q_u \ge \check{e}_{u,k}b_r$, $q_{p_{u,k}} \ge \hat{e}_{u,k}b_r$, and $\gamma_{u,k}\kappa_{u,k} \ge b_r$ then
12:                    if $\tau_{p_{u,k}} < \tau_{p_{u,c}}$ then
13:                        let $c = k$ // least contribution potential
14:            if $c$ exists and Eq. (6.1e) is satisfied then
15:                let $d_u = d_u - b_r$ // get $r$ from $c$
16:                let $q_u = q_u - \check{e}_{u,c}b_r$
17:                let $q_{p_{u,c}} = q_{p_{u,c}} - \hat{e}_{u,c}b_r$
18:                let $B_{u,c} = B_{u,c} - b_r$
19:                let $x_{u,\lfloor r/L \rfloor, r \bmod L, c} = 1$
20:        if $d_u \le 0$ or $q_u \le 0$ or $\mathbf{Z}_u$ is empty then
21:            remove $u$ from $\mathbf{H}$
```

Figure 6.5: The pseudocode of our CDRR algorithm for solving the distribution planning problem.

6.4.5 Near-Optimality of the Proposed Algorithm

We perform numerical analysis to quantify the performance of CDRR. We use our DP algorithm to optimally solve the formulation in Eq. (6.1), and use it as a benchmark. We adopt real datasets: GeoLife [193], San Francisco [150], and SIGCOMM [149] trajectory traces. The details of the datasets are given in Sec. 6.5. The datasets give us number of users and contacts per user. We then vary the number of multimedia content, and solve the distribution planning problem using the CDRR and DP algorithms. We repeat each experiment 5 times and report their running time and user experience.

Table 6.3 gives the sample running time and user experience from the GeoLife trace. We draw three observations from the tables. First, the proposed CDRR algorithm runs in real time, and scales to more multimedia content. Second, the DP algorithm leads to prohibitively long running time with large number of multimedia content. Third, we observe that CDRR achieves at least 93% of the user experience compared to DP. In summary, DP can optimally solve our problem under small problem size, but it is not feasible for large problems because of the long running time. In contrast, CDRR can solve larger problem with near-optimal user experience in short time. We note that we only report the *expected* user experience in this subsection, and more detailed simulation results are provided in Sec. 6.5. In that section, less-than-perfect optimality of CDRR is further compensated by the practical heuristics presented in Sec. 6.4.6.

6.4.6 Practical Considerations

Our system implementation contains the following practical optimizations.

Determining the downloading order. For each contact, a mobile user downloads units based on the plan. After the units planned for a contact are all finished, the units planned for other contacts are downloaded. Once all units on the distribution plan are downloaded (or the plan is empty, which means the mobile user has not received his/her plan), the mobile device shows available units to the user, and allows him/her to select the contents to request. When requesting multiple units, the order is crucial: it is preferred to devote resources to those units that result in higher user experience improvement normalized to unit size. Therefore, in each contact, each mobile device computes ρ_i/b_i of the next outstanding layer of each content. The mobile device requests unit i^* with the highest ρ_{i^*}/b_{i^*}. This is repeated until the contact is over or resources (energy and disk) are used up.

Segmenting video layers. Because multimedia content could be quite large, we define a maximal segment size F as a system parameter. For unit with size b larger than F, we divide it into $\lceil b/F \rceil$ segments, where the first $\lceil b/F \rceil - 1$ segments are in the size of F. Doing segmentation is to avoid unnecessary retransmission after interrupted transfers. The user experience improvement of a unit is equally split among all segments in that unit; more comprehensive approaches are also possible. Segmentation is done after the distribution plans are computed because incorporating the concept of segments in the distribution planning problem increases the problem size, which leads to high computational overhead.

6.5 Trace-Driven Simulations

In this section we use real datasets to analyze the performance of the distribution planning algorithm in a detailed simulator and show that it outperforms other algorithms by wide margins. various metrics including: (i) user experience, (ii) watched user experience, (iii) missed units, (iv) disk efficiency, (v) energy efficiency, and (vi) watched units.

6.5.1 Datasets

We employ three datasets: (i) user contacts, (ii) multimedia content, and (iii) user interests to drive our simulator. In order to evaluate our distributed system in different environments, we adopt three user contact datasets: GeoLife [193], San Francisco [150], and SIGCOMM [149]. In GeoLife dataset, there are multiple transportation modes, including walk, bike, bus, taxi, train, and subway. In San Francisco dataset, the transportation mode is taxi, and the participants of SIGCOMM dataset walk in the conference. The 4-year GeoLife and 30-day San Francisco datasets contain the GPS trajectories of 178 and 500 participants, respectively. Using the locations of the participants, we estimate the contact duration using a WiFi range of 55 m, measured under the setup proposed in Wang et al. [175] with HTC Desire 620 smartphone. We use the average throughput measured by Friedman et al. [85] for each contact. The SIGCOMM dataset contains 76 mobile users' BlueTooth contacts for 3 days. The characteristics of these three datasets are diverse. Therefore, the evaluation results using these three datasets will shed some lights on the performance of our solution in *diverse* environments, including challenged networks.

For the multimedia content dataset, we collect 300 news reports from NBC in 2015, and divide each news report into five layers: text, audio, low-, medium-, and high-resolution videos. The size of each layer is calculated. The shortest and longest news reports last for 12 and 287 secs, respectively. We adopt Topia Term Extractor [46] to extract keywords from the articles. The resulting keywords are used by the Content Matcher. Last, we derive the user interests by leveraging the user queries in the LETOR [30] dataset. In particular, we randomly pick a user query, and take the keywords in it to mimic the user's interests. The keywords in LETOR dataset are different from our NBC dataset, and we map the keywords using their orders of appearance.

6.5.2 Simulator Implementation

We implement a detailed simulator using a mixture of Python, Java, and Matlab. The distribution planning algorithm is executed once a day in our simulations. Once the distribution plans are computed, we carry out the simulations following the ground truth given in the datasets.

State-of-the-art algorithms. For comparisons, we implement two algorithms: (i) Epidemic that transmits all the units when a contact occurs [169] and (ii) CSI that sends the units of interested multimedia content to mobile users based on mobile similarity [108].

Table 6.4: Statistics of User Contact Datasets

	Contacts Per Day		**Contact Duration (sec)**	
	Mean	**Std**	**Mean**	**Std**
GeoLife	2.7	7.1	91	399
SIGCOMM	19	29	526	2606
San Francisco	258	209	25	50

Performance metrics. We consider the following performance metrics, and report the average performance with 95% confidence intervals whenever applicable.

- **User experience**: The average user experience (between 0 and 100%) of all the user demanded news reports. We also report **watched user experience** that only considers watched multimedia content.
- **Missed units**: The number of unavailable news units among all the user demanded ones.
- **Disk efficiency**: The ratio of per-user user experience and per-user disk consumption.
- **Energy efficiency**: The ratio of per-user user experience and per-user energy consumption.
- **Watched units**: The number of watched multimedia content units.

We note that performance metrics used for live video streaming, such as initial buffering time, number of rebuffering instances, and number of dropped frames, are not directly applicable to our multimedia content delivery application. This is because we only shows *fully-downloaded* content to users. Therefore, none of the aforementioned, and unfortunate, situations that are common to live video streaming ever occur in our platform.

Content Matcher and Contact Predictor. We also implement several machine learning algorithms in Content Matcher and Contact Predictor. The algorithms use historical data up to the past a days to predict contacts and user interests. The Contact Predictor keeps track of the historical contacts, and predicts the future contacts based on frequencies. The Content Matcher first classifies the news reports into several categories and calculates the viewing probability using a Bayesian approach inspired by Google news recommendation [129]. We utilize the BBC dataset [92], which classifies 2225 news reports in 5 categories, to train a classification model following a frequency-based approach, which uses numbers of keyword occurrences in each category for classification. We note that these machine learning algorithms are not developed by us, nor are the most advanced ones. We adopt them to be conservative: our platform will achieve even better performance with updated machine learning algorithms.

Simulation scenarios. We run 3-day simulations using GeoLife, San Francisco, and SIGCOMM datasets. The GeoLife dataset is very sparse as only 3.33% of user-day GPS trajectories are non-empty and the dataset spans over the greater Beijing area. Therefore, we focus on the 88 km^2 downtown area, and create a 3-day trace by choosing the top 30 active days of each mobile user. We assume that only 10% of the residents participated in GeoLife data collection, so that we aggregate 30 days trace to 3 days. We remove the mobile users who never get into the downtown area, which yields a trace with 870 mobile users. 80 fog devices are randomly deployed in the crowded locations. For San Francisco and SIGCOMM, we promote 50 and 10 mobile users with the most contacts to be the fog devices, respectively. Table 6.4 shows some statistics of the generalized datasets. We observe that GeoLife dataset has the lowest connectivity, while SIGCOMM and San Francisco have 7 and 96 times more contacts compared to GeoLife. The contact duration of San Francisco is lower than other datasets because the speed of taxi is much higher than other transportation modes. SIGCOMM has the highest contact duration because the attendees in SIGCOMM conference walk and discuss with each other in a small area. These three datasets with different characteristics help us to evaluate our distributed system under diverse environments.

Parameters. We vary the number of random NBC daily news reports within $\{10, 25, 50, 100, 200\}$. We also assume that $\frac{1}{3}$ energy is used by communications, and $\frac{1}{3}$ of communication energy is used by our app. We then consider the disk budget in $\{15, 30, 60, 125, 250, 500\}$ MB, and the energy budget in $\{100, 200, 400, 800, 1500\}$ J. By default, we pick 100 news reports everyday and set disk (energy) budget to be 125 MB (400 J). Moreover, the per-byte WiFi energy consumption is 9×10^{-7} J [85], and the maximal segment size is 5 MB. For prediction, we let $a = 3$ (days) for predicting the viewing probability and contacts. Last, we set the user experience following Table 6.1.

6.5.3 Results

The performance of our CDRR algorithm is near optimal under unlimited resources. Fig. 6.6 shows the service quality and resource usage under unlimited resources. We use Epidemic algorithm as a benchmark which gives optimal results in terms of user experience and delivery delay with unlimited resources. To achieve the optimality, Epidemic uses excessive energy and disk to flood news reports. Figs. 6.6(a) and 6.6(b) show that our CDRR algorithm achieves near optimal ($< 3\%$ gap) user experience and low delivery delay ($< 6\%$ gap), compared to Epidemic algorithms. Regarding system overhead reported in Figs. 6.6(c) and 6.6(d), CDRR saves up to 8% energy consumption and 8% used disk space compared to Epidemic algorithm. CSI saves about 67% energy and disk consumption on average, but suffers from 10% longer delivery delay and 20% lower user experience compared to Epidemic. In summary, while CDRR is not designed for environments with unlimited resources, it performs almost optimally in terms of user experience,

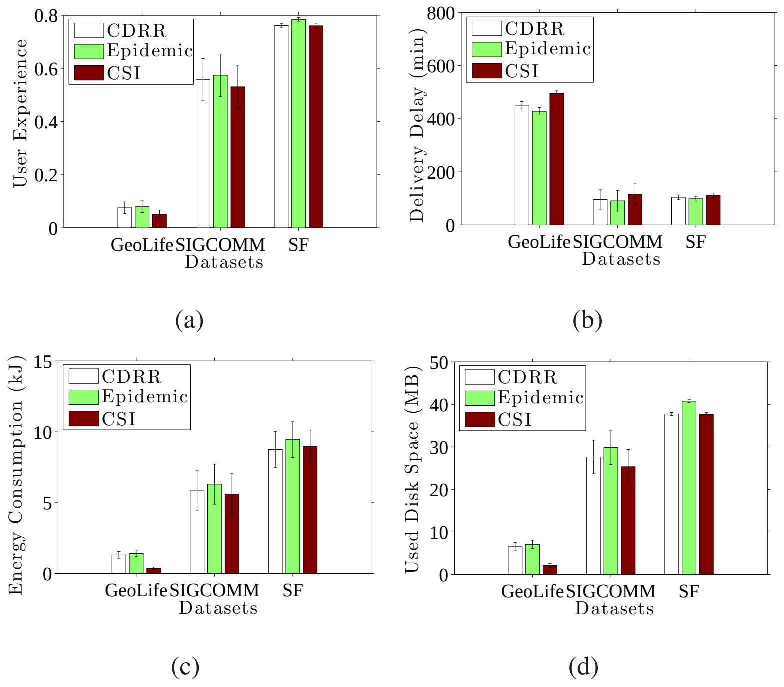


Figure 6.6: CDRR with unlimited resources is near optimal, average: (a) user experience, (b) delivery delay, (c) energy consumption, and (d) used disk space over 3 days.

yet achieves short delivery delay. CSI has lower resource usage, but suffers from higher delivery delay and lower user experience. In real life, the resources are *always* limited, and thus we take limited energy, disk, and network budgets into considerations in the rest of this chapter.

The proposed CDRR algorithm improves the service quality. Fig. 6.7 reports the service quality of individual mobile users in user experience, watched user experience, and missed units among three datasets over 3 days. Figs. 6.7(a) and 6.7(b) show that our algorithm outperforms all other algorithms in terms of user experience and watched user experience. The gap of Epidemic and CSI to our algorithm is at least 20% in terms of user experience and watched user experience. This is because Epidemic and CSI do not take the characteristics of multi-layer news reports into considerations. Fig. 6.7(c) gives the number of missed units, which shows that Epidemic and CSI miss at least 10% and 11% more demanded units than our CDRR algorithm, respectively. Next, we plot sample empirical CDF (Cumulative Distribution Function) curves from SIGCOMM in Figs. 6.7(d), 6.7(d), and 6.7(f), which clearly show that our algorithm results in much higher user experience and fewer missed units. In summary, Fig. 6.7 demonstrates that our CDRR algorithm significantly improves the service quality.

The proposed CDRR algorithm is resource efficient. We report the resource efficiency of our proposed CDRR in Fig. 6.8. Figs. 6.8(a) and 6.8(b) present the energy efficiency and disk efficiency of our CDRR, which are the ratios between user experience

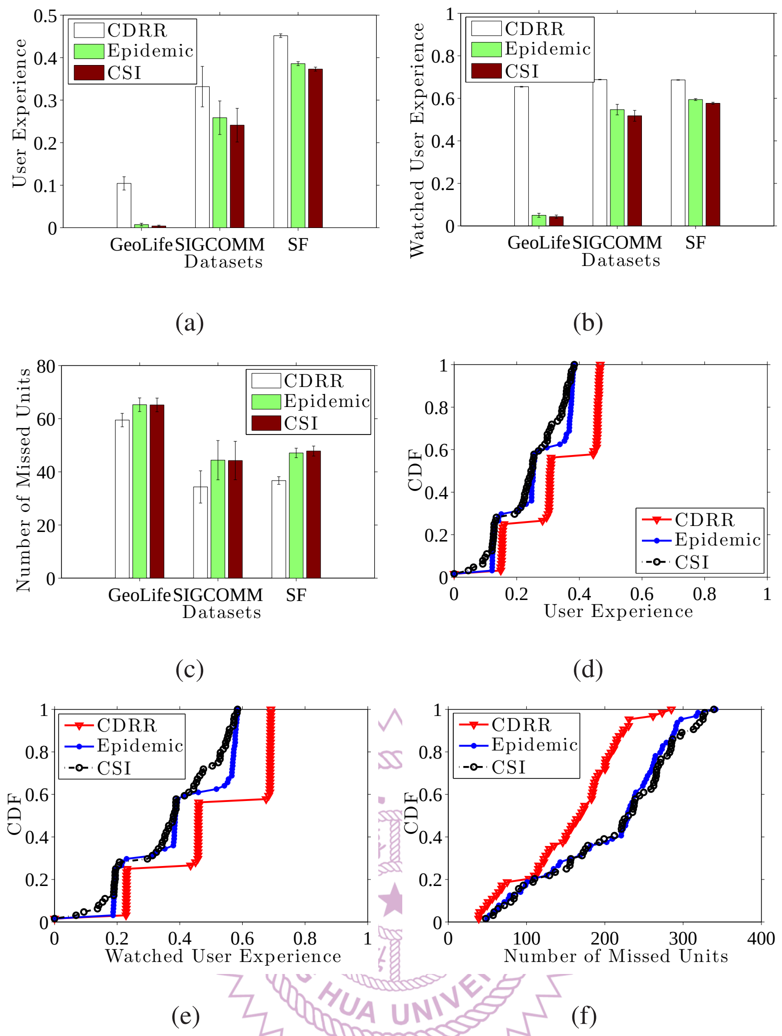


Figure 6.7: Service quality improvement of our CDRR algorithm: (a) user experience, (b) watched user experience, (c) missed units, (d) user experience CDF, (e) watched user experience CDF, and (f) missed units CDF over 3 days. Sample CDFs in (d), (e), and (f) are from SIGCOMM dataset.

and energy/disk consumption. The figures show that our CDRR algorithm is more energy-efficient than all other algorithms. In particular, at least 33% higher energy efficiency is observed compared to Epidemic and CSI algorithms. Moreover, the disk efficiency of our CDRR algorithm outperforms others by at least 39%. Figs. 6.8(a) and 6.8(b) reveal that CDRR delivers high service quality in a resource-efficient manner. Next, we plot the watched units in Fig. 6.8(c), which reveals that Epidemic and CSI suffer from lower watched units: at least 70% and 76%, compared to our CDRR algorithm. This partly explains why the CDRR algorithm is resource-efficient: it downloads more *useful* units. In summary, Fig 6.8 shows that our CDRR algorithm is resource efficient.

Implications of different datasets. In Table 6.5, we report the average performance comparisons between our CDRR algorithm and the other two baseline algorithms across

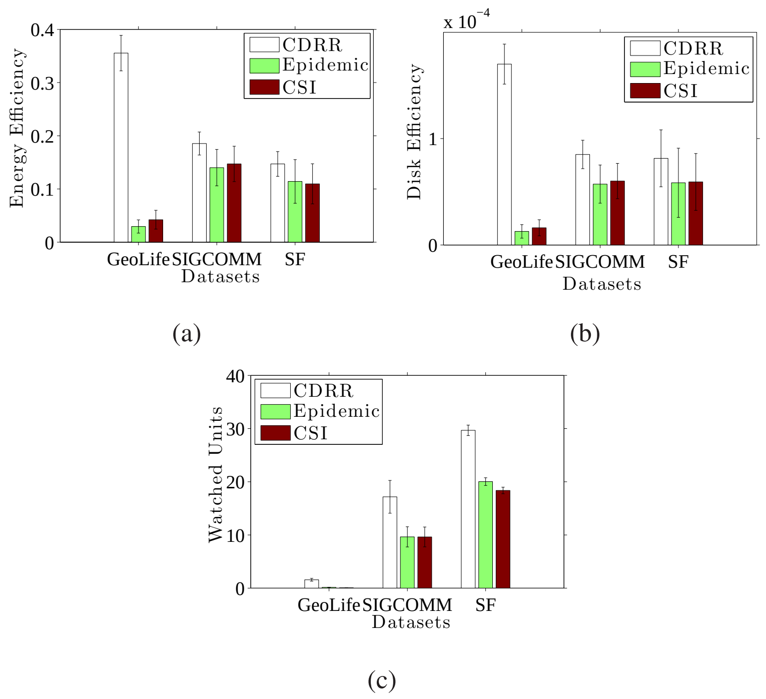


Figure 6.8: Resource efficiency of our CDRR algorithm: (a) energy efficiency, (b) disk efficiency, and (c) watched units over 3 days.

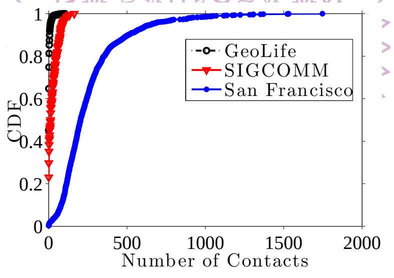
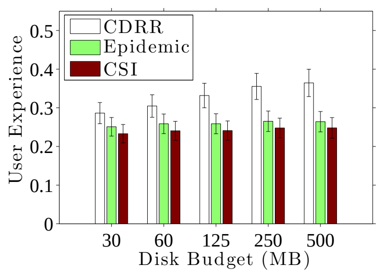


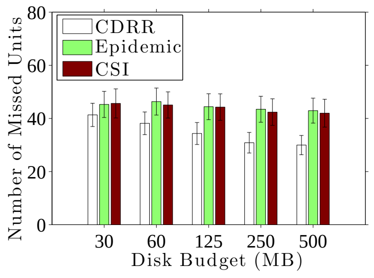
Figure 6.9: Number of per-user contacts in 3 datasets.

all considered simulations. This table shows that our CDRR algorithm significantly outperforms others in all the aspects using all three datasets. A closer look reveals that our CDRR algorithm has its limitations with San Francisco dataset. We only outperform others by up to 21% in terms of user experience. This can be explained by Fig. 6.9, which shows the mobile users in San Francisco dataset are better connected, compared to GeoLife and SIGCOMM datasets. Since the number of contacts is very high, *any* distribution plans will work reasonably well. However it is not an issues, because typical challenged networks, especially those in developing countries and rural areas are not well-connected.

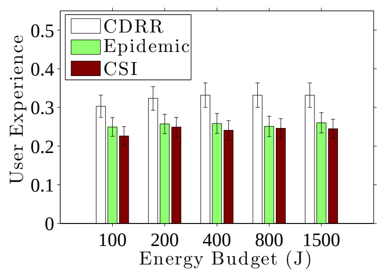
The proposed CDRR algorithm is scalable. Next, we vary the disk budget, energy budget, and number of daily news reports to compare the performance of our CDRR algorithm against the other algorithms with SIGCOMM dataset in Fig. 6.10. Fig. 6.10(a) presents the user experience under different disk budgets, which shows that higher disk



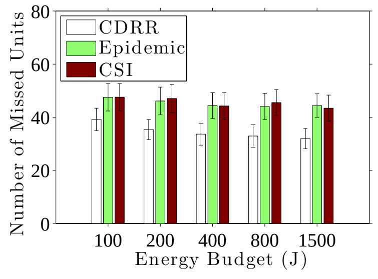
(a)



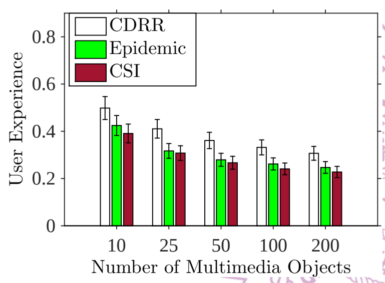
(b)



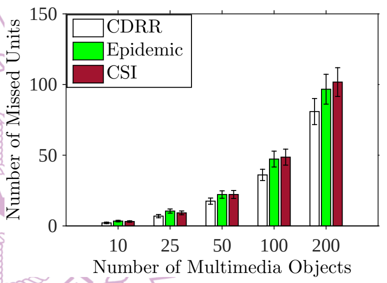
(c)



(d)



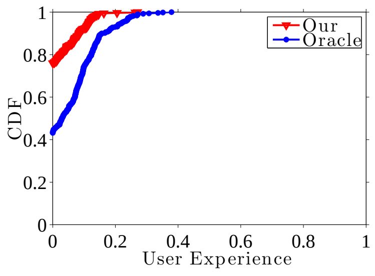
(e)



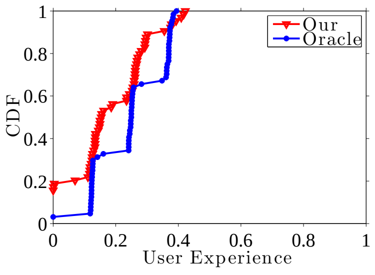
(f)

Figure 6.10: Scalability of our CDRR algorithm under different resource budgets and number of multimedia objects: (a) user experience, and (b) number of missed units with diverse disk budgets; (c) user experience, and (d) number of missed units with diverse energy budgets; (e) user experience, and (f) number of missed units with diverse number of news reports. Sample results from SIGCOMM dataset.

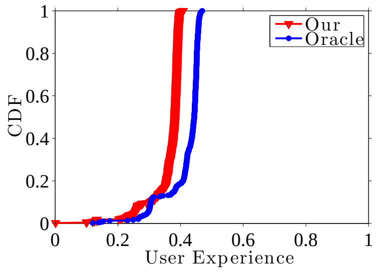
budgets lead to higher user experience with our CDRR algorithm, but it is not the case with Epidemic and CSI algorithms. This can be explained by Fig. 6.10(b), which shows that the CDRR algorithm utilizes higher disk budgets efficiently to reduce missed units. The other two algorithms, however, do not leverage the additional disk budgets. Compared with our CDRR algorithm, Epidemic and CSI miss 40% and 42% more units under 500 MB disk budget. Next, we plot the user experience under different energy budgets in Fig. 6.10(c). This figure shows that our CDRR algorithm capitalizes higher energy budget for better user experience, while CSI and Epidemic algorithms do not result in the same trend. This can be explained by Fig. 6.10(d), which reveals that the CDRR algorithm utilizes higher energy budgets efficiently to reduce the number of missed units. Compared



(a)



(b)



(c)

Figure 6.11: Effectiveness of our Content Matcher and Content Predictor: (a) GeoLife, (b) SIGCOMM, and (c) San Francisco datasets.

Table 6.5: Performance Comparisons of CDRR over Epidemic and CSI

Metric	GeoLife		SIGCOMM		SF	
	Epidemic	CSI	Epidemic	CSI	Epidemic	CSI
User Experience	14X	26X	1.28X	1.38X	1.2X	1.21X
Watched User Experience	13X	15X	1.26X	1.33X	1.56X	1.19X
Missed Units	1.1X	1.11X	1.24X	1.26X	1.21X	1.25X
Watched Units	11X	26X	1.78X	1.78X	1.7X	1.76X
Energy Efficiency	13X	12X	1.33X	1.36X	1.33X	1.33X
Disk Efficiency	14X	15X	1.5X	1.52X	1.39X	1.41X

with our CDRR algorithm, Epidemic and CSI algorithms miss 24% and 26% more units under 1500 J energy budget. Finally, we plot the user experience under different numbers of news reports in Fig. 6.10(e). This figure shows that more news reports lead to lower user experience. This can be explained by Fig. 6.10(f), which reveals that more news reports will degrade the user experience because of more missed units. However, our CDRR algorithm outperforms Epidemic and CSI by at least 17% and 27% under any number of news reports, respectively. In Fig. 6.10(f), compared to our CDRR algorithm, Epidemic and CSI algorithm miss up to 20% and 26% more units. In summary, Fig. 6.10 shows that our CDRR algorithm scales better with more resources and news reports, compared to other algorithms.

Effectiveness of our Content Matcher and Content Predictor. We quantify the effectiveness of the machine-learning algorithms implemented in the Content Matcher and Content Predictor as follows. We augment our simulator to prohibit mobile users

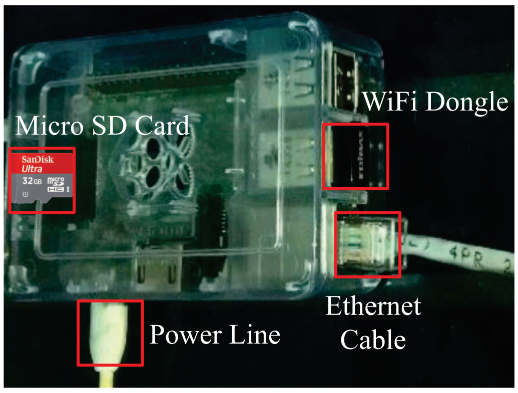


Figure 6.12: The fog device built on Raspberry PI.

from requesting for any content that are not on their distribution plans, so as to focus on the impact of the machine learning algorithms. For comparisons, we assume perfect predictions using the user contact datasets, and refer to it as Oracle in the figures. We emphasize that Oracle is an impractical upper bound for benchmarking purpose only. We report the empirical CDF curves of user experience from the 3 datasets in Fig. 6.11. This figure reveals that our machine learning algorithms achieve similar performance with Oracle with SIGCOMM and San Francisco datasets: on average, only 26% and 17% gaps are observed, respectively. For the GeoLife dataset, our algorithms suffer from a larger gap of 69%, which can be attributed to more challenging scenarios as indicated by the inferior connectivity reported in Fig. 6.9. We note that, in such challenging scenarios, our CDRR algorithm significantly outperforms Epidemic and CSI as summarized in Table 6.5.

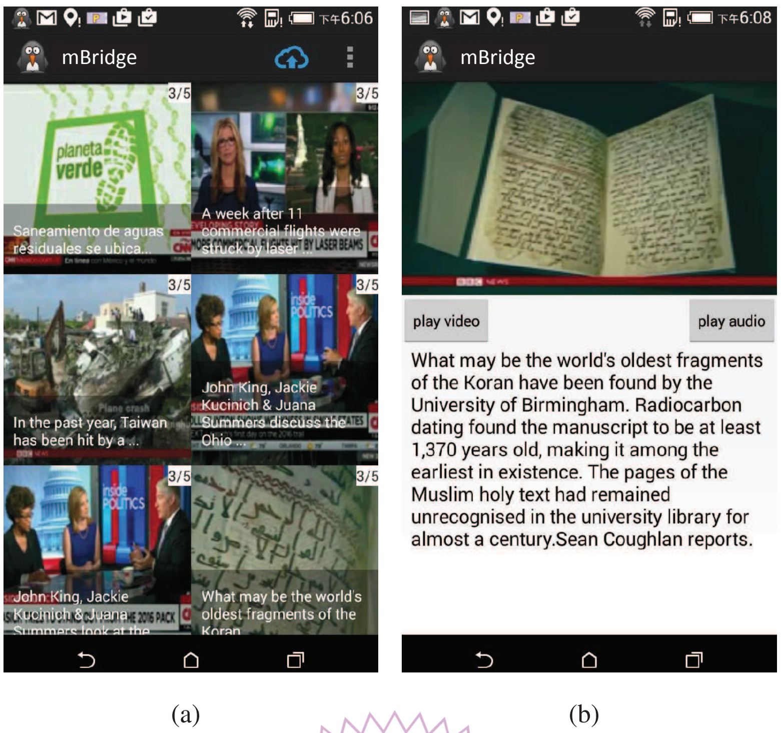
6.6 Real Implementation

This section presents a complete prototype system[1] that distributes multimedia news reports to multiple users. We also compare our algorithm against others under real-life settings.

6.6.1 Implementation on Linux and Android

We have implemented a complete testbed of the proposed content delivery applications running on cloud-to-things continuum platforms using Linux machines and Android mobile devices. The fog controller is built on a Linux workstation, and we realize our CDRR algorithm on it. fog devices can be built on any Linux embedded devices and general-purpose computers. We adopt Raspberry PI for fog devices, as shown in Fig. 6.12. Using

[1]Parts of the prototype system were presented in Hong et al. [106].



Saneamiento de aguas residuales se ubica...

A week after 11 commercial flights were struck by laser

In the past year, Taiwan has been hit by a ...

John King, Jackie Kucinich & Juana Summers discuss the Ohio ...

John King, Jackie Kucinich & Juana Summers look at the

What may be the world's oldest fragments of the Koran

What may be the world's oldest fragments of the Koran have been found by the University of Birmingham. Radiocarbon dating found the manuscript to be at least 1,370 years old, making it among the earliest in existence. The pages of the Muslim holy text had remained unrecognised in the university library for almost a century.Sean Coughlan reports.

Figure 6.13: Mobile client: (a) a list of news reports and (b) a downloaded news report.

Raspberry PI as the fog device results in the following benefits: (i) smaller form factor, (ii) more cost effective, and (iii) easier deployments. The size of a Raspberry PI is only 100 cm^2. Each Raspberry PI, including a WiFi dongle, a case, a network cable, a memory card, and a power line, only costs about 55 US dollars at the time of writing. Compared to PCs, Raspberry PIs are easier to be transported to different places and countries.

The fog devices are connected to the fog controller via wired networks. We configure the fog devices to be WiFi access points, and program them to bridge the fog controller and mobile devices. We implement an Android app, which follows the distribution plan

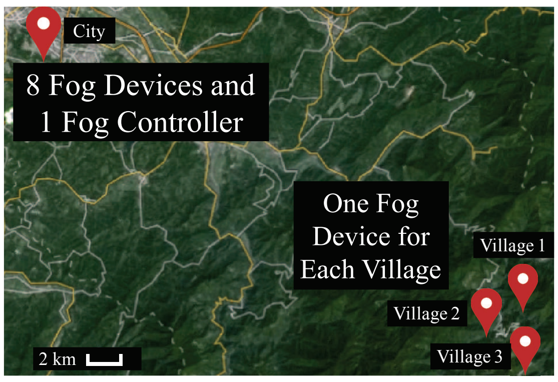


Figure 6.14: The locations of the deployed fog devices.

to download multimedia content whenever it runs into fog devices. It also records timestamped events and sends them via fog devices to the fog controller as profiles. Fig. 6.13 shows sample screenshots of our app. The fog controller analyzes the profiles for various inputs, such as contacts, and computes distribution plans at 5 a.m. as a `cron` job. The computed plans are sent to mobile users via fog devices. To preserve user privacy, we anonymize the collected profiles, and allow mobile users to opt out from the data collection any time. We also allow users to configure several settings, such as the disk and energy budgets.

6.6.2 Experimental Setup

We set up a fog controller and eleven fog devices at four different locations: three different rural villages and a city (on our university campus). Fig. 6.14 shows a map with the locations of fog devices. There are 15 users[2], including university students, university employees, and farmers who use our Android app. In particular, they install our app from Google Play. The app comes with the default disk budget of 200 MB and the default energy budget of 80% battery capacity.

Every midnight, the fog controller automatically downloads the latest news reports, including text, audio, and videos from CNN, BBC, and Apple Daily. It transcodes the news videos to three resolutions: 240p, 360p, and 480p using `FFmpeg`. For each user, we use his/her profile collected in the past 7 days to predict his/her behavior. We then use the predicted behaviors to compute the distribution plans and send the plans to mobile users when they have the first contact with a fog device. Our mobile app downloads news reports following the distribution plan, and the user may watch news reports anytime.

6.6.3 Experimental Scenarios

In the first two weeks of our experiment, the fog controller downloads 46 news reports on average every day, which is equivalent to 900+ MB total size. On average, each user spends 2.8 hours within the coverage of fog devices everyday. The average number of contacts of each user in each day is 7.8, the average contact duration is 101 seconds, and each user moves 3.6 km on average everyday. The distance between the university to the villages is 38.8 km and there are no users commuting between the campus and the villages. The size of the university and the villages are about 1.2 km^2 and 4 km^2, respectively. There are 67% and 87% of users using the default disk and energy budgets, respectively. With the default disk budget, each user can download news reports for up to about three layers. We note that when the app runs out of the disk budget, it pops up

[2]In total, there are 31 users, but 16 of them decide to use our app without uploading their profiles.

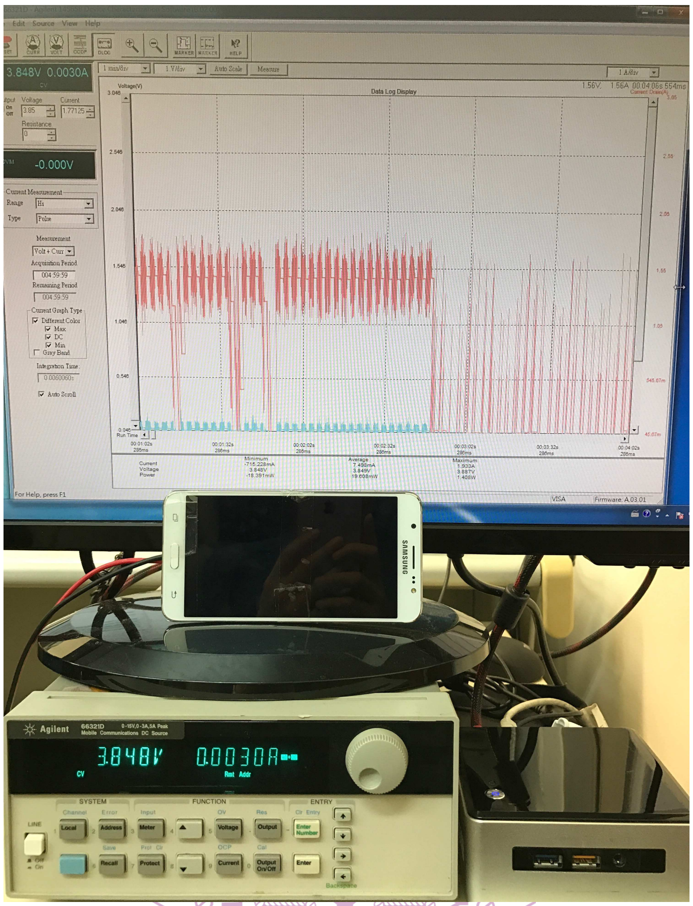


Figure 6.15: Our emulation testbed for measuring energy consumption.

a dialog reminding the user about the possibility of increasing the disk budget for better video quality. We observe that more than 2/3 of users stick with the default disk budget, which shows that many users are satisfied with the low-resolution videos. This is inline with our observation of user experience improvements decreasing along with the number of layers in Sec. 6.3.

6.6.4 Emulation Setup

One limitation of our deployed testbed is that the application specific optimizer can only execute a *single* distribution planning algorithm at a time and we only implement our CDRR algorithm in the testbed. To compare our algorithm against the baseline algorithms, we use the collected profiles to drive our simulator multiple times, and save the *distribution plans* generated by different algorithms. We then set up an *emulation testbed*, which consists of a Linux FTP server, a smartphone, and a power meter, so as to measure the *actual* energy consumption resulted by different algorithms. Fig. 6.15 illustrates the emulation testbed. We configure the FTP server to limit the bandwidth following the

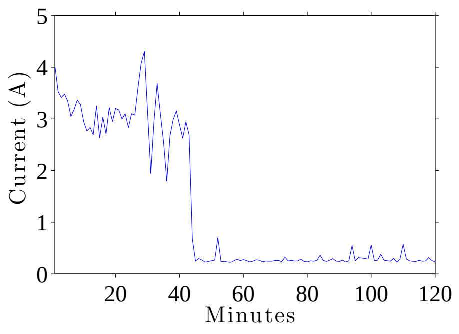


Figure 6.16: Sample measured current levels from user 1.

traces in Friedman et al. [85]. We then augment our mobile app to download the multimedia news reports following the distribution plans. We run the mobile app on a Samsung Galaxy J7 smartphone connected to an Agilent 66321D power meter. The power meter is configured to serve as a power source at 3.85 V, and connected to a PC via an USB cable. We record the voltage and current at 200 Hz. The records are then used to calculate the energy consumption.

In addition, we use the distribution plans from the diverse algorithms to conduct a user study for real user experience. Questionnaires, similar to the ones used in Sec. 6.3, are prepared for the user study. Running 2-week emulations with 3 different algorithms for all 15 users is time consuming without revealing too many insights. For example, for a less active user, he/she doesn't have too many contacts, and none of the algorithms work for him/her. Therefore, we focus on top three active users and their top seven active days for a 1-week emulation, which is equivalent to 21 days in totals. We repeat each day of emulation with three algorithms, resulting in 63 1-day news reports. We recruit 63 participants, among them 70% are males and the average age is 24 years old. To avoid overloading the participants, each participant watches 1-day news report and fills in the questionnaire using the Web interface shown in Fig. 6.3. Each 1-day news report takes a participant about 10-40 mins. The participants then fill the questionnaire to give MOS scores.

6.6.5 Emulation Results

Our implementation is energy efficient. Fig. 6.16 shows 2-hour sample measurements of the current from user 1. In this figure, the user's smartphone downloads news reports for 43 minutes before being idling. We compute the communication energy consumption by deducting the idling energy consumption from the total energy consumption. We report the mean and standard deviation of daily energy consumption resulted by different

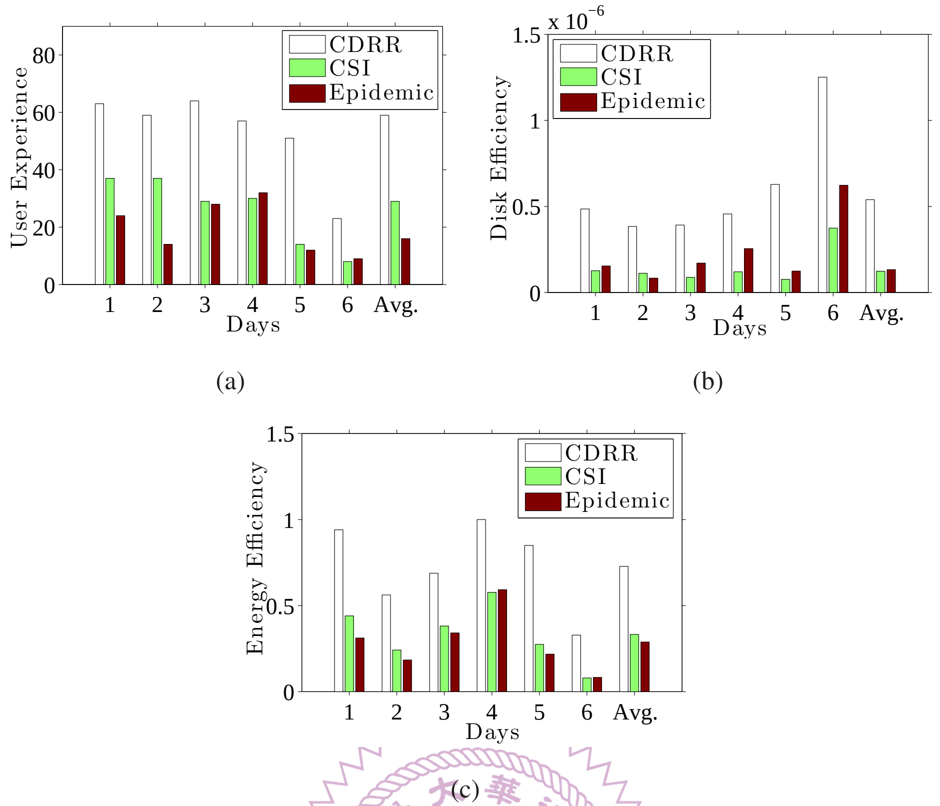


Figure 6.17: Better service quality and resource efficiency of our CDRR algorithm: (a) quality improvement, (b) disk efficiency, and (c) energy efficiency over a 1-week emulation. Sample results from user 1 are shown.

algorithms in Table 6.6. This table shows that completing a 1-day distribution plan only consumes up to 153 J, which is 0.3% of the battery capacity (3300 mAh, 3.85 V). We also report the energy consumption per MB in the same table. It shows that the gaps among different algorithms is at most 7%, which is insignificant.

Our CDRR algorithm leads to better service quality and resource efficiency. Although different distribution planning algorithms result in similar energy consumption, they may lead to diverse user experience. We plot the sample user experience from user 1 in Fig. 6.17(a), which shows that our CDRR algorithm outperforms CSI and Epidemic by 1.1 and 2.7 times on average in terms of user experience. Figs. 6.17(b) and 6.17(c) report the sample disk efficiency and energy efficiency from user 1. In terms of disk efficiency, our CDRR algorithm outperforms CSI and Epidemic by 3.4 and 4.1 times; in terms of energy efficiency, our CDRR algorithm outperforms them by 1.2 and 1.5 times. In Table 6.7, we give the average performance improvement of our CDRR algorithm over the other two baseline algorithms for individual users. Among the three users, our CDRR algorithms outperforms others by up to 206%, 472%, and 188% in terms of user experience, disk efficiency, and energy efficiency.

Table 6.6: Statistics of Daily Communication Energy Consumption (J) Over a 1-week Emulation

Total Energy Consumption						
	CDRR		**CSI**		**Epidemic**	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**
User 1	79.5	21.7	82.2	28.7	79.1	19.8
User 2	99.1	23.6	100.2	24.8	91.4	18.6
User 3	55.5	4.1	50.4	2.6	50.2	7.3
Per MB Energy Consumption						
	CDRR		**CSI**		**Epidemic**	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**
User 1	0.46	0.04	0.46	0.1	0.44	0.02
User 2	0.45	0.04	0.44	0.04	0.43	0.01
User 3	0.43	0.02	0.40	0.01	0.41	0.02

Table 6.7: Performance Improvements of CDRR over Epidemic and CSI

Metric	**User 1**		**User 2**		**User 3**	
	CSI	**Epidemic**	**CSI**	**Epidemic**	**CSI**	**Epidemic**
User Experience	128%	206%	89%	66%	143%	65%
Disk Efficiency	349%	43%	42%	143%	69%	472%
Energy Efficiency	148%	188%	129%	87%	177%	77%

6.7 Discussion

In this chapter, we studied the problem of distributing multimedia content over challenged networks to mobile devices. We running our content delivery application on our cloud-to-things continuum platform, which carefully plans the distribution of multimedia content to mobile users. The critical component is the distribution planning algorithm, which intelligently distributes multi-layer multimedia objects over challenged networks using opportunistic communications. Our CDRR achieves near-optimal results in terms of user experience, despite the complexity of the distribution planning problem (NP-Complete). We conducted extensive simulations using real datasets. The simulation results indicate that our proposed CDRR algorithm results in: (i) better user experience, which outperforms other algorithms by at least 20%, (ii) higher energy and disk efficiency, which outperforms other algorithms by at least 33% and 39%, respectively, (iii) fewer missed units which outperforms other algorithms by at least 10%, and (iv) more watched units, which are at least 70% more than other algorithms. Moreover, we implemented and deployed a prototype testbed in a university and three rural villages. Experiments reveal that our proposed algorithms outperform the baseline algorithms by up to 206%, 472% and 188% in terms of user experience, disk efficiency, and energy efficiency, respectively.

Chapter 7

Future Directions

A report from Gartner states that *"Currently, around 10% of enterprise-generated data is created and processed outside a traditional centralized data center or cloud."* By 2022, Gartner predicts this figure will reach 50% [49]. Moreover the market of fog computing is expected to increase 10 times from 2017 to 2022 [14]. These reports point out that performing operations on cloud-to-things continuum platforms is a clear trend. In the future, with the cloud-to-things continuum platforms, various resources from not only traditional cloud data centers, but also fog devices scattering across wide geographical locations can be easily accessed anytime and anywhere. When we are at home, we can ask our intelligent IoT devices to detect safety events, such as carbon monoxide spikes, and send notifications to hospitals to protect us. When we are driving, we can ask smart street lights to analyze road conditions and coordinate all the cars to avoid traffic jams. When we are playing complicated games over mobile phones, we can ask cellular base stations to render the complicated scenes to have better gaming quality and save mobile phones' energy. When we are walking alone on a street at midnight, we can ask a patrolling tiny drone to detect some dangerous events, such as brawling and gun shots. In this chapter, we list future directions to complete such a large-scale, comprehensive, and innovative cloud-to-things continuum platform.

7.1 Application Developer Support

In this thesis, we have introduced two actors (the provider and users) in the cloud-to-things continuum platforms. In order to build a comprehensive cloud-to-things platform, another actors, called application developers need to be added into an ecosystem as illustrated in Fig. 7.1. The application developers implement many applications and publish them through the provider to the users. However, in cloud-to-things continuum platform, the provider has to offer a large number of applications, which is challenging for a small

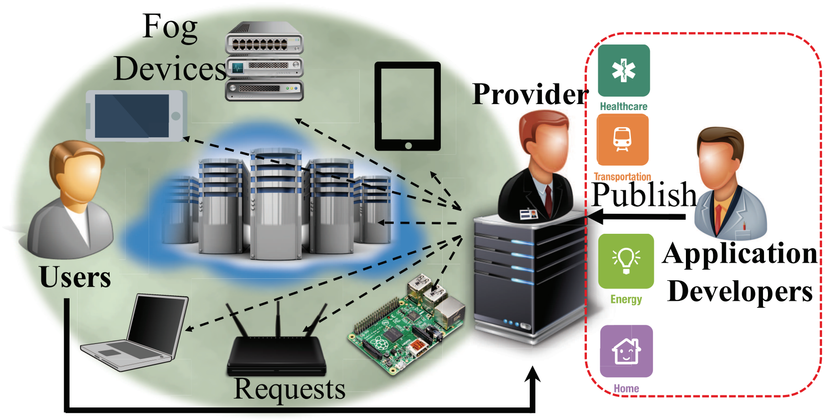


Figure 7.1: A cloud-to-things continuum ecosystem.

group of application developers. Hence, we need to give the application developers some supports to make their life easier when implementing the applications. Doing so, we can lower the learning curves to be an application developer and allow various people, such as students, engineers, and anyone who can follow the supporting tools to implement the applications. The resulting ecosystem will be similar to Google Play and Apple Store, where the application developers publish and sell their applications to the users.

Having the application support results in many benefits. First, because the number of application developers is increased and the developers have various backgrounds, many and various applications are provided to the users, and the users will automatically interact with the application developers to figure out killer cloud-to-things continuum applications. Second, it is also helpful for retaining the users because they have many options of applications. They can switch among the applications based on their requirements that are changed over time. Third, it also increases benefits earned by the providers and it further makes more providers and companies interesting in realizing the comprehensive cloud-to-things continuum platform.

One possible solution to achieve the application developer support is to design a *programming model*. In order to implement applications, which can run on such heterogeneous platform in a distributed way, the programming model has to support three features: (i) decomposition, (ii) communications, and (iii) performance optimization. With the decomposition feature, the provider can easily and dynamically split the applications into smaller operators. The splitted operators are distributedly deployed to multiple fog devices. When doing so, the communication channels among the operators need to be transparently set up. Finally, performance optimization strategies, such as stream processing [97] to optimize the applications running across multiple fog devices, are also required. The resulting programming model can be readily followed by application developers to implement tailored applications for cloud-to-things continuum platforms.

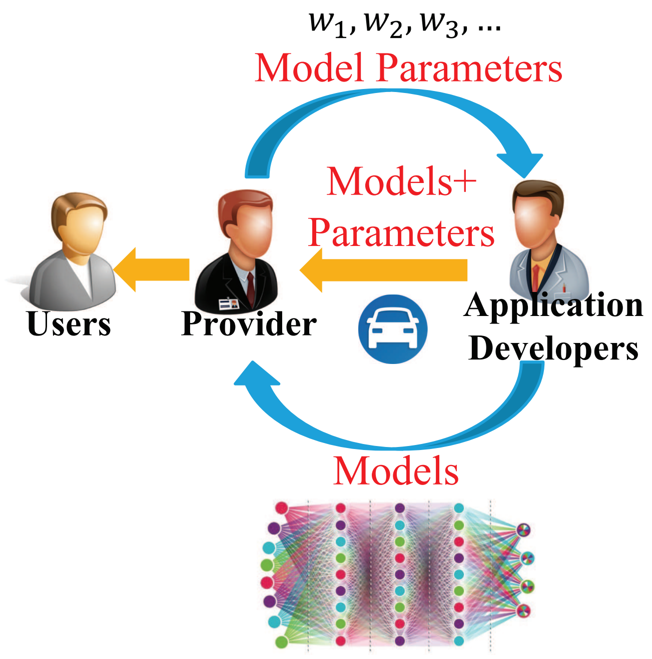


Figure 7.2: An illustration of Training-as-a-Service (TaaS).

7.2 Data Management and Privacy Protection

In the clout-to-things continuum platforms, many sensors, such as camera, temperature, and gas sensors generate tremendous amount of data. Various data are collected and used by different people, such as researchers, application developers, users, companies, and providers. Hence, we need a data management mechanism to help them readily access the data. Moreover, some of the data are privacy-sensitive, so we need to protect these data before accessing. For example, a smart street light may collect some images that contain plates and faces, which are large and privacy-sensitive. If a data management and privacy protection strategy is proposed, these people can efficiently access the protected large amount of data and start to use them.

A well-designed data management mechanisms results in many benefits. First, the data management mechanism can reduce redundant data, e.g., we may not need to collect readings from two temperature sensors close to each other (say one meter) at the same time. The readings may not have much difference to derive more information from the sensors. Second, the data management mechanism may reduce the cost and network traffic congestions while sending and storing the large amount of data to/on remote servers.

For example, the data can be prioritized and the data with the lowest priority can be sent at midnight to avoid network traffic congestions. About privacy protection strategy, it is required while using the privacy-sensitive data to avoid violating the law. Moreover, protecting the privacy helps to eliminate doubts of users while using the applications running on the cloud-to-things continuum platform.

Designing the data management and privacy protection strategies requires three important components: (i) Training as a Service (TaaS), (ii) data analyzing API, and (iii) sensor-rich indexing mechanism. The TaaS is designed for the application developers who need to access the privacy sensitive data. For example, when an application developer designs a car tracking application, he/she needs to analyze images containing plates and derive a model for the application. However, the provider cannot directly send the privacy-sensitive data to the application developer for training the model. Hence, as illustrated in Fig. 7.2, the provider provide a new service called TaaS. The application developers follows TaaS to send their car tracking models to the data owner (the provider) for training. The provider then sends back the training results (model parameters) to the application developers who publish their applications through the provider, so that the users can request for these applications. Furthermore, the provider also provides a data analyzing API. The API allows application developers to analyze the data without revealing the original privacy-sensitive data. It also helps the application developers understand the characteristics of the data for better application design. Moreover, when the accuracy of the model trained by provider is poor, the application developers can use the API for root-cause analysis. In order to support the data analyzing API, we require a sensor-rich indexing mechanism, which can efficiently access required data for analyzing. For example, in the cloud-to-things continuum platform, we have many sensors giving informations about the weather, such as wind speed, temperature, humidity, and so on. We can design a sensor-rich indexing mechanism to access these data with a single request. The resulting data management and privacy protection strategies can maximize the value of the various and large amount of data.

7.3 Design-Time Optimization

In this thesis, we have solved three *run-time* optimization problems, while the *design-time* optimization problem remains open. Run-time optimization algorithms are launched after the cloud-to-thing continuum platform has been built. The design-time optimization problems are solved while designing the platform. More specifically, we can propose design time optimization algorithms to make deployment decisions of sensors, network devices, and fog devices. Taking smart street light as an example, the street lights are fog devices, which have computing, networking, sensors, storage, and actuators. If we plan to deploy PM 2.5 sensors on the street lights to create detailed pollution maps, we have to decide where to install the sensors because it is too dense to install the PM 2.5 sensors on every single street light while it may be to sparse to install the sensors on every cross streets. Deploying on every single street lights would leads to high cost with marginal performance gain. In contrast to it, we may not satisfy requirements to plot a detailed pollution map.

Moreover, when we make plans of sensor and actuator deployments at design-time, we can further consider representative analytics. Take smart street lights as an example again, we may solve a camera planning problem at design-time to optimize the computer vision applications. More specifically, the camera planning algorithms decide where to install which types of camera with what orientations. Different kinds of camera have different resolutions, field-of-view, focus, and so on. Hence, the decisions made by design-time optimization algorithms affects features, such as angle, covered range, and resolution of captured images. These features further make huge influence of the analytics, that is, a well-planed sensor deployment decisions may increase accuracy of the analytics, reduce the computation time, and shorten the training time.

Because we have various sensors and analytics on our platform, we need to propose a *generalized* design-time optimization algorithm. The algorithm can be used on any or similar types of sensors and analytics. Otherwise, if we propose the algorithms for every sensors and analytics, it will take quite a long time and will be a challenging and tedious task. The possible starting point of the generalized design-time optimization algorithm is to figure out a generalized feature list. For example, the distance between sensors and its sensing targets is one of generalized features of most sensor types. When the distance increases the noise of sensing data is larger and the performance of corresponding analytics become worse. However, when the distance of the all the sensors are too short, the density of deployed sensors will be too large, which leads to very high cost. Listing these features is a starting point of designing the generalized design-time optimization algorithm. With it, we can further optimize the cloud-to-things continuum platforms and provide much better user experiences for the users.

7.4 Outdoor Testbed Deployment

Many studies have proved that fog computing reduces latency, network traffics, and energy consumptions [139]. However, evaluations of the studies are from simulations or lab-scale prototypes like what we have shown in Sec. [**?**]. In order to have a more accurate evaluations, and figure out limitations and other future directions, we need to have

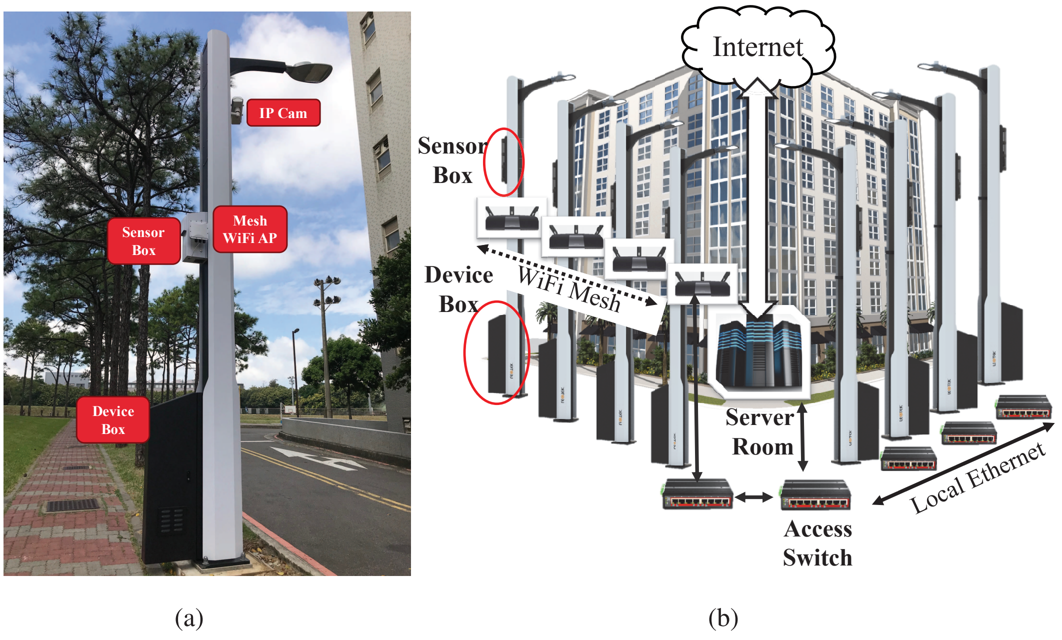


Figure 7.3: A SmartPole block: (a) devices and (b) network topology.

a real deployment of the cloud-to-things continuum platform. We have deployed an outdoor cloud-to-things continuum testbed, based on the cluster of eight street lamp poles. We refer to these street lamps as *smart poles* and the cluster as *block*. In our deployment, the block of smart poles are set up surrounding the computer science building at National Tsing Hua University (NTHU); in large deployments, a block of smart poles may be deployed along a street block. Each pole, as illustrated in Fig. 7.3(a), has two boxes: a sensor box and a device box. The sensor box contains a Raspberry Pi, which is equipped with multiple sensors, including temperature, PM 2.5, voltage, and current sensors. The device box contains an Ethernet switch and an Industrial PC (IPC). We note that all eight smart poles come with Raspberry Pi 3 and switches, but only two of them have IPCs serving as edge servers. Moreover, four smart poles have surveillance IP cameras to capture images and videos for video analysis. For communications, as shown in Fig. 7.3(b), all poles access to the Internet through an access switch. Four smart poles connect to the access switches through local Ethernet and the other four connect to it through WiFi mesh network.

Fig. 7.4 reports sample performance results from our outdoor smart pole testbed. The IPCs in the device boxes come with Intel 4-core i3 CPU and 8 GB RAM. One IPC accesses the Internet through Ethernet switches and another one through WiFi Mesh APs.

Requests are deployed almost instantly in our smart pole testbed. Fig. 7.4(a) gives the deployment time of diverse number of requests. In the experiments, we submit $\{10, 20, 30, 40, 50\}$ requests to measure the deployment time. The requests are object recognition analytics with random selected QoS values and the docker image of the object

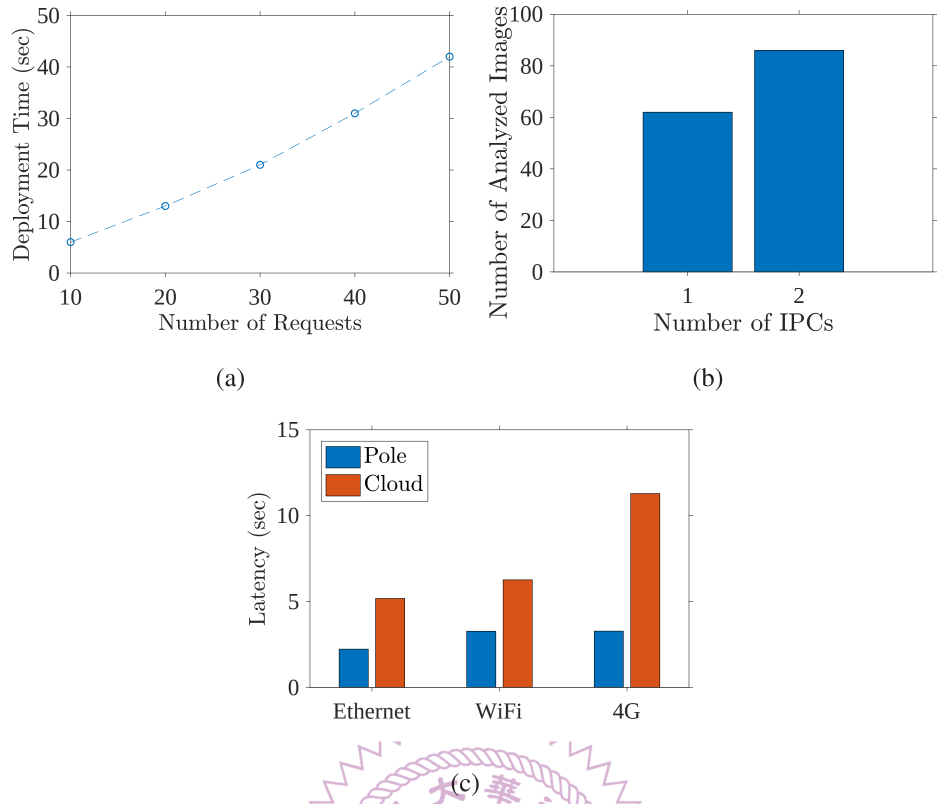


Figure 7.4: Measurement results of our outdoor deployment in: (a) deployment time, (b) number of analyzed images, and (c) latency.

recognizer is cached at the IPCs. As shown in the figure, on average, each request can be deployed in one second, with almost linear increase. This is good enough for most IoT analytics.

Distributed execution levels to higher throughput. Fig. 7.4(b) reports the number of analyzed images from an object recognition analytics in a minute. The experiment shows that when the analytics is split into two smaller operators running on two IPCs results in 32% throughput improvement.

Placing the analytics close to data originates and users results in shorter latency. Fig. 7.4(c) presents the end-to-end delay while running the object recognition analytics in the cloud (with an i5 Intel workstation) and a pole (with an i3 Intel PC) with diverse network conditions of the access link (from the access switch to the Internet). The images are from a camera installed on a smart pole and the recognized results are sent to a user in the Computer Science building. The network conditions are emulated by Wonder Shaper [51] and tc [44]. The Wonder Shaper is used to throttle the bandwidth and tc is used to add the network latency. The bandwidth of Ethernet is set to 60 Mbps, WiFi is set to 32 Mbps, and 4G is set to 22.67 Mbps following a recent study [63]. The latency to the cloud is measured using Amazon network testing web page [56]. We calculate average network latencies over 34 Amazon cloud servers under Ethernet, WiFi, and 4G.

The resulting network latencies are: Ethernet:190 ms, WiFi: 241 ms, and 4G: 288 ms. The experiment results show that running the analytics on the smart pole (edge) reduces at least 93% network latency.

Smart pole is built to achieve a smart campus to increase students' safety, monitor the university's environments, and help students' school life. For example, we have air pollution sensors to draw a pollution map, water quality sensors to detect lake quality, and image analytics applications to detect emergent events. Moreover, in the future , we can install displays and microphones, which allows students to interact with the testbed and receive activities. Beside the target of implementing the smart campus, smart pole plays a key role to help researchers from academia and industries to evaluate their new designs, algorithms, and products. Comparing to lab-scale testbeds, a real deployment is under a harsh environments, e.g., (i) the fog devices deployed on the street lights may be broken because of high temperature and humidity, (ii) the communication qualities may be degraded by the interfered wireless channels, and (iii) the noisy sensing data may increase the analytic difficulty and degrade the accuracy. These possible issues cannot be aware from lab-scale testbeds and are required to be solved to realize a comprehensive cloud-to-things continuum platform. Moreover, the experience of building up the smart pole can be used to reproduce a new smart campus, build a smart apartment complex, and extend to built a larger-scale smart city.

Chapter 8

Conclusion

Studying cloud-to-things continuum platforms is a timely topic because researchers from academia and industry start move services, applications, and data analytics from traditional cloud data centers to fog devices, that are closer to end users. However, the cloud data centers cannot to be totally replaced. Hence, aggregate, communicate, and collaborate fog devices in the cloud-to-things continuum is getting popular. In this thesis, we design an intelligent framework and solve optimization problems for the popular cloud-to-things continuum platform. The fog devices are located anywhere with rich resources and capacities, which turn innovative applications. We optimize three resource allocation problems: (i) application deployment problem, (ii) delay-sensitive application optimization problem, and (iii) delay-insensitive application optimization problem. The first problem is designed for providers to serve as many users as possible. The other two problems are designed for users to adapt to system dynamics and optimize applications at run-time to maximize user experience.

We solve the application deployment problem while considering heterogeneous fog devices with diverse resources, required application resources, location requirements, and user-specified QoS targets. We formulate the problem into two formulations for ideal and generalized cloud-to-things continuum, respectively. We design an approximation algorithm with $1/|U|$ approximation factor to solve the ideal problem. The approximation factor is mathematically and experimentally proved in this thesis. The goal of this problem is to maximize number of served users. Our evaluations show that our algorithms outperform the state-of-the-art algorithms by at least 134%, 167%, 161%, and 124%, in terms of the number of satisfied requests, the CPU resource consumption, the RAM resource consumption, and the network resource consumption, respectively.

After deploying the applications, in order to adapt to system dynamics, we add an application specific optimizer into our framework. It solves delay-sensitive application optimization problem and delay-insensitive application optimization problem. For the delay-sensitive application, we focus on optimizing game streaming applications running on our platform. The optimization goal is to maximize gaming experience by adapting to network dynamics of ongoing sessions. Thus, we propose an optimal bitrate adaptation algorithm to dynamically configure the bitrate of game streaming applications in the ongoing sessions running in polynomial time. The proposed algorithm outperforms the baseline algorithms by up to 46% and 30%.

For the delay-insensitive application, we focus on optimizing multimedia content delivery applications over challenged networks on our platform. The optimization goal is to maximize the user experience, which is calculated based on understanding levels to solve digital divide. We propose an optimal DP algorithm which runs in pseudo-polynomial time. Moreover, we also design another efficient algorithm to solve the problem in polynomial time. The goal of our algorithms is to maximize user experience. In fact, our efficient algorithm outperforms the baseline algorithms by at least 20%, 33%, and 39% in terms of user experience, energy efficiency, and disk efficiency.

Besides our optimization algorithms, which is the core of cloud-to-things continuum platforms, we also discuss future directions and remaining tasks of realizing a more comprehensive cloud-to-things platform. Solving the future directions, the comprehensive platform allows more application developers to implement novel applications and even improve existing applications, such as cloud applications, e.g., moving current IoT analytics from data centers to fog devices can reduce network latency and traffics. Moreover, having the platform, novel applications, such as self-driving cars and wearable augmented reality can be realized. The cloud-to-things continuum platform, in the future, leads us to the next era, which makes our life easier, fantastic and safer.

Bibliography

[1] 150 amazing Amazon statistics and facts (August 2018). https://expandedramblings.com/index.php/amazon-statistics/.

[2] Amazon EC2. https://aws.amazon.com/ec2/.

[3] Artificial Intelligence plus the Internet of Things (IoT). https://www.techemergence.com/artificial-intelligence-plus-the-internet-of-things-iot-3-examples-worth-learning-from/.

[4] AWS Greengrass. https://aws.amazon.com/tw/greengrass/.

[5] AWS Snowball Edge. https://aws.amazon.com/snowball-edge/.

[6] Bountry workers (in Chinese). http://bountyworkers.net/.

[7] BRITE. https://www.cs.bu.edu/brite/.

[8] Cisco visual networking index: Forecast and methodology. https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf.

[9] Combining Artificial Intelligence with the Internet of Things could make your business smarter. https://www.ibm.com/blogs/insights-on-business/gbs-strategy/ai-iot-smarter-business/.

[10] Docker. https://www.docker.com.

[11] Docker Swarm. https://hub.docker.com/_/swarm/.

[12] DTN2. http://sourceforge.net/projects/dtn/files/DTN2/dtn-2.9.0/.

[13] The evolution of wireless sensor networks. https://www.silabs.com/documents/public/white-papers/evolution-of-wireless-sensor-networks.pdf.

[14] Fog computing market worth 203.48 million usd by 2022. https://www.marketsandmarkets.com/PressReleases/fog-computing.asp.

[15] GaiKai web page. http://www.gaikai.com/.

[16] GamingAnywhere: An open source cloud gaming project. http://gaminganywhere.org.

[17] Google App Engine. https://cloud.google.com/appengine/.

[18] Here's a map of all Azure and AWS data centers. https://www.datacenterknowledge.com/archives/2016/09/21/heres-a-map-of-all-azure-and-aws-data-centers.

[19] How AI and IoT must work together. https://venturebeat.com/2018/03/15/investors-share-their-predictions-for-ai-and-machine-learning-in-2018/.

[20] How do we accelerate Internet access in Africa? http://ventureburn.com/2014/01/how-do-we-accelerate-internet-access-in-africa/.

[21] IBM CPLEX optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[22] IBR-DTN. http://trac.ibr.cs.tu-bs.de/project-cm-2012-ibrdtn.

[23] IEEE adopts OpenFog reference architecture as official standard for fog computing. https://www.openfogconsortium.org/news/ieee-adopts-openfog-reference-architecture-as-official-standard-for-fog-computing/.

[24] Internet of Things (IoT): number of connected devices worldwide from 2012 to 2020 (in billions). https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[25] Internet solutions division strategy for cloud computing. https://s3.amazonaws.com/files.technologyreview.com/p/pub/legacy/compaq_cst_1996_0.pdf.

[26] It's time to take a closer look at China's mobile industry. http://www.businessinsider.com/the-key-china-mobile-industry-statistics-2013-12?op=1.

[27] ITU ICT facts and figures the world in 2015. http://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx.

[28] Kubernetes. http://kubernetes.io/.

[29] KVM. http://www.linux-kvm.org/.

[30] LETOR 4.0 dataset. http://research.microsoft.com/en-us/um/beijing/projects/letor/letor4dataset.aspx.

[31] librosa. https://github.com/librosa/librosa.

[32] LXC. https://linuxcontainers.org.

[33] Microsoft Azure IoT edge. https://azure.microsoft.com/en-us/services/iot-edge/.

[34] Mobile-edge computing. https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf.

[35] MQTT. http://mqtt.org.

[36] NetIndex real time global broadband and mobile data. http://www.netindex.com/.

[37] OnLive web page. http://www.onlive.com/.

[38] OpenCV. http://opencv.org.

[39] OpenFog. https://www.openfogconsortium.org.

[40] OpenFog reference architecture for fog computing. https://www.openfogconsortium.org/ra/.

[41] Revenues from the artificial intelligence (AI) market worldwide from 2016 to 2025 (in million u.s. dollars). https://www.statista.com/statistics/607716/worldwide-artificial-intelligence-market-revenues/.

[42] SaltStack. https://saltstack.com/.

[43] Social, digital and mobile in India. http://wearesocial.net/blog/2014/07/social-digital-mobile-india-2014/.

[44] tc command. https://linux.die.net/man/8/tc.

[45] TensorFlow. https://www.tensorflow.org.

[46] Topia's term extractor. https://pypi.python.org/pypi/topia.termextract/.

[47] Ubitus web page. http://www.ubitus.net.

[48] VMware. https://www.vmware.com.

[49] What edge computing means for infrastructure and operations leaders. https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders/.

[50] Who coined cloud computing? https://www.technologyreview.com/s/425970/who-coined-cloud-computing/.

[51] Wonder Shaper. http://lartc.org/wondershaper/.

[52] Xen. http://www.xenproject.org/.

[53] Z-Wave. https://www.z-wave.com.

[54] Zigbee. https://www.zigbee.org.

[55] Distributed TensorFlow. https://www.tensorflow.org/deploy/distributed, 2015.

[56] Amazon web services network test. https://cloudharmony.com/speedtest-for-aws, 2018.

[57] T. Abdelkader, K. Naik, A. Nayak, N. Goel, and V. Srivastava. A performance comparison of delay-tolerant network routing protocols. *IEEE Transactions on Network*, 30(2):46–53, 2016.

[58] Z. Abrams and J. Liu. Greedy is good: On service tree placement for in-network stream processing. In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, Lisboa, Portugal, July 2006.

[59] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 8(4):393–422, 2002.

[60] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[61] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the Internet of Things. In *Proc. of ACM SIGCOMM workshop on Mobile Cloud Computing (MCC)*, Helsinki, Finland, August 2012.

[62] J. Boyce, Y. Ye, J. Chen, and A. Ramasubramonian. Overview of SHVC: Scalable extensions of the high efficiency video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(1):20–34, 2016.

[63] A. Brogi, S. Forti, and A. Ibrahim. Deploying fog applications: How much does it cost, by the way? In *Proc. of International Conference on Cloud Computing and Services Science (CLOSER)*, Madeira, Portugal, March 2018.

[64] J. Burgess, B. Gallagher, D. Jensen, and B. Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, Barcelona, Spain, April 2006.

[65] B. Burns, O. Brock, and B. Levine. MV routing and capacity building in disruption tolerant networks. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, Miami, FL, March 2005.

[66] W. Cai, M. Chen, and V. Leung. Towards gaming as a service. *IEEE Transactions on Internet Computing*, 18(3):12–18, 2014.

[67] V. Cardellini, V. Grassi, F. Presti, and M. Nardelli. Optimal operator placement for distributed stream processing applications. In *Proc. of ACM International Conference on Distributed and Event-based Systems (DEBS)*, Irvine, CA, June 2016.

[68] V. Cardellini, V. Grassi, L. Presti, and M. Nardelli. On QoS-aware scheduling of data stream applications over fog computing infrastructures. In *Proc. of IEEE Symposium on Computers and Communication (ISCC)*, Larnaca, Cyprus, July 2015.

[69] V. Cardellini, F. P. V. Grassi, V., and M. Nardelli. Optimal operator replication and placement for distributed stream processing systems. *ACM SIGMETRICS Performance Evaluation Review*, 44(4):11–22, 2017.

[70] N. Changuel, B. Sayadi, and M. Kieffer. Control of distributed servers for quality-fair delivery of multiple video streams. In *Proc. of ACM Multimedia (MM)*, Nara, Japan, October 2012.

[71] G. Chatzimilioudis, N. Mamoulis, and D. Gunopulos. A distributed technique for dynamic operator placement in wireless sensor networks. In *Proc. of IEEE International Conference on Mobile Data Management (MDM)*, Kansas, MO, May 2010.

[72] A. Chatzistergiou and D. Viglas. Fast heuristics for near-optimal task allocation in data stream processing over clusters. In *Proc. of ACM International Conference on Conference on Information and Knowledge Management (CIKM)*, Shanghai, China, November 2014.

[73] L. Chen, Y. Zhou, and D. Chiu. Smart streaming for online video services. *IEEE Transactions on Multimedia*, 17(4):485–497, 2015.

[74] Y. Chen, C. Chang, and W. Ma. Asynchronous rendering. In *Proc. of ACM SIGGRAPH symposium on Interactive 3D Graphics and Games (I3D)*, Washington, DC, February 2010.

[75] P. Cheng, K. Lee, M. Gerla, and J. Härri. GeoDTN+Nav: Geographic DTN routing with navigator prediction for urban vehicular environments. *Mobile Networks and Applications*, 15(1):61–82, 2010.

[76] E. Cho, S. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proc. of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, San Diego, CA, August 2011.

[77] M. Claypool, D. Finkel, A. Grant, and M. Solano. Thin to win? network performance analysis of the onlive thin client game system. In *Proc. of ACM Annual Workshop on Network and Systems Support for Games (NetGames)*, Venice, Italy, October 2012.

[78] N. Do, C. Hsu, and N. Venkatasubramanian. HybCAST: Rich content dissemination in hybrid cellular and 802.11 ad hoc networks. In *Proc. of IEEE Symposium on Reliable Distributed Systems SRDS*, Irvine, CA, October 2012.

[79] N. Do, C. Hsu, and N. Venkatasubramanian. Video dissemination over hybrid cellular and ad hoc networks. *IEEE/ACM Transactions on Mobile Computing*, 13(2):274–286, 2014.

[80] T. Duong, X. Li, R. Goh, X. Tang, and W. Cai. QoS-aware revenue-cost optimization for latency-sensitive services in IaaS clouds. In *Proc. of IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Dublin, Ireland, October 2012.

[81] P. Eisert and P. Fechteler. Low delay streaming of computer graphics. In *Proc. of IEEE International Conference on Image Processing (ICIP)*, San Diego, CA, October 2008.

[82] P. Endo, A. Palhares, N. Pereira, G. Goncalves, D. Sadok, J. Kelner, B. Melander, , and J. Mangs. Resource allocation for distributed cloud: concepts and research challenges. *IEEE Network*, 25(4):42–46, 2011.

[83] K. Fall. A delay-tolerant network architecture for challenged Internets. In *Proc. of ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, Karlsruhe, Germany, August 2003.

[84] A. Fischer, J. Botero, M. Beck, H. Meer, and X. Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys and Tutorials*, 15(4):1888–1906, 2013.

[85] R. Friedman, A. Kogan, and Y. Krivolapov. On power and throughput tradeoffs of WiFi and Bluetooth in smartphones. *IEEE Transactions on Mobile Computing*, 12(2):1363–1375, 2013.

[86] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa. Cooperative caching for efficient data access in disruption tolerant networks. *IEEE Transactions on Mobile Computing*, 13(3):611–625, 2014.

[87] I. Ghergulescu, A.-N. Moldovan, and C. Muntean. Energy-aware adaptive multimedia for game-based e-learning. In *Proc. of IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Beijing, China, June 2014.

[88] F. Giesen, R. Schnabel, and R. Klein. Augmented compression for server-side rendering. In *Proc. of International Fall Workshop on Vision, Modeling, and Visualization (VMV)*, Konstanz, Germany, October 2008.

[89] Y. Go, O. Kwon, and H. Song. An energy-efficient HTTP adaptive video streaming with networking cost constraint over heterogeneous wireless networks. *IEEE Transactions on Multimedia*, 17(9):1646–1657, 2015.

[90] M. Golkarifard, J. Yang, Z. Huang, A. Movaghar, and P. Hui. Dandelion: A unified code offloading system for wearable computing. *IEEE Transactions on Mobile Computing*, 2018. Early Access.

[91] M. Gonzalez, C. Hidalgo, and A. Barabasi. Understanding individual human mobility patterns. *Nature*, 453:779–782, 2008.

[92] D. Greene and P. Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proc. of ACM International Conference on Machine Learning (ICML)*, Pittsburgh, PA, June 2006.

[93] R. Guruprasad and S. Dey. Battery aware video delivery techniques using rate adaptation and base station reconfiguration. *IEEE Transactions on Multimedia*, 17(9):1630–1645, 2015.

[94] H. Hanano, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito. Video ads dissemination through WiFi-cellular hybrid networks. In *Proc. of IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Galveston, TX, March 2009.

[95] K. Harras and K. Almeroth. Controlled flooding in disconnected sparse mobile networks. *Wireless Communication and Mobile Computing*, 9(1):21–33, 2009.

[96] J. Herrera and J. Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[97] M. Hirzel, R. Soule, S. Schneider, B. Gedik, and R. Grimm. A catalog of stream processing optimizations. *ACM transactions on Computing Surveys*, 46(4), 2014.

[98] O. Holthe, O. Mogstad, and L. Ronningen. Geelix LiveGames: Remote playing of video games. In *Proc. of IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, January 2009.

[99] H. Hong, D. Chen, C. Huang, K. Chen, and C. Hsu. QoS-aware virtual machine placement for cloud games. In *Proc. of ACM Annual Workshop on Network and Systems Support for Games (NetGames)*, Denver, CO, December 2013.

[100] H. Hong, D. Chen, C. Huang, K. Chen, and C. Hsu. Placing virtual machines to optimize cloud gaming experience. *IEEE Transactions on Cloud Computing*, 3(1):42 – 53, 2014.

[101] H. Hong, J. Chuang, and C. Hsu. Animation rendering on multimedia fog computing platforms. In *Proc. of IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg,Luxembourg, December 2016.

[102] H. Hong, T. El-Ganainy, C. Hsu, K. Harras, and M. Hefeeda. Disseminating multilayer multimedia content over challenged networks. *IEEE Transactions on Multimedia*, 20(2):245–360, 2018.

[103] H. Hong, T. Fan-Chiang, C. Lee, K. Chen, C. Huang, and C. Hsu. GPU consolidation for cloud games: Are we there yet? In *Proc. of ACM Annual Workshop on Network and Systems Support for Games (NetGames)*, Nagoya, Japan, December 2014.

[104] H. Hong, C. Hsu, T. Tsai, C. Huang, K. Chen, and C. Hsu. Enabling adaptive cloud gaming in an open-source cloud gaming platform. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2078–2091, 2015.

[105] H. Hong, P. Tsai, A. Cheng, M. Uddin, N. Venkatasubramanian, and C. Hsu. Supporting Internet-of-Things analytics in a fog computing platform. In *Proc. of IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Hong Kong, China, December 2017.

[106] H. Hong, S. Wang, C. Tan, T. El-Ganainy, K. Harras, C. Hsu, and M. Hefeeda. Challenged content delivery network: Eliminating the digital divide. In *Proc. of ACM Multimedia (MM) Demo*, Brisbane, Australia, October 2015.

[107] C. Hsu, H. Hong, T. Elgamal, K. Nahrstedt, and N. Venkatasubramanian. Multimedia fog computing: Minions in the cloud and crowd. In *Frontiers of Multimedia Research*, chapter 10, pages 255–286. Association for Computing Machinery and Morgan & Claypool, January 2018.

[108] W. Hsu, D. Dutta, and A. Helmy. CSI: A paradigm for behavior-oriented profilecast services in mobile networks. *Ad Hoc Networks*, 10(8):1586–1602, 2012.

[109] Y. Hu, D. Niu, and Z. Li. A geometric approach to server selection for interactive video streaming. *IEEE Transactions on Multimedia*, 18(5):840–851, 2016.

[110] C. Huang, K. Chen, D. Chen, H. Hsu, and C. Hsu. GamingAnywhere: The first open source cloud gaming system. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 10(1s):10:1–10:25, 2014.

[111] C. Huang, P. Chen, Y. Huang, K. Chen, and C. Hsu. Measuring the client performance and energy consumption in mobile cloud gaming. In *Proc. of ACM Annual Workshop on Network and Systems Support for Games (NetGames)*, Nagoya, Japan, December 2014. Poster.

[112] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen. Gaminganywhere: An open cloud gaming system. In *Proc. of ACM Multimedia Systems (MMSys)*, Oslo, Norway, February 2013.

[113] C.-Y. Huang, C.-H. Hsu, D.-Y. Chen, and K.-T. Chen. Quantifying user satisfaction in mobile cloud games. In *Proc. of ACM Workshop on Mobile Video Delivery (MoVid)*, Singapore, March 2014.

[114] Y. Huang, Z. Luan, R. He, and D. Qian. Operator placement with QoS constraints for distributed stream processing. In *Proc. of IEEE International Conference on Network and Service Management (CNSM)*, Paris, France, October 2011.

[115] P. Hui, J. Crowcroft, and E. Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 10(11):1576–1589, 2011.

[116] S. Isaacman and M. Martonosi. Low-infrastructure methods to improve Internet access for mobile users in emerging regions. In *Proc. of ACM International Conference Companion on World Wide Web (WWW)*, Hyderabad, India, March 2011.

[117] V. Jindal. History and architecture of wireless sensor networks for ubiquitous computing. *International Journal of Advanced Research in Computer Engineering & Technology*, 7(2):214–217, 2018.

[118] A. Jurgelionis, P. Fechteler, P. Eisert, F. Bellotti, H. David, J. Laulajainen, R. Carmichael, V. Poulopoulos, A. Laikari, P. Perala, A. Gloria, and C. Bouras. Platform for distributed 3D gaming. *International Journal of Computer Games Technology*, 2009:1:1–1:15, 2009.

[119] S. Kang and M. Mutka. Efficient mobile access to internet data via a wireless peer-to-peer network. In *Proc. of IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Orlando, FL, March 2004.

[120] R. Kapoor, L. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi. Capprobe: A simple and accurate capacity estimation technique. In *Proc. of ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, Portland, OR, August 2004.

[121] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, and A. Markopoulou. MicroCast: Cooperative video streaming on smartphones. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Low Wood Bay, UK, June 2012.

[122] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, Orlando, FL, March 2012.

[123] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze. Cloud federation. In *Proc. of International Conference on Cloud Computing, GRIDs, and Virtualization*, Rome, Italy, September 2011.

[124] Y. Lee, K. Chen, H. Su, and C. Lei. Are all games equally cloud-gaming-friendly? an electromyographic approach. In *Proc. of IEEE/ACM Annual Workshop on Network and Systems Support for Games (NetGames)*, Venice, Italy, Oct 2012.

[125] Y.-T. Lee and K.-T. Chen. Is server consolidation beneficial to MMORPG? a case study of World of Warcraft. In *Proc. of IEEE International Conference on cloud computing (CLOUD) 2010*, Miami, FL, February 2010.

[126] M. Li, M. Claypool, and R. Kinicki. WBest: a bandwidth estimation tool for ieee 802.11 wireless networks. In *Proc. of IEEE Conference on Local Computer Networks (LCN)*, Montreal, Canada, October 2008.

[127] X. Li and C. Qian. A survey of network function placement. In *Proc. of IEEE Annual Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, January 2016.

[128] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye. Mining periodic behaviors for moving objects. In *Proc. of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Washington, DC, July 2010.

[129] J. Liu, P. Dolan, and E. Pedersen. Personalized news recommendation based on click behavior. In *Proc. of ACM international conference on Intelligent user interfaces (IUI)*, Hong Kong, China, February 2010.

[130] Z. Lu and Y. Wen. Distributed and asynchronous solution to operator placement in large wireless sensor networks. In *Proc. of IEEE International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, Chengdu, China, December 2012.

[131] Z. Lu, Y. Wen, R. Fan, L. Tan, and J. Biswas. Toward efficient distributed algorithms for in-network binary operator tree placement in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 31(4):743–755, 2013.

[132] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. Pocketweb: Instant web browsing for mobile devices. *ACM SIGARCH Computer Architecture News*, 40(1):1–16, 2012.

[133] R. Mahmud, R. Kotagiri, and R. Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*, chapter 5, pages 103–130. Springer, October 2018.

[134] L. Mainetti, L. Patrono, and A. Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *Proc. of IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Split, Croatia, September 2011.

[135] R. Mohan, J. Smith, and C. Li. Adapting multimedia internet content for universal access. *IEEE Transactions on Multimedia*, 1(1):104–114, 1999.

[136] K. Mokhtarian and M. Hefeeda. Capacity management of seed servers in peer-to-peer streaming systems with scalable video streams. *IEEE Transactions on Multimedia*, 15(1):181–194, 2012.

[137] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: A location predictor on trajectory pattern mining. In *Proc. of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Paris, France, June 2009.

[138] V. Mota, F. Cunha, D. Macedo, J. Nogueira, and A. Loureiro. Protocols, mobility models and tools in opportunistic networks: A survey. *Computer Communications*, 48:5 – 19, 2014.

[139] C. Mouradian, D. Naboulsi, S. Yangui, R. Glitho, M. Morrow, and P. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 20(1):416–464, 2018.

[140] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. Ferrag, N. Choudhury, and V. Kumar. Security and privacy in fog computing: Challenges. *IEEE Access*, 5:19293–19304, 2017.

[141] M. Mukherjee, L. Shu, and D. Wang. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys and Tutorials*, 20(3):1826–1857, 2018.

[142] I. Naas, R. Parvedy, J. Boukhobza, and L. Lemarchand. iFogStor: An IoT data placement strategy for fog infrastructure. In *Proc. of IEEE International Conference on Fog and Edge Computing (ICFEC)*, Madrid, Spain, May 2017.

[143] J. Ni, K. Zhang, X. Lin, and X. Shen. Securing fog computing for Internet of Things applications: Challenges and solutions. *IEEE Communications Surveys and Tutorials*, 20(1):601–628, 2017.

[144] Y. Nimmagadda, K. Kumar, and Y. Lu. Adaptation of multimedia presentations for different display sizes in the presence of preferences and temporal constraints. *IEEE Transactions on Multimedia*, 12(7):650–664, 2010.

[145] H. Ntareme, M. Zennaro, and B. Pehrson. Delay tolerant network on smartphones: Applications for communication challenged areas. In *Proc. of ACM Extreme Conference on Communication (ExtremeCom)*, Manaus, Brazil, September 2011.

[146] E. Pagani, L. Valerio, and G.Rossi. Weak social ties improve content delivery in behavior-aware opportunistic networks. *Ad Hoc Networks*, 25(B):314–329, 2015.

[147] A. Pathak and K. Prasanna. Energy-efficient task mapping for data-driven sensor network macroprogramming. *IEEE Transactions on Computers*, 59(7):955–968, 2010.

[148] C. Perera, Y. Qin, J. Estrella, S. Reiff-Marganiec, and A. Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys*, 50(3):32:1–32:43, 2017.

[149] A. Pietilainen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. MobiClique: Middleware for mobile social networking. In *Proc. of ACM Workshop on Online Social Networks (WOSN)*, Barcelona, Spain, August 2009.

[150] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser. A parsimonious model of mobile partitioned networks with clustering. In *Proc. of IEEE International Conference on Communication Systems and Networks and Workshops (COMSNETS)*, Bangalore, India, January 2009.

[151] F. Pires and B. Baran. A virtual machine placement taxonomy. In *Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Shenzhen, China, May 2015.

[152] F. Qian, K. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Web caching on smartphones: Ideal vs. reality. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Low Wood Bay, UK, June 2012.

[153] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. Pathchirp: Efficient available bandwidth estimation for network paths. In *Proc. of Passive and Active Monitoring Workshop (PAM)*, volume 4, San Diego, CA, April 2003.

[154] B. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *Proc. of IEEE International Joint Conference on INC, IMS and IDC (NCM)*, Seoul, South Korea, August 2009.

[155] S. Rizou, F. Durr, and K. Rothermel. Providing qos guarantees in large-scale operator networks. In *Proc. of IEEE International Conference on High Performance Computing and Communications (HPCC)*, Melbourne, Australia, September 2010.

[156] H. Saha, A. Mandal, and A. Sinha. Recent trends in the internet of things. In *Proc. of IEEE In Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, January 2017.

[157] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[158] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwalder. Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In *Proc. of ACM International Conference on Distributed and Event-based Systems (DEBS)*, Irvine, CA, June 2016.

[159] B. Schilling, B. Koldehofe, and K. Rothermel. Efficient and distributed rule placement in heavy constraint-driven event systems. In *Proc. of IEEE International Conference on High Performance Computing and Communications (HPCC)*, Banff, AB, Canada, September 2011.

[160] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, 2007.

[161] M. Sharifi, S. Kafaie, and O. Kashefi. A survey and taxonomy of cyber foraging of mobile devices. *IEEE Communications Surveys & Tutorials*, 14(4):1232–1243, 2012.

[162] Y. Shen, C. Jiang, Q. Quek, and Y. Ren. Device-to-device-assisted communications in cellular networks: An energy efficient approach in downlink video sharing scenario. *IEEE Transactions on Wireless Communications*, 15(2):1575–1587, 2016.

[163] Y. Shen, C. Jiang, T. Quek, and Y. Ren. Location-aware device communication design: exploration and exploitation on energy. *IEEE Wireless Communications*, 23(2):46–52, 2016.

[164] S. Shi, C. Hsu, K. Nahrstedt, and R. Campbell. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *Proc. of ACM Multimedia (MM)*, Scottsdale, AZ, November 2011.

[165] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar. Towards QoS-aware fog service placement. in fog and edge computing. In *Proc. of IEEE International Conference on Fog and Edge Computing (ICFEC)*, Madrid, Spain, May 2017.

[166] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner. Resource provisioning for IoT services in the fog. In *Proc. of IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, Macau, China, November 2016.

[167] M. Taneja and A. Davy. Resource aware placement of IoT application modules in fog-cloud computing paradigm. In *Proc. of IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, May 2017.

[168] N. Tziritas, T. Loukopoulos, S. Khan, and C. Xu. Distributed algorithms for the operator placement problem. *IEEE Transactions on Computational Social Systems*, 2(4):182–196, 2015.

[169] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical report, Duke University, 2000.

[170] T. Verbelen, S. Pieter, T. Filip, and D. Bart. Cloudlets: Bringing the cloud to the mobile user. In *Proc. of ACM Workshop on Mobile Cloud Computing and Services (MCS)*, Lake District, UK, June 2012.

[171] J. Wang, C. Jiang, Z. Bie, Q. Quek, and Y. Ren. Mobile data transactions in device-to-device communication networks: Pricing and auction. *IEEE Wireless Communications Letters*, 5(3):300–303, 2016.

[172] S. Wang and S. Dey. Modeling and characterizing user experience in a cloud server based mobile gaming approach. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, Honolulu, HW, December 2009.

[173] S. Wang and S. Dey. Rendering adaptation to address communication and computation constraints in cloud mobile gaming. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, Miami, FL, December 2010.

[174] S. Wang and S. Dey. Cloud mobile gaming: Modeling and measuring user experience in mobile wireless networks. *ACM Transactions on Mobile Computing and Communications Review*, 16(1):10–21, 2012.

[175] S. Wang, C. Fan, Y. Huang, and C. Hsu. Toward optimal crowdsensing video quality for wearable cameras in smart cities. In *Proc. of IEEE International Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Hong Kong, China, April 2015.

[176] T. Wang, P. Hui, S. Kulkarni, and P. Cuff. Cooperative caching based on file popularity ranking in delay tolerant networks. In *Proc. of ACM Extreme Conference on Communication (ExtremeCom)*, Zurich, Switzerland, March 2012.

[177] Y. Wang, J. Kim, S. Chang, and H. Kim. Utility-Based video adaptation for universal multimedia access (UMA) and content-based utility function prediction for Real-Time video transcoding. *IEEE Transactions on Multimedia*, 9(2):213–220, 2007.

[178] Y. Wang, J. Ostermann, and Y. Zhang. *Video Processing and Communications*. Prentice Hall, 2001.

[179] D. Winter, P. Simoens, L. Deboosere, F. Turck, J. Moreau, B. Dhoedt, and P. Demeester. A hybrid thin-client protocol for multimedia streaming and interactive gaming applications. In *Proc. of ACM International Workshop on Network and Operating Systems Support for Digital Audio and Vid eo (NOSSDAV)*, Newport, RI, May 2006.

[180] D. Wu, Z. Xue, and J. He. iCloudAccess: Cost-effective streaming of video games from the cloud with low latency. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(8):1405–1416, 2014.

[181] X. Wu, J. Yang, Y. Ran, and H. Xi. Adaptive scalable video transmission strategy in energy harvesting communication system. *IEEE Transactions on Multimedia*, 17(12):2345–2353, 2015.

[182] Z. Xiang, Q. Zhang, W. Zhu, Z. Zhang, and Y. Zhang. Peer-to-peer based multimedia distribution service. *IEEE Transactions on Multimedia*, 6(2):343–355, 2004.

[183] Y. Xu and S. Mao. A survey of mobile cloud computing for rich media applications. *IEEE Transactions on Wireless Communications*, 20(3):46–53, 2013.

[184] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292–2330, 2008.

[185] L. Ying, Z. Liu, D. Towsley, and H. Xia. Distributed operator placement and data caching in large-scale sensor networks. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, Phoenix, AZ, April 2008.

[186] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712, 2016.

[187] A. Zhang, J. Chen, L. Zhou, and S. Yu. Graph theory-based QoE-Driven cooperation stimulation for content dissemination in Device-to-Device Communication. *IEEE Transactions on Emerging Topics in Computing*, 4(4):556–567, 2016.

[188] A. Zhang, L. Wang, and L. Zhou. Location-based distributed caching for device-to-device communications underlaying cellular networks. *Wireless Communications and Mobile Computing*, 16(13):1859–1875, 2015.

[189] G. Zhang, W. Liu, X. Hei, and W. Cheng. Unreeling Xunlei Kankan: Understanding hybrid CDN-P2P video-on-demand streaming. *IEEE Transactions on Multimedia*, 17(2):229–242, 2014.

[190] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[191] Y. Zhang, C. Tan, and L. Qun. Cachekeeper: A system-wide web caching service for smartphones. In *Proc. of ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, Zurich, Switzerland, September 2013.

[192] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proc. of ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc)*, Tokyo, Japan, May 2004.

[193] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma. Understanding mobility based on GPS data. In *Proc. of ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, Seoul, Korea, September 2008.

[194] L. Zhou. Mobile device-to-device video distribution: Theory and application. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 12(3), 2016.