國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

在智慧城市閘道器上之物聯網分析程式容器下載與頻寬分配研究

Image Download and Rate Allocation of Internet-of-Things Analytics at Gateways in Smart Cities

王鈺鎔
Yu-Jung Wang

學號：106062600
Student ID:106062600

指導教授：徐正炘 博士
Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 109 年 07 月
July, 2020

國立清華大學
資訊工程研究所

碩士論文

在智慧城市閘道器上之物聯網分析程式容器下載與頻寬分配研究

王鈺鎔 撰

108
07

Acknowledgments

I would like to express my gratitude to all the people who helped me in the course of my research. I would also like to express my sincere gratitude to my advisor Prof. Cheng-Hsin Hsu. Without his guidance and suggestions, I would not have accomplished what I have, nor learned so much. I would like to thank my labmates in Networking and Multimedia Systems Laboratory, especially Ching-Ling Fan and Hua-Jun Hong, who helped me a great deal in the course of my research. Without their careful tutoring, I would not have been able to finish this work. I would like to thank another labmate, Yu-Chen Hsieh, who also helped me and encouraged me a lot in in the course of my research. Lastly, I want to thank my parents for providing me with their firm support and standing behind my decisions along the way.

致謝

在此我要感謝所有幫助過我的人，如果沒有你們的幫助我一定沒有辦法順利完成我的論文。 我要特別感謝我的指導教授：徐正炘教授。如果沒有他的給予我的指導以及建議， 我一定沒辦法完成如此多的事情以及學到如此多的東西。 我也要感謝網路與多媒體系統實驗室的同學們， 特別是洪華駿和樊慶玲在一路上的研究中幫助我非常的多。 我還要感謝另一位同學謝侑臻，在一路上的研究中也是幫助我和鼓勵我良多。 最後我要感謝我的母親，提供我堅定不移的支持，同時也支持我所作的每一個決定。

Abstract

Internet-of-Things (IoT) devices are connected to the Internet through a gateway, which can host IoT analytics encapsulated in containers to convert raw sensor data into more condensed processed data. In this thesis, we study two research problems to maximize the overall Quality-of-Service (QoS) level of all IoT analytics that run on both data center servers and gateways. The first problem is selecting additional IoT analytics to deploy on a gateway to save upload bandwidth due to uploading raw sensor data. The second problem is allocating the residue upload bandwidth among all IoT analytics to maximize the overall QoS level. We propose several algorithms to solve these two research problems. Moreover, we implement several classical layer replacement policies and discuss their performance. We have implemented real testbeds to evaluate our proposed system and algorithms. Our experiment results reveal that our proposed algorithms: (i) capitalize the download bandwidth and storage space of the gateway in order to save the upload bandwidth consumption, (ii) achieve high QoS levels without overloading the network and gateway, (iii) outperform the other two baseline algorithms by 18% and 37% in QoS levels in low upload network bandwidth environment, and (iv) outperform the other two baseline algorithms by 162% and 61% in the utilization rate of upload bandwidth in high upload network bandwidth environment.

中文摘要

物聯網 (IoT) 裝置透過閘道器連接至網路， 並且閘道器讓被包裝成容器的物聯網分析程式能夠轉換原始的感測器資料成為更為濃縮的處理過的資料。 在這個論文裡， 我們研究兩個研究問題去最大化跑在資料中心伺服器上和閘道器上的物聯網分析程式的總體服務品質 (QoS)。 第一個問題是根據需要上傳的原始的感測器資料， 挑選一部分的物聯網分析程式去佈建在閘道器上，用以節省所需的上傳頻寬。 第二個問題是分配剩下的上傳頻寬給所有的物聯網分析程式，用以最大化總體的服務品質。 我們提出了一些演算法去解決這兩個研究問題。 除此之外，我們實作了一些經典的分層替換策略並且探討了他們的表現。我們已經實作了真實的平台用以測試我們提出的系統和演算法。 我們的實驗結果揭示了我們提出的演算法： (i) 運用閘道器的下載頻寬和儲存空間來節省上傳頻寬的消耗， (ii) 在沒有過載網路和閘道器的情況下，取得高服務品質級別， (iii) 在低上傳頻寬的環境下，比起其他兩個基準算法，服務品質級別分別高出了18%和37%， (iv) 在高上傳頻寬的環境下，比起其他兩個基準算法，上傳頻寬的使用率分別高出了162%和61%。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Along with the tremendous increase in the number of Internet-of-Things (IoT) devices, analyzing IoT sensor data has led to new market opportunities [4]. We refer to such analysis tasks as *IoT analytics* [43], which extract useful information from the ocean of sensor data. Smart city[1] [49] is one of the most popular IoT usage scenarios in which IoT analytics enable more efficient resource usage, higher Quality-of-Service (QoS), and lower management costs. In smart cities, IoT devices are often clustered, and thus it makes sense to deploy a CPE (Customer Premises Equipment) for each cluster of IoT devices. CPEs are essentially *gateways* that interconnect IoT devices and the Internet, as illustrated in Figure 1.1. More precisely, IoT devices are connected to their gateways via short-range wireless (or wired) networks, while gateways are attached to heterogeneous Internet access links.

A naive way to deploy IoT analytics is to send all sensor data through gateways and access links to data center servers for analysis. Doing so, however, incurs a large amount of network traffic over the access links, which could be costly if city-wide 4G/5G cellular links are installed. Uploading data from rich-media sensors, like microphones, cameras, and Lidar (Light detection and ranging) could cause: (i) high Internet access cost and (ii) degraded QoS due to network congestion. To cope with these issues, we dynamically *deploy* some IoT analytics to gateways because: (i) modern gateways offer nontrivial computational power, and (ii) download bandwidth of access links may not be fully utilized. For IoT analytics running on gateways, only processed data are sent over gateways and access links, and thus the upload bandwidth consumption is reduced. As shown in Figure 1.1, IoT analytics are packed into *container images* that can be launched on different computers on-demand [23, 24].

Deploying containers on gateways is not an easy task, because container images may

[1]For brevity, we use the term *smart city*, while our discussion can be generalized to *smart spaces* in various scales, such as smart buildings, smart campuses, and smart communities.

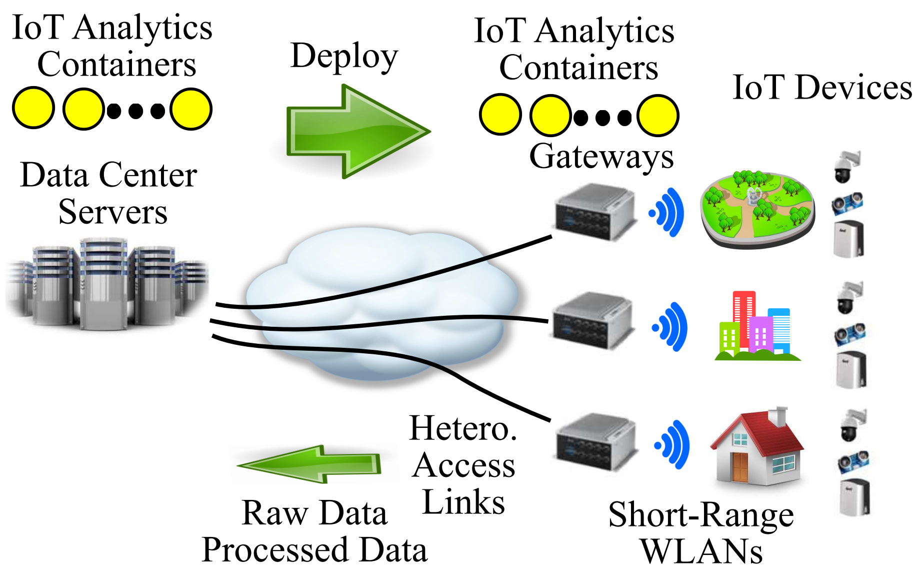


Figure 1.1: We propose to deploy some IoT analytics on gateways to reduce the bandwidth consumption.

be quite large. For example, a Ubuntu Linux base image has a size of 120+ MB, while adding Mobilenet [25] to it increases its size by 122–638 MB. These container images are organized into *layers*, where a layer may be shared by multiple container images. The image layers are downloaded and stored in the *image pool* at the gateway, which has limited storage space. Therefore, before deploying an IoT analytics to a gateway, we need to carefully account for both the pros (saved upload bandwidth) and cons (consumed download bandwidth and storage space).

1.1 Contributions

In this thesis, we strive to solve the following two research problems to maximize the overall QoS level of all IoT analytics that run on both data center servers and gateways.

- **Image download problem** selects additional IoT analytics to deploy to a gateway to save as much upload bandwidth as possible.

- **Rate allocation problem** allocates the upload bandwidth among IoT analytics on both the data center servers and gateways to maximize the overall QoS level.

In particular, we propose an optimal, approximative, and heuristic algorithm to solve the image download problem. We also propose a heuristic algorithm to solve the rate allocation problem. We discuss four classical layer replacement policies as well. We have implemented our solution on open-source projects, including Kubernetes [7], Docker [3], and TensorFlow [10]. We set up testbeds to evaluate the performance of our proposed algorithms under different system parameters.

Our experiment results show the merits of our proposed algorithms, e.g.:

- We achieve as high as 0.99 weighted QoS level in the scale of [0,1] because we deploy as many IoT analytics containers to the gateway as possible.
- For image download problems, our heuristic algorithm saves as much upload bandwidth as the optimal algorithm while achieving similar QoS levels.
- For rate allocation problems, our proposed algorithm outperforms the two baseline algorithms by 18% and 37% in weighted QoS levels in a low upload network bandwidth environment, and it also outperforms the two baseline algorithms by 162% and 61% in the utilization of upload bandwidth in a high upload network bandwidth environment.

All the results are achieved without overloading the access link and gateway. For example, we have observed at most a 400 ms running time in our algorithms.

1.2 Thesis Organization

This thesis is organized as follows. We give the background of the architectures of edge computing and Internet of Things in Chapter 2. We state the research problem in Chapter 3. Chapter 4 gives our system architecture. We solve the two research problems: (i) image download problem in Chapter 5 and (ii) rate allocation problem in Chapter 6. We then discuss the representative layer replacement policies in Chapter 7 We implement our solutions and report the evaluation results in Chapter 8. We survey the literature in Chapter 9. Chapter 10 concludes the thesis and describes our future work.

Chapter 2

Background

In this chapter, we introduce the architecture and the features of edge computing and IoT. We next discuss how to package IoT analytics into containers and the advantages of containers. In the end, we briefly present two container management tools, Docker and Kubernetes, which are used in our implementation.

2.1 Edge Computing

The mobile devices and various sensors are becoming more and more prevalent. The resulting massive data from sensors and devices becomes a heavy load on the cloud servers. In addition, most of these devices need to respond to users in a short time, which sometimes suffers from the long routes from the cloud servers to the end-users. In response to the long distance between the cloud servers and end-users in cloud computing, edge computing seeks to provide services closer to end-users. Edge computing puts the analytics in the local devices or servers, which means it has the benefits of low latency and quick response time. Thus, edge computing plays an important role in many fields like surveillance, virtual reality, etc.

Figure 2.1 shows the architecture of edge computing. Generally, edge computing contains three layers [48]:

- **Front-end.** The front-end includes edge sensors and edge devices, which provide interactions for the end-users. Edge devices in the front-end have to respond in real-time to users. However, these edge devices usually have limited resources, which makes it hard to accomplish all the requested tasks. Therefore, edge devices in the front-end still need help from the near-end and far-end.

- **Near-end.** The near-end includes gateways and edge servers. Gateways deal with the most of network traffic in edge computing. Edge servers have more resources to run the analytics, which means edge servers in the near-end can run harder and more sophisticated analytics like sound classification and object detection. Also, edge servers are close to users. Although the network latency of the analytics in edge servers increases a little bit, users still can get better real-time services.

- **Far-end.** The far-end includes cloud servers. These servers typically are far away from end-users. Compared to the edge devices and edge servers, cloud servers have the most resources. Hence, cloud servers are able to run analytics, such as data mining and machine learning. Nonetheless, the long distance between cloud servers and users dramatically increases the network latency, which limits us from putting all the analytics in cloud servers.

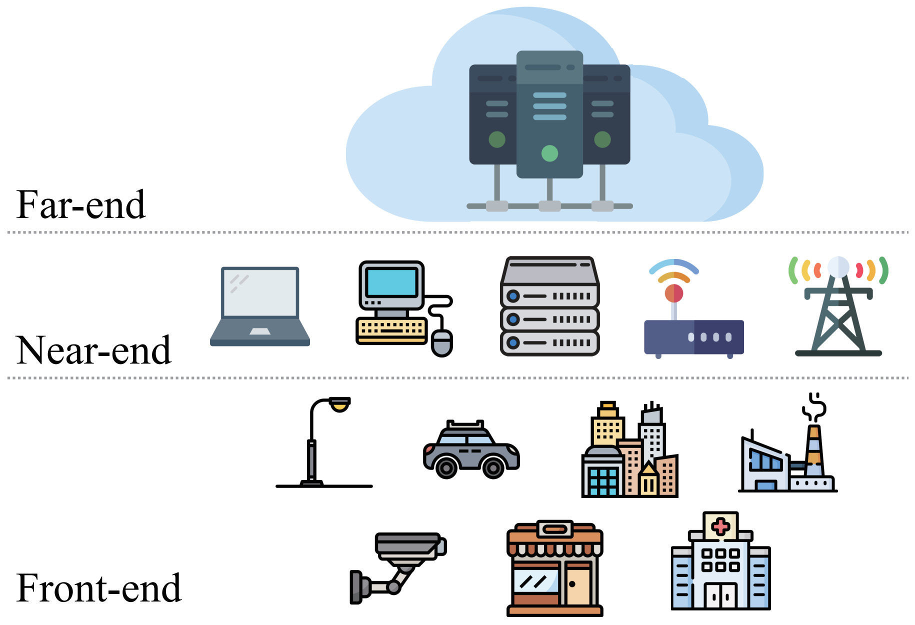


Figure 2.1: The architecture of edge computing [48].

2.2 Internet of Things

Originally a new idea for Radio Frequency Identification (RFID) to connect items to the Internet in 1999 [15], IoT is getting more and more popular today. IoT is contained everywhere in our life, such as the so-called smart home [31] and smart city [27, 49]. In 2014, Amazon released its smart speakers, Amazon Echo [1], which is a smart assistant designed to help people with everyday tasks like ordering pizzas, taking a phone call, playing music, etc. Also, Google and Alibaba Group introduced their smart speakers, Google Home [5] and Tmall Genie [11], in 2016 and 2017 respectively. According to the statics [6], the number of IoT connected devices is forecasted to grow to 75.44 billion in 2025.

Figure 1.1 gives the typical architecture of IoT. IoT is composed of three main components:

- **Sensors/Devices.** In IoT, billions of sensors/devices have been developed in many fields. These IoT sensors/devices can provide various types of data for IoT analytics, such as images, audio, air conditioning, etc. Compared to the data produced by humans, sensors/devices can automatically and continuously produce diverse types of data, which is a heavy burden for cloud servers. Therefore, sensors/devices are interconnected to gateways instead of directly connecting to cloud servers.

- **Gateways.** As middlewares between sensors and cloud servers, gateways transfer and buffer the raw data from sensors to cloud servers. In addition, some gateways have restricted resources to run some IoT analytics. Running IoT analytics on gateways not only makes the loading of cloud servers lighter but saves the bandwidth of cloud servers. Some IoT analytics need huge *raw data* from sensors but output the simple *processed data* to the users. For example, transferring a 4-second and 22-kHz audio needs around 300 KB of bandwidth, but transferring the recognition result of the audio only uses about 10 bytes of bandwidth,

- **Cloud Servers.** Cloud servers provide services to users by running IoT analytics and receive data from sensors through gateways. Cloud servers have much more resources than gateways to satisfy numerous and various IoT analytics.

2.3 IoT Analytics

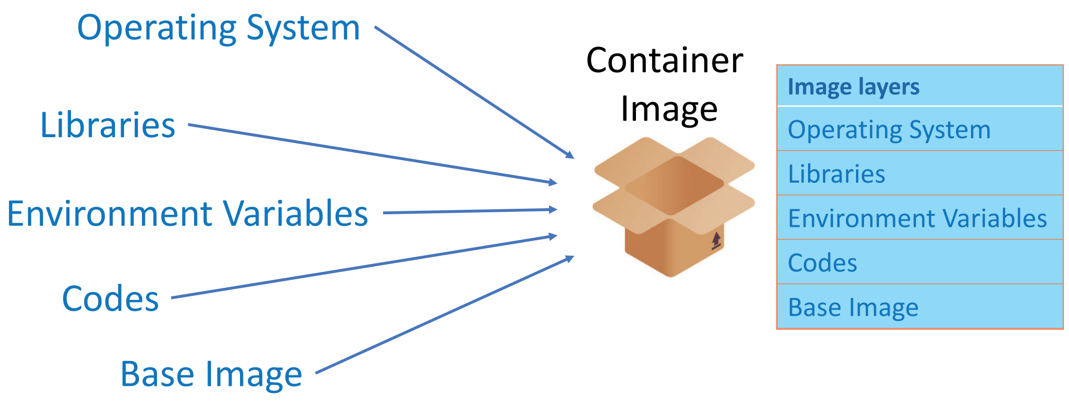


Figure 2.2: Illustrations of IoT analytics containers.

The popularity of IoT devices leads to another issue - how to rapidly and easily collect data and manage resources to run the applications in IoT devices. We use container virtualization techniques to solve the above issue. We can package applications and the needed environments, including operating systems, libraries, environment variables, etc into containers, which is shown in Figure 2.2. After analytics containers are deployed to devices, the environments are set up in advance, which can significantly shorten the startup time of analytics. Moreover, some analytics containers can share the same environment, such as the same operating system, which can dramatically reduce the storage occupancy. Therefore, a container is a lightweight process in IoT devices. To create IoT analytics containers, we then introduce two containers management tools in the following two subsections 2.3.1 and 2.3.2, which are also employed in our implementation.

2.3.1 Docker

One of the most famous container technology is Docker [3]. Docker container technology is developed from Linux Containers (LXC) [8] technology. In addition to LXC virtual technology, Docker also adds storage drivers and the concept of base images. The concept provides good architecture and dependency in order to cache Docker images. Docker containers begin from images that can be seen as the code of containers. An image can start many containers, and these containers can do different actions if needed. A new image can be built from an existing image as a base and with new commands on the base. Also, this image can be the base image of many other images. We create two new images, python3 and pip3, by two Docker files in Tables 2.1 and 2.2 as an example. In Table 2.1, to create python3, we first pull an image (ubuntu:16.04) from a remote Docker pool and use ubuntu:16.04 as a base image. Then, we create the working directory and install Python3 through Linux commands. After creating python3, we follow similar steps, using python3 as a base image and install pip3 through Linux commands, given in Table 2.2. Now, we have three images, ubuntu:16.04, python3, and pip3. Notice that Docker fails to run a container from the image pip3 if one of ubuntu:16.04 or python3 is missing.

2.3.2 Kubernetes

Kubernetes is originally designed by Google to manage the clusters of numerous IoT analytics containers deployed on heterogeneous IoT devices and gateways at different places. Kubernetes provides automatical deployment, scaling, and management for IoT analytics containers and IoT devices. Users can define pods and services for analytics containers to deploy different analytics containers to heterogeneous IoT devices. Users can also define different roles for each IoT device and combine each IoT device in clusters. Kubernetes

Table 2.1: The Docker File of Python3

```
From ubuntu:16.04

RUN mkdir python3
WORKDIR /python3

RUN apt-get -y update
RUN apt-get install -y python3
```

Table 2.2: The Docker File of Pip3

```
From python3:latest

RUN mkdir pip3
WORKDIR /pip3

RUN apt-get -y update
RUN apt-get install -y python3-pip
```

can thus monitor the condition of the analytics containers and the resources in each IoT device. Moreover, users can easily scale up the number of IoT analytics containers since Kubernetes can automatically create replicas of analytics containers. Beside, Kubernetes also provides services like automatically restarting the dead analytics containers or updating analytics containers without termination.

Chapter 3

Research Problem

In this chapter, we first present the considered problem in Section 3.1. We then decompose it into two subproblems in Section 3.2.

3.1 Problem Statement

Both the data center servers and gateways are capable of executing IoT analytics containers to turn *raw sensor data* into (more condensed) *information*, or *processed sensor data*, as illustrated in Figure 1.1. We use B_a and B_d to denote the upload and download bandwidth of the access link reserved for IoT analytics. We let A be the number of analytics containers and L be the number of layers. We use an $A \times L$ boolean matrix $\mathbf{M}$ to map layers to containers, i.e., $m_{a,l} = 1$ iff l ($l = 1, 2, \dots, L$) is part of a ($a = 1, 2, \dots, A$). The size of the image layer l is represented by u_l. Therefore, the size of a is $\sum_{l=1}^{L} m_{a,l} u_l$. The decision variable $e_a = 1$ denotes deploying a on the gateway, and $e_a = 0$ denotes otherwise.

To execute a on a gateway, all its layers must be downloaded to the image pool of the gateway. The image pool has a limited size S. We use boolean value h_l ($l = 1, 2, \dots, L$) to denote whether layer l is already on the gateway, i.e., $h_l = 1$ iff l is in the image pool. We assume IoT analytics have *control knobs*, referred to as the *QoS knobs* to tradeoff their QoS and resource consumption. The control knobs are normalized in $[0, 1]$. Executing container a with QoS knob k_a on a data center server consumes $r_a(k_a)$ kbps upload bandwidth due to raw sensor data, while doing so on the gateway consumes $p_a(k_a)$ kbps upload bandwidth due to processed sensor data. Container a produces results with a QoS level of $q_a(k_a)$ under QoS knob k_a. We use $\check{k}_a$ and $\hat{k}_a$ to denote the minimal and maximal QoS knob of a, which are empirically derived. Table 3.1 summarizes the symbols used throughout the thesis. We write our main research problem as follows.

Problem 1. *Given A IoT analytics containers, where each container a has a weight w_a. Our problem is to determine: (i) the boolean variable e_a that indicates whether downloading/deploying container a to the gateway without overflowing the image pool size (S) and (ii) the target QoS knob k_a without overloading upload and download bandwidths (B_u and B_d). The goal is to maximize the overall weighted QoS level ($\sum_{a \in A} w_a q_a(k_a)$).*

3.2 Problem Decomposition

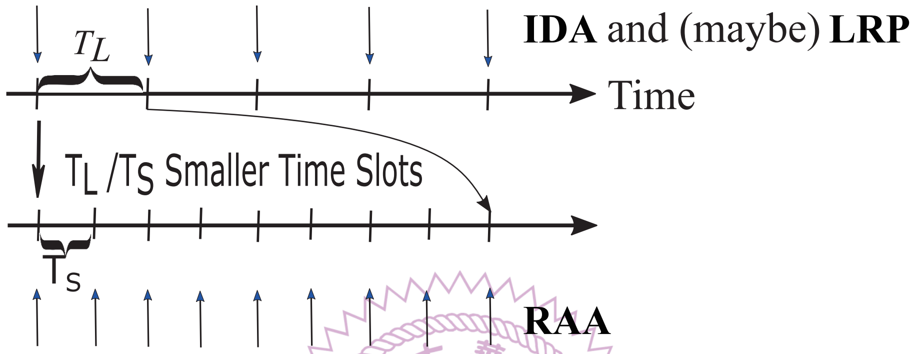


Figure 3.1: The execution time of the three subproblems (P1, P2, and P3).

Solving Problem 1 is not easy because of the different properties of the two decision variables: e_a and k_a. The decisions on k_a must be made frequently to cope with the dynamic environments. On the other hand, changing e_a may result in nontrivial download/storage overhead, and thus should be performed less frequently. On top of that, some image layers are available in the image pool, while others need to be downloaded. Therefore, we decompose Problem 1 into: (i) the *image download problem* (IDA) that determines e_a, (ii) the *rate allocation problem* (RAA) that determines k_a, and (iii) the *layer replacement policy* (LRP) that releases free space when the image pool is full. Figure 3.1 presents the executions timeline of the three subproblems. The image download problem is solved once every T_L second. It picks a few IoT analytics containers to deploy to the gateway, while the missing layers must be downloaded within T_L second (finished before the next execution time). The rate allocation problem is solved more often: once every T_S seconds, where $T_S \leq T_L$. It sets the optimal QoS knobs of individual IoT analytics containers without overloading the upload bandwidth. We assume T_L is a multiple of T_s for simplicity. Unlike the image download problem and the rate allocation problem solved periodically, the layer replacement policy checks the usage of the image pool before the image download problem. It runs only when the usage is over the threshold.

Table 3.1: Symbols Used in This Thesis

Symbol	**Description**
A	Number of IoT analytics
L	Total number of layers in the whole system
S	Total image pool space of gateways
$M_{A\times L}$	Indicator of image layer l is in analytic a
h_l	Indicator of image layer l is on gateway
u_l	Size of image layer l
T_L	Image Download Algorithm time slot duration
T_S	Rate Allocation Algorithm time slot duration
B_u	Total uplink bandwidth
B_d	Total downlink bandwidth
$\mathbf{A}_C$	The set of analytics running on data center servers
$\mathbf{A}_G$	The set of analytics running on gateways
$r_a(\cdot)$	Uplink bandwidth consumption of raw data for analytic a
$p_a(\cdot)$	Uplink bandwidth consumption of processed data from analytic a
e_a	Deploying analytic a on gateways
ϵ	Approximation parameter of $(1-\epsilon)$-approximation algorithm
$q_a(\cdot)$	QoS level of analytic a
w_a	Weight of analytic a
k_a	QoS knob of analytic a
$\hat{k}_a$	Maximum QoS knob to run analytics a
$\check{k}_a$	Minimum QoS knob to run analytics a
α	Step size of the proposed rate allocation algorithm
W_h	High-water mark of the storage usage in gateway
W_l	Low-water mark of the storage usage in gateway

Chapter 4

System Architecture

This chapter presents our system architecture. Figure 4.1 gives an overview of our proposed system. We describe the key system components below.

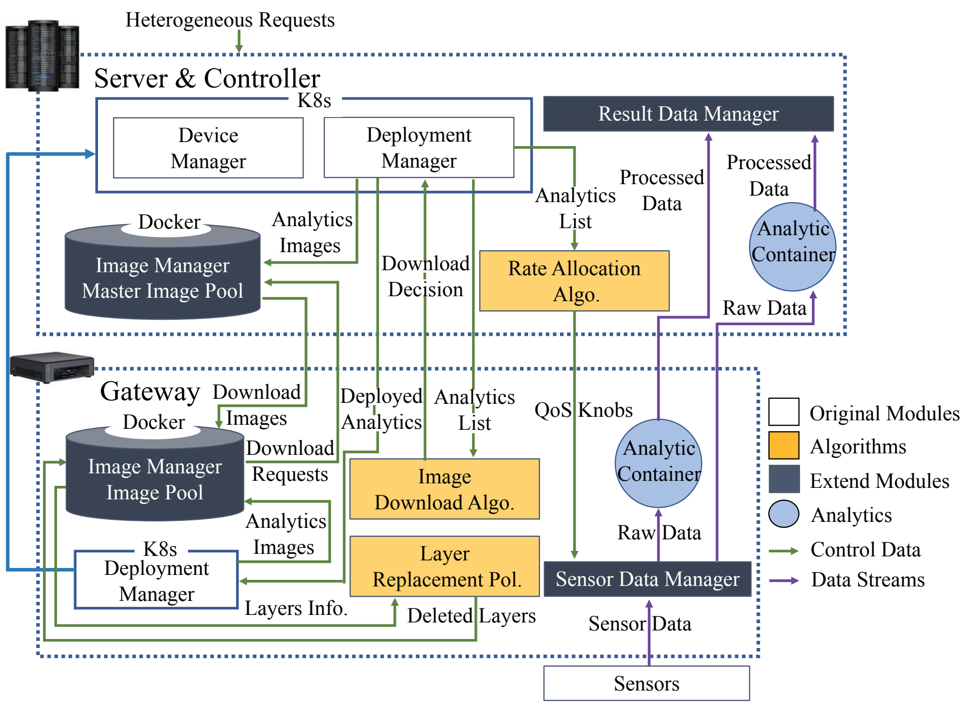


Figure 4.1: The architecture of our proposed system.

4.1 Server and Controller

- *Device manager* keeps track of the device status, such as resource utilization and network condition.

- *Deployment manager* keeps track of IoT analytics status and supports tasks like launching an IoT analytics container. When deploying an IoT analytic, the deployment manager will first check whether the analytics image, including all the needed layers, is on the master image pool or not, then run the IoT analytics container.

- *Image manager and master image pool* reside on the server. The image manager supports tasks like providing images and layers to the image manager on gateways. The master image pool stores all images and layers.

- *Rate allocation algorithm* determines the rate allocation among IoT analytics by QoS knobs for each IoT analytic to maximize the overall QoS. The rate allocation algorithm then sends the QoS knobs to the sensor data manager on gateways to adjust sensor data.

- *Result data manager* collects the processed data of IoT analytics both on the server and gateways.

- *Analytics container* is the IoT analytic deployed by the deployment manager. Analytics containers first take raw data from the sensor data manager on gateways, process the raw data, and then send the proposed data to the result data manager.

4.2 Gateway

- *Image Manager and Image pool* reside on gateways. The image manager on the gateway supports tasks such as downloading IoT analytics images and layers from the master image pool on the server or deleting IoT analytics images and layers in the image pool. The image pool on the gateway stores a subset of all IoT analytics layers in the master image pool on the controller.

- *Deployment manager* supports accepting the deployment tasks from the deployment manager on the controller and launching IoT analytics containers. When deploying an IoT analytic, the deployment manager will first check if the IoT analytics image, including all the needed IoT analytics layers, is on the image pool. If not, the image manager will download the missing IoT analytics layers from the master image pool on the server.

- *Image download algorithm* determines which IoT analytics containers to deploy to the gateway in the upcoming T_L seconds. The download decision is next sent to the deployment manager on the controller.

- *Layer replacement policy* chooses the victim IoT analytics layers to be deleted when the usage of the image pool on the gateway is over the high-water mark. Layer replacement policy then requests the image manager to delete the victim IoT analytics layers.

- *Analytics container* is the IoT analytic deployed by the deployment manager. Analytics containers first take raw data from the sensor data manager, process the raw data, and then send the proposed data to the result data manager on the server.

- *Sensor data manager* receives sensor data from sensors and downsamples the sensor data as the raw data according to the QoS knobs from the rate allocation algorithm. The sensor data manager then sends the raw data to IoT analytics containers.

- *Sensors* detect and collect data from outside and send sensor data to the sensor data manager on gateways.

Chapter 5

Image Download Problem and Algorithms

This chapter formulates and solves the image download problem. Table 5.1 summarizes the symbols of the image download problem and algorithms.

Table 5.1: Symbols of Image Download Problem and Algorithms

Symbol	Description
A	Number of IoT analytics
L	Total number of layers in the whole system
S	Total image pool space of gateways
$M_{A\times L}$	Indicator of image layer l is in analytic a
h_l	Indicator of image layer l is on gateway
u_l	Size of image layer l
T_L	Image Download Algorithm time slot duration
B_d	Total downlink bandwidth
$r_a(\cdot)$	Uplink bandwidth consumption of raw data for analytic a
$p_a(\cdot)$	Uplink bandwidth consumption of processed data from analytic a
k_a	QoS knob of analytic a
e_a	Deploying analytic a on gateways
$s_{A\times L}$	Downloaded layer size of image layer l in analytic a
z_a	Saved upload bandwidth of analytic a
R	Remaining resource
ϵ	Approximation parameter of $(1-\epsilon)$-approximation algorithm

5.1 Problem Formulation

Problem 2. *Let k_a be the default QoS knob of analytics container a from all A containers that can be deployed to the gateway. Determine whether each a should be deployed at the gateway to maximize the saved upload traffic without exceeding the image pool size S.*

Lemma 1 (Hardness)**.** *The image download problem is NP-hard.*

Proof. We show this by reducing the *m-dimensional knapsack problem (m-DKP)* [29, Ch. 9] to our problem as follows. The m-DKP is an NP-hard problem where we pack some objects from all objects in a bag in order to maximize the total value of the selected objects subject to m resource constraints, or say capacities. Let $m = 2$, and we first map the two resource capacities to the two resource constraints of our problem: S and $B_d T_L$. For each object o with a value z_o, we create an IoT analytic a with $r_a(k_a) = z_o$ and $p_a(k_a) = 0$. We also set total layer size $\sum_{l=1}^{L} m_{a,l} u_l$ and downloaded layer size$\sum_{l=1}^{L} \max(m_{a,l} - h_l, 0) u_l$ to the two dimensions of the 2-DKP problem. The objective function of 2-DKP is to maximize the sum of the chosen objects' z_o. We set the objective function of our problem as maximizing the sum of the saved bandwidth $r_a(k_a) - p_a(k_a)$ of the deployed analytics, which is equivalent to minimizing the resulting uplink traffic. Lastly, by setting S and $B_d T_L$ and because of the 2-DKP resource capacities, we reduce the problem in polynomial time. Thus, we have proved the hardness. ☐

To solve NP-hard image download problem, we write Problem 2 into the following Integer Linear Programming (ILP) formulation:

$$\max \sum_{a=1}^{A} [r_a(k_a) - p_a(k_a)] e_a \tag{5.1a}$$

$$s.t. \sum_{a=1}^{A} e_a \sum_{l=1}^{L} m_{a,l} u_l \leq S; \tag{5.1b}$$

$$\sum_{a=1}^{A} e_a \sum_{l=1}^{L} m_{a,l} (1 - h_l) u_l \leq B_d T_L; \tag{5.1c}$$

$$e_a \in \{0, 1\} \ \forall a = 1, 2, \ldots, A.$$

Eq. (5.1a) accounts for the saved bandwidth, which is the difference between the raw and processed data amounts. Eqs. (5.1b) and (5.1c) are the constraints on the image pool size and download network bandwidth. It is not hard to see that Eqs. (5.1b) and (5.1c) can be merged into: $\sum_{a=1}^{A} e_a \sum_{l=1}^{L} m_{a,l}(1 - h_l) u_l \leq \min\{S - \sum_{l=1}^{L} h_l u_l, B_d T_L\}$, which reduces the 2-DKP problem in Eq. (5.1) into a 1-DKP problem.

5.2 Dynamic Programming Algorithm

Algorithm 1 Dynamic Programming Algorithm (IDA$_\mathrm{D}$)

```
Input: downloaded layer size $s = \{s_{1,1}, s_{1,2}, \dots, s_{1,L}, s_{2,1}, \dots, s_{A,L}\}$, saved upload
    bandwidth $z = \{z_1, z_2, \dots, z_A\}$, remaining resource $R$, selected container $a$.
Output: total saved upload bandwidth $t^\star$, deployed containers $D^\star$.
1: if $R \leq 0$ or $n \leq 0$ then //resource is used up or all the containers are checked
2:     $t^\star = 0$, $D^\star = \{\}$
3: else if $\sum_{l=1}^{L} s_{a,l} > R$ then //remaining resource is not enough to deploy $a$
4:     //return the total saved upload bandwidth and deploy containers from containers
           $1, 2, \dots, a-1$ with remaining resource $R$
5:     $t^\star, D^\star = \mathrm{IDA_D}(s, z, R, a-1)$
6: else
7:     //find the total saved upload bandwidth and deploy containers from containers
           $1, 2, \dots, a-1$ with remaining resource $R$
8:     $t_n, D_n = \mathrm{IDA_D}(s, z, R, a-1)$
9:     //find the total saved upload bandwidth and deploy containers from containers
           $1, 2, \dots, a-1$ with remaining resource $R - \sum_{l=1}^{L} s_{a,l}, a-1$
10:    $t_i, D_i = \mathrm{IDA_D}(s, z, R - \sum_{l=1}^{L} s_{a,l}, a-1)$
11:    if $t_i \leq t_n$ then
12:        $t^\star = t_n, D^\star = D_n$
13:    else
14:        $t^\star = t_i + z_a, D^\star = D_i \cup a$
15: return $t^\star, D^\star$
```

We first develop an optimal algorithm using dynamic programming [29, Ch. 2], which is called the IDA$_\mathrm{D}$ algorithm. We present the pseudocode of IDA$_\mathrm{D}$ in Algorithm 1. We start our algorithm by setting downloaded layer size $s = \{s_{a,l}\} = \{m_{a,l}(1-h_l)u_l\}$ for all $l = 1, 2, \dots, L$, $a = 1, 2, \dots, A$, saved upload bandwidth $z = \{z_a\} = \{r_a(k_a) - p_a(k_a)\}$ for all $a = 1, 2, \dots, A$, remaining resource $R = \min\{S - \sum_{l=1}^{L} h_l u_l, B_d T_L\}$, selected container $a = A$. IDA$_\mathrm{D}$ recursively finds the total saved upload bandwidth and deployed containers with remaining resource R. In each recursion, if the remaining resource is sufficient to deploy the container a, and the total saved upload bandwidth with a is larger than that without a, IDA$_\mathrm{D}$ returns the total saved upload bandwidth and deployed containers with a. Otherwise, IDA$_\mathrm{D}$ returns the total saved upload bandwidth and deployed containers without a. The time complexity of IDA$_\mathrm{D}$ is pseudo-polynomial. Moreover, IDA$_\mathrm{D}$ gives us the optimal solution $D^\star$ and optimal value $t^\star$. We then let $e_a = 1$ for each container a in the $D^\star$. Otherwise, $e_a = 0$.

5.3 $(1-\epsilon)$-Approximation Algorithm

Algorithm 2 $(1-\epsilon)$-Approximation Algorithm (IDA$_\text{A}$)

Input: downloaded layer size $s = \{s_{1,1}, s_{1,2}, \ldots, s_{1,L}, s_{2,1}, \ldots, s_{A,L}\}$, saved upload bandwidth $z = \{z_1, z_2, \ldots, z_A\}$, remaining resource R. approximation parameter ϵ.

Output: total saved upload bandwidth t', deployed containers D'.

```
1: initialize: $t' = 0$, $z' = \{z'_a\}$ for all $a = 1, 2, \ldots, A$
2: $K = \epsilon \frac{\max_{a\in[1,A]}\{z_a\}}{A}$
3: for $a = 1; a \le A; a++$ do
4:     $z'_a = \lfloor \frac{z_a}{K} \rfloor$ //give the bound of $z_a$ by rounding it by $K$
5: //run IDA$_\text{D}$ with the new saved upload bandwidth $z' = \{z'_1, z'_2, \ldots, z'_A\}$
6: $t, D' =$ IDA$_\text{D}$($s$, $z'$, $R$, $A$)
7: for $a \in D'$ do
8:     $t' = t' + z_a$
9: return $t', D'$
```

While IDA$_\text{D}$ gives optimal solutions, it suffers from the pseudo-polynomial running time. Therefore, we next develop an $(1-\epsilon)$-approximation algorithm with Fully Polynomial Time Approximation Scheme (FPTAS) [29, Ch. 2], which is called IDA$_\text{A}$. Algorithm 2 gives the pseudocode of IDA$_\text{A}$. In the beginning, we set the downloaded layer size s, saved upload bandwidth z, and remaining resource R same to those of Algorithm 1. Moreover, we let ϵ be the approximation parameter. IDA$_\text{A}$ calculates the rounding denominator $K = \epsilon \frac{\max_{a\in[1,A]}\{z_a\}}{A}$. IDA$_\text{A}$ then takes the floor of the saved upload bandwidth z_a divided by the rounding denominator K as the new saved upload bandwidth z'_a for each container a. Finally, we run IDA$_\text{D}$ with the new saved upload bandwidth z' and get the total saved upload bandwidth t' and deployed containers D'. We then let $e_a = 1$ for each container a in the D'. Otherwise, $e_a = 0$.

Rounding the saved upload bandwidth z_a of each container a makes $z_a = iK$ where $i \ge 0$ is an integer, which thus gives a bound of the total saved upload bandwidth t. The total saved upload bandwidth $t \le A\frac{\max_{a\in[1,A]}\{z_a\}}{K} = \frac{A^2}{\epsilon}$. For a given container a, IDA$_\text{D}$ compares the total saved upload bandwidth at most $\frac{A^2}{\epsilon}$ times, and IDA$_\text{D}$ checks A containers. Hence, IDA$_\text{D}$ has a polynomial running time of $O(A\frac{A^2}{\epsilon}) = O(\frac{A^3}{\epsilon})$.

Since $\sum_{a\in D^\star} z_a - \sum_{a\in D'} z'_a \le \sum_{a\in D^\star} z_a - \sum_{a\in D^\star} z'_a \le AK$, we can derive $t' = \sum_{a\in D'} z'_a \ge \sum_{a\in D^\star} z_a - AK = t^\star - \epsilon \max_{a\in[1,A]}\{z_a\} \ge t^\star - \epsilon t^\star = (1-\epsilon)t^\star$. Therefore, IDA$_\text{D}$ has an approximation factor $(1-\epsilon)$.

5.4 Greedy Algorithm

Algorithm 3 Greedy Algorithm (IDA$_\mathrm{G}$)

```
Input: downloaded layer size $s = \{s_{1,1}, s_{1,2}, \ldots, s_{1,L}, s_{2,1}, \ldots, s_{A,L}\}$, saved upload
    bandwidth $z = \{z_1, z_2, \ldots, z_A\}$, remaining resource $R$.
Output: total saved upload bandwidth $t$, deployed containers $D$.
 1: initialize: $t = 0$, $D = \{\}$
 2: for $a = 1; a \leq A; a++$ do
 3:     //saved upload bandwidth normalized to the consumed download bandwidth of $a$
 4:     $n_a = \left\lceil z_a \right\rceil / \left\lceil \frac{\sum_{l=1}^{L} s_{a,l}}{R} \right\rceil$,
 5: Sort the containers by $n_a$ in the descending order.
 6: for each container $a$ in the descending order do
 7:     if $R \leq 0$ then //remaining resource is used up
 8:         break
 9:     else if $\sum_{l=1}^{L} s_{a,l} \leq R$ then //remaining resource is enough to deploy $a$
10:         $D = D \cup a$
11:         $t = t + z_a$
12:         $R = R - \sum_{l=1}^{L} s_{a,l}$
13: return $t$, $D$
```

To solve the problem more efficiently, we finally propose a greedy algorithm IDA$_\mathrm{G}$. The pseudocode of IDA$_\mathrm{G}$ is shown in Algorithm 3. We start by setting downloaded layer size s, saved upload bandwidth z, and remaining resource R same to those of Algorithm 1 and Algorithm 2. IDA$_\mathrm{G}$ initializes the total saved upload bandwidth $t = 0$ and deployed containers $D = \{\}$ in line 1. Between line 2 and line 4, IDA$_\mathrm{G}$ computes n_a, representing the saved upload bandwidth normalized to the consumed download bandwidth of a. IDA$_\mathrm{G}$ then sorts the containers by n_a in the descending order in line 5. From line 6 to line 12, IDA$_\mathrm{G}$ repeatedly picks analytic a from the sorted list. If the residual resources are sufficient to deploy a on the gateway, IDA$_\mathrm{G}$ chooses the container a as one of deployed containers D and increases the total saved upload bandwidth t by the saved upload bandwidth z_a. The loop stops when all the gateway resources are used up, or when all the containers are checked. In the end, IDA$_\mathrm{G}$ returns the total saved upload bandwidth t and deployed containers D in line 13. We then let $e_a = 1$ for each container a in the D. Otherwise, $e_a = 0$.

IDA$_\mathrm{G}$ has a polynomial running time of $O(A \log A)$ due to the sorting of containers in line 5. Since a container image will be downloaded only once, we combine the containers with the same container image into a single container before running IDA$_\mathrm{D}$, IDA$_\mathrm{A}$, and IDA$_\mathrm{G}$.

Chapter 6

Rate Allocation Problem and Algorithms

This chapter formulates and solves the rate allocation problem. Table 6.1 summarizes the symbols of the rate allocation problem and algorithm.

Table 6.1: Symbols of Rate Allocation Problem and Algorithm

Symbol	Description
B_u	Total uplink bandwidth
$\mathbf{A}_C$	The set of analytics running on data center servers
$\mathbf{A}_G$	The set of analytics running on gateways
$r_a(\cdot)$	Uplink bandwidth consumption of raw data for analytic a
$p_a(\cdot)$	Uplink bandwidth consumption of processed data from analytic a
w_a	Weight of analytic a
$q_a(\cdot)$	QoS level of analytic a
k_a	QoS knob of analytic a
$\hat{k}_a$	Maximum QoS knob to run analytics a
$\check{k}_a$	Minimum QoS knob to run analytics a
α	Step size of the proposed rate allocation algorithm

6.1 QoS and Bandwidth Models of IoT Analytics

Different IoT analytics have different QoS metrics. For concrete discussion, we consider two sample IoT analytics: object detector and sound classifier[1], and employ *accuracy* as their QoS metric. These two analytics can trade QoS levels (lower accuracy) for bandwidth consumption (lower bitrate) through adjusting an analytic-dependent QoS knob k_a. We define the accuracy as the fraction of correct classifications. For the object detector, we employ an object detection network [14] to test 300 images from 20 classes [13]. The average image resolution is 390×467 originally, and the downsampled image resolution is determined by k_a. For the sound classifier, we adopt a sound classification network [2] to test 8732 audio files from 10 classes [12]. The original sampling rate is 22 kHz, and the downsampled sampling rate is determined by k_a.

We measure: (i) accuracy, (ii) raw data size, and (iii) processed data size for different k_a values in $[\check{k}_a, \hat{k}_a]$. Note that the sound classifier stops working when $k_a < 0.4$ (8 kHz), and thus we define $\check{k}_a = 0.4$, $\hat{k}_a = 1$ for the sound classifier. We also define $\check{k}_a = 0$, $\hat{k}_a = 1$ for the object detector. We observe that the QoS values of the object detector and the sound classifier can be respectively captured by:

$$q_a^o(k_a) = p_{a,1} e^{p_{a,2} k_a} + p_{a,3}; \tag{6.1}$$

$$q_a^s(k_a) = p_{a,1} \ln(p_{a,2} k_a) + p_{a,3}, \tag{6.2}$$

where $p_{a,1}$–$p_{a,3}$ are model parameters. Moreover, both the raw and processed data bandwidths can be captured by:

$$r_a(k_a) = p_{a,4} k_a + p_{a,5}; \tag{6.3}$$

$$p_a(k_a) = p_{a,6} k_a + p_{a,7}, \tag{6.4}$$

where $p_{a,4}$–$p_{a,7}$ are model parameters. We report the sample results and models in Figure 6.1.

When deriving the model parameters, we ensure that the function is monotonically increasing in $[\check{k}_a, \hat{k}_a]$. In particular, we consider degree-1 polynomial, exponential function, and logarithm. We compute the adjusted R^2 of the three functions for each model and select the function with the maximal adjusted R^2 value. Table 6.2 gives the parameters of the resulting models and adjusted R^2 values.

[1]These analytics are merely samples, our proposed approach is applicable to other analytics, including those developed in the future.

Table 6.2: QoS and Bandwidth Model Parameters of Sample Analytics

Analytics	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$	**Adj. R^2**	$p_{a,4}$	$p_{a,5}$	**Adj. R^2**	$p_{a,6}$	$p_{a,7}$	**Adj. R^2**
Sound Classifier	0.01	3.99	0.16	0.9630	687.56	1.01	1	0	0.05	Undef.
Object Recognizer	0.16	494	0	0.9828	52.52	5.45	0.9974	30.01	4.17	0.9956

6.2 Problem Formulation

For the sake of presentation, we partition all IoT analytics containers into two sets: $\mathbf{A}_C$ and $\mathbf{A}_G$ for containers running on data center servers and gateways, respectively.

Problem 3. *Given the QoS models $q_a(\cdot)$, the bandwidth models $r_a(\cdot)$ and $p_a(\cdot)$ for all analytics container a, find the best QoS knobs $k_a, \forall a \in \mathbf{A}_C \cup \mathbf{A}_G$, to maximize the weighted QoS value without exceeding the upload bandwidth B_u.*

The problem can be mathematically written as:

$$\max \sum_{a \in \mathbf{A}_C \cup \mathbf{A}_G} w_a q_a(k_a) \tag{6.5a}$$

$$s.t. \sum_{a \in \mathbf{A}_C} r_a(k_a) + \sum_{a \in \mathbf{A}_G} p_a(k_a) \leq B_u; \tag{6.5b}$$

$$\check{k}_a \leq k_a \leq \hat{k}_a \; \forall a \in \mathbf{A}_C \cup \mathbf{A}_G. \tag{6.5c}$$

The objective function in Eq. (6.5a) follows that of Problem 1. The constraints in Eq. (6.5b) ensure that the upload bandwidth is not exceeded.

6.3 Rate Allocation Algorithm

The algorithms for solving Problem 3 heavily depend on the properties of the model functions. Our empirical models in Eq. 6.1–6.4 are monotonically increasing on k_a. This *should* be valid if the control knob is well-defined in $[\check{k}_a, \hat{k}_a]$: higher bandwidth consumption *should* lead to higher accuracy.

We propose a greedy rate allocation algorithm with a step size α, which is called RAA. We give the pseudocode of RAA in Algorithm 4. In line 1, RAA initializes $\mathbf{A}'_C$ and $\mathbf{A}'_G$ by $\mathbf{A}_C$ and $\mathbf{A}_G$. Also, RAA initializes the QoS knob k_a by $\check{k}_a$ for each container a in the cloud servers and gateways. From line 2 to line 5, RAA computes $g(k_a)$, which is the ratio of weighted QoS value and bandwidth according to where the container is deployed. In the while loop between line 6 and line 10, RAA repeatedly selects the container a with the maximal $g(k_a)$. RAA then increases the QoS knob k_a by the step size α. If the QoS knob k_a reaches the maximal QoS knob $\hat{k}_a$, RAA removes the container a from $\mathbf{A}'_C \cup \mathbf{A}'_G$. The while loop stops when the upload bandwidth B_u is used up, or when the QoS knobs of all the containers reach $\hat{k}_a$, i.e., $\mathbf{A}'_C \cup \mathbf{A}'_G$ is empty. In the end, RAA returns the QoS knob k_a for all a in the cloud servers and gateways.

Algorithm 4 Rate Allocation Algorithm (RAA)

Input: Weight w_a, minimal QoS knob $\check{k}_a$, maximal QoS knob $\hat{k}_a$, QoS model $q_a(\cdot)$, raw bandwidth models $r_a(\cdot)$, processed bandwidth models $p_a(\cdot)$ $\forall a \in \mathbf{A}_C \cup \mathbf{A}_G$, upload bandwidth B_u, step size α.

Output: Optimal QoS knobs decision k_a $\forall a \in \mathbf{A}_C \cup \mathbf{A}_G$.

```
 1: initialize: $\mathbf{A}'_C = \mathbf{A}_C$, $\mathbf{A}'_G = \mathbf{A}_G$, $k_a = \check{k}_a$ $\forall a \in \mathbf{A}_C \cup \mathbf{A}_G$
 2: if $a \in \mathbf{A}_C$ then
 3:     Let $g(k_a) = w_a q_a(k_a)/r_a(k_a)$
 4: else //$a \in \mathbf{A}_G$
 5:     Let $g(k_a) = w_a q_a(k_a)/p_a(k_a)$
 6: while $0 < B_u$ and $\mathbf{A}'_C \cup \mathbf{A}'_G \neq \varnothing$ do
 7:     find the container $a$ with the maximal $g(k_a)$ $\forall a \in \mathbf{A}'_C \cup \mathbf{A}'_G$
 8:     $k_a = k_a + \alpha$.
 9:     if $k_a \geq \hat{k}_a$ then
10:         remove $a$ from $\mathbf{A}'_C \cup \mathbf{A}'_G$
11: return $k_a$ $\forall a \in \mathbf{A}_C \cup \mathbf{A}_G$
```

6.4 Analysis

Finding the maximal $g(k_a)$ has a complexity of $O(1)$ if the containers are stored in a heap. We repeatedly find the maximal $g(k_a)$ at most $\frac{1}{\alpha}|\mathbf{A}_C \cup \mathbf{A}_G|$ times. Hence, the greedy algorithm has a polynomial running time of $O(\frac{1}{\alpha}|\mathbf{A}_C \cup \mathbf{A}_G|)$, which is linear to A for typical α values.

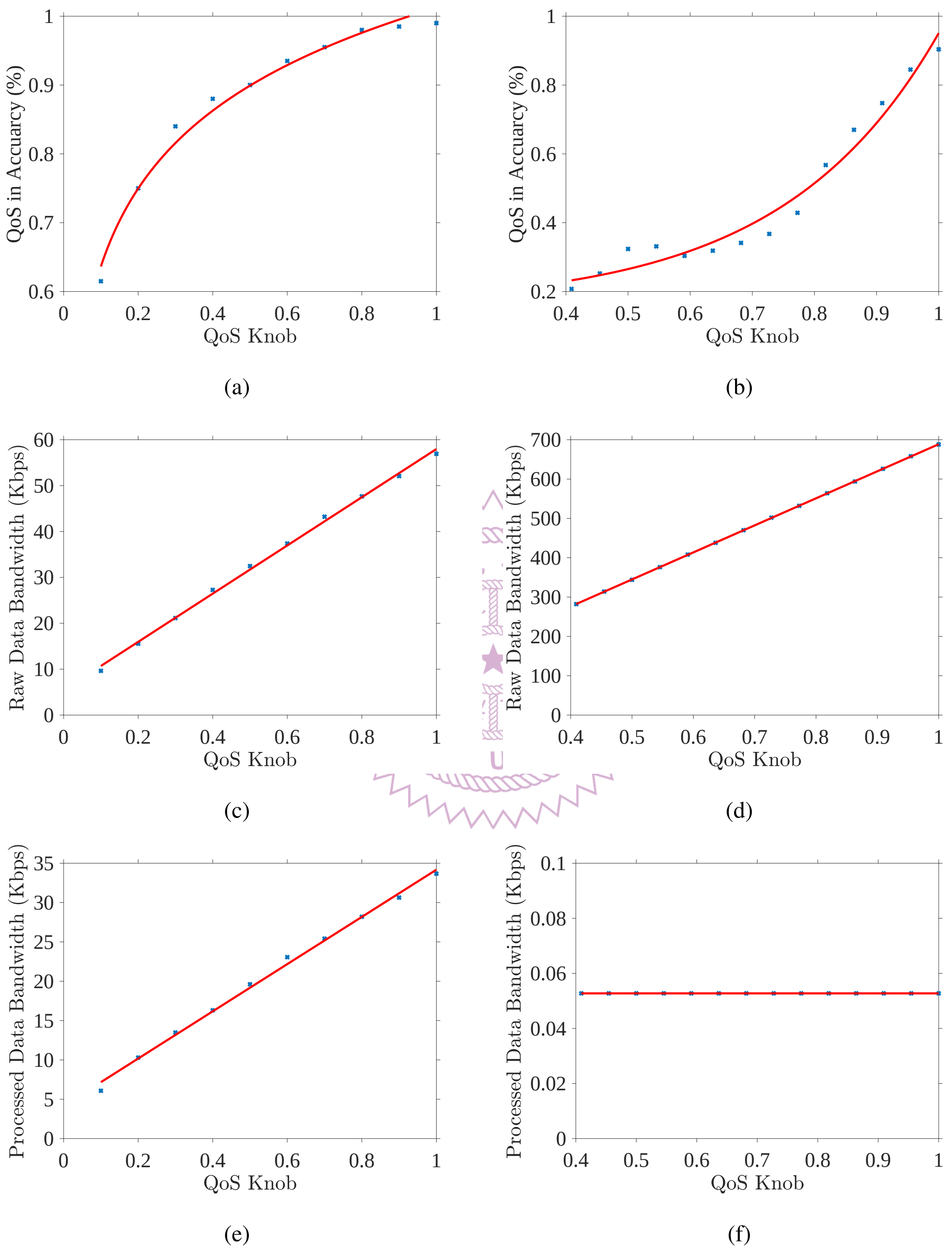


Figure 6.1: Sample models from the object recognizer: (a), (c), (e); and the sound classifier: (b), (d), (f). (a) and (b) are the QoS models; (c) and (d) are the raw data bandwidth models; and (e) and (f) are the processed data bandwidth models.

Chapter 7

Layer Replacement Policies

Each analytics container consists of a set of image layers with a linear dependency among them. That is, a higher layer depends on all the layers beneath it. Layers are the units of those downloaded and cached, while an analytics container cannot be launched if not all its image layers are already saved/cached in the local image pool. Typically, the highest layer encapsulates the application, which is the IoT analytics in our scenarios. The lower layers provide utilities and libraries. While the higher layers of different analytics containers are quite different, e.g., with different neural networks, their lower layers may be common, e.g., the same Linux utilities and C libraries. Figure 7.1 shows the layers of two sample analytics containers, where some lower layers are *shared*, which are good candidates to be cached in the image pools to reduce the download bandwidth consumption. Notice that the lower layers of different analytics containers may encapsulate different *versions* of the *same* library. These layers are essentially *different* and cannot be shared.

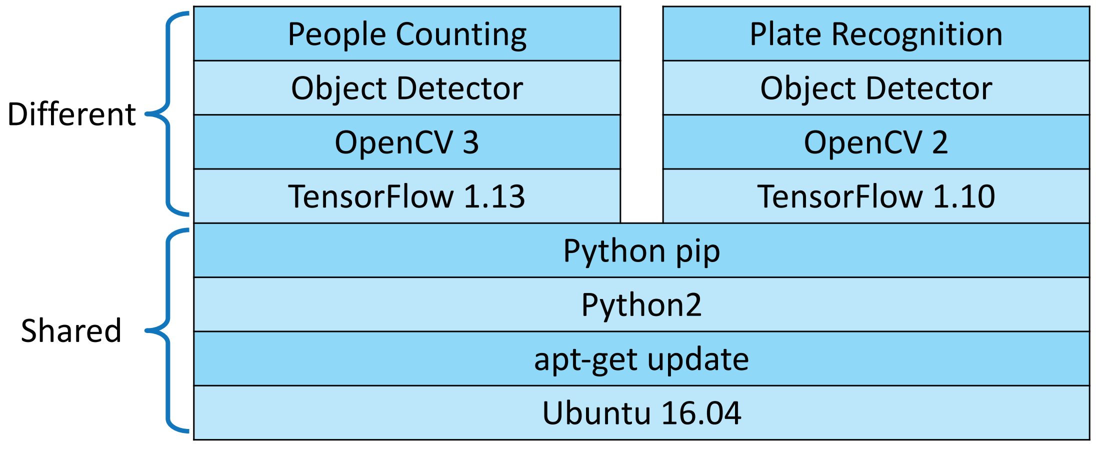


Figure 7.1: Illustrations of container image layers of two sample IoT analytics.

The symbol table of layer replacement policies is shown as Table 7.1. Before deciding what IoT analytics should be deployed on the gateway, the system checks the

Table 7.1: Symbols of Layer Replacement Policies

Symbol	**Description**
L	Total number of layers in the whole system
S	Total image pool space of gateways
h_l	Indicator of image layer l is on gateway
u_l	Size of image layer l
W_h	High-water mark of the storage usege in gateway
W_l	Low-water mark of the storage usege in gateway

gateway's storage occupation. To decide whether Layer Replacement Policy (LRP) runs or not and how much it removes if it runs, we employ *watermarking* policy [28, Ch. 8]. Precisely, the layer replacement policy starts deleting victim layers when the storage usage reaches high-water mark (W_h), i.e., $\sum_{l=1}^{L} h_l u_l \geq W_h S$, the layer replacement policy chooses a victim layer l' and removes it ($h_{l'} = 0$). Next, the layer replacement policy checks if the storage usage is lower than or equal to the low-water mark (W_l), i.e., $W_l S \geq (\sum_{l=1}^{L} h_l u_l) - u_{l'}$. If not, the layer replacement policy keeps doing the above removing process until the storage usage is down to the low-water mark W_l, i.e., $W_l S \geq \sum_{l=1}^{L} h_l u_l$.

We consider the following classic cache replacement policies [40, Ch. 4] to identify a *victim* layer:

- *Least-Recently-Used (LRU):* The layer of the least recently used image is selected.
- *Most-Recently-Used (MFU):* The layer of the most recently used image is selected.
- *Least-Frequently-Used (LFU):* The layer of the least frequently used image is selected.
- *Most-Frequently-Used (MFU):* The layer of the most frequently used image is selected.

The victim layer is picked from the top layer of the selected image. For example, the top layer of one of the sample analytics, people counting in Figure 7.1, has been chosen as the victim layer and deleted in the previous round. That is, the top layer, people counting, has been removed. If the sample analytic is chosen again in this round, the layer replacement policy will pick the second top layer, object detector, as the victim layer.

Note that the layer replacement policy only deletes the layer where no running analytic is using. For instance, we continue choosing victim layers and want to choose the layer, python-pip, as the next victim layer. Another sample analytics, plate recognition in Figure 7.1, is running on the gateway. Since plate recognition is already using this layer, we will choose another layer instead.

Chapter 8

Evaluations

We implement our proposed solutions on a real testbed. We then conduct extensive experiments to evaluate its performance.

8.1 Implementations

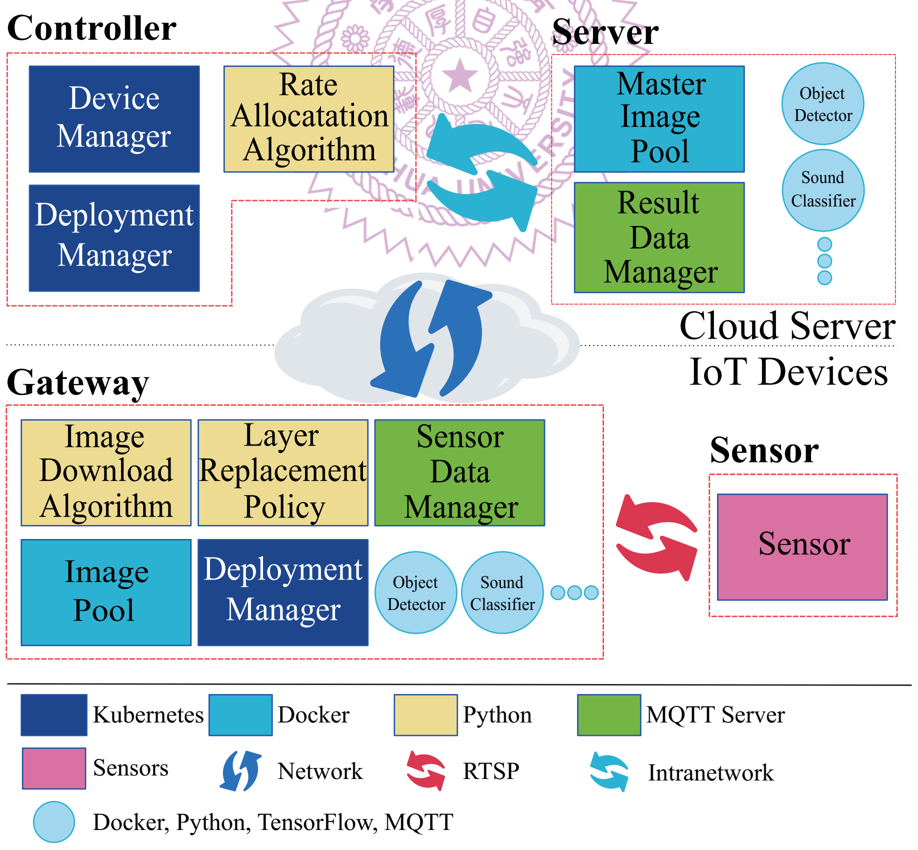


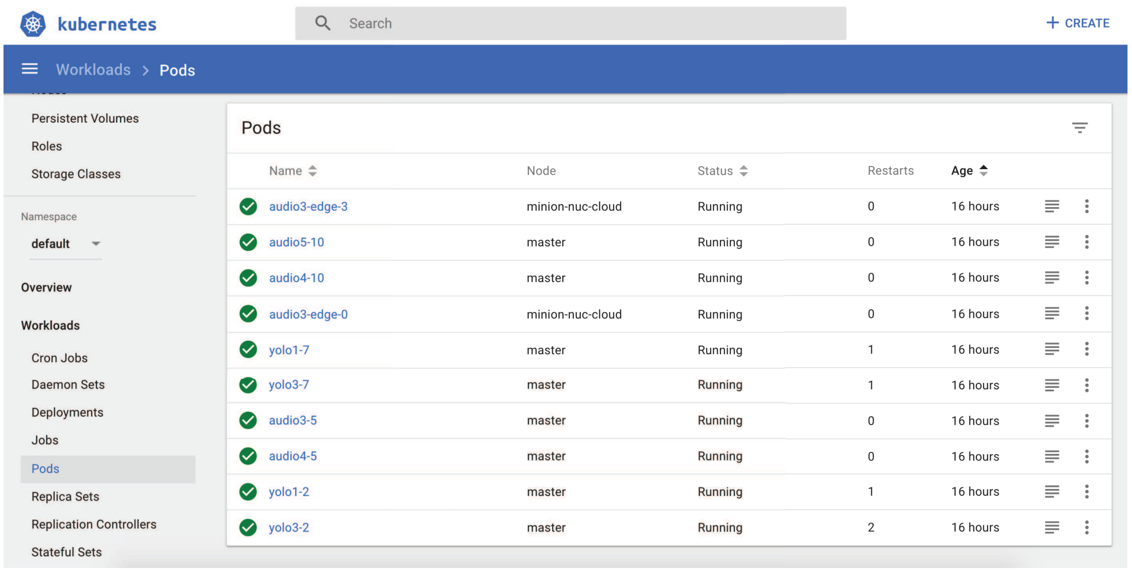
Figure 8.1: The proposed IoT testbed.

We have implemented our proposed system, as shown in Figure 8.1. The management platform is built on Kubernetes. The Kubernetes server runs on the controller, and the minions run on the data center server and gateway. The analytics containers are built on TensorFlow and Docker and executed on the server and gateway. Figure 8.2 gives the user an interface for monitoring the usage and analytics on our system. We have augmented Kubernetes, Docker, and TensorFlow to meet our needs. For example, the vanilla Docker does not support removing a single image layer from the image pool. We adopt Moby [9] to add and remove individual layers. We use a table to record the deleted layers and layer information. Nevertheless, our system needs to download the top layers when deleting the bottom layers, which takes additional time. In addition, we do not consider the extraction time when downloading image layers. Extraction consumes a lot of time, especially in the high download network bandwidth environment. To ignore deleting and extraction time, we add the deleting and extraction time to the end of each time slot. We have also added additional features. For example, we have developed Python scripts to capture sensor data streams, such as live video from cameras and audio from microphones, and save them into trace files. Another set of scripts replay these trace files using protocols like Real Time Streaming Protocol (RTSP) to mimic sensors in the field. By doing so, we can incur exactly the same workload on different algorithms for fair comparisons. We use Message Queuing Telemetry Transport (MQTT) protocol to exchange data among sensors and analytics containers.

Table 8.1: Sample IoT Analytics Containers

Container	Size (GB)	# of Layers	Container	Size (GB)	# of Layers
SC 1	0.72	19	**OD 1**	0.88	16
SC 2	0.81	20	**OD 2**	0.96	16
SC 3	1.27	19	**OD 3**	1.43	16
SC 4	1.35	20	**OD 4**	1.5	16
SC 5	0.81	19	**OD 5**	0.97	16
SC 6	0.9	20	**OD 6**	1.02	16
SC 7	1.35	19	**OD 7**	1.51	16
SC 8	1.44	20	**OD 8**	1.59	16
SC 9	0.81	21	**OD 9**	0.97	19
SC 10	1.16	22	**OD 10**	1.41	21
SC 11	1.67	23	**OD 11**	1.83	21
SC 12	1.93	24	**OD 12**	2.1	22

We have created 24 IoT analytics containers using : (i) the two sample analytics (object detector and sound classifier), (ii) different Ubuntu versions (16.04.5, 16.04.6, and 18.04.4), (iii) different Python versions (2 versus 3), and (iv) different TensorFlow versions (1.14.0 versus 1.15.0). Table 8.1 summaries the sample analytics containers. These containers are composed of 457 image layers (253 of them are unique). We give the sample result of the two sample analytics in Figure 8.3. We have implemented our proposed algorithms using Python. For the rate allocation problem, we have implemented two baseline algorithms: Unweighted Allocation (UA) and Weighted Allocation (WA) algorithms. The UA algorithm equally allocates bandwidth among analytics containers. The WA algorithm allocates bandwidth that is proportional to the weight of each analytic container. We have also implemented multiple layer replacement policies: (i) Least-Recently-Used (LRU), (ii) Most-Recently-Used (MRU), (iii) Least-Frequently-Used (LFU), and (iv) Most-Frequently-Used (MFU).



Pods

	Name	Node	Status	Restarts	Age
✓	audio3-edge-3	minion-nuc-cloud	Running	0	16 hours
✓	audio5-10	master	Running	0	16 hours
✓	audio4-10	master	Running	0	16 hours
✓	audio3-edge-0	minion-nuc-cloud	Running	0	16 hours
✓	yolo1-7	master	Running	1	16 hours
✓	yolo3-7	master	Running	1	16 hours
✓	audio3-5	master	Running	0	16 hours
✓	audio4-5	master	Running	0	16 hours
✓	yolo1-2	master	Running	1	16 hours
✓	yolo3-2	master	Running	2	16 hours

Figure 8.2: User interface for monitoring the usage and analytics on our system.

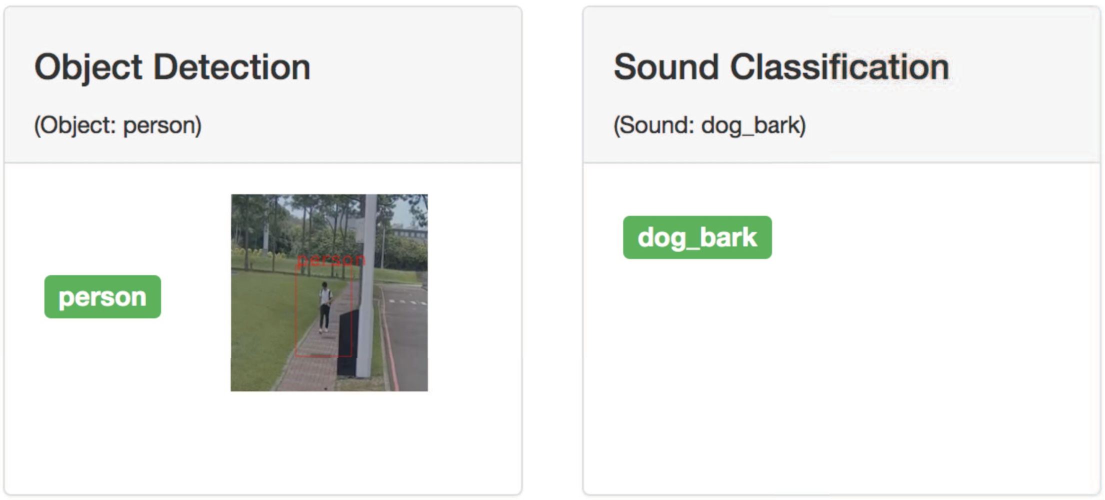


Figure 8.3: Sample results of: (a) object detector and (b) sound classifier.

8.2 Testbeds

We have deployed a campus testbed, as shown in Figure 8.4. The testbed contains eight street lamps, which are interconnected by Ethernet and WiFi mesh networks. Multiple sensors, including cameras, microphones, air-quality, and micro-weather sensors (not shown in the figure) are mounted on the lamps. Two lamps are equipped with gateways, which are essentially rugged PCs. The lamps are connected to the controller and a public master image pool hosted on DockerHub. The campus testbed allows us to validate the functionalities of the proposed IoT gateway and optimization algorithms.

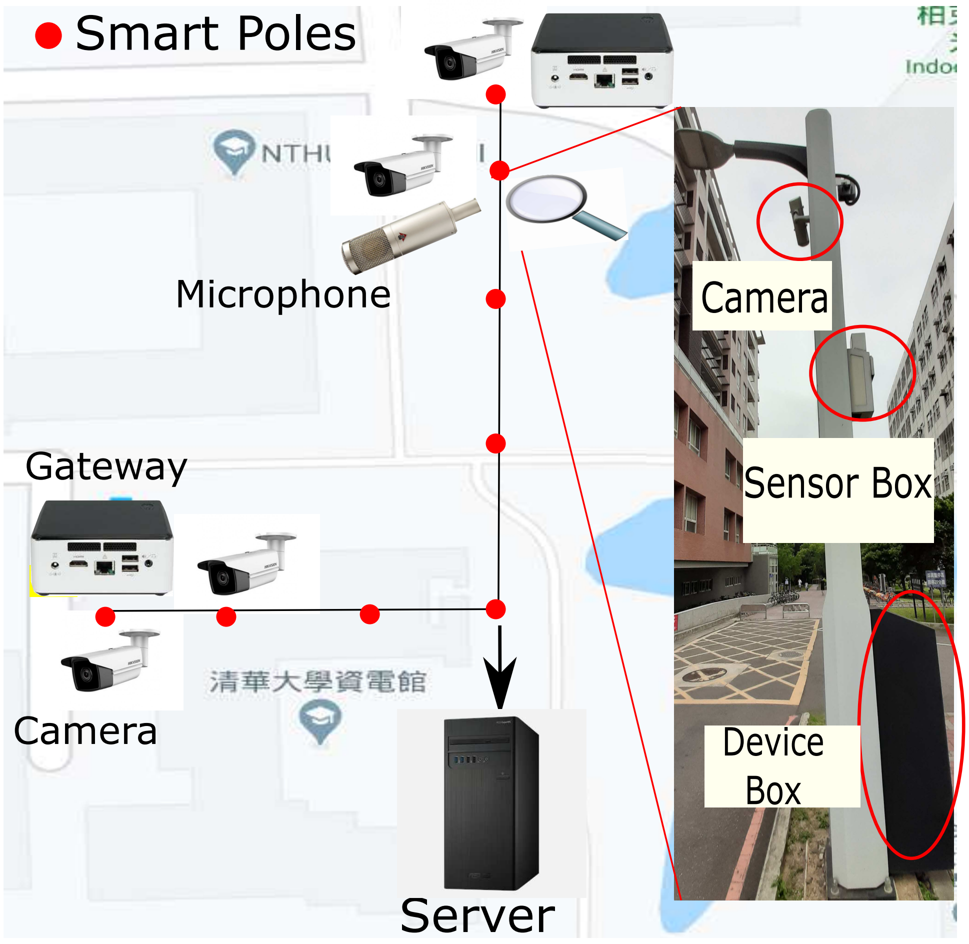


Figure 8.4: Our campus testbed with smart street lamps.

For exercising different parameters, we have also set up a lab testbed, as shown in Figure 8.5. Different from the campus testbed, we have set up our own master image pool and adopt the Linux TC tool to throttle the network bandwidth to mimic access links. We record video/audio streams from the campus testbed and replay them in the lab testbed to fairly compare the performance of different algorithms under diverse parameters.

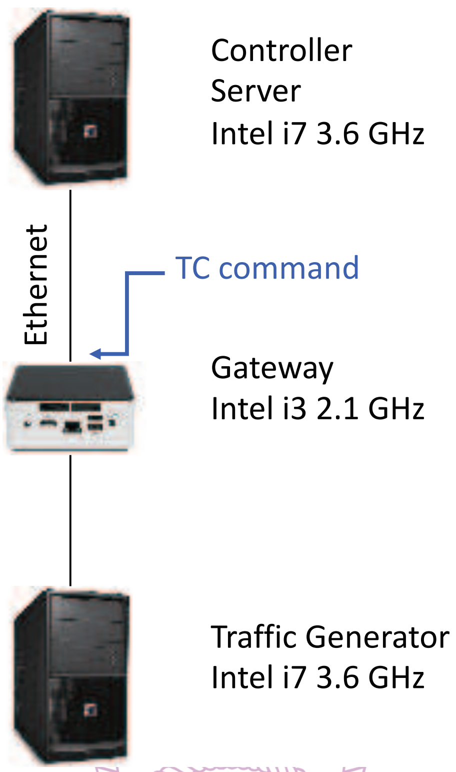


Figure 8.5: Our lab testbed with an emulated access link.

8.3 Setup

By default, we let the image pool size be 3 GB. We let the network bandwidth (B_u, B_d) = (5, 100) Mbps. The image download algorithms are executed once every 5 minutes, and the rate allocation algorithms are executed once every 1 minute. We generate the IoT analytics requests using a Poisson process with 1/3-min inter-arrival time. Each request randomly selects one of the twenty-four analytics containers. The average departure time of IoT analytics requests is randomly chosen in $[1, 10]$ minutes. Each experiment run lasts for 30 minutes. For the rate allocation problem, we use weights, which are random floating-point numbers in the interval $[0, 1]$. Table 8.2 gives the weights of the two sample IoT analytics containers. We compare the saved upload bandwidth among IDA_A with the approximation parameter ϵ from 0.1 to 0.5 and show their results in Figure 8.6. We let the approximation parameter $\epsilon = 0.4$ since it averagely has the most saved upload bandwidth. We also compare the weighted QoS and running time among RAA with different step sizes (α = 0.01, 0.05, 0.1, 0.15), which are shown in Figure 8.7(a) and 8.7(b). While the QoS of the four step size in Figure 8.7(a) is similar, the step size 0.01 takes the least running time in Figure 8.7(b). Therefore, we let the step size $\alpha = 0.01$.

We employ IDA_D, RAA, and LRU algorithms if not otherwise specified. Apart from looking at the experiment by default settings, we also adjust some parameters in experiments. We consider the following experiments. For Image download algorithms (IDA_D,

Table 8.2: Weights of Sample IoT Analytics Containers

Analytic / Run	SC 1	SC 2	SC 3	SC 4	SC 5	SC 6	SC 7	SC 8	SC 9	SC 10	SC 11	SC 12
1	0.22	0.06	1.0	0.9	0.16	0.01	0.25	0.63	0.07	0.49	0.52	0.24
2	0.6	0.51	0.48	0.9	0.63	0.52	0.83	0.56	0.05	0.09	0.38	0.91
3	0.33	0.45	0.07	0.4	0.95	0.82	0.88	0.86	0.98	0.53	0.75	0.7
4	0	0.9	0.86	0.11	0.08	0.23	0.53	0.96	0.2	0.96	0.54	0.03
5	0.73	0.37	0.33	0.05	0.21	0.07	0.02	0.96	0.45	0.08	0.15	0.82

Analytic / Run	OD 1	OD 2	OD 3	OD 4	OD 5	OD 6	OD 7	OD 8	OD 9	OD 10	OD 11	OD 12
1	0	0	0.25	0.94	0.71	0.53	0.09	0.54	0.7	0.99	0.59	0.72
2	0.54	0.51	0.6	0.25	0.03	0.23	0.03	0.02	0.26	0.83	0.35	0.26
3	0.6	0.76	0.03	0.74	0.74	0.78	0.33	0.1	0.78	0.6	0.29	0.35
4	0.2	0.38	0.4	0.37	0.94	0.31	0.61	0.37	0.58	0.92	0.73	0.92
5	0.42	0.91	0.47	0.3	0.24	0.17	0.55	0.2	0.3	0.35	0.21	0.89

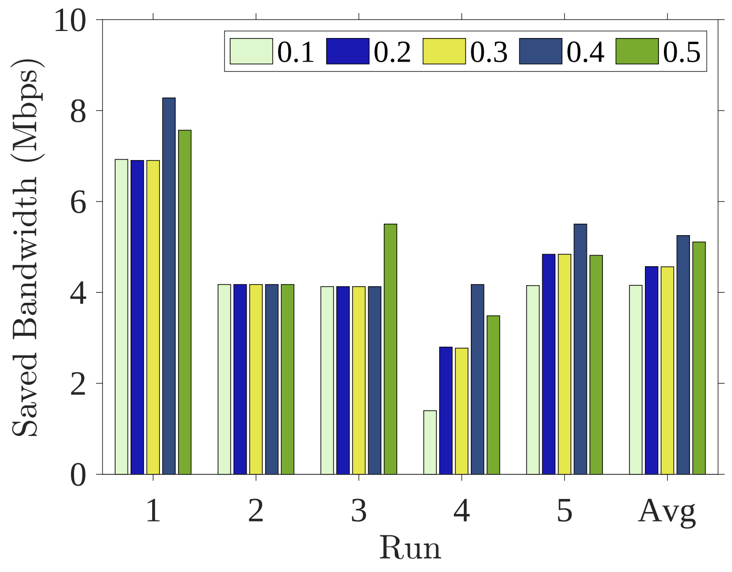
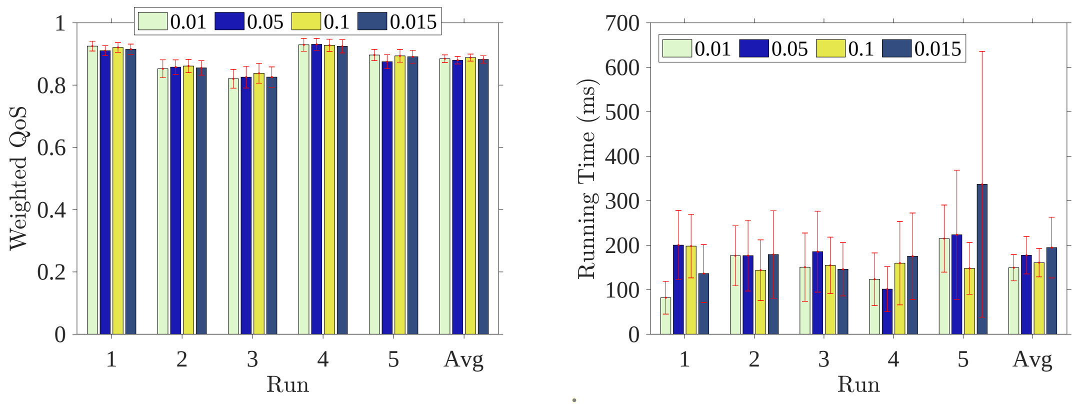


Figure 8.6: Saved upload bandwidth of IDA_A with different approximation parameters ϵ.

IDA_A, and IDA_G), we employ default settings, but with different: (1) image pool size: 3 GB (default), 6 GB, and (2) inter-arrival time: 1/3 min (default), 1/2 min. For rate allocation algorithms (RAA, WA, and UA), we use default settings, but with different upload network bandwidth: 5 Mbps (default), 10 Mbps. For layer replacement policies, we compare LRU, MRU, LFU, and MFU with default settings and extend the experiment time to 60 minutes. Moreover, we run each experiment five times with five different input requests.

To evaluate our proposed algorithms, we report the performance results using the following metrics:

- *Weighted QoS value* across all analytics containers on the data center server and gateway.



(a) (b)

Figure 8.7: Comparisons between RAA with different different step sizes α: (a) weighted QoS and (b) running time.

- *Saved upload bandwidth*, which indicates the reduced upload network workload by deploying containers to the gateway.
- *Hit saved upload bandwidth*, which indicates the saved upload bandwidth of the containers successfully deployed to the gateway.
- *Miss saved upload bandwidth*, which indicates the saved upload bandwidth of the containers unsuccessfully deployed to the gateway.
- *Expected saved upload bandwidth*, which indicates the saved upload bandwidth of the containers that IDA algorithms are supposed to deploy to the gateway. Expected saved upload bandwidth is also the sum of the hit saved upload bandwidth and the miss saved upload bandwidth.
- *Hit rate*, which is the ratio of the hit saved upload bandwidth and the expected saved upload bandwidth.
- *Number of analytics containers on the cloud server and gateway*, which includes the number of analytics containers running on the cloud server and the number of analytics containers running on the cloud server gateway.
- *Number of analytics containers on the gateway*, which is the number of analytics deployed to the gateway.
- *Upload/Download bandwidth consumption,* which indicates the incurred network workload.
- *Running time* of various algorithms.

8.4 Results

In this section, we first compare the performance of the three image download algorithms IDA_D, IDA_A, and IDA_G. We then compare the performance of our proposed rate allocation algorithm with the two baselines.. Last, we compare the performance of four representative layer replacement policies LRU, MRU, LFU, and MFU. We report average values with 95% confidence intervals whenever possible.

8.4.1 Default Sample Run Analysis

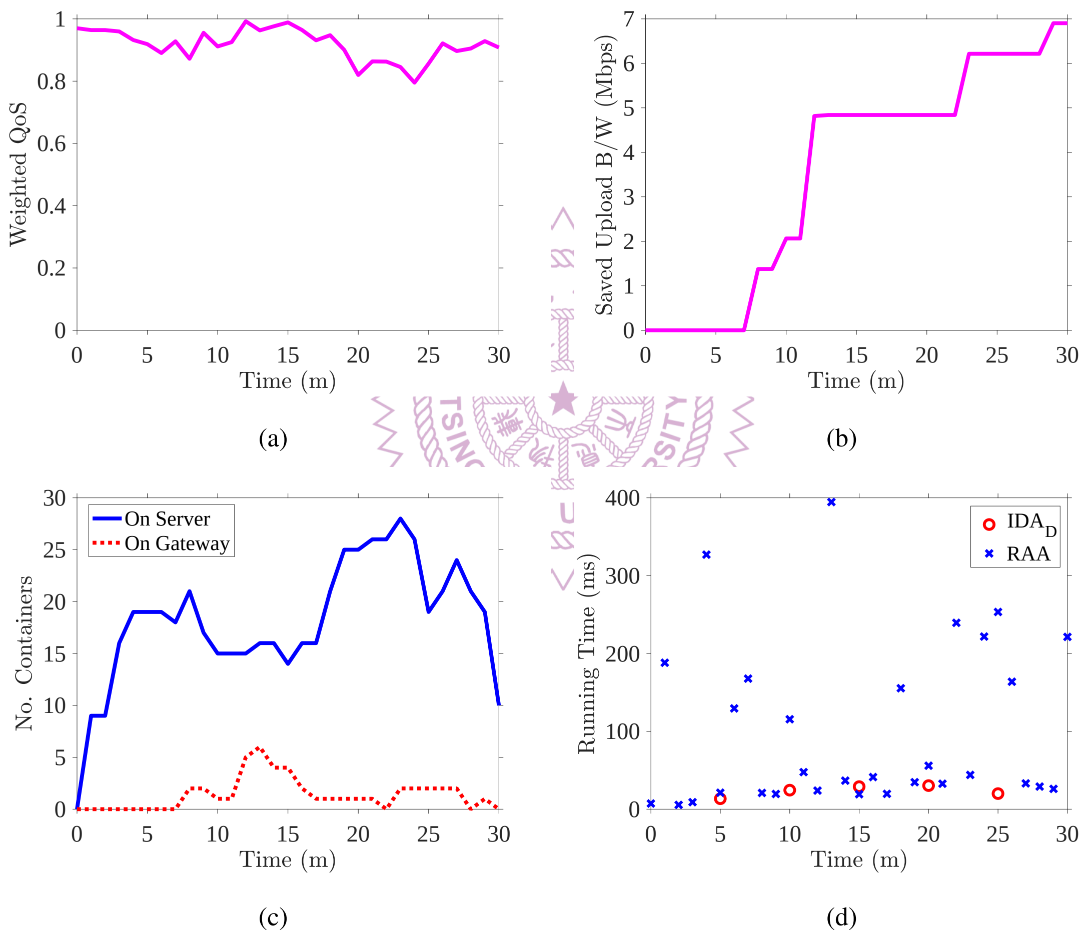
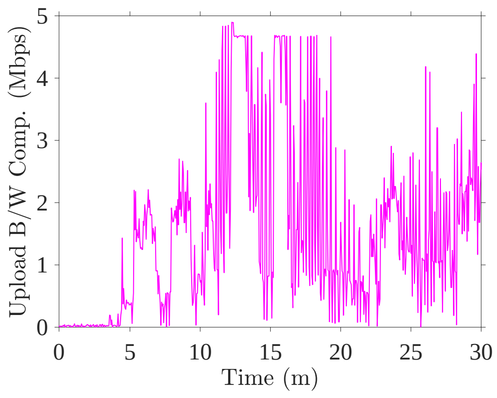
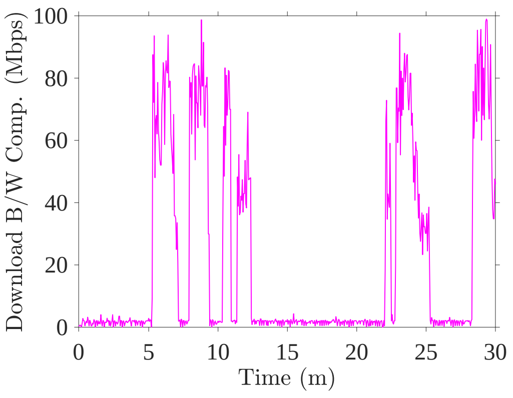


Figure 8.8: A sample run with IDA_D measuring: (a) weighted QoS, (b) saved upload bandwidth, (c) number of containers on the cloud server and gateway, and (d) running time.

Our proposed algorithms achieve high weighted QoS. Figure 8.8(a) reports the QoS level averaged per minute. This figure reveals that our algorithms achieve fairly high weighted QoS levels: at least 0.8 and at most 0.99 are observed. This is non-trivial, considering the QoS level is normalized in the range of $[0, 1]$. A closer look indicates that our proposed algorithms capitalize on the otherwise wasted download bandwidth to



(a)



(b)

Figure 8.9: A sample run with IDA_D measuring: (a) upload bandwidth consumption and (b) download bandwidth consumption.

deploy IoT analytics on the gateway. In particular, Figure 8.8(b) shows that increasingly more upload bandwidth (at most 6.9 Mbps) is saved over time. Figure 8.8(c) gives the number of containers over time, and Figure 8.8(d) depicts that the maximal running time of IDA_D and RAA is very short: at most 400 ms, which is negligible compared to their minute-scale invocation periods. We also validate that our algorithms do not overload the upload and download bandwidth, which are shown in Figure 8.9(a) and 8.9(b).

8.4.2 Image Download Algorithm Analysis

In this section, we compare the weighted QoS, saved upload bandwidth, number of analytics containers on the gateway, and running time of three IDA algorithms across five runs in the four different experiments in Figure 8.10–Figure 8.14. In these figures, the first group, IDA Def., is the experiments using default settings. Other groups IDA 6GB and IDA Arr2 are the experiments which exploit the default settings, but separately have a 6-GB image pool size and a 1/2-min inter-arrival time.

IDA_D, IDA_A, and IDA_G all have high weighted QoS. We compare the weighted QoS of three IDA algorithms in the three different experiments in Figure 8.10. From Figure 8.10, we observe that: (1) the three algorithms work well in terms of overall QoS, and (2) increasing image pool size has little influence on weighted QoS. The reason for the two above observations is that the number of IoT analytics containers deployed on the gateway is much less than that on the cloud server. For example, in Figure 8.8(c), the difference in the number of IoT analytics containers on the cloud server and gateway is large. In contrast, Figure 8.10 reveals that increasing the inter-arrival time severely affects the weighted QoS since the number of analytics containers dramatically decreases.

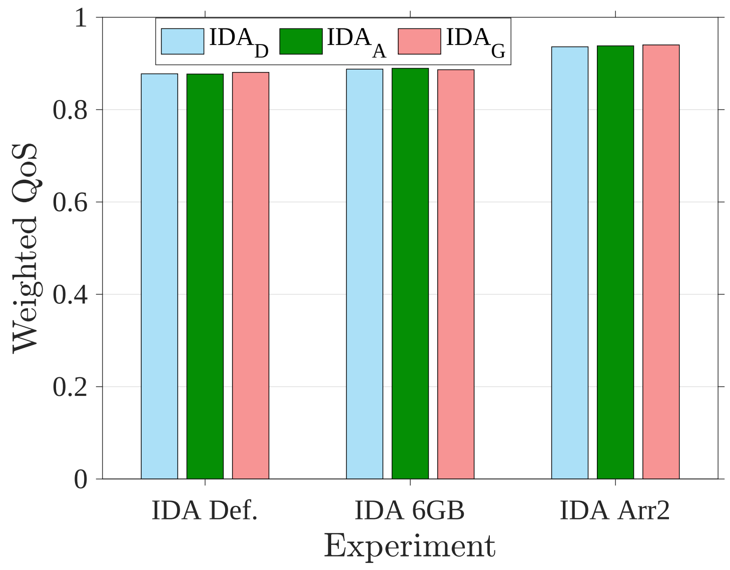
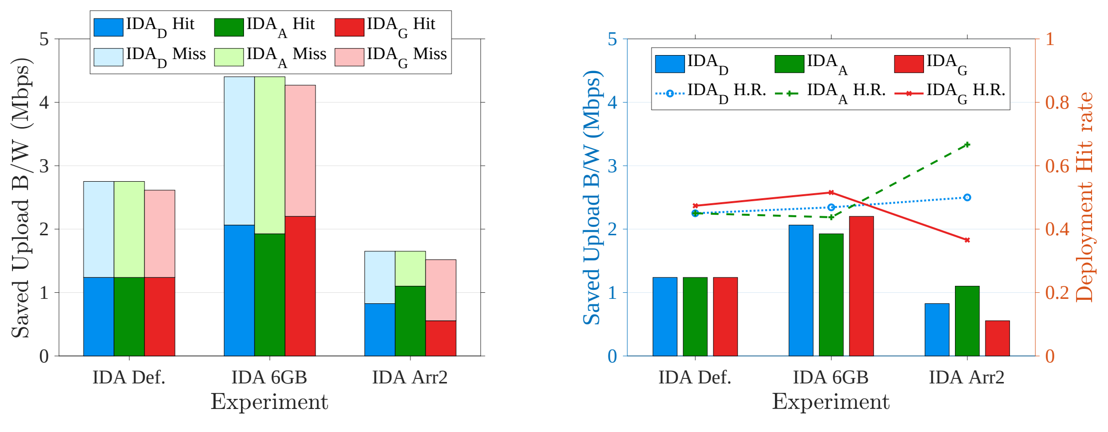


Figure 8.10: Weighted QoS of the three IDA algorithms in experiments: default settings (3-GB, 100-Mbps, 1/3-min), default settings with a 6-GB image pool size, and default settings with a 1/2-min inter-arrival time.

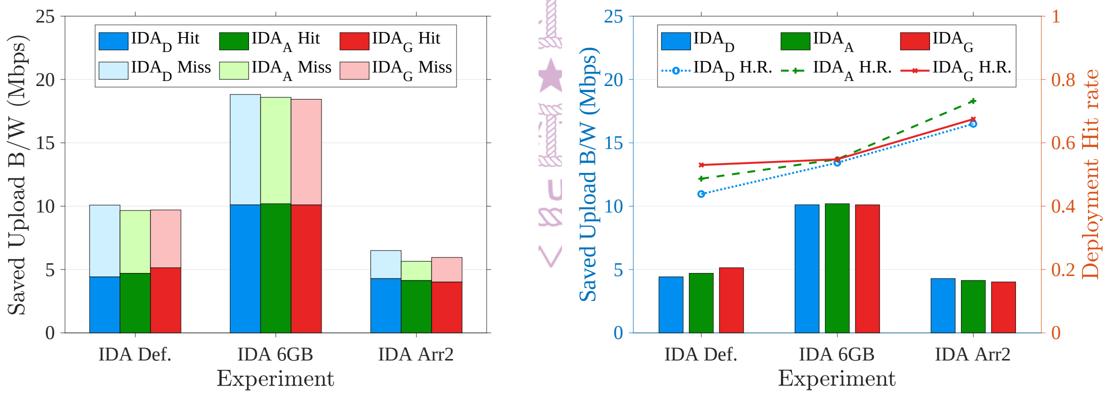
IDA$_D$ has the optimal result in expected saved upload bandwidth as the input requests are the same. Since some analytics containers depart before the deployment manager finishes downloading their container images, the deployment manager may fail to deploy some containers to the gateway. We call these failed containers *miss containers* and call successfully deployed containers *hit containers* for convenience. Also, the sum of hit and miss containers is the *expected containers*. The *hit* and *miss saved upload bandwidth* is the upload bandwidth that the hit and miss containers can save, separately. The *expected saved upload bandwidth* is the sum of the hit and miss saved upload bandwidth, and the ratio of hit saved upload bandwidth to the expected saved upload bandwidth is the hit rate. We compare the hit and miss saved upload bandwidth of analytics containers on the gateway of the three IDA algorithms in the first time slot of the three different experiments in Figure 8.11(a). We compare the saved upload bandwidth in the first time slot of experiments because, in the first time slot, the input requests of the three IDA algorithms are the same. For the expected saved upload bandwidth in Figure 8.11(a), IDA$_D$ gives the optimal result, and IDA$_A$ and IDA$_G$ give similar results. We also compare the hit saved upload bandwidth and hit rate of the three IDA algorithms in the first time slot of the three different experiments in Figure 8.11(b). IDA$_G$ has a higher hit rate in the first two experiments in Figure 8.11(b). We investigate deeper and find that IDA$_G$ gives smaller analytics containers higher priority when the input requests are the same. That is, IDA$_D$ and IDA$_G$ sometimes encounter the problem that small analytics containers depart when waiting for the downloading of big analytics containers. All in all, IDA$_D$, IDA$_A$,



(a) (b)

Figure 8.11: (a) Hit and miss saved upload bandwidth and (b) hit saved upload bandwidth and hit rate of the three IDA algorithms in the first time slot of experiments with default settings (3-GB, 100-Mbps, 1/3-min), default settings with a 6-GB image pool size, and default settings with a 1/2-min inter-arrival time.

and IDA$_G$ have similar saved upload bandwidth.



(a) (b)

Figure 8.12: (a) Overall hit and miss saved upload bandwidth and (b) Overall hit saved upload bandwidth and hit rate of the three IDA algorithms in experiments with default settings (3-GB, 100-Mbps, 1/3-min), default settings with a 6-GB image pool size, and default settings with a 1/2-min inter-arrival time.

IDA$_D$, IDA$_A$, and IDA$_G$ all perform well in terms of saved upload bandwidth. Apart from comparing the outcome in the first time slot, we compare the overall hit and miss saved upload bandwidth of the three IDA algorithms in the three different experiments in Figure 8.12(a). We compare the overall hit saved upload bandwidth and hit rate of the three IDA algorithms in the three different experiments as well in Figure 8.12(b). In every time slot, the results of the three IDA algorithms are decided by the current analytics containers status, which depends on the results in the previous time slots. Hence, the overall comparison results of saved upload bandwidth of the three IDA algorithms in Figure 8.12(a) and 8.12(b) may be different from those in Figure 8.11(a) and 8.11(b). In Figure 8.12(a), we find that IDA$_\mathrm{D}$ has the most expected saved upload bandwidth. In addition, IDA$_\mathrm{D}$ has the lowest hit rate in Figure 8.12(b). However, IDA$_\mathrm{D}$, IDA$_\mathrm{A}$, and IDA$_\mathrm{G}$ conclusively save similar amounts of upload bandwidth.

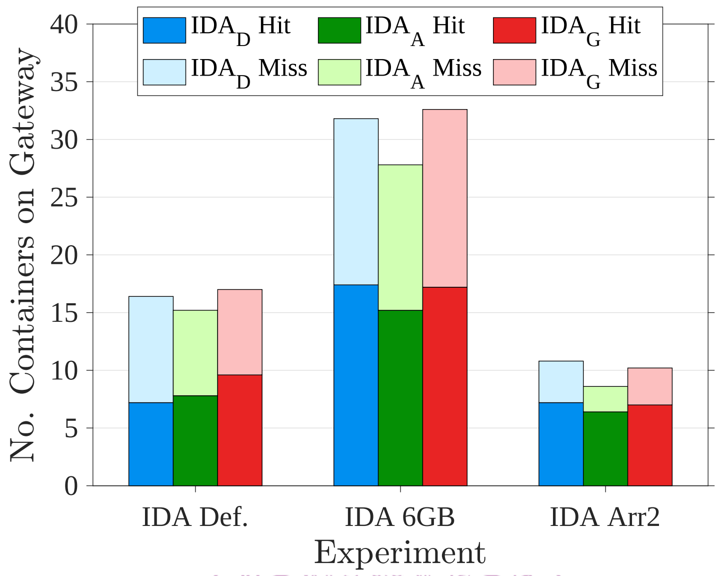


Figure 8.13: Overall hit and miss number of analytics containers on the gateway of the three IDA algorithms in experiments: default settings (3-GB, 100-Mbps, 1/3-min), default settings with a 6-GB image pool size, and default settings with a 1/2-min inter-arrival time.

IDA$_\mathrm{G}$ deploys more analytics containers to the gateway. We compare the number of analytics containers on the gateway of the three IDA algorithms in the three different experiments in Figure 8.13. In Figure 8.13, IDA$_\mathrm{G}$ averagely has the most expected analytics containers on the gateway. Moreover, IDA$_\mathrm{G}$ averagely deploys the most analytics containers on the gateway. Figure 8.13 reports that IDA$_\mathrm{G}$ outperforms IDA$_\mathrm{D}$ and IDA$_\mathrm{A}$ at most by 29% and 17%. A deeper investigation indicates that this is because IDA$_\mathrm{G}$ gives smaller analytics containers higher priority. Smaller analytics have higher chance to be downloaded before the deadline, which is crucial in dynamic (real) environments.

IDA$_\mathrm{D}$, IDA$_\mathrm{A}$, and IDA$_\mathrm{G}$ all finish in a short time. For instance, Figure 8.14 confirms the short running time of all algorithms in the three experiments: < 40 ms. Since the scale of every request is small, and the total analytics containers are less diverse, the three algorithms have similar running times.

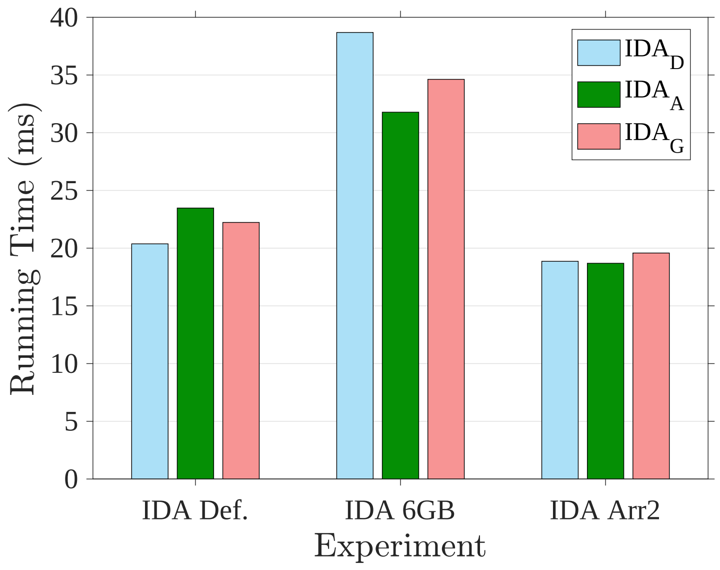
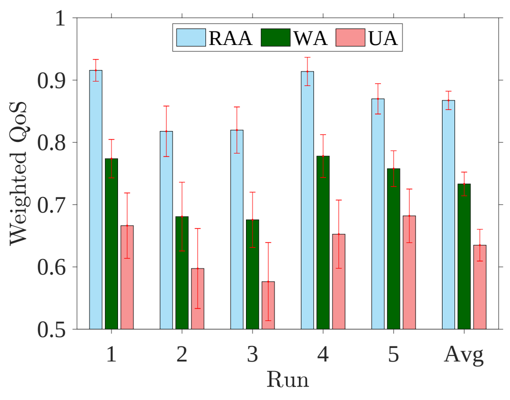


Figure 8.14: Running time of the three IDA algorithms in experiments: default settings (3-GB, 100-Mbps, 1/3-min), default settings with a 6-GB image pool size, and default settings with a 1/2-min inter-arrival time.

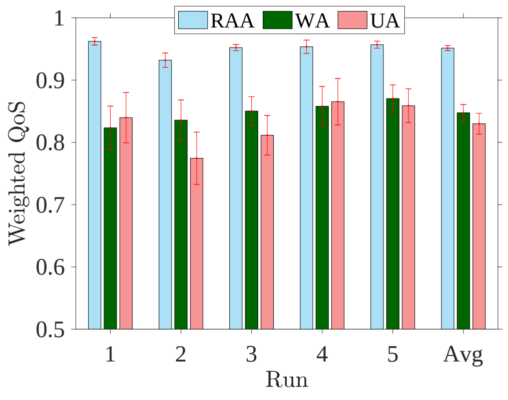
8.4.3 Rate Allocation Algorithm Analysis

Our proposed RAA algorithm achieves a higher weighted QoS. We compare the weighted QoS between RAA and two baselines (WA and UA) across five runs in different upload network bandwidths (5 and 10 Mbps) in Figure 8.15. Figure 8.15(a) reports that when the upload network bandwidth is 5 Mbps, our RAA algorithm outperforms the WA and UA algorithms by about 18% and 37% in weighted QoS. Figure 8.15(b) shows that when the upload network bandwidth is 10 Mbps, our RAA algorithm outperforms the WA and UA algorithms by about 12% and 15% in weighted QoS. Furthermore, Figure 8.15(a) also reveals that RAA can still maintain the high weighted QoS around 0.82 in low upload network bandwidth. On the other hand, the two baselines are severely influenced by the upload network bandwidth. Figure 8.15(b) reports that high upload network bandwidth provides a stabler and higher weighted QoS. For example, the difference between the lowest and highest weighted QoS of RAA in Figure 8.15(b) is roughly 0.03, which is much less than the difference 0.1 of RAA in Figure 8.15(a).

Our proposed RAA algorithm has higher utilization rate of upload network bandwidth. We compare the upload network bandwidth consumption between RAA and two baselines (WA and UA) across five runs in different upload network bandwidths (5 and 10 Mbps) in Figure 8.16. Figure 8.16(a) reports that when the upload network bandwidth is 5 Mbps, our RAA algorithm outperforms the WA and UA algorithms by about 53% and 31% in the upload network bandwidth consumption. Figure 8.16(b) reports that when the



(a)



(b)

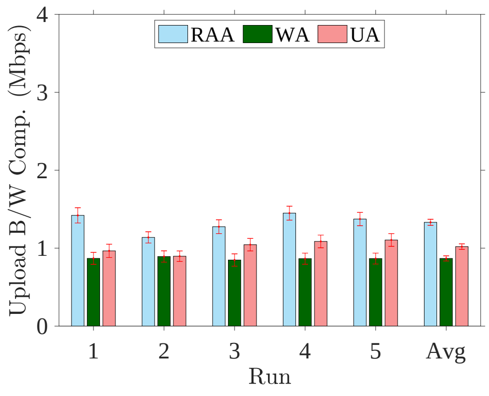
Figure 8.15: Weighted QoS of RAA and two baselines (WA and UA) in experiments: (a) default settings with a 5-Mbps upload network bandwidth and (b) default settings with a 10-Mbps upload network bandwidth.

upload network bandwidth is 10 Mbps, our RAA algorithm outperforms the WA and UA algorithms by about 162% and 61% in the upload network bandwidth consumption. To be more precise, our RAA algorithm maintains 27% and 26% utilization rates of upload network bandwidth in Figure 8.16(a) and 8.16(b). However, WA and UA have 17% and 20% utilization rates of upload network bandwidth in Figure 8.16(a), and their utilization rates drop to 10% and 16% in Figure 8.16(b).

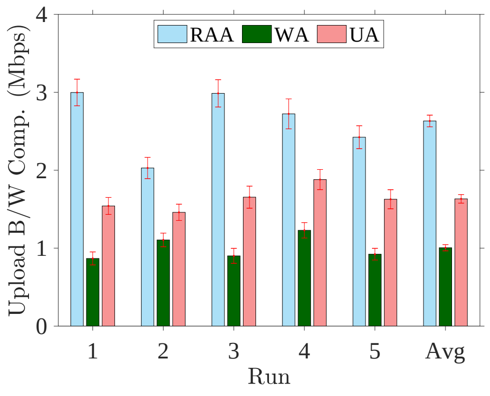
All the RAA algorithms have a short running time. We compare the running time between RAA and two baselines (WA and UA) across five runs in different upload network bandwidths (5 and 10 Mbps) in Figure 8.17. More precisely, the average running time of RAA, WA, and UA are all less than 200 ms in both figures, which is negligible.

8.4.4 Layer Replacement Policy Analysis

LRU saves more upload bandwidth. Figure. 8.18 shows that different replacement policies have little impact on the weighted QoS. Figure. 8.19 reports LRU outperforms MRU, LFU, and MFU by about 14%, 23%, and 16% respectively in the saved upload bandwidth. In addition, LRU and MRU are the best and the second best policies to save the most upload bandwidth. We take a closer look and find that the container images which are downloaded in the same time slot will share more layers with each other. Therefore, LRU and MRU will preserve more complete container images, which can avoid too many pieces of container images occupying the space of the image pool. Furthermore, LRU will also avoid deleting the layers, whose bottom layers are shared with the running containers. Thus, LRU averagely saves the most upload bandwidth in Figure. 8.19.

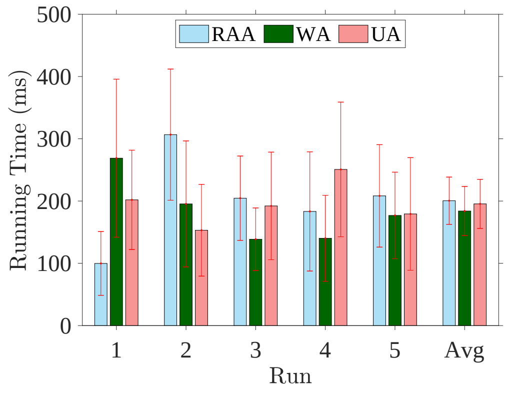


(a)

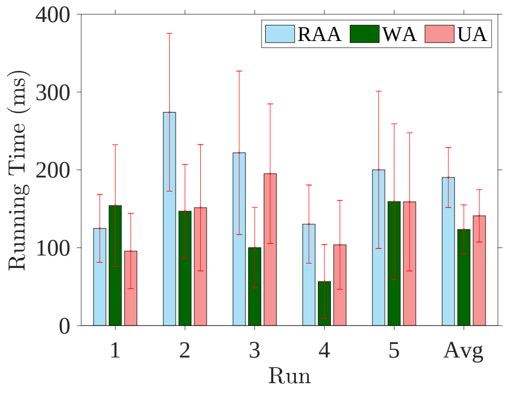


(b)

Figure 8.16: Upload bandwidth consumption of RAA and two baselines (WA and UA) in experiments: (a) default settings with a 5-Mbps upload network bandwidth and (b) default settings with a 10-Mbps upload network bandwidth.



(a)



(b)

Figure 8.17: Running time of RAA and two baselines (WA and UA) in experiments: (a) default settings with a 5-Mbps upload network bandwidth and (b) default settings with a 10-Mbps upload network bandwidth.

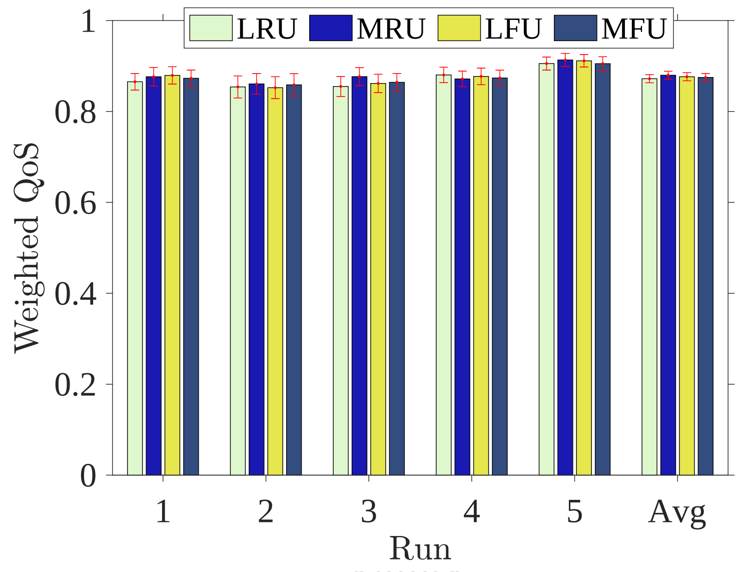


Figure 8.18: Overall weighted QoS of layer replacement policies.

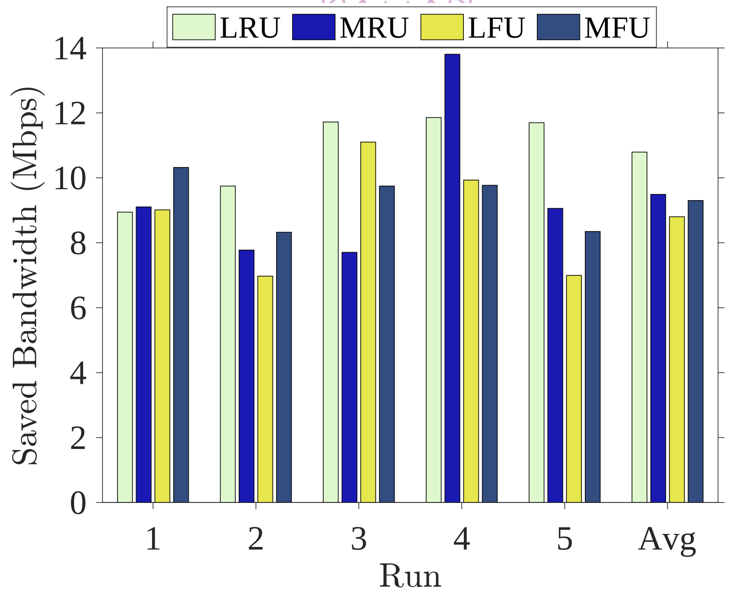


Figure 8.19: Overall saved upload bandwidth of layer replacement policies.

Chapter 9

Related Work

In this chapter, we survey the literature. We first present the studies focusing on IoT platforms without edge devices. We then present the studies where edge devices are incorporated into IoT platforms. Last, we focus on the studies related to IoT analytics.

9.1 IoT Platforms without Edge Devices

IoT platforms in real-world deployments and measurements for environmental and traffic monitoring have been studied in the literature [16,26,42]. These studies focus on mitigating high computational complexity due to large-scale dynamic data collected from many IoT devices. Zhai et al. [51] consider the problem of minimizing total power consumption under constrained user-specified rates. The problem is challenging as the Shannon capacity formula shows that the data rate is nonlinear. They transform the problem into a convex optimization problem for optimal power and rate allocation. They evaluate their system in simulators without real-world deployment. Zhai et al. [50] investigate the power consumption minimization and dynamic user scheduling problem for Non-Orthogonal Multiple Access (NOMA) networks. In this paper, a low-complexity algorithm is proposed based on Lyapunov optimization and branch-and-bound techniques. Their simulations evaluate the performance of their algorithm without real implementations. Furthermore, Lv et al. [33] solve the problem of maximizing the sum- and fair-throughput in both the downlink and uplink, from the networking aspect. The above studies focus on some optimization criteria, and there are also papers that discuss the trade-off among multiple objectives, like power allocation, throughput, communication overhead [17, 19, 36]. Hierarchical system architectures or dynamic scheduling algorithms are employed in these papers. Like the aforementioned studies, however, data processing (such as IoT analytics) is done in the cloud rather than edge devices, which results in long response time and degraded QoS.

Table 9.1: Related Work

Reference	Offloading Decision	Resources Allocation	Deleted Policy	Virtualization	Objective	Decision Maker
Fan et al. [20]	✓	✗	✗	VM	Delay	Cloud
Zhao et al. [52]	✓	✗	✗	VM	Delay	Cloud
Moghaddam et al. [34]	✓	Uplink	✗	VM	Price	Cloud
Faruque et al. [21]	✓	Uplink	✗	VM	Energy	IoT Device
Yousefpour et al. [47]	✓	✗	✗	VM	Delay	Cloud
Liu et al. [32]	✓	Computation	✗	VM	Energy	Cloud
Deng et al. [18]	✓	Computation	✗	VM	Energy	Cloud
Wang et al. [44]	✓	Uplink Computation	✗	VM	Delay	IoT Device
Wei et al. [45]	✓	Uplink Computation	✗	VM	Delay	IoT Device
Our Proposal	✓	Uplink	✓	Container	QoS	Cloud IoT Device

9.2 IoT Platforms with Edge Devices

Edge devices are incorporated into some IoT platforms to mitigate some of the drawbacks of pure-cloud based IoT platforms. Kiani and Ansari [30] employ the concept of *cloudlet* to extend remote data centers so as to bring the cloud closer to end-users. Hierarchical infrastructure is used to meet the QoS requirements from users. Through simulations, they demonstrate that their proposal outperforms the baselines in terms of, e.g., latency reduction. Fan and Ansari [20] and Zhao et al. [52] both investigate the placement problem of cloudlets in SDN networks. Their proposed algorithms adapt to devices with diverse and dynamic workloads. While the concept of cloudlet brings some computing and storage resources to the end-users, it lacks horizontal integration [37].

In contrast, *fog computing* aims to integrate a distributed deployment of multiple edge devices, in order to reduce the transfer time among IoT devices, edge servers, and cloud servers. Xu and Helai [46] introduce the Cloud-Edge-Beneath (CEB) architecture for large-scale deployment in smart cities. CEB consists of three layers: the beneath layer (physical layer), edge layer, and cloud layer. They conduct experiments with diverse numbers of sensor nodes and compare the average CPU utilization as time goes on. While the experiments validate the functionality of this system, no optimization strategies exist in their system. Morabito et al. [35] adopt lightweight virtualization technologies to deploy software on real IoT devices. In their measurement study, they quantify the response time, CPU utilization, and power consumption of different tasks and networks, with and without different virtualization technologies. Pahl et al. [38, 39] introduce another edge-cloud architecture based on the Platform-as-a-Service paradigm and implement it on IoT devices. Their architecture supports cluster management for reducing power consumption and processing latency. The above system papers focus on building fog computing systems without rigorous performance optimization.

More recently, different optimization objectives, such as energy consumption, latency, and QoS requirements, have been considered in literature. Table 9.1 shows the research with different optimization objectives. Moghaddam and Leon-Garcia [34] solve the problem of energy management. The fog nodes act as the retail power market in the proposed model, providing energy to end-users, and the system would optimize the customer's schedule based on the power consumption of each application. The simulations evaluate the bandwidth allocation and delay of the fog-based model and cloud-based model. Also, they confirm that the proposed fog-based model decreases bandwidth consumption and delay time. Faruque and Vatanparvar [21] address the energy management issue through fog computing, especially for the implementation of Home Energy Management (HEM) and microgrid-level energy management. With fog computing, the system can overcome real-time connectivity and data privacy challenges, and the experiments show that they are able to monitor and control energy consumption. Yousefpour et al. [47] solve the problem of minimizing the service delay by delay-minimizing fog offloading policy. The policy is to offload tasks based on their response time, which depends on the queue length and request types. The simulation evaluates the performance on the proposed framework, but the threshold of fog nodes is static. Liu et al. [32] and Deng et al. [18] investigate the relation between power consumption and delay performance. They trade delay performance off against power consumption. The previous work use queuing theory to Mobile Devices (MD), fog, and cloud center while the last one decomposes the problem into three subproblems: the tradeoff for the fog computing system, the tradeoff for cloud computing system, and the tradeoff for the fog-computing system. The simulations of both studies show the transmission power and delay performance is better than other schemes, which only optimize power utilization or delay reduction. But with the complexity of data increasing, we need a more suitable application for complicated analysis. Wang et al. [44] solve the problem of minimizing the system latency by coupling the offloading, transmission, and computing resource allocation. The experiments evaluate the system latency and system utility, but the downlink is not considered, which may affect the startup time of the fog application. Wei et al. [45] tackles the issues of minimizing the system latency by content strategy, offloading policy, and radio resource allocation. They have proposed an actor-critic deep RL model to make the decisions. They consider both uplink and downlink. The simulations evaluate the system latency and the system utility, but they don't consider the deleted images policy when the edge devices are full.

9.3 IoT Analytics on Edge Devices

IoT analytics is the data analysis application to obtain value from a large volume of data. With lightweight virtualization technologies, IoT analytics could be deployed on small devices, such as mobile devices or embedded devices. He et al. [22] introduce the multi-tier fog computing model with IoT analytics in order to maximize utility for QoS aware services. In this paper, the architecture includes two tier fog nodes, dedicated fog (D-fog) and ad-hoc fog (A-fog). They develop job admission control/offloading and resource allocation schemes, which decide where the analytics service is executed (D-fog, A-fog, or cloud), and how many resources (CPU, memory) it can use. The simulations evaluate the service utility and job arrival time, but they only consider the uplink bandwidth as the limitation.

Our earlier work [41] considered a simplified version of the considered rate allocation problem over a 3G cellular link, where the IoT analytics containers were preloaded on gateways, and diverse QoS functions were not considered.

Chapter 10

Conclusion

In this thesis, we solved the image download and rate allocation problem at an IoT gateway in smart spaces like smart cities. Our proposed image download algorithms strive to reduce the upload bandwidth consumption due to the bulky raw sensor data, while our rate allocation algorithm takes the heterogeneity of IoT analytics into consideration to maximize the overall weighted QoS level. We evaluated our proposed algorithms on our campus and lab testbeds built upon several open-source projects. The experiment results show the merits of our proposed system and algorithms on increasing the overall QoS level (between 0.8 and 0.99 in the scale of [0,1]) without overloading the network and gateway (terminate in < 400 ms). For the image download problem, we recommend using IDA_G in a dynamic environment, which makes smaller analytics have a higher chance to be downloaded before the deadline. For the rate allocation problem, our proposed algorithm RAA outperforms the two baseline algorithms by 18% and 37% in weighted QoS levels in a low upload network bandwidth environment, and it also outperforms the two baseline algorithms by 162% and 61% in utilizing upload bandwidth in a high upload network bandwidth environment. For layer replacement policies, LRU saves more upload bandwidth than that of the other three policies.

The current thesis can be extended in the following directions:

- **Considering more resource types.** The current system only considers a few resource types like image pool size and bandwidth. Some other resource types, like computation resources, can also be accounted.
- **Augmenting the source code of the Docker engine.** The current system does not fully implement the layer deletion and layer extraction functionalities in the Docker engine. If we augment the Docker engine and directly put these functionalities in it, the overall performance can be further improved.
- **Conducting more experiments.** Additional experiments can be conducted to fur-

ther understand the performance of our system and algorithms under diverse, dynamic, and realistic conditions. For instance, we can extend larger experiments driven by real traces from our campus testbed. We can even run our algorithms on the street lamps handling live data feeds from sensors.

Bibliography

[1] Amazon echo. https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-Alexa-Black/dp/B00X4WHP5E/.

[2] Audio classification: Multilayer neural networks using TensorFlow. https://github.com/nextco/audio-classification.

[3] Docker web page. https://www.docker.com/.

[4] Global IoT analytics market to grow at a CAGR of +30.9% during forecast period 2018-2025. https://www.marketresearchfuture.com/reports/iot-analytics-market-1757.

[5] Google home. https://store.google.com/gb/product/google_home/.

[6] Internet of things (iot) connected devices installed base worldwide from 2015 to 2025. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[7] Kubernetes web page. https://kubernetes.io/.

[8] Linux containers web page. https://linuxcontainers.org/.

[9] Moby: open framework created by docker to assemble specialized container systems. https://mobyproject.org/.

[10] Tensorflow web page. https://www.tensorflow.org/.

[11] Tmall genie. https://bot.tmall.com/.

[12] Urban sound 8k dataset. https://urbansounddataset.weebly.com/urbansound8k.html.

[13] Visual object classes challenge 2012. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/#data.

[14] YOLO: Real-time object detection. https://pjreddie.com/darknet/yolo/.

[15] K. Ashton. That 'internet of things'thing. *RFID journal*, 22(7):97–114, 2009.

[16] L.-J. Chen, Y.-H. Ho, H.-H. Hsieh, S.-T. Huang, H.-C. Lee, and S. Mahajan. ADF: An anomaly detection framework for large-scale PM2.5 sensing systems. *IEEE Internet of Things Journal*, 5(2):559–570, April 2018.

[17] Z. Chu, F. Zhou, Z. Zhu, R. Q. Hu, and P. Xiao. Wireless powered sensor networks for Internet of Things: Maximum throughput and optimal power allocation. *IEEE Internet of Things Journal*, 5(1):310–321, February 2018.

[18] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal*, 3(6):1171–1181, December 2016.

[19] S. Ezdiani, I. S. Acharyya, S. Sivakumar, and A. Al-Anbuky. Wireless sensor network softwarization: Towards WSN adaptive QoS. *IEEE Internet of Things Journal*, 4(5):1517–1527, October 2017.

[20] Q. Fan and N. Ansari. Application aware workload allocation for edge computing-based IoT. *IEEE Internet of Things Journal*, 5(3):2146–2153, June 2018.

[21] M. A. A. Faruque and K. Vatanparvar. Energy management-as-a-service over fog computing platform. *IEEE Internet of Things Journal*, 3(2):161–169, April 2016.

[22] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang. Multitier fog computing with large-scale IoT data analytics for smart cities. *IEEE Internet of Things Journal*, 5(2):677–686, April 2018.

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

[24] H. Hong, P. Tsai, and C. Hsu. Dynamic module deployment in a fog computing platform. In *in Proc. of Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Kanazawa, Japan, 2016.

[25] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arXiv preprint arXiv:1704.04861.*, 2017.

[26] J. Huang, Y. Meng, X. Gong, Y. Liu, and Q. Duan. A novel deployment scheme for green Internet of Things. *IEEE Internet of Things Journal*, 1(2):196–205, April 2014.

[27] Z. Ji, I. Ganchev, O. Droma, L. Zhao, and X. Zhang. A cloud-based car parking middleware for iot-based smart cities: Design and implementation. *Sensors*, 14(12):22372–22393, November 2014.

[28] D. Katsaros and Y. Manolopoulos. Cache management for web-powered databases. In J. W. Rahayu and D. Taniar, editors, *Web-Powered Databases*, chapter 8, pages 203–244. IGI Global, Hershey, PA, USA, 2003.

[29] H. Kellerer and U. Pferschy. *Knapsack Problems*. Springer, 2004.

[30] A. Kiani and N. Ansari. Toward hierarchical mobile edge computing: An auction-based profit maximization approach. *IEEE Internet of Things Journal*, 4(6):2082–2091, December 2017.

[31] B. Li and J. Yu. Research and application on the smart home based on component technologies and internet of things. *Procedia Engineering*, 15:2087–2092, 2011. CEIS 2011.

[32] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi. Multiobjective optimization for computation offloading in fog computing. *IEEE Internet of Things Journal*, 5(1):283–294, February 2018.

[33] K. Lv, J. Hu, Q. Yu, and K. Yang. Throughput maximization and fairness assurance in data and energy integrated communication networks. *IEEE Internet of Things Journal*, 5(2):636–644, April 2018.

[34] M. H. Y. Moghaddam and A. Leon-Garcia. A fog-based internet of energy architecture for transactive energy management systems. *IEEE Internet of Things Journal*, 5(2):1055–1069, April 2018.

[35] R. Morabito, I. Farris, A. Iera, and T. Taleb. Evaluating performance of containerized IoT services for clustered devices at the network edge. *IEEE Internet of Things Journal*, 4(4):1019–1030, August 2017.

[36] B. Mostafa, A. Benslimane, M. Saleh, S. Kassem, and M. Molnar. An energy-efficient multiobjective scheduling model for monitoring in Internet of Things. *IEEE Internet of Things Journal*, 5(3):1727–1738, June 2018.

[37] Openfog reference architecture for fog computing. Standard, OpenFog Consortium, 2017.

[38] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee. A container-based edge cloud paas architecture based on raspberry pi clusters. In *Proc. of IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloudW'16)*, page 117–124, Vienna, Austria, October 2016.

[39] C. Pahl and B. Lee. Containers and clusters for edge cloud architectures–a technology review. In *Proc. of IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, Rome, Italy, August 2015.

[40] D. Patterson and J. Hennessy. *Computer Organization and Design: the Hardware Software Interface*. Morgan Kaufmann, 2004.

[41] M. Rahman, A. Rahman, H. J. Hong, L. W. Pan, M. Y. S. Uddin, N. Venkatasubramanian, and C. H. Hsu. An adaptive iot platform on budgeted 3g data plans. *Journal of Systems Architecture*, 97:65–76, August 2019.

[42] P. M. Santos, J. Rodrigues, S. Cruz, T. Lourenço, P. d'Orey, Y. Luis, C. Rocha, S. Sousa, S. Crisóstomo, C. Queirós, S. Sargento, A. Aguiar, and J. Barros. Portolivinglab: An iot-based sensing platform for smart cities. *IEEE Internet of Things Journal*, 5(2):523–532, April 2018.

[43] S. Verma, Y. Kawamoto, Z. Fadlullah, H. Nishiyama, and N. Kato. A survey on network methodologies for real-time analytics of massive iot data and open research issues. *IEEE Communications Surveys & Tutorials*, 19(3):1457–1477, April 2017.

[44] P. Wang, C. Yao, Z. Zheng, G. Sun, and L. Song. Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems. *IEEE Internet of Things Journal*, 6(2):2872–2884, April 2019.

[45] Y. Wei, F. Yu, M. Song, and Z. Han. Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor–critic deep reinforcement learning. *IEEE Internet of Things Journal*, 6(2):2061–2073, April 2019.

[46] Y. Xu and A. Helal. Scalable cloud–sensor architecture for the Internet of Things. *IEEE Internet of Things Journal*, 3(3):285–298, June 2016.

[47] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue. On reducing IoT service delay via fog offloading. *IEEE Internet of Things Journal*, 5(2):998–1010, April 2018.

[48] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. A survey on the edge computing for the internet of things. *IEEE Access*, 6:6900–6919, 2018.

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

[50] D. Zhai, R. Zhang, L. Cai, B. Li, and Y. Jiang. Energy-efficient user scheduling and power allocation for NOMA-based wireless networks with massive IoT devices. *IEEE Internet of Things Journal*, 5(3):1857–1868, June 2018.

[51] X. Zhai, X. Guan, C. Zhu, L. Shu, and J. Yuan. Optimization algorithms for multiaccess green communications in Internet of Things. *IEEE Internet of Things Journal*, 5(3):1739–1748, June 2018.

[52] L. Zhao, W. Sun, Y. Shi, and J. Liu. Optimal placement of cloudlets for access delay minimization in (SDN-based) Internet of Things networks. *IEEE Internet of Things Journal*, 5(2):1334–1344, April 2018.