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

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

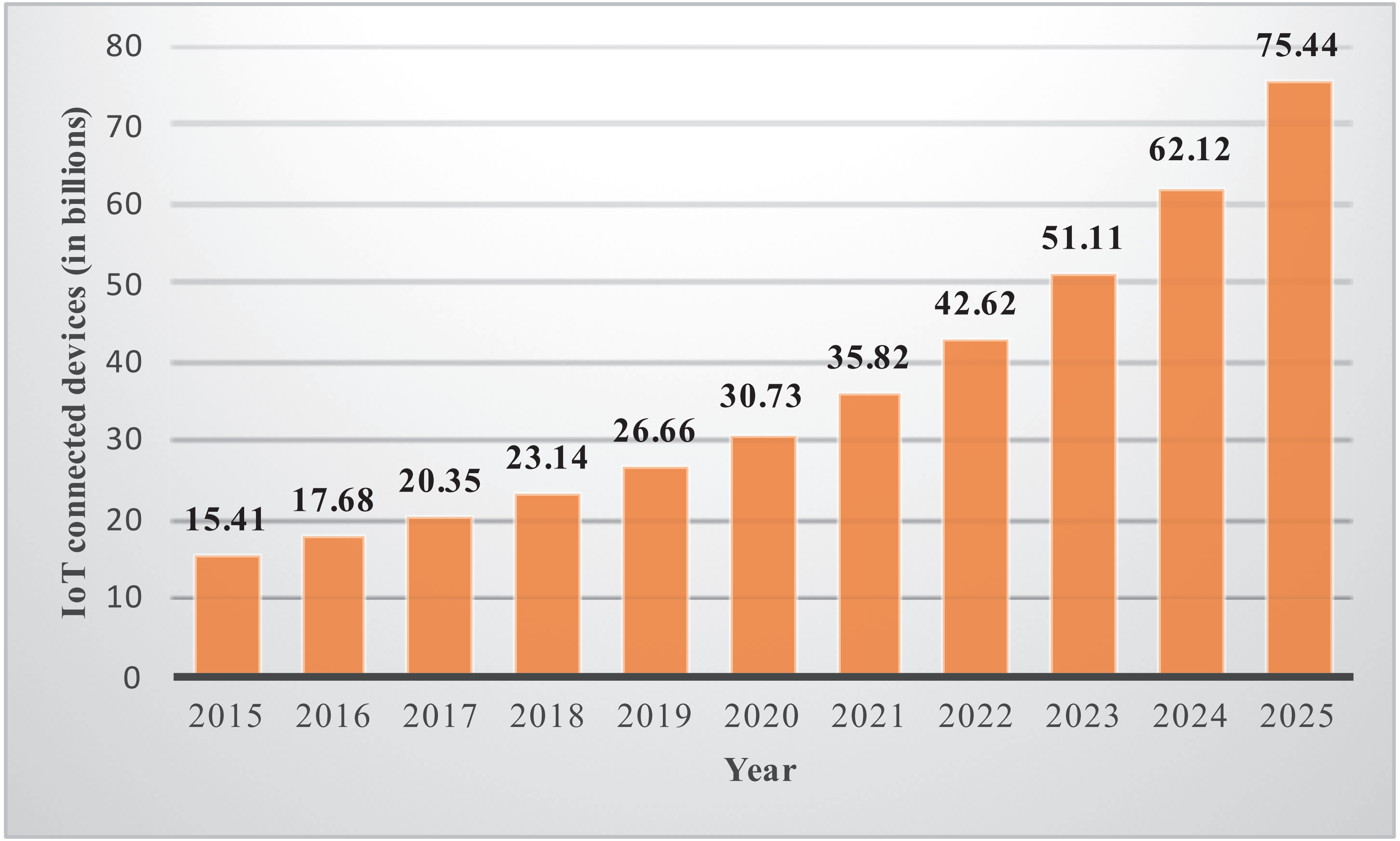
Yu-Jung Wang
Advisor: Cheng-Hsin Hsu
Networking Multimedia Systems Lab
CS Dept. National Tsing-Hua University

Outline

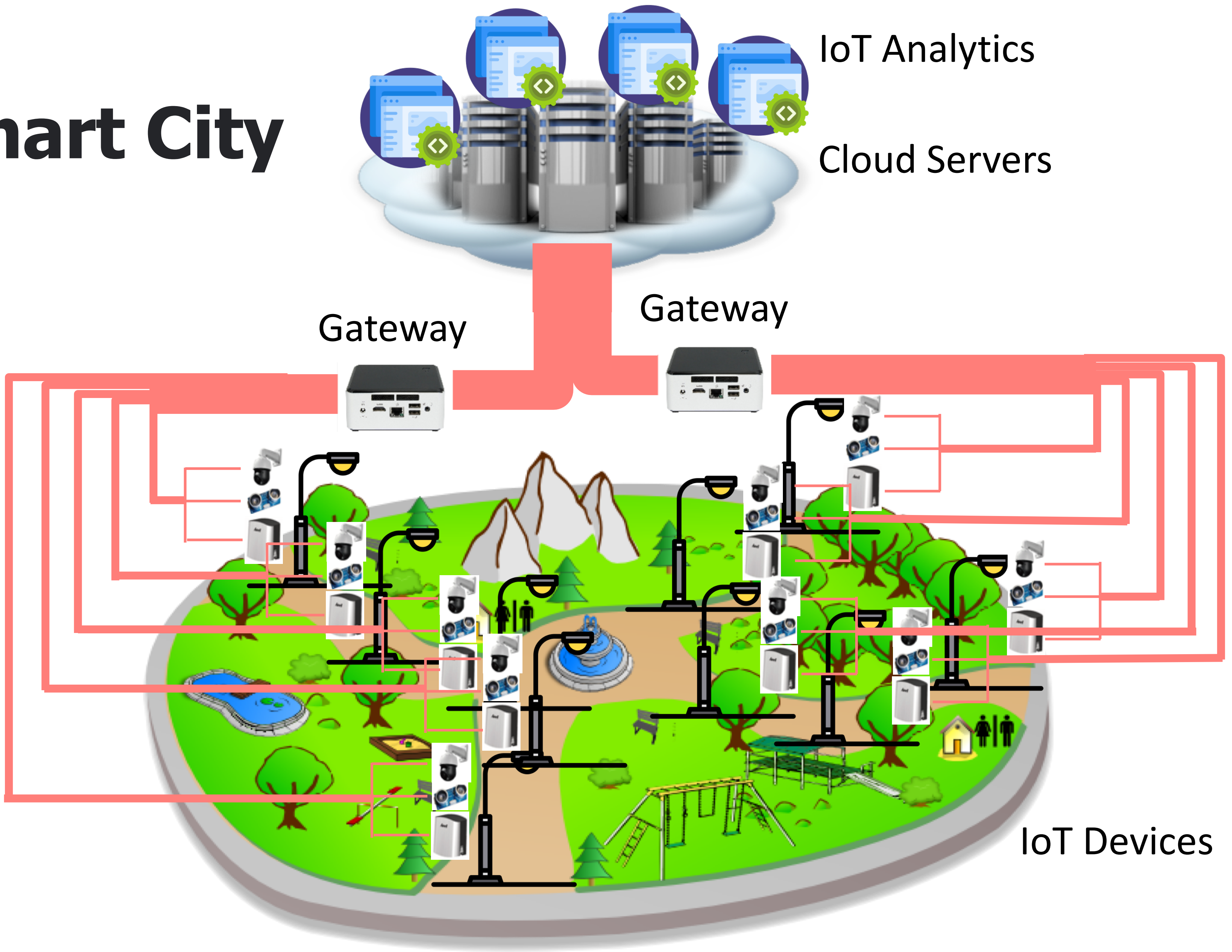
- Introduction
- Background
- System Architecture
- Image Download Problem
  - Dynamic Programing
  - (1-ε)-Approximation
  - Greedy
- Rate Allocation Problem
- Evaluation
- Conclusion

- Introduction
- Background
- System Architecture
- Image Download Algorithm
  - Dynamic Programing
  - (1-ε)-Approximation
  - Greedy
- Rate Allocation Algorithm
- Evaluation
- Conclusion

Internet of Things (IoT) is getting popular



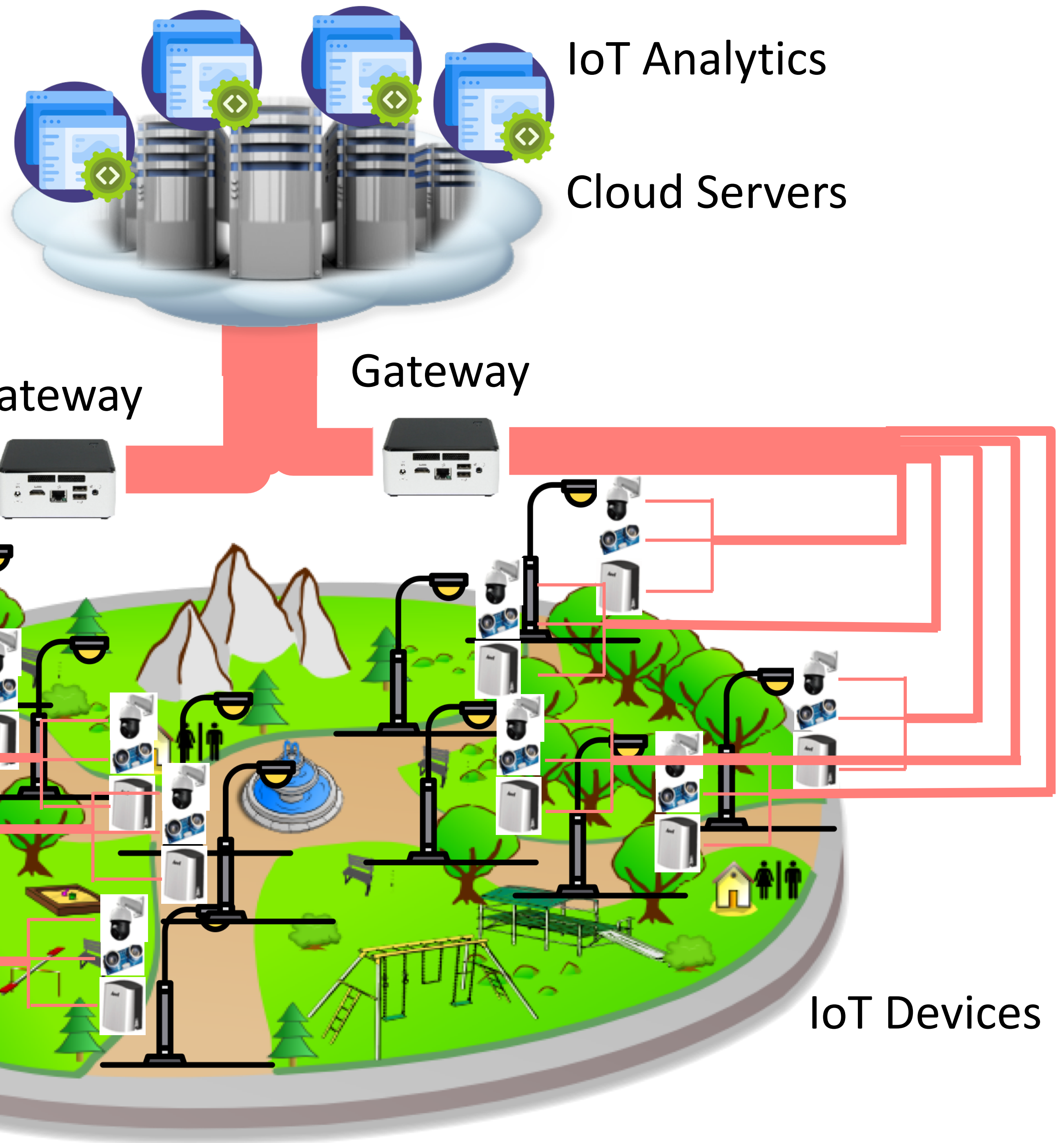
Smart City



Problems of Sending All Data to Cloud

- Excessive Internet access cost
- Degraded QoS due to network congestion
- Heavy burden of computing resource

Smart City

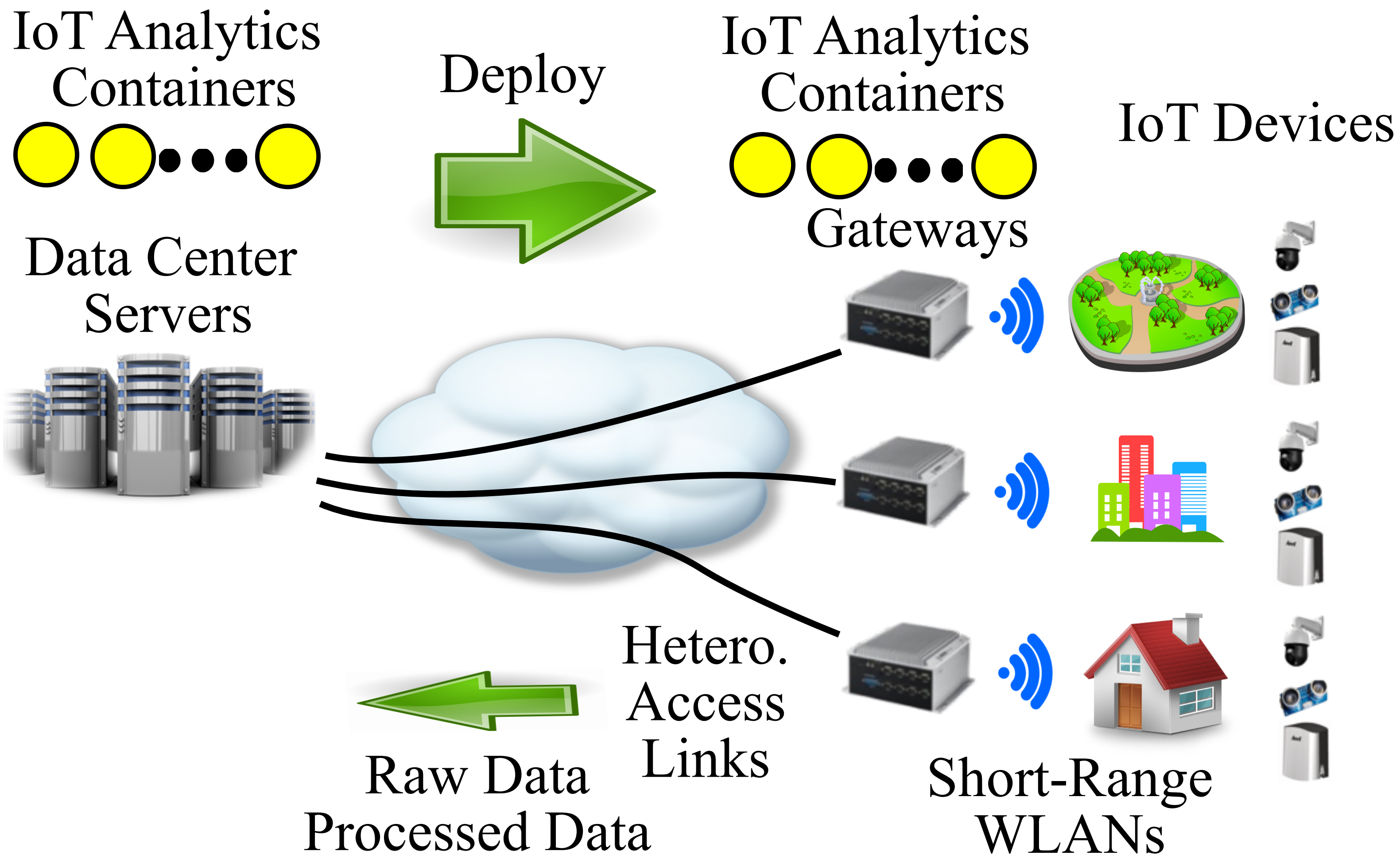


Dynamically Deploy IoT Analytics to Gateways

Advantages:

- Reducing work load of cloud servers
- Reducing upload bandwidth consumption
- Better utilizing download bandwidth of access links

Dynamically Deploy IoT Analytics to Gateways



Research Problems

Image Download Problem

selects additional IoT analytics to deploy on 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

Contributions

Achieve as high as 1 weighted QoS level in the scale of [0,1]

We deploy as many IoT analytics containers on the gateway as possible.

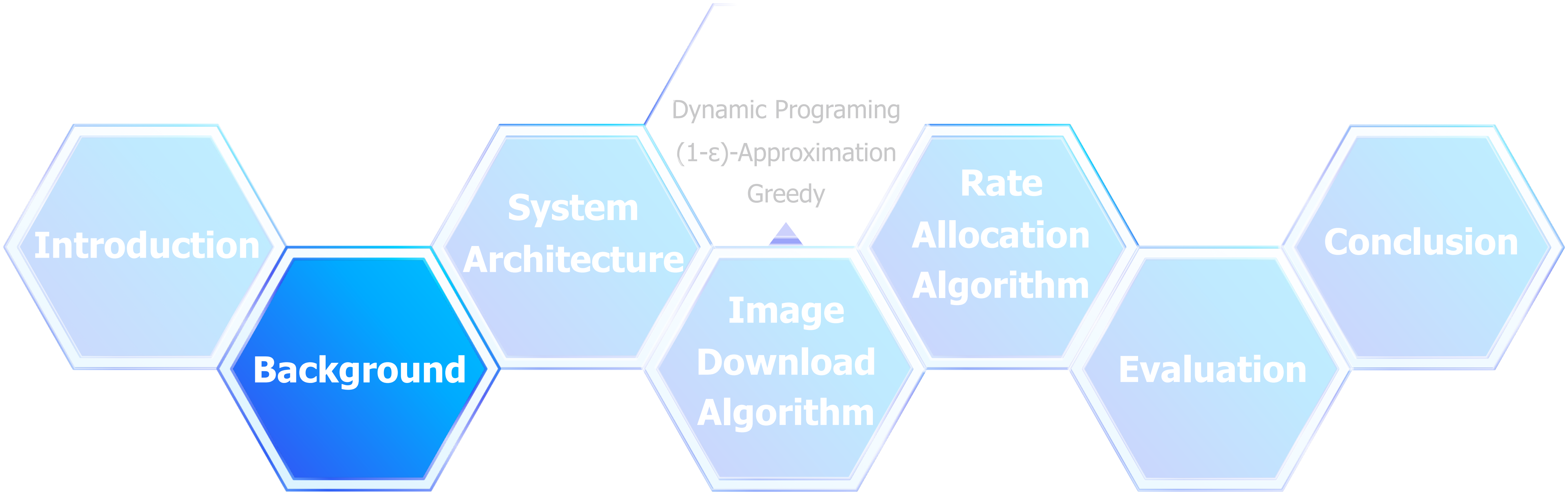
Image download problem

Our heuristic algorithms saves as much upload bandwidth as the optimal algorithm while achieving similar QoS levels.

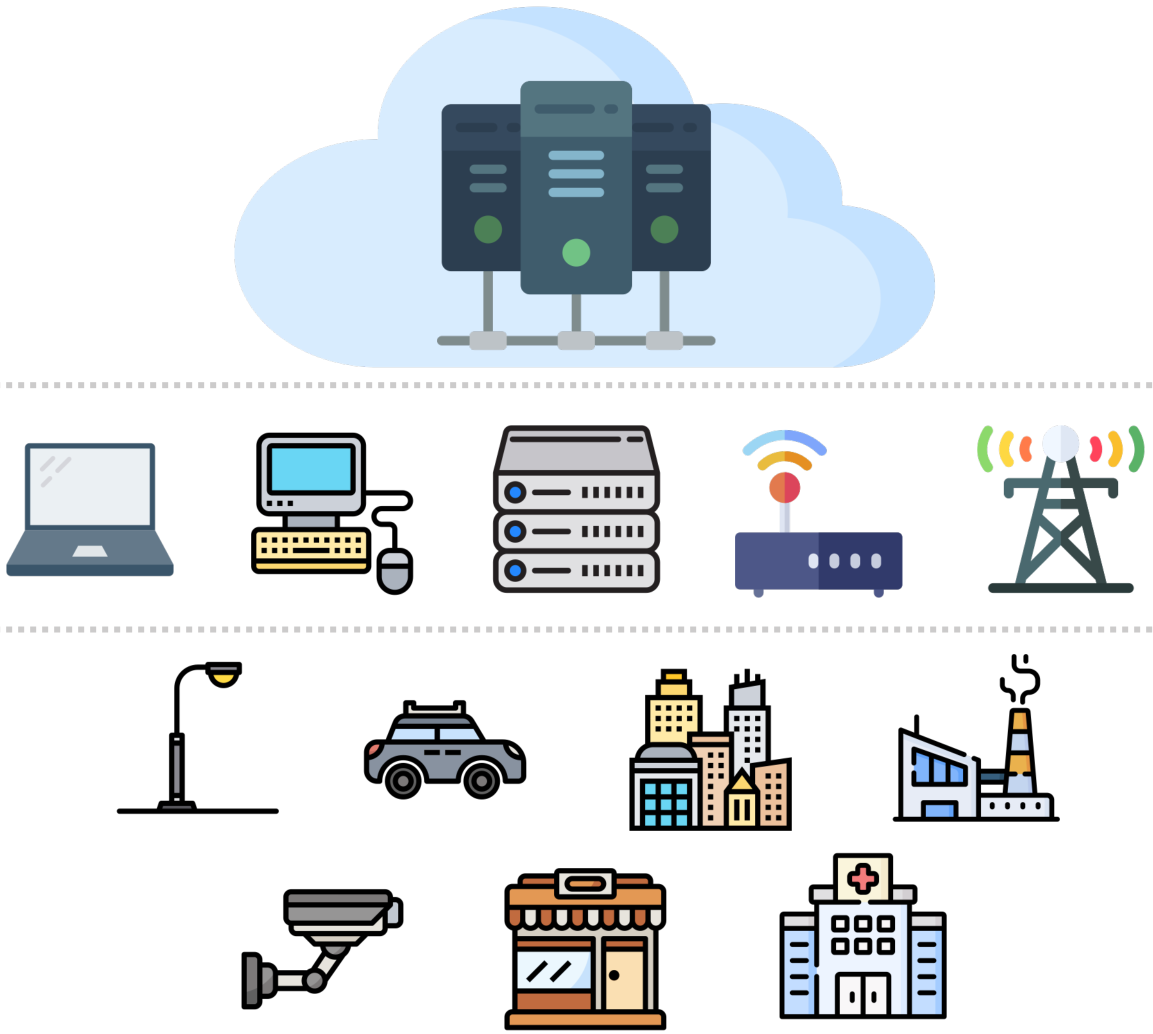
Rate Allocation Problem

Our proposed algorithm outperforms the two baseline algorithms

1. by 23% and 37% in weighted QoS levels,
2. by 168% and 74% in utilization of upload bandwidth.


Dynamic Programing
(1-ε)-Approximation
Greedy
Introduction
Background
System Architecture
Image Download Algorithm
Rate Allocation Algorithm
Evaluation
Conclusion

Edge Computing



Far-end

Near-end

Front-end

	Edge Computing	IoT
Far-end	Cloud Servers	Cloud Servers
Near-end	Gateways/ Edge Servers	Gateways
Front-end	Edge Devices	IoT Devices

Container Images

Pack IoT analytics into container images can

- Easily package heterogeneous environments
- Make IoT analytics be easily and Rapidly deployed

Container engine and management tools:

- Docker
- Kubernetes

Docker

Operating System

Libraries

Environment Variables

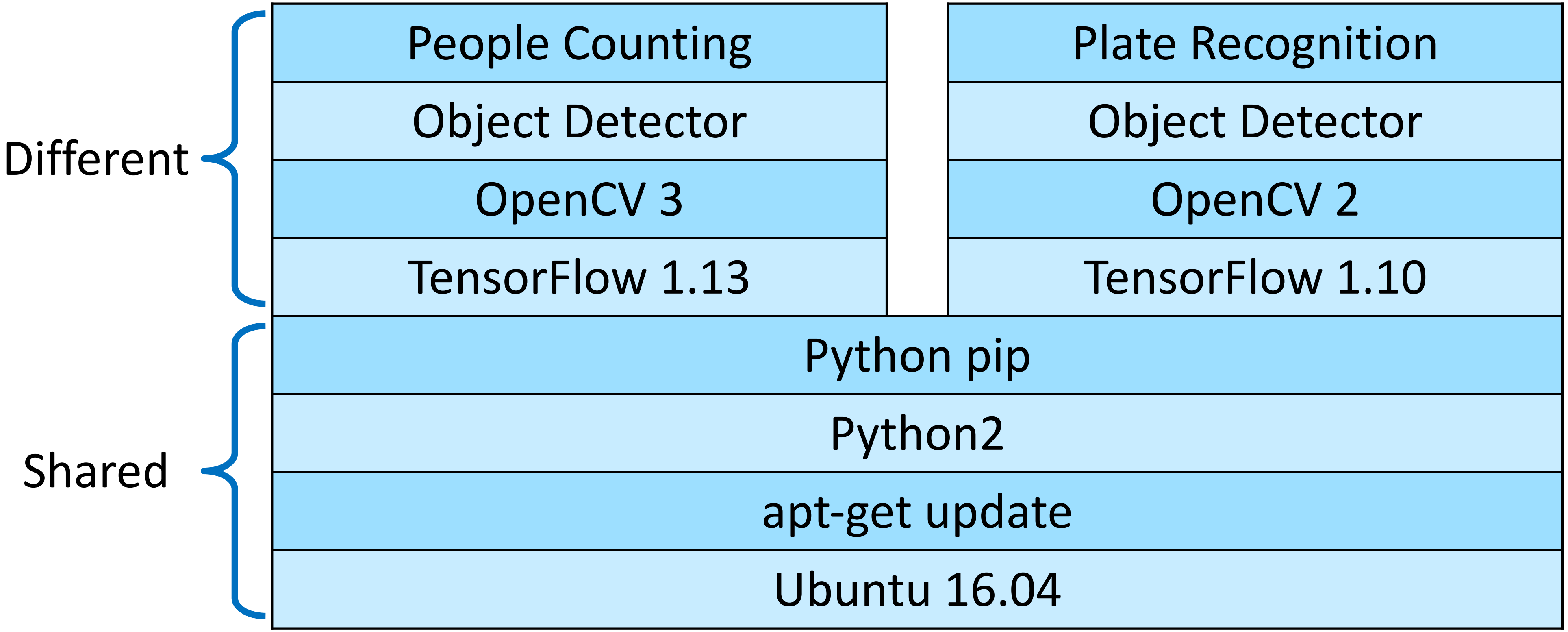
Codes

Base Image

Container Image

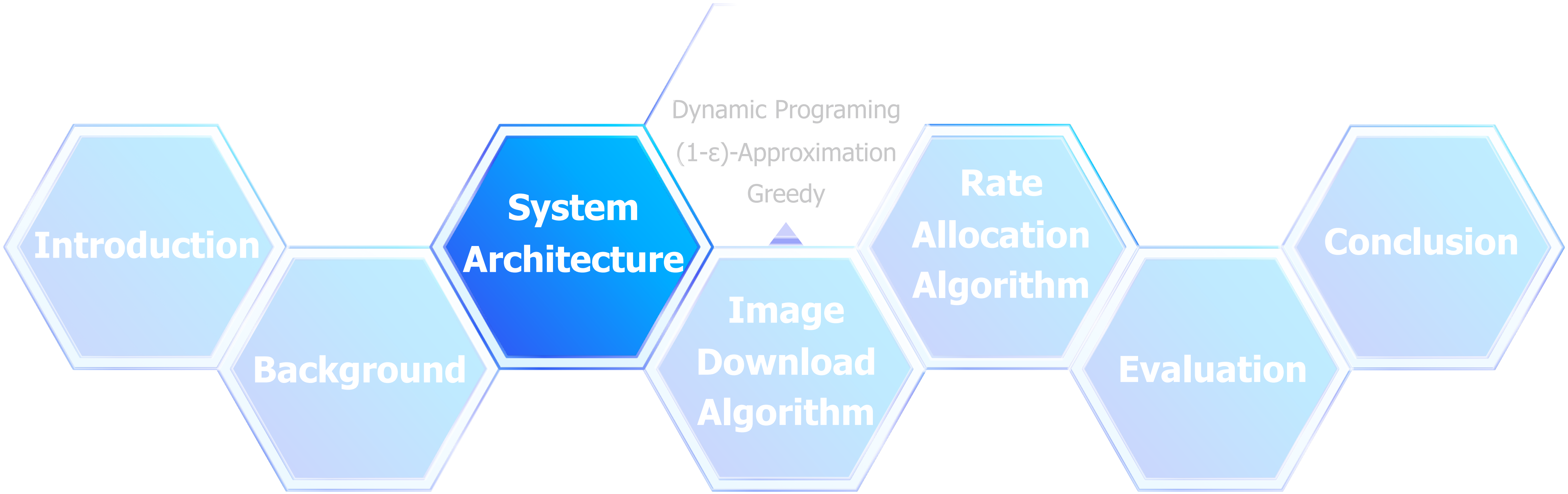
Image layers
Operating System
Libraries
Environment Variables
Codes
Base Image

Layer Dependency



Kubernetes

- Defines different roles for each IoT device
- Combine each IoT device in clusters
- Monitor the condition of analytics containers
- Monitor resources of each IoT device
- Automatically restart the dead containers


Introduction
Background
System Architecture
Image Download Algorithm
Dynamic Programing
(1-ε)-Approximation
Greedy
Rate Allocation Algorithm
Evaluation
Conclusion

Implemented Algorithms

IDA

Image Download Algorithm

1. Dynamic Programming Algorithm (IDA_D)
2. $(1-\varepsilon)$-Approximation Algorithm (IDA_A)
3. Greedy Algorithm (IDA_G)

RAA

Rate Allocation Algorithm

1. Rate Allocation Algorithm (RAA)
2. Weighted Allocation Algorithm (WA)
3. Unweighted Allocation Algorithm (UA)

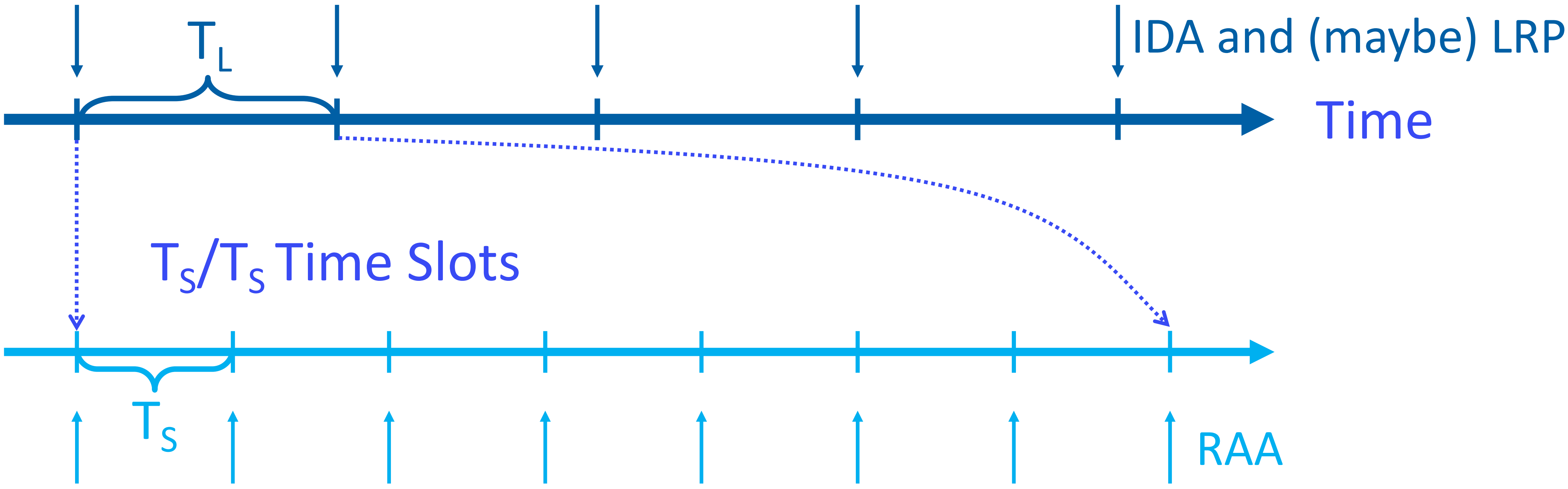
LRP

Layer Replacement Policy

Default setting:

Least-Recently-Used (LRU)

Execution Time of Algorithms



Basic System Architecture

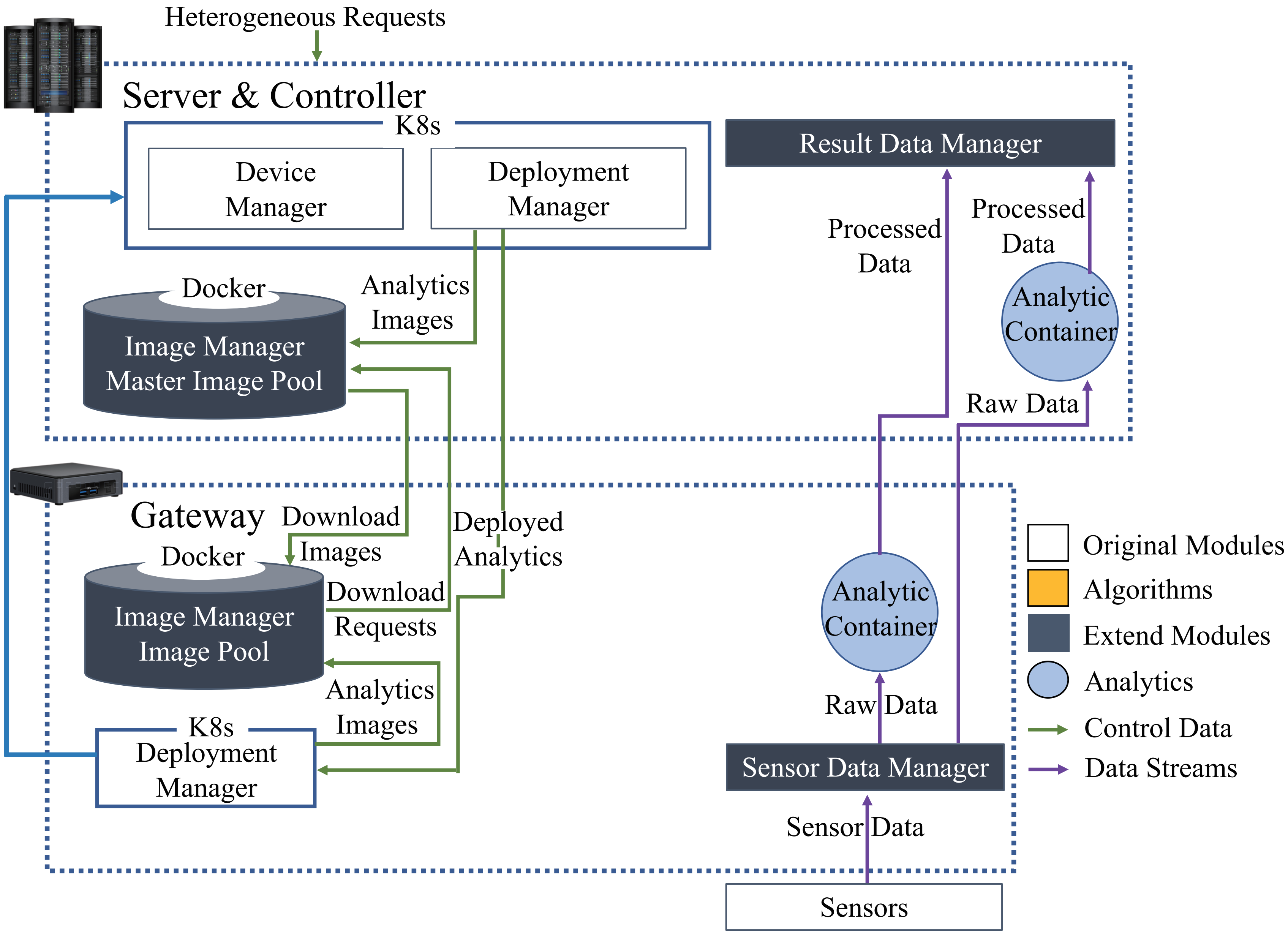
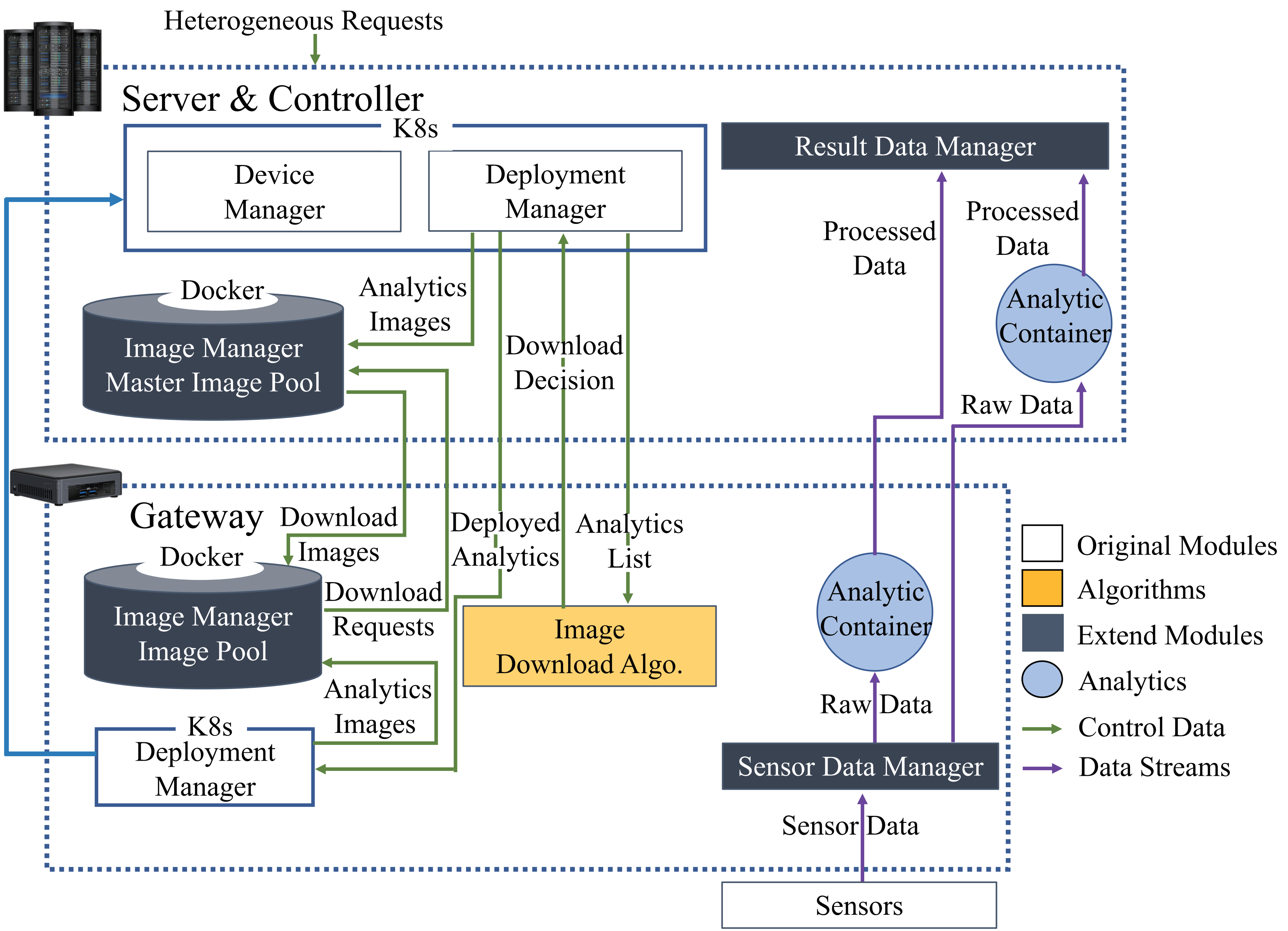
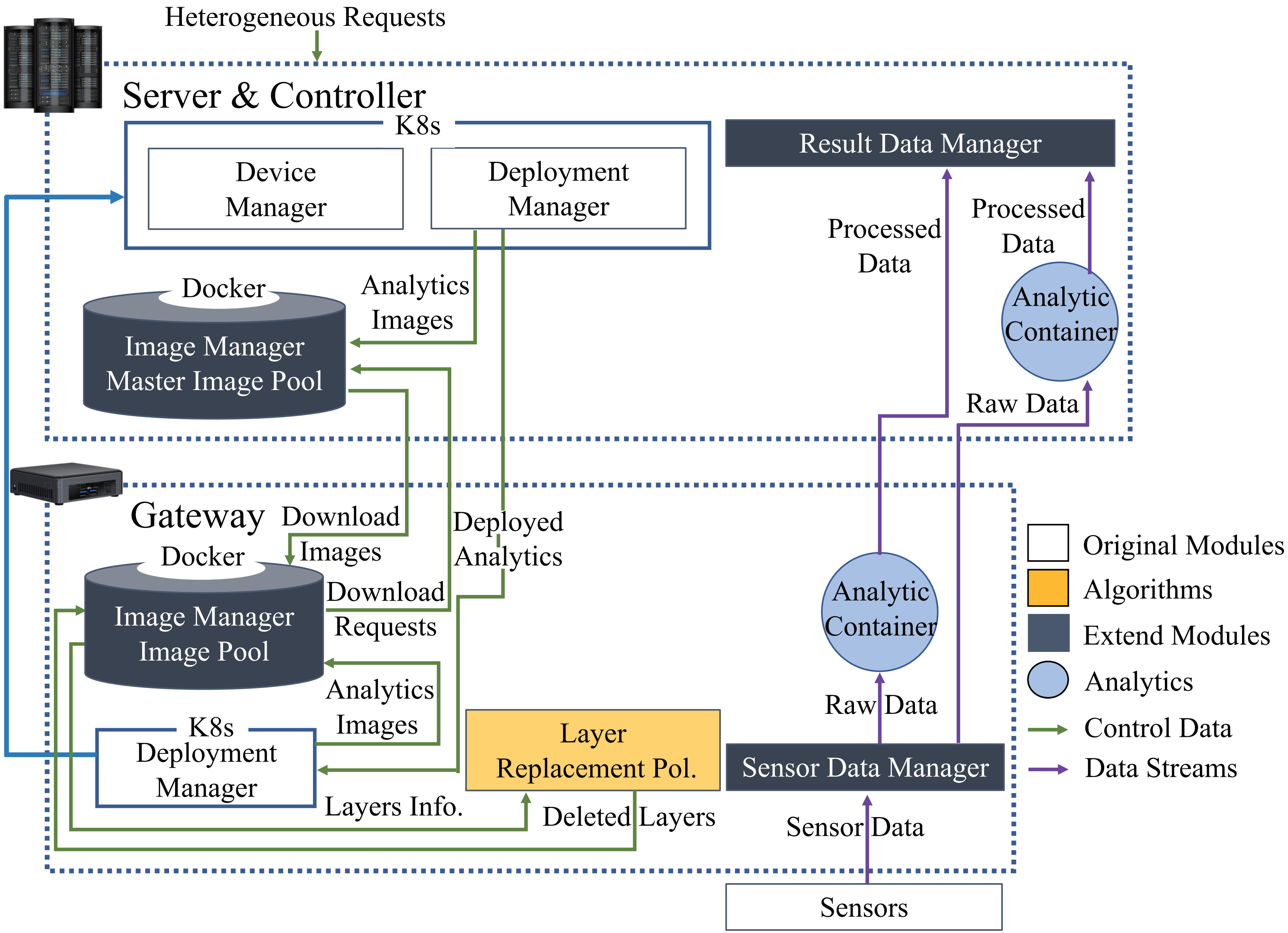


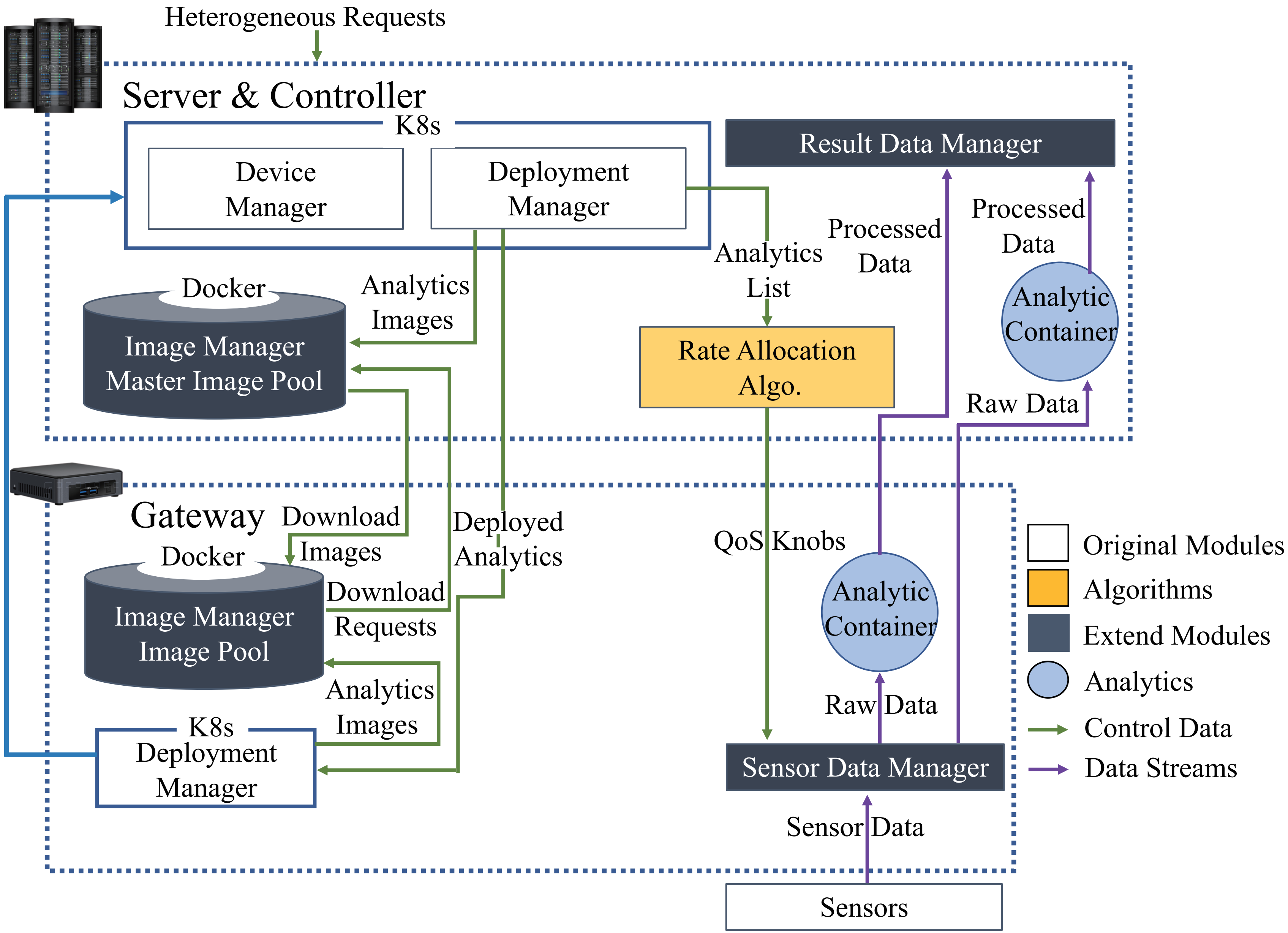
Image Download Algorithm



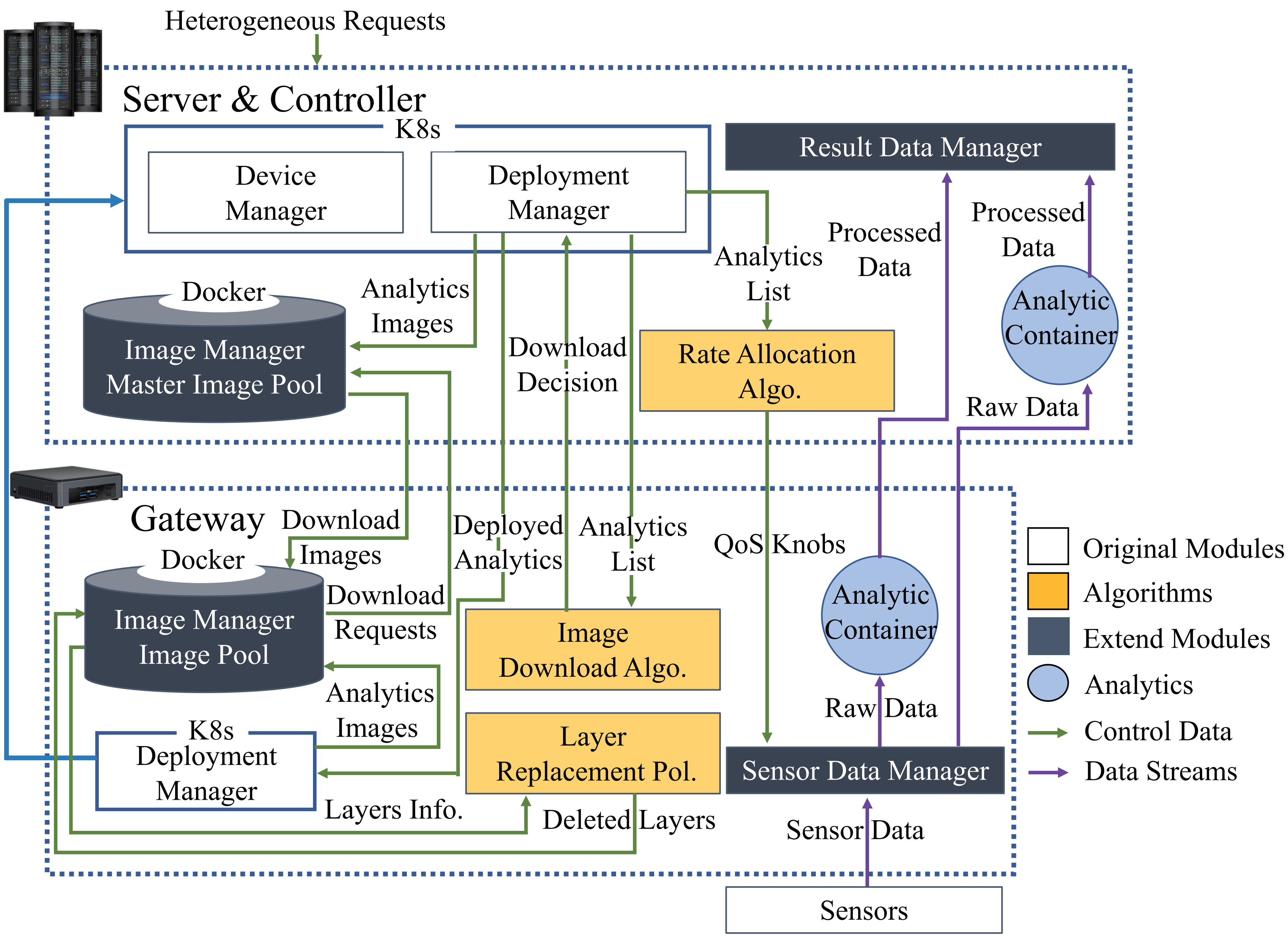
Layer Replacement Policy

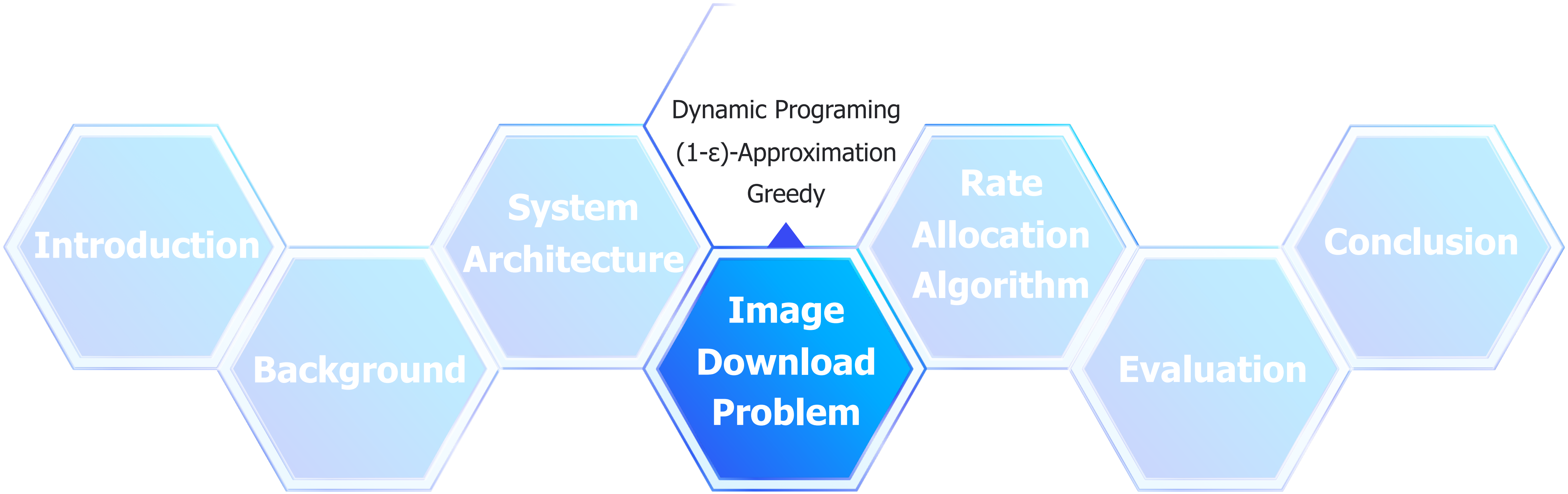


Rate Allocation Algorithm



Overall System Architecture




Introduction
Background
System Architecture
Dynamic Programing
(1-ε)-Approximation
Greedy
Image Download Problem
Rate Allocation Algorithm
Evaluation
Conclusion

Symbols of Image Download Problem

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	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
e_a	Deploying analytic a on gateways
ϵ	Approximation parameter of $(1-\epsilon)$-approximation algorithm

Problem Formulation

Integer Linear Programming (ILP) formulation:

Saved upload bandwidth z

Saved upload bandwidth

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

Deploy decision

Downloaded layer size s

Total layer size

Residual Image pool size

Maximal download amount in T_L

$$s.t. \sum_{a=1}^{A} e_a \sum_{l=1}^{L} m_{a,l}(1 - h_l) u_l \leq \min\{S - \sum_{l=1}^{L} h_l u_l, B_d T_L\}$$

Remaining resource R

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

Dynamic Programming Algorithm ($\mathrm{IDA_D}$)

Time complexity: pseudo-polynomial

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 $D^\star$.

if $R \leq 0$ **or** $n \leq 0$ **then**
    $t^\star = 0, D^\star = \{\}$

Resource is used up or all the containers are checked

else if $\sum_{l=1}^{L} s_{a,l} > R$ **then**
    $t^\star, D^\star = \mathrm{IDA_D}(s, z, R, a-1)$ //Not choose *a*

Remaining resource is not enough to deploy *a*

else
    $t_n, D_n = \mathrm{IDA_D}(s, z, R, a-1)$ //Not choose *a*
    $t_i, D_i = \mathrm{IDA_D}(s, z, R - \sum_{l=1}^{L} s_{a,l}, a-1)$ //Choose *a*
    if $t_i \leq t_n$ **then** //Return the better one
        $t^\star = t_n, D^\star = D_n$
    else
        $t^\star = t_i + z_a, D^\star = D_i \cup a$

Remaining resource is enough to deploy *a*

return $t^\star, D^\star$

$(1-\varepsilon)$-Approximation Algorithm ($\mathrm{IDA_A}$)

Algorithm 4 $\mathrm{IDA_A}$

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'.

initialize: $t' = 0$, $z' = \{z'_a\}$ for all $a = 1, 2, \ldots, A$

$K = \epsilon \frac{\max_{a \in [1,A]} \{z_a\}}{A}$ //Rounding denominator

for $a = 1; a \le A; a{+}{+}$ **do**

    $z'_a = \lfloor \frac{z_a}{K} \rfloor$ //Give the bound by rounding it by K

$t, D' = \mathrm{IDA_D}(s, z', R, A)$ //Run $\mathrm{IDA_D}$ with new saved upload bandwidth $z' = \{z'_1, z'_2, \ldots, z'_A\}$

for $a \in D'$ **do**

    $t' = t' + z_a$

return t', D'

Rounding the saved upload bandwidth of each container

Time complexity: $O(\frac{A^3}{\varepsilon})$

Approximation factor: $(1-\varepsilon)$

Greedy Algorithm (IDA_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.

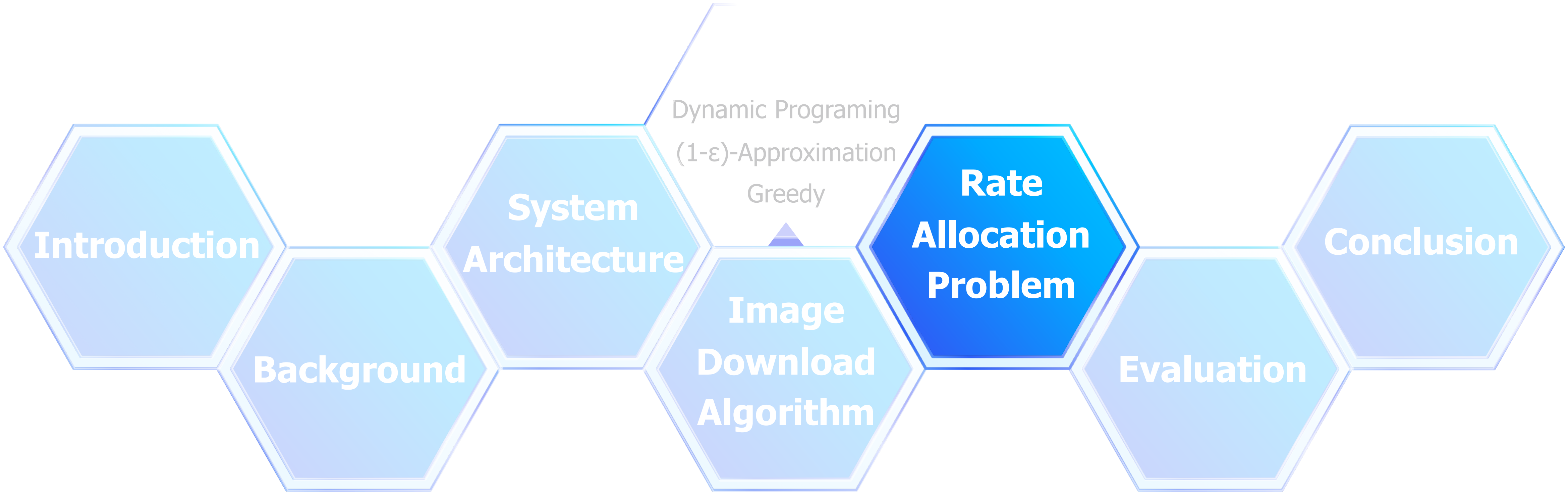
initialize: $t = 0$, $D = \{\}$
for $a = 1; a \leq A; a{+}{+}$ **do**
    $n_a = \left[z_a\right] / \left[\frac{\sum_{l=1}^{L} s_{a,l}}{R}\right]$,
Sort the containers by n_a in the descending order.
for each container a in the descending order **do**
    if $R \leq 0$ **then**
        break
    else if $\sum_{l=1}^{L} s_{a,l} \leq R$ **then**
        $D = D \cup a$
        $t = t + z_a$
        $R = R - \sum_{l=1}^{L} s_{a,l}$
return t, D

Time complexity: $O(A \log A)$

Sort the saved upload bandwidth normalized to the consumed download bandwidth

Remaining resource is used up

Remaining resource is enough to deploy *a*


Dynamic Programing
(1-ε)-Approximation
Greedy
Introduction
Background
System Architecture
Image Download Algorithm
Rate Allocation Problem
Evaluation
Conclusion

Symbols of Rate Allocation Algorithm

Symbol	Description
A	Number of IoT analytics
T_S	Allocation Algorithm time slot duration
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
$q_a(\cdot)$	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

Problem Statement and Formulation

The problem can be mathematically written as:

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

Processed data b/w

Raw data b/w

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

Upload network bandwidth

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

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 $\forall a \in \mathbf{A}_C \cup \mathbf{A}_G$.

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

Time complexity: $O(\frac{1}{\alpha}|A_C \cup A_G|)$

Calculate $g(k_a) = \frac{\text{weighted QoS value}}{\text{bandwidth}}$ of each container

Repeatedly find the maximal $g(k_a)$ of each container, and increase the QoS knob k_a by the step size α.

Introduction

Background

System Architecture

Dynamic Programing
(1-ε)-Approximation
Greedy

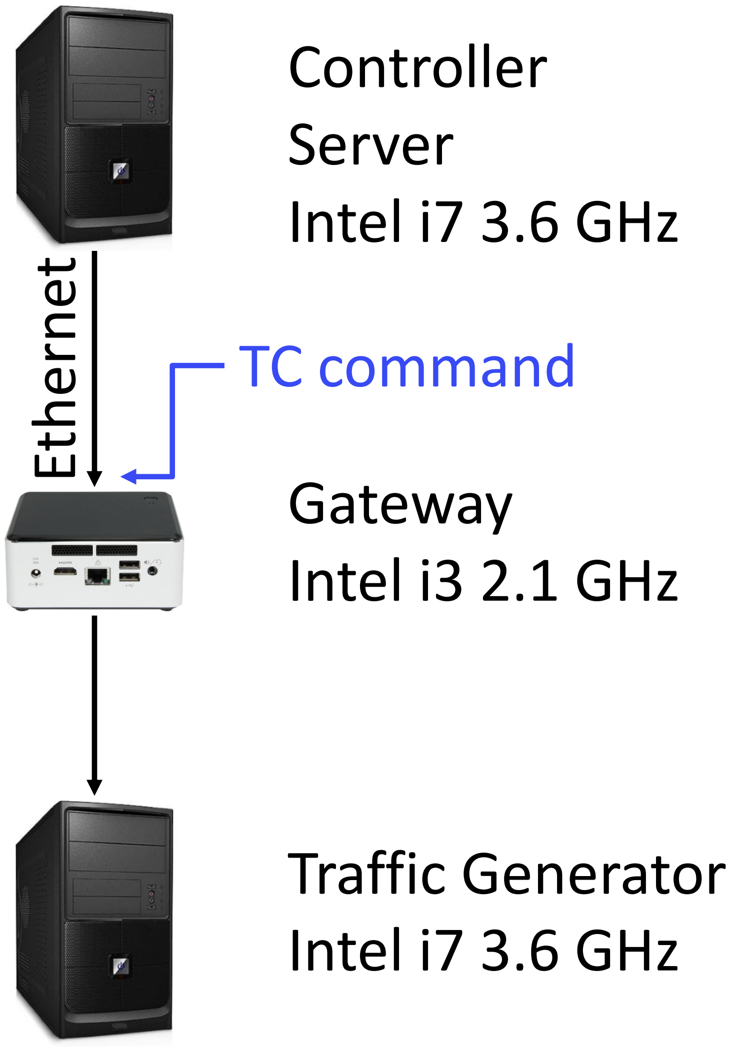
Image Download Algorithm

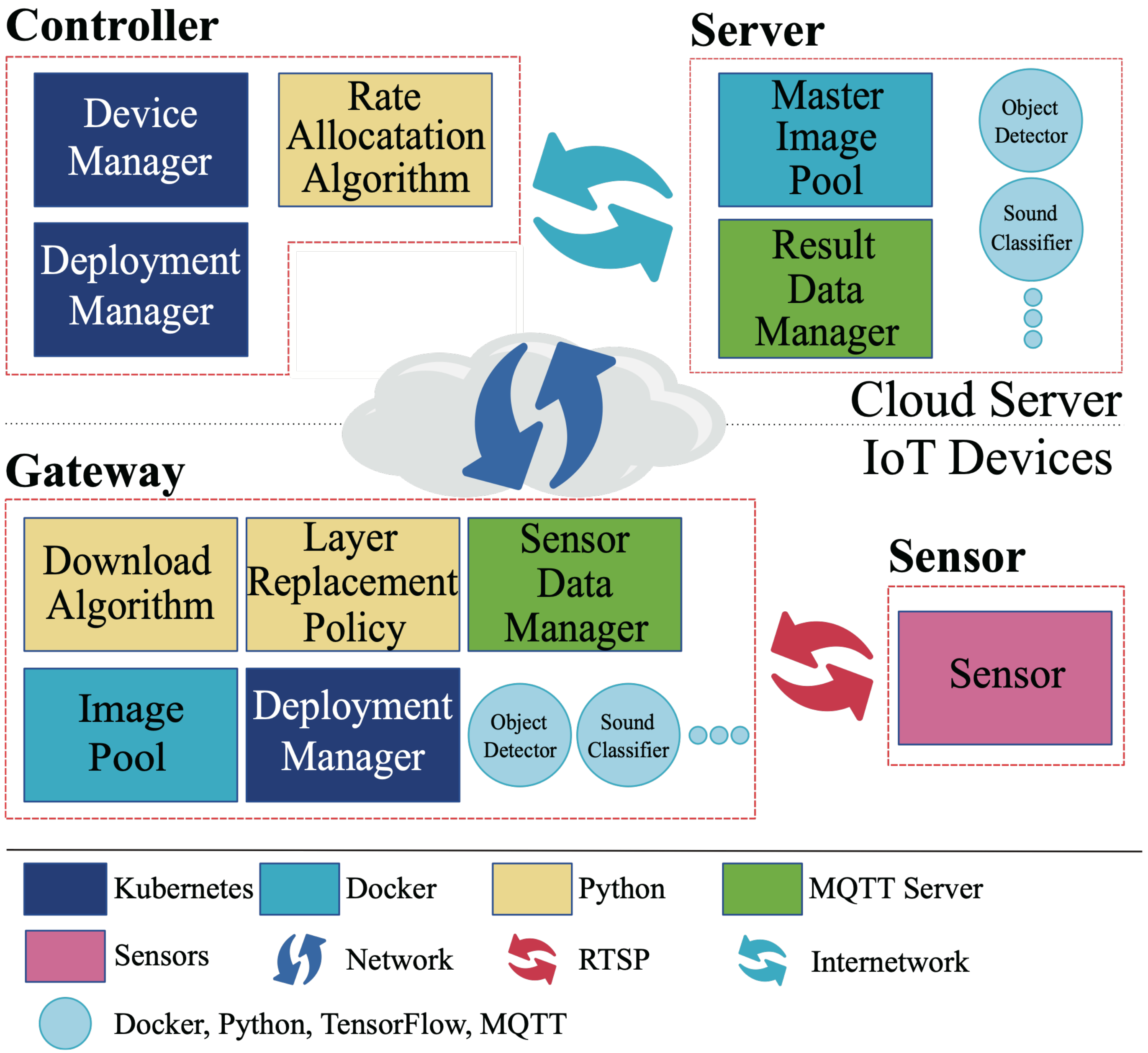
Rate Allocation Algorithm

Evaluation

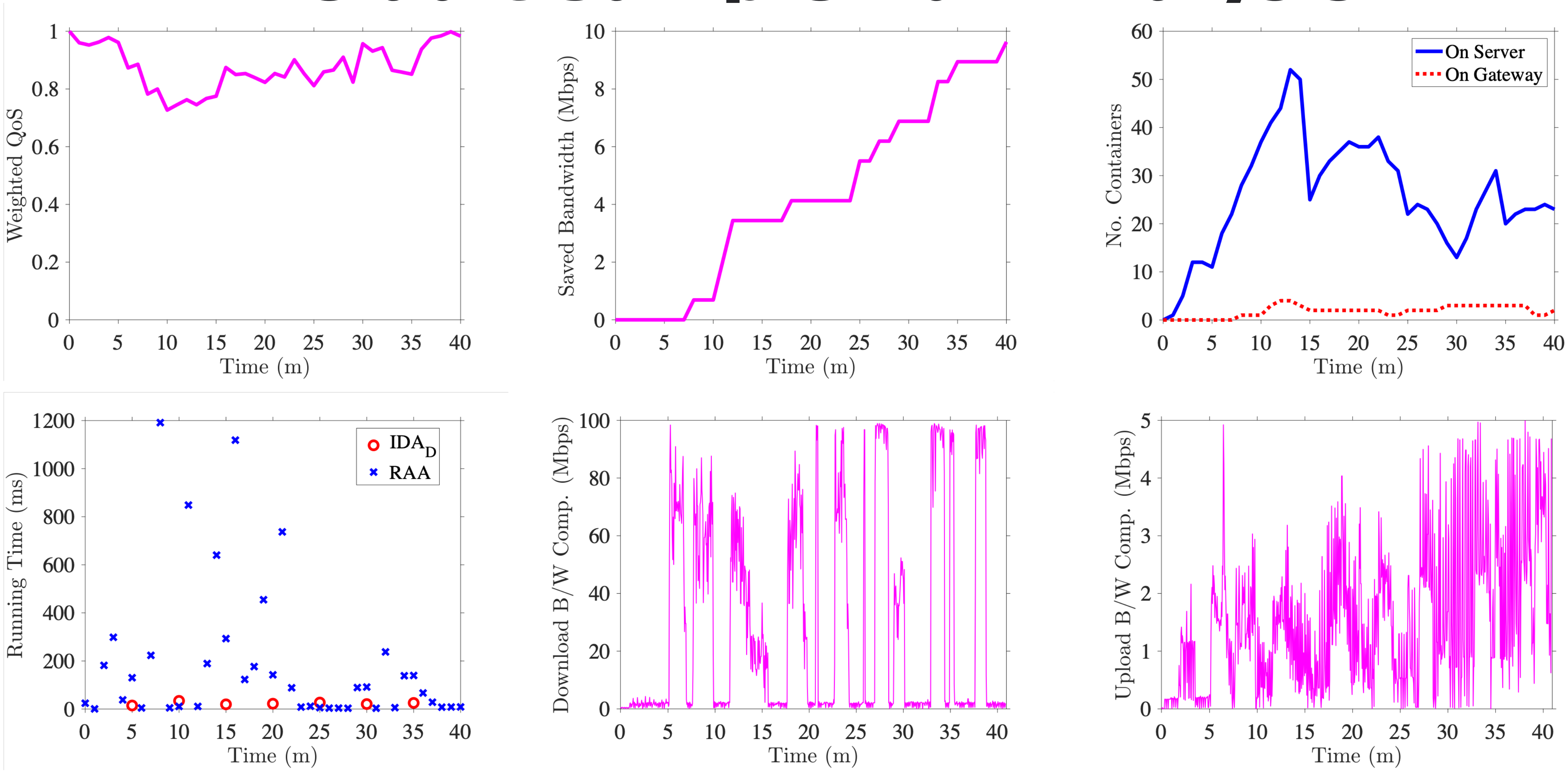
Conclusion

Testbed


Controller
Server
Intel i7 3.6 GHz
Ethernet
TC command
Gateway
Intel i3 2.1 GHz
Traffic Generator
Intel i7 3.6 GHz

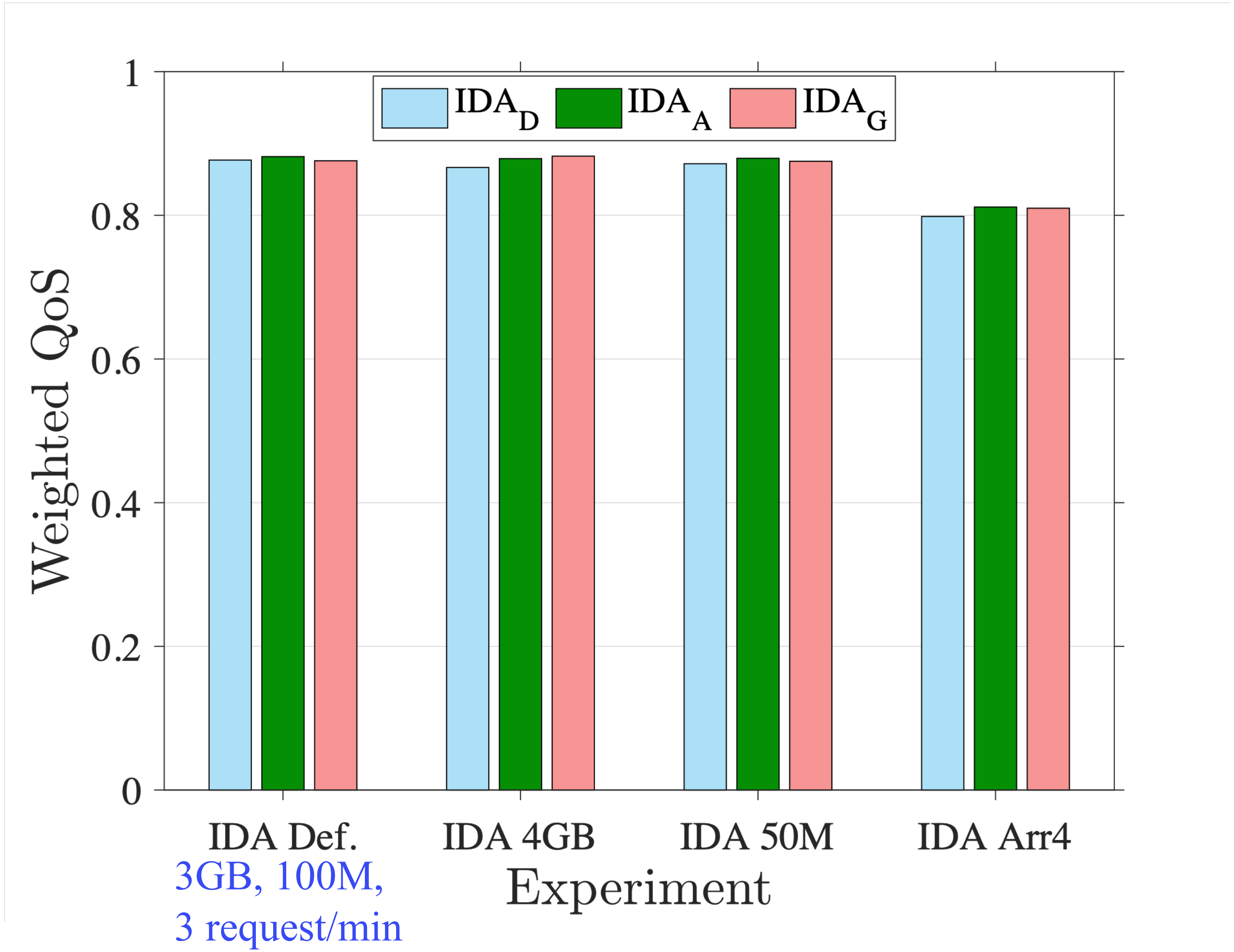

Controller
Device Manager
Rate Allocatation Algorithm
Deployment Manager
Server
Master Image Pool
Object Detector
Sound Classifier
Result Data Manager
Cloud Server
IoT Devices
Gateway
Download Algorithm
Layer Replacement Policy
Sensor Data Manager
Image Pool
Deployment Manager
Object Detector
Sound Classifier
Sensor
Sensor
Kubernetes
Docker
Python
MQTT Server
Sensors
Network
RTSP
Internetwork
Docker, Python, TensorFlow, MQTT

Default Sample Run Analysis



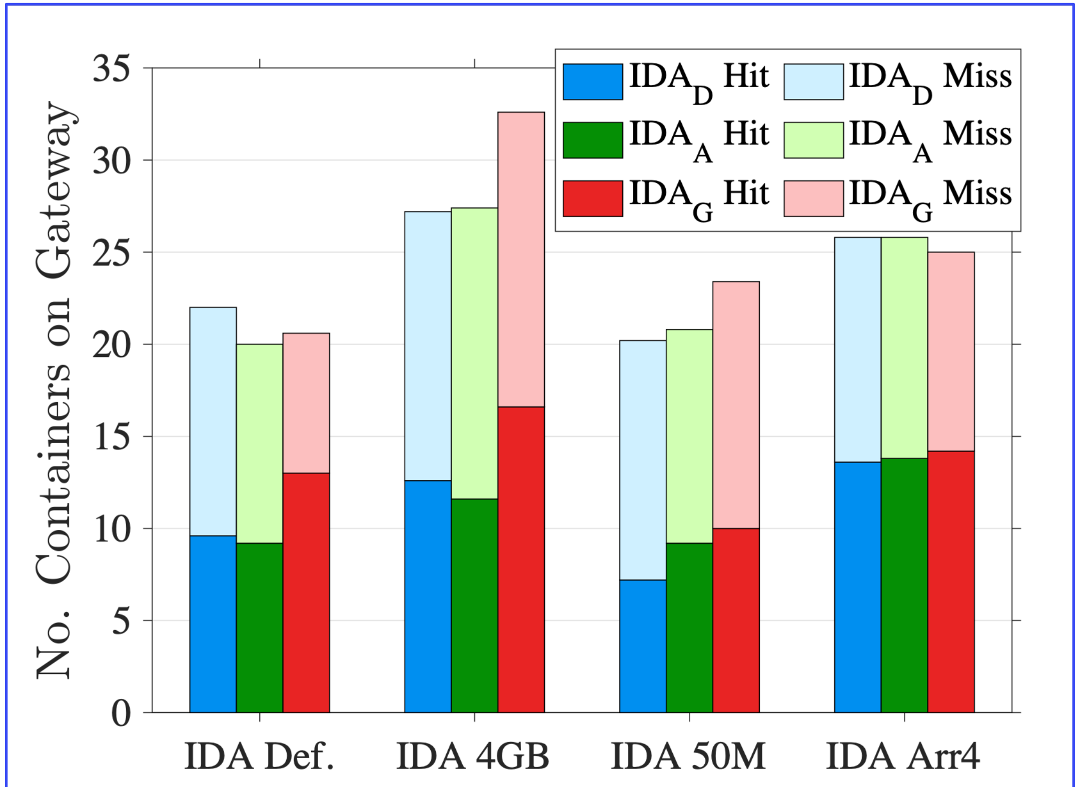
- ➢ Our proposed algorithms achieve high weighted QoS: 0.72 - 1
- ➢ Negligible running time of IDA_D and RAA: at most 1200 ms
- ➢ Do not overload the upload and download bandwidth

Weighted QoS of IDA

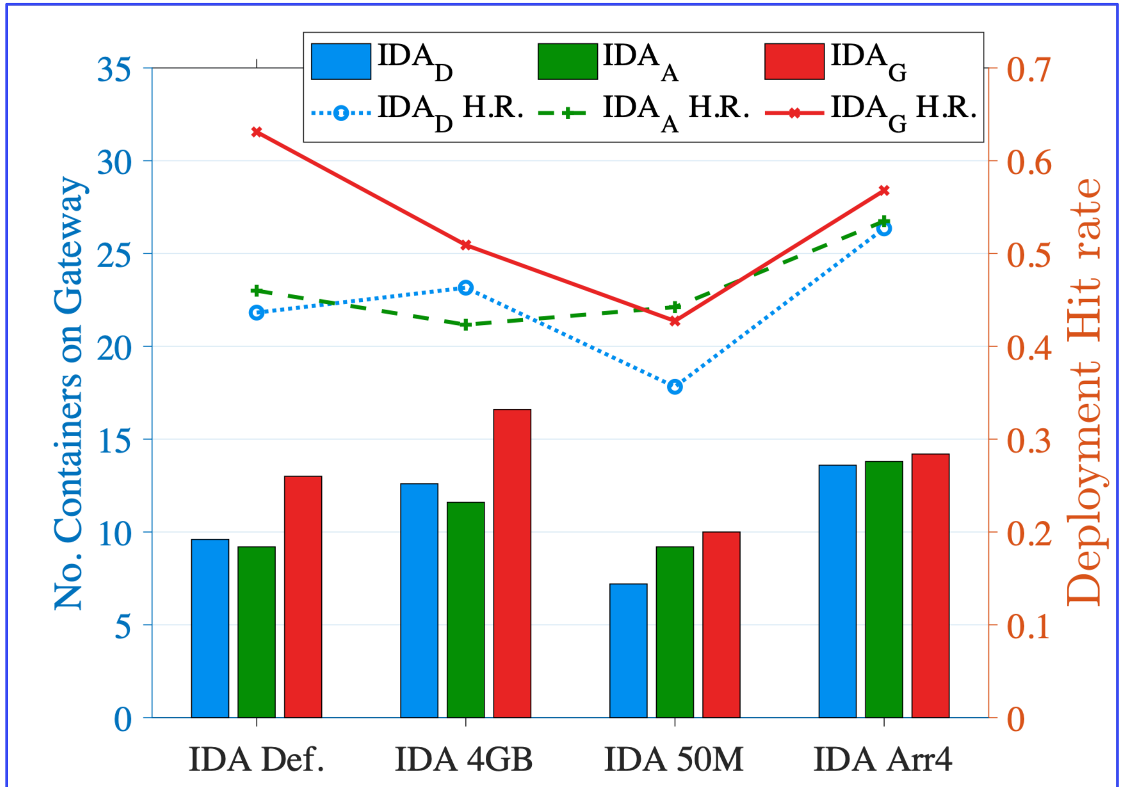


- IDA$_D$, IDA$_A$, and IDA$_G$ all have high weighted QoS.
- # of the containers deployed on the gateway is much less than that on the cloud server.

Deployed Number of IDA



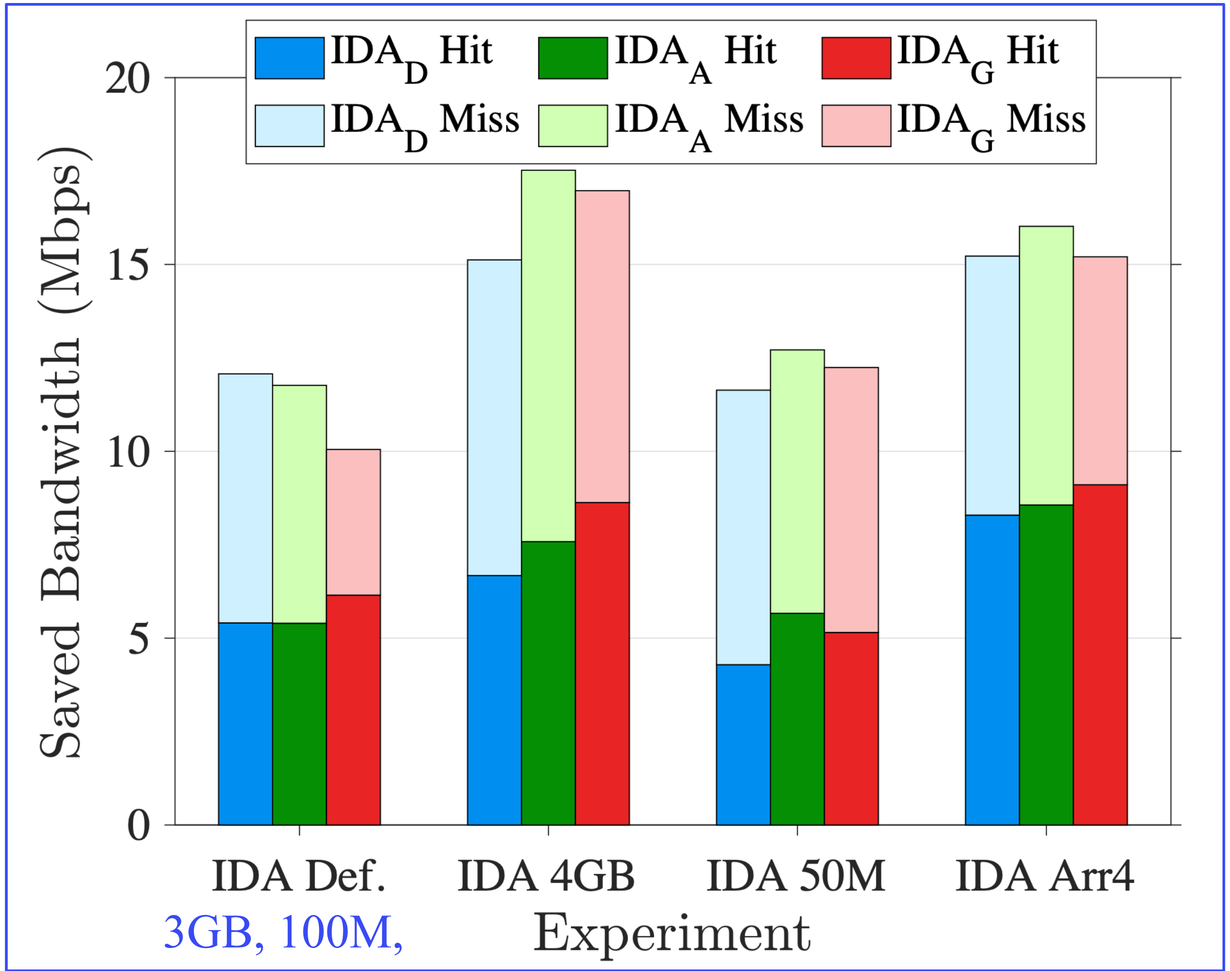
3GB, 100M, 3 request/min

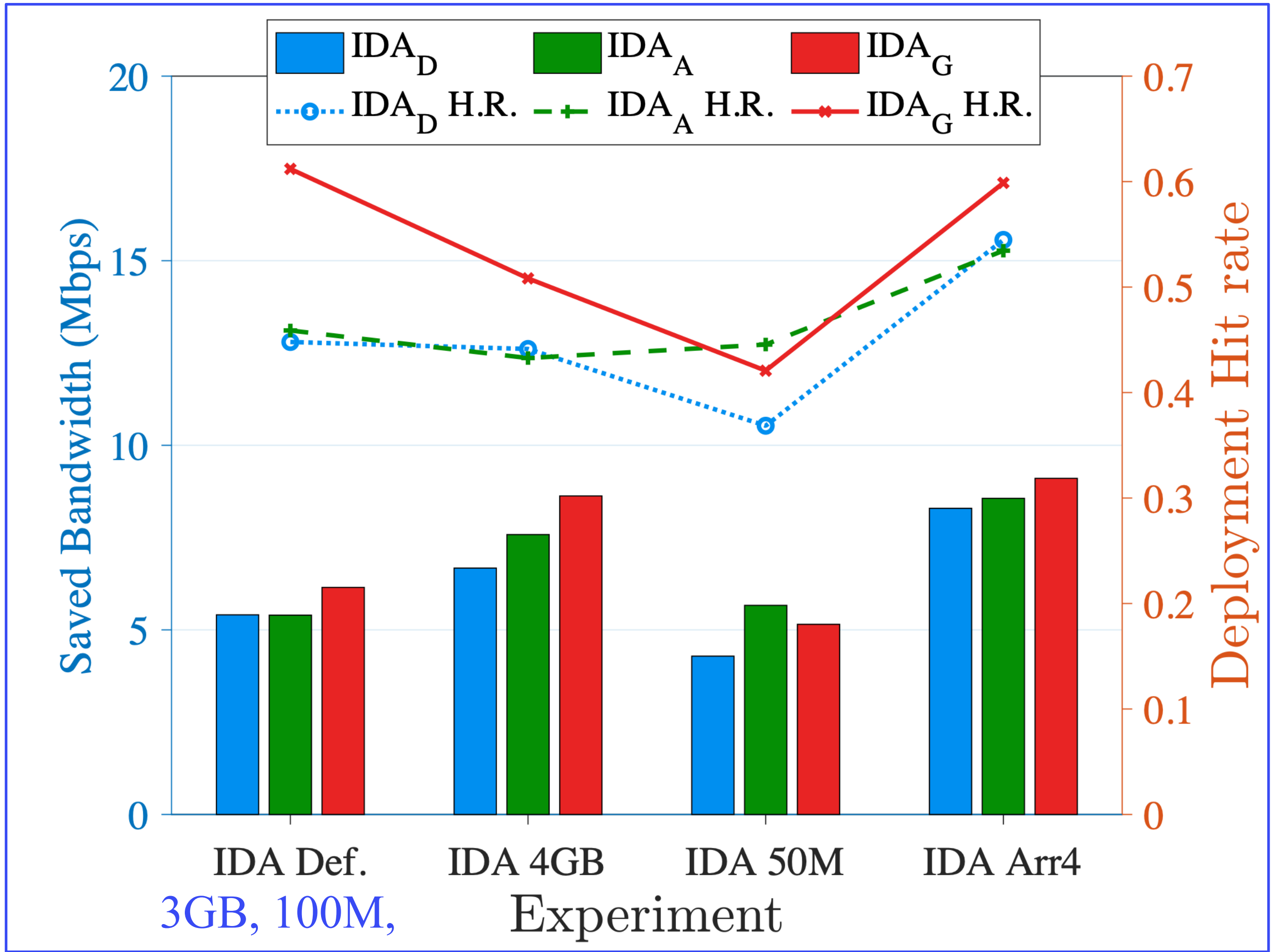


3GB, 100M, 3 request/min

- IDA_D, IDA_A, and IDA_G all consider “current” condition.
- IDA_G has the highest hit rate.
- IDA_G outperforms IDA_D and IDA_A by
  - 35% and 41% in IDA Def,
  - 32% and 43% in IDA 4GB,
  - 39% and 9% in IDA 50M.

Saved Upload B/W of IDA





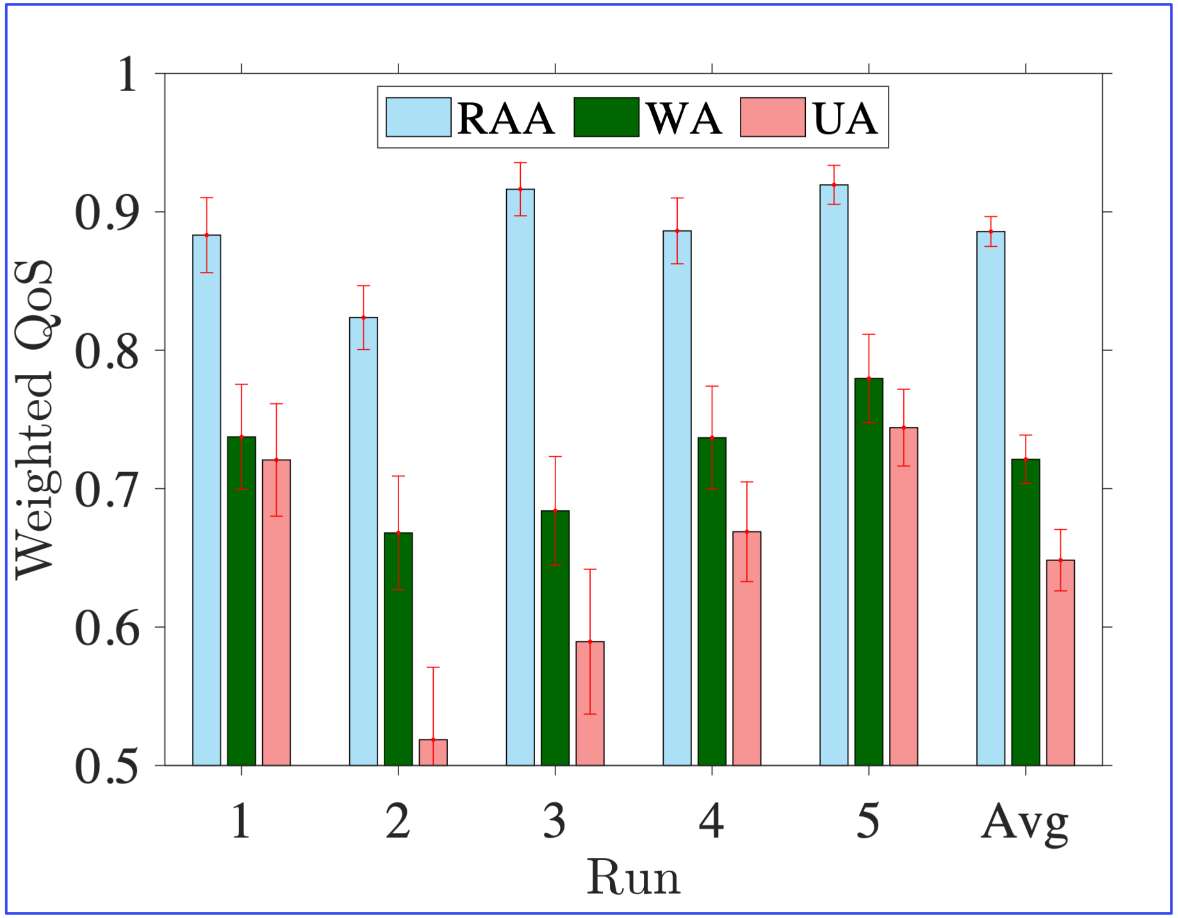
3GB, 100M,
3 request/min

3GB, 100M,
3 request/min

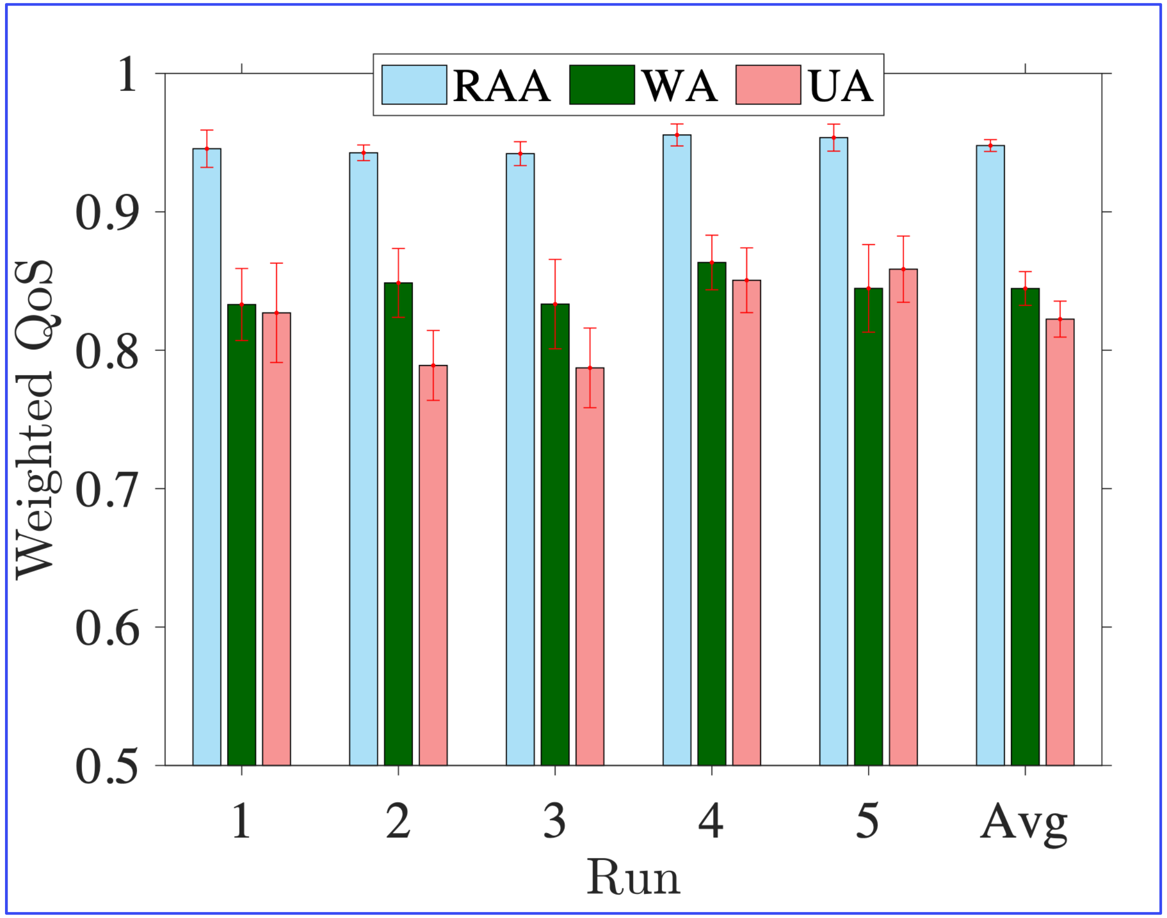
- IDA_D, IDA_A, and IDA_G all perform well in in terms of saved upload bandwidth.
- IDA_D, IDA_A, and IDA_G all consider “current” condition.
- IDA_G has the highest hit rate.

Weighted QoS of RAA

5-Mbps upload B/W,



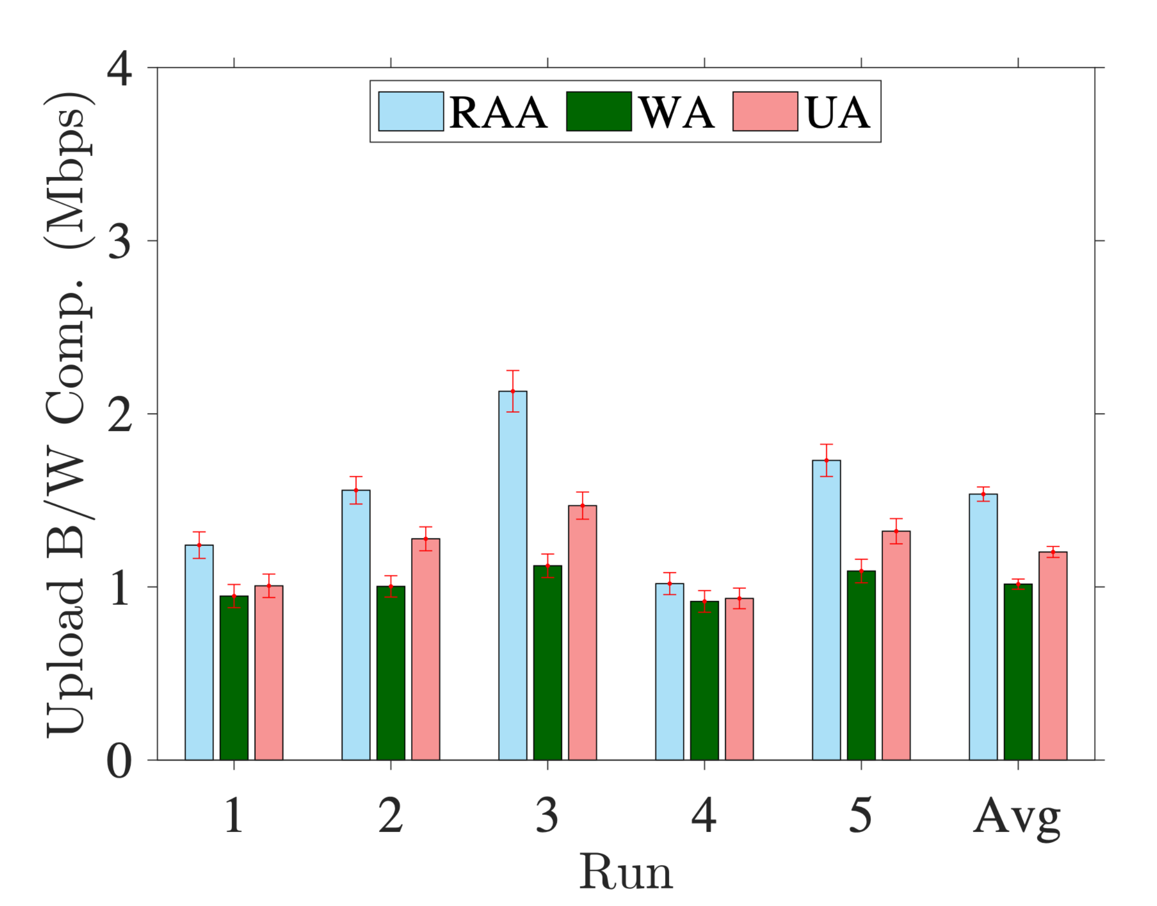
10-Mbps upload B/W,



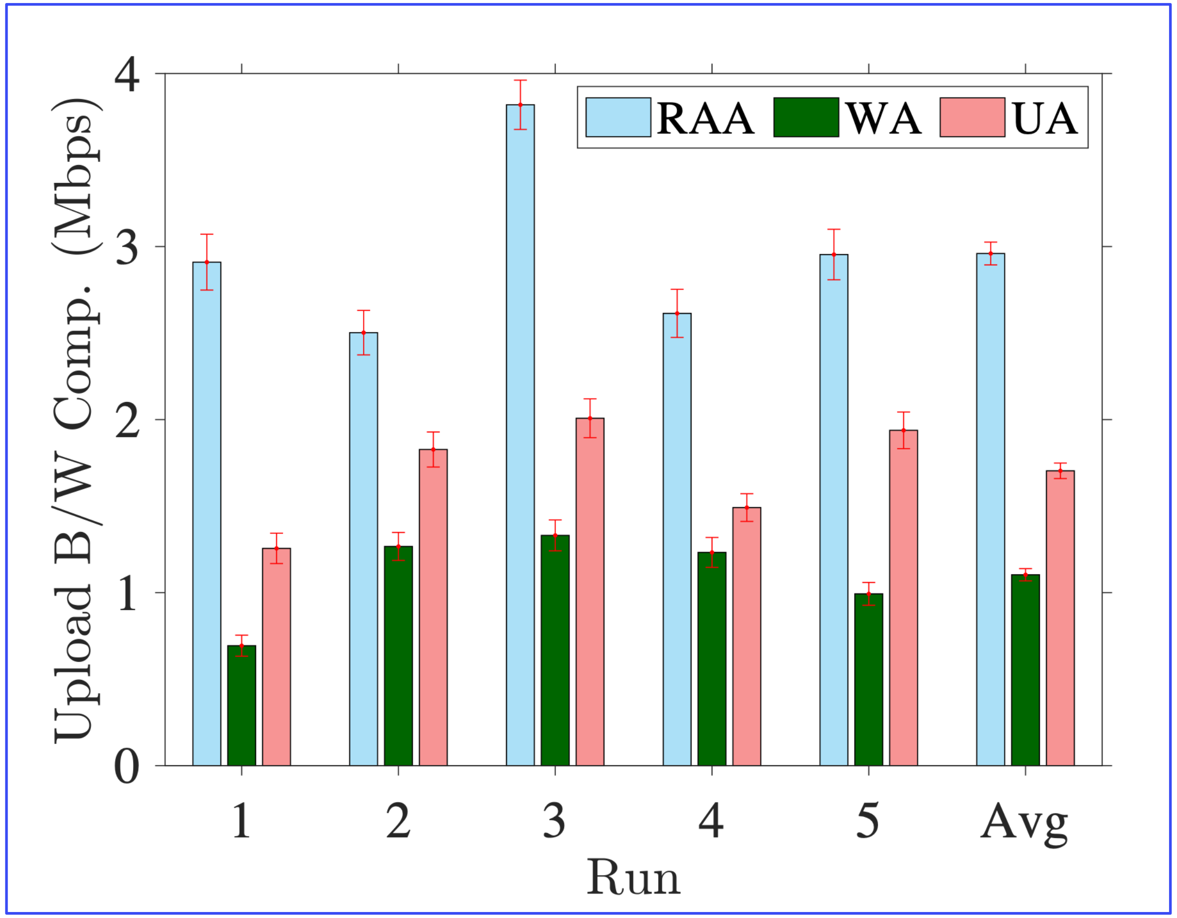
- RAA algorithm outperforms the WA and UA algorithms by
  - about 23% and 37% in 5-Mbps upload B/W,
  - about 12% and 15% in 10-Mbps upload B/W.

Utilization of Upload B/W of RAA

5-Mbps upload B/W,



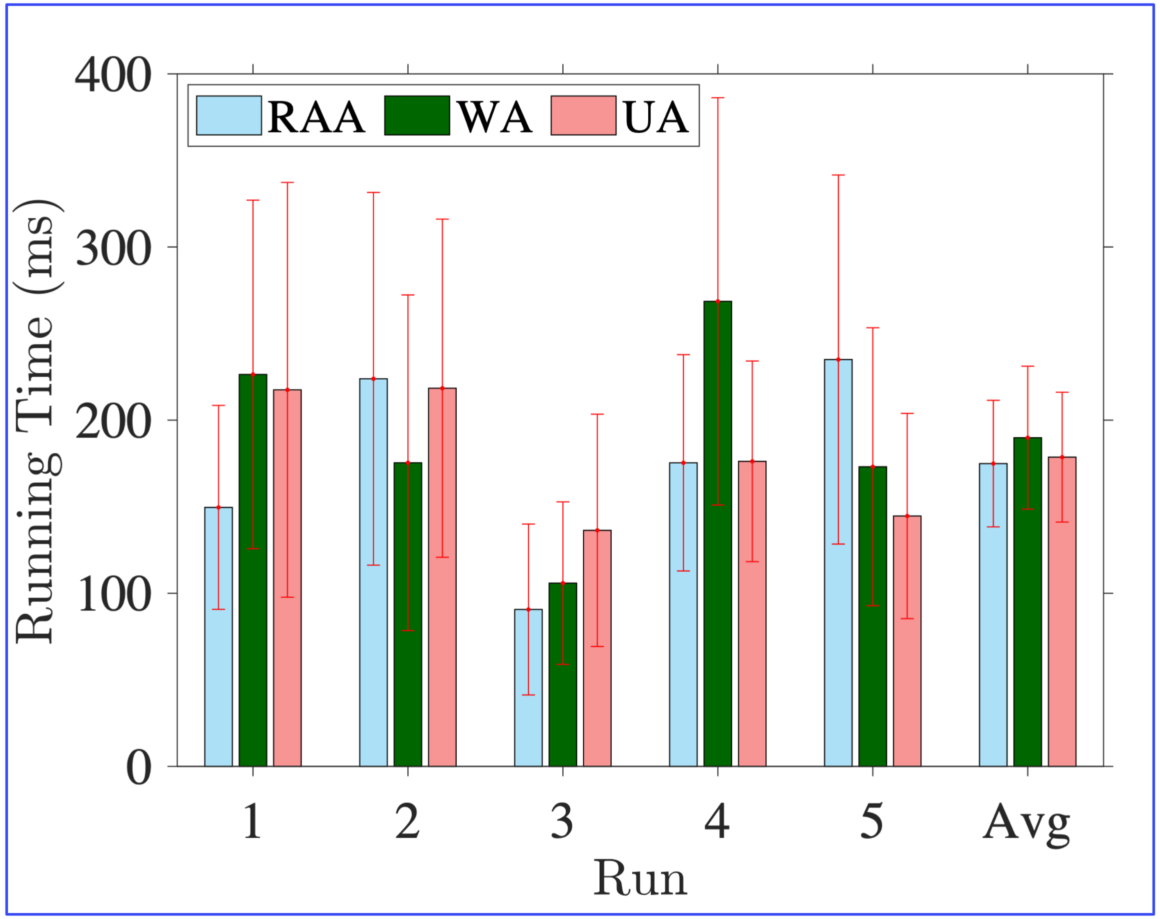
10-Mbps upload B/W,



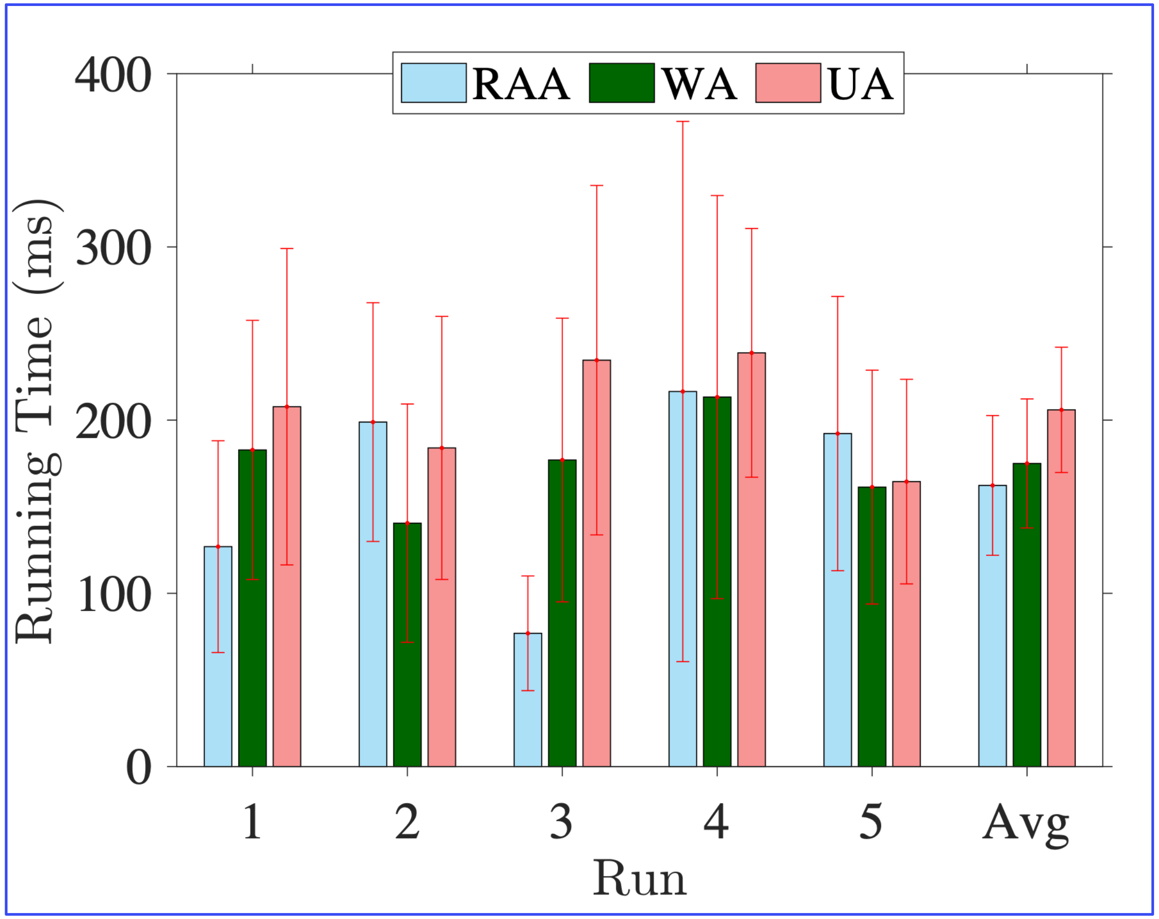
- RAA algorithm outperforms the WA and UA algorithms by
  - about 51% and 28% in 5-Mbps upload B/W,
  - about 168% and 74% in 10-Mbps upload B/W.

Running Time of RAA

5-Mbps upload B/W,



10-Mbps upload B/W,



➢ All the RAA algorithms averagely take short running time: < 300 ms

Introduction

Background

System Architecture

Dynamic Programing
(1-ε)-Approximation
Greedy

Image Download Algorithm

Rate Allocation Algorithm

Evaluation

Conclusion

Conclusion

1. We evaluated our proposed algorithms on our campus and lab testbeds built upon several open-source projects.
2. The experiment results show our proposed system and algorithms increase the overall QoS level (between 0.72 and 1 in the scale of [0,1]) without overloading the network and gateway (terminate in < 1.2 s).
3. For image download problem, our heuristic algorithms saves as much upload bandwidth as the optimal algorithm while achieving similar QoS levels.
4. For rate allocation problem, our proposed algorithm outperforms the two baseline algorithms by
   - 23% and 37% in weighted QoS levels
   - 168% and 74% in utilization of upload bandwidth

Future Works

- Utilizing the source code of Docker engine for better performance.
- Exploring more probability of different layer replacement policies.
- Larger experiments driven by real traces from our campus testbed.

Q&A

Yu-Jung Wang
yurongwang.tw@gmail.com

NAME OF YOUR

NAME OF YOUR

Layer Replacement Policy

Classical layer replacement policies:

- Least-Recently-Used (LRU)
- Most-Recently-Used (MFU)
- Least-Frequently-Used (LFU)
- Most-Frequently-Used (MFU)

Problem Statement

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.*

Problem Formulation

Integer Linear Programming (ILP) formulation:

Raw data b/w

Processed Data b/w

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

Deploy decision

Total Image size

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

Image pool size

Total layer size

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

Maximal download amount in T_L

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

New Problem Formulation

Integer Linear Programming (ILP) formulation:

Save upload bandwidth

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

Deploy decision

Maximal download amount

Residual Image pool size

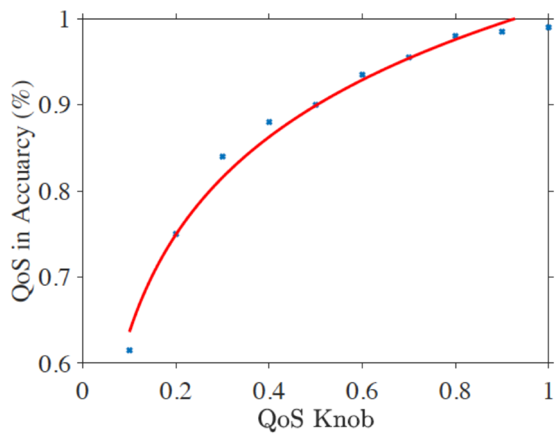
Total layer size

$$s.t. \sum_{a=1}^{A} e_a \sum_{l=1}^{L} m_{a,l}(1-h_l)u_l \le \underbrace{\min\{S - \sum_{l=1}^{L} h_l u_l, B_d T_L\}}_{\text{Remaining resource R}}$$

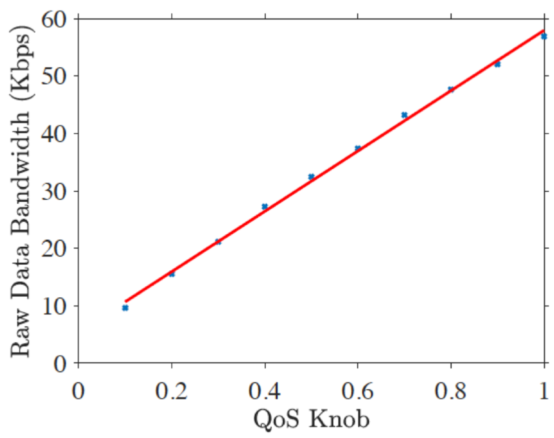
$$e_a \in \{0,1\} \; \forall a = 1, 2, \ldots, A.$$

Sample IoT Analytics

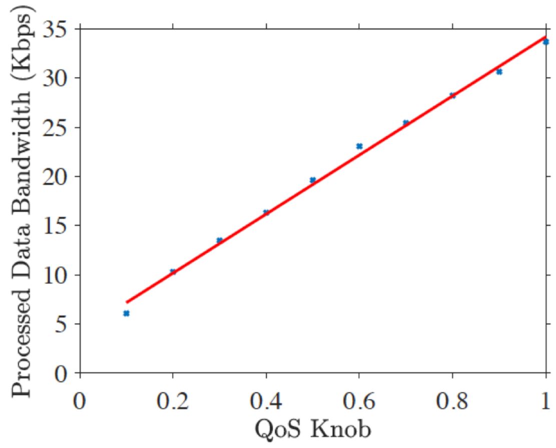
Object Detector



(a)

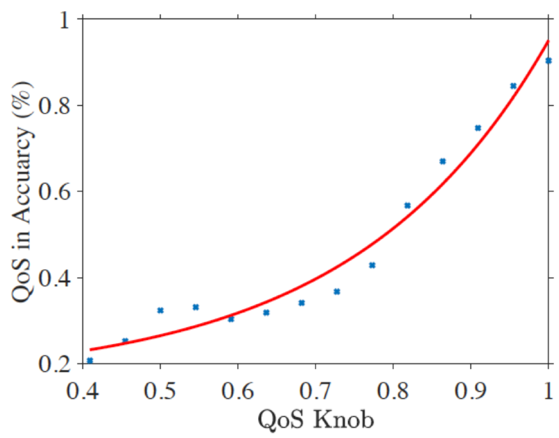


(b)

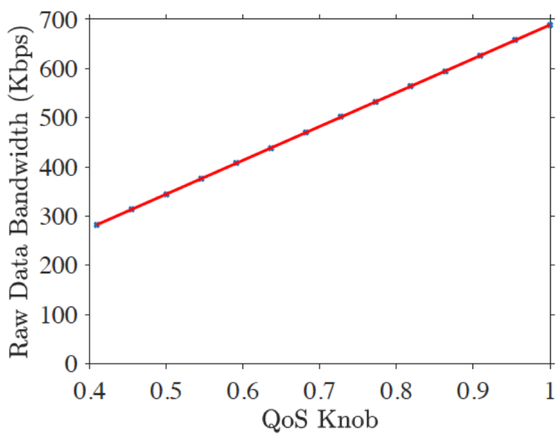


(c)

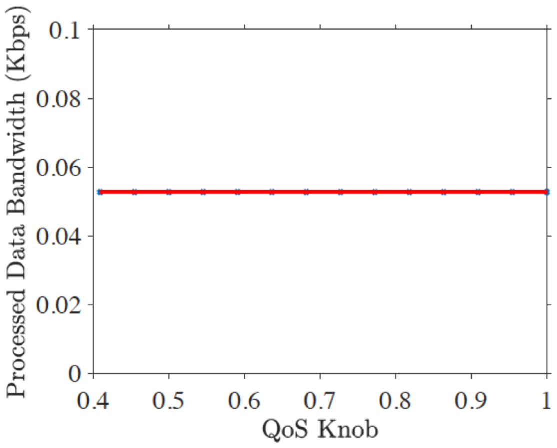
Sound Classifier



(d)



(e)



(f)

QoS and Bandwidth Models of IoT Analytics

QoS model of object detector and sound classifier:

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

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

Raw and processed data bandwidth models of object detector and sound classifier:

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

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

monotonically increasing in $[\check{k}_a, \hat{k}_a]$

Parameters:

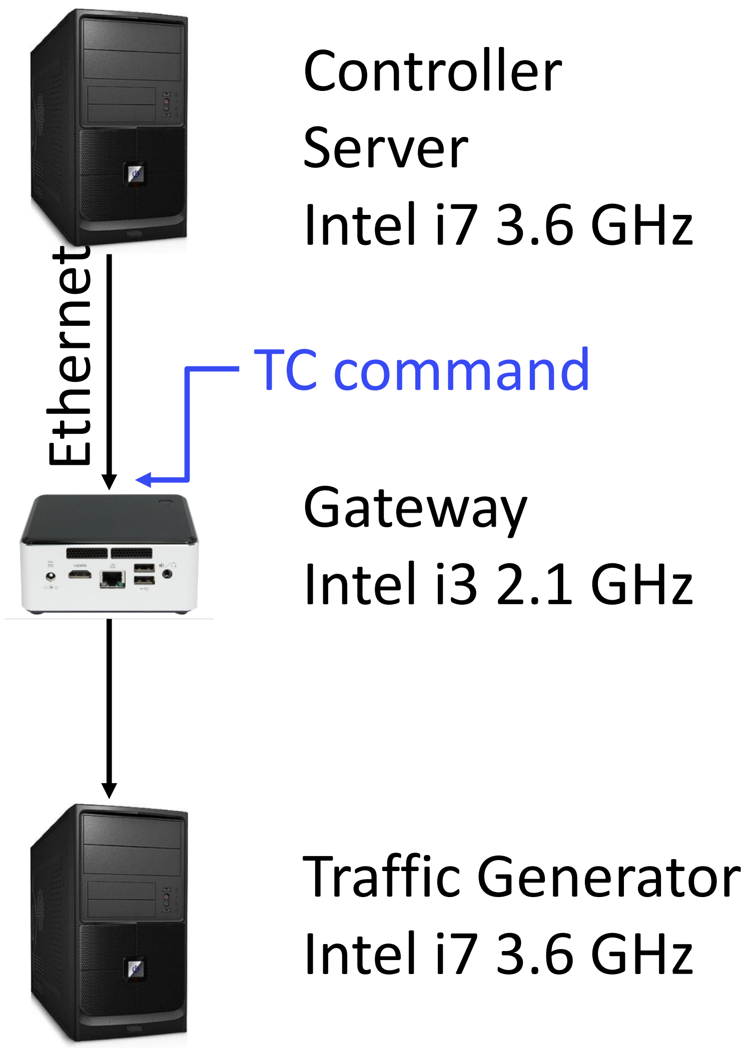
Analytics	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$	Adj. R^2	$p_{a,4}$	$p_{a,5}$	Adj. R^2	$p_{a,6}$	$p_{a,7}$	Adj. R^2
Sound Classifier	0.01	3.99	0.16	0.9630	687.56	1.01	1	0	0.05	Undef.
Object Recognizer	0.16	494	0	0.9828	52.52	5.45	0.9974	30.01	4.17	0.9956

Sample IoT analytics containers

24 IoT analytics containers using :

1. Two sample analytics (object detector and sound classifier)
2. Different Ubuntu versions (16.04.5, 16.04.6, and 18.04.4)
3. Different Python versions (2 versus 3)
4. Different TensorFlow versions (1.14.0 versus 1.15.0)

Testbed



Controller
Server
Intel i7 3.6 GHz
Ethernet
TC command
Gateway
Intel i3 2.1 GHz
Traffic Generator
Intel i7 3.6 GHz

Sample IoT analytics containers

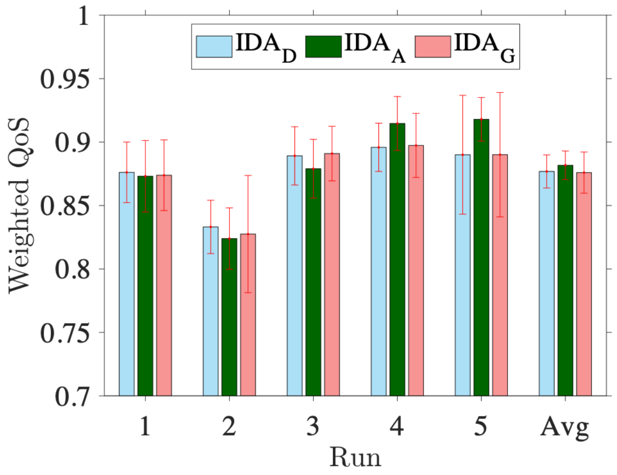
Container	Size (GB)	# of Layers	Container	Size (GB)	# of Layers
SC 1	0.72	19	**OD 1**	0.88	16
SC 2	0.81	20	**OD 2**	0.96	16
SC 3	1.27	19	**OD 3**	1.43	16
SC 4	1.35	20	**OD 4**	1.5	16
SC 5	0.81	19	**OD 5**	0.97	16
SC 6	0.9	20	**OD 6**	1.02	16
SC 7	1.35	19	**OD 7**	1.51	16
SC 8	1.44	20	**OD 8**	1.59	16
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

Setup

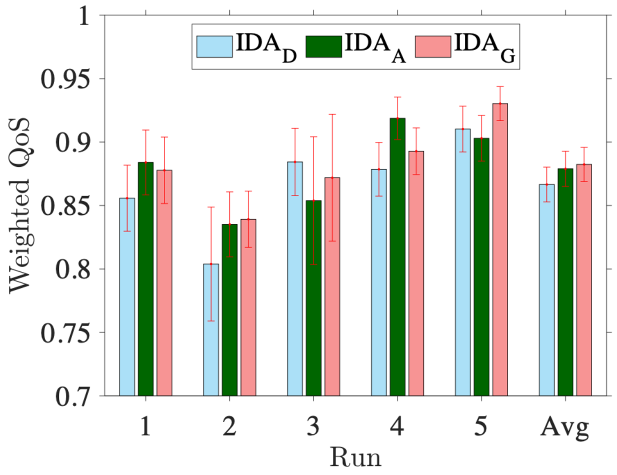
- Image pool size: 3 GB
- Network bandwidth (B_u, B_d): (5, 100) Mbps
- T_L: 5 minutes
- T_S: 1 minute
- IoT analytics requests: Poisson process with 1/3-min inter-arrival time
- Departure time: [1, 10] minutes
- Each experiment run lasts for 40 minutes
- Weights: random floating point numbers in [0, 1]
- Approximation parameter ε: 0.3
- Step size $\alpha = 0.1$

Image Download Algorithm Analysis

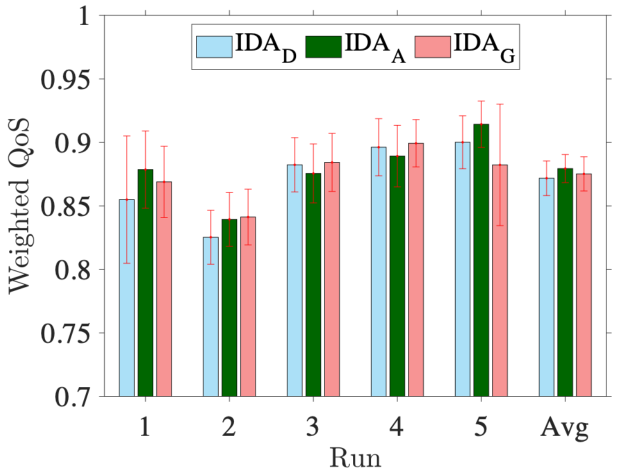
3 GB, 100 Mbps, 3 requests/min



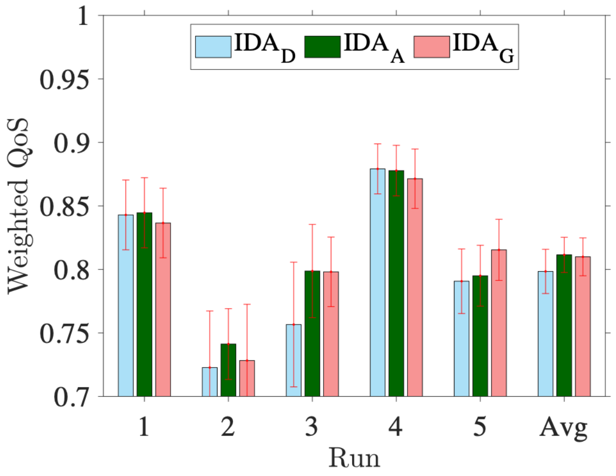
4 GB



50 Mbps

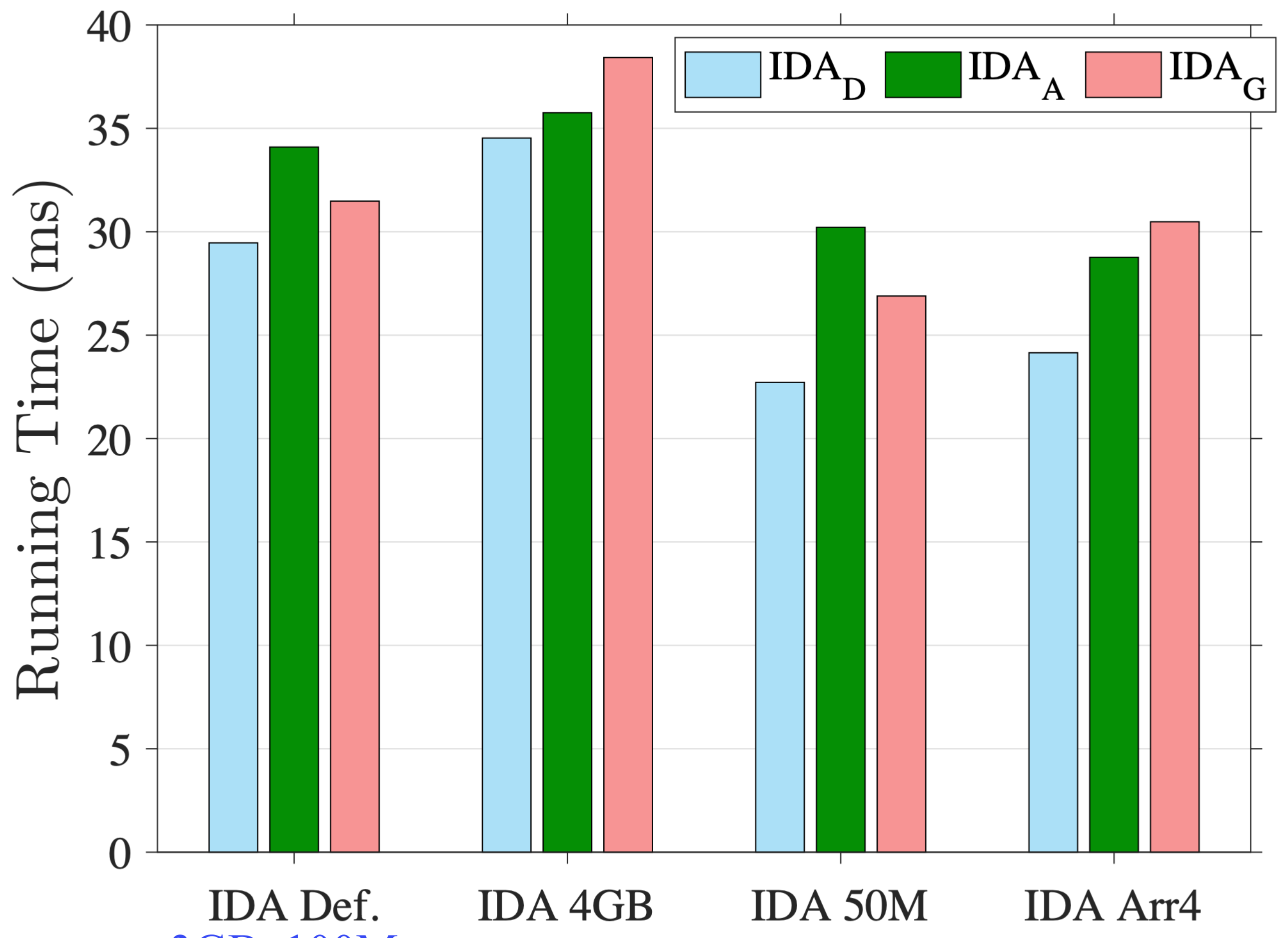


4 requests/min



- IDAD, IDAA, and IDAG all have high weighted QoS.
- # of the containers deployed on the gateway is much less than that on the cloud server.

Running Time of IDA



3GB, 100M,
3 request/min

➢ IDA_D, IDA_A, and IDA_G all averagely finish in short time: < 40 ms.