EDGE-SIDE RESOURCE MANAGEMENT FOR DATA-DRIVEN APPLICATIONS

CHUANG HU

PhD

The Hong Kong Polytechnic University

2020

The Hong Kong Polytechnic University
Department of Computing

Edge-side Resource Management for Data-driven Applications

Chuang HU

A thesis submitted in partial fulfilment of the requirements

for the degree of Doctor of Philosophy

July 2019

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Chuang Hu
_____ (Name of student)

Abstract

Data-driven applications exploit data mining and machine learning technologies to dig the great potential value of the data. Edge computing is promoted to meet increasing performance needs of data-driven applications using computational and storage resources close to the end devices at the edge of the current network. To achieve higher performance in this new paradigm, one has to consider how to combine the efficiency of resource usage at all three layers of architecture: end devices, edge devices, and the cloud. Indeed, end devices or edge devices are resource-constrained devices, whereas the cloud has almost unlimited but far away resources. Providing and/or managing the resource at the edge will enable the end device to spare resources and speed up computations and allows using resources it does not possess. Hence, there is a need for an efficient resource management at the edge.

In this research, we study the resource management in the edge side and make the following original contributions in this field.

Firstly, we focus on optimizing the communication resource management for the data-driven applications which need to transfer the data of end devices to the cloud at the edge side. The emerging smart after-sales maintenance is one such application. Manufacturers/vendors collect the data of their sold-products to the cloud so that they can conduct analysis and improve their operation, maintenance, and services of their products. Manufacturers are looking for a self-contained solution for data transmission since their products are typically deployed in a large number of different buildings, and it is neither feasible to negotiate with each building to use the buildings network (e.g., WiFi) nor practical to establish its own network infrastructure. A dedicated channel from an ISP can be rent to act as a thing-to-cloud

communication (TCC) link for each end device. Since the readily available 3G/4G is over costly for most end devices, ISPs are developing new choices. Nevertheless, it can be expected that the choices from ISPs will not be fine-grained enough to match hundreds or thousands of requirements on different costs and data volumes from the end devices. To address issue, in this thesis, we propose the communication sharing sTube+, sharing tube. Stube+ organizes a greater number of end devices, with heterogeneous data communication and cost requirements, to efficiently share fewer choices of communication resources, i.e. TCC links, and transmit their data to the cloud. We take a design of centralized price optimization and distributed network control. More specifically, we architect a layered architecture for data delivery, develop algorithms to optimize the overall monetary cost, and prototype a fully functioning system of sTube+. We also develop a case study on smart maintenance of chillers and pumps, using sTube+ as the underlying network architecture.

Secondly, we study computational allocation and optimization for DNN-based data-driven applications. DNN inference imposes heavy computation burden to end devices, but offloading inference tasks to the cloud causes transmission of a large volume of data. Motivated by the fact that the data size of some intermediate DNN layers is significantly smaller than that of raw input data, we design the DNN surgery, which allows partitioned DNN processed at both the edge and cloud while limiting the data transmission. DNN surgery considers the network bandwidth between the end device and the cloud, as well as their processing capabilities when deciding to handle the layer of the DNN either on the end device or in the cloud. The challenge is twofold: (1) Network dynamics substantially influence the performance of DNN partition, and (2) State-of-the-art DNNs are characterized by a directed acyclic graph (DAG) rather than a chain so that partition is greatly complicated. In order to solve the issues, we design a Dynamic Adaptive DNN Surgery (DADS) scheme, which optimally partitions the DNN under different network condition. We conduct a comprehensive study of the partition problem under the lightly loaded condition and heavily loaded condition. Under the lightly loaded condition, DNN Surgery Light (DSL) is developed, which minimizes the overall delay to process one frame. The

minimization problem is equivalent to a min-cut problem so that a globally optimal solution is derived. In the heavily loaded condition, DNN Surgery Heavy (DSH) is developed, with the objective to maximize throughput. However, the problem is NP-hard so that DSH resorts an approximation method to achieve an approximation ratio of 3.

Finally, we propose a Transmission-Analytic Processing Unit (TAPU), a novel accelerator using multi-image FPGA to provide both computation resource and communication resource for edge device. A multi-image FPGA can pre-store multiple images in the FPGA flash and fast switch between images. We can then configure one image for computation, and the other images for network functions. Thus, we can multiplex the accelerator by controlling the switch of the images. System design of TAPU from the hardware to the software is present. In the hardware design, we discuss the FPGA choice and abstracts a set of transparent APIs for the developers. For the software, offline modules are designed to determine which functions are offloaded to FPGA, and runtime modules are designed to determine how to switch images adapting to the runtime variations. We choose to accelerate network functions first, and use the residual computation capacity to accelerate computation. We develop two schemes to estimate the residual computation capacity of TAPU for non-preemption case and general case respectively. Then to fully use the residual computation capacity, we design an inference task offloading algorithm for video analytic task assignment to the FPGA. These algorithms collectively exploit the capacity of the FGPA for DNN inference acceleration to the maximum.

In summary, we propose three methods to overcome the communication, computation and resource challenges in edge/cloud computing for data-driven applications from edge-side resource management perspective. The proposed methods are leveraged in important data-driven applications. Prototype of the systems were developed to evaluate the effectiveness of the solutions. We identify the requirements and address the challenges therein, providing effective frameworks and solutions for practitioners.

Publications

1. **Chuang Hu**, Wei Bao, Dan Wang, and Fengming Liu, "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge", *Proceedings of the 38th IEEE International Conference on Computer Communication (INFOCOM)*, Paris, France, April 29 - May 2, 2019.

2. **Chuang Hu**, Wei Bao, and Dan Wang, "IoT Communication Sharing: Scenarios, Algorithms and Implementation", *Proceedings of the 37th IEEE International Conference on Computer Communication (INFOCOM)*, Honolulu, HI, October 24-28, 2018.

3. **Chuang Hu**, Wei Bao, Dan Wang, Yi Qian, Muqiao Zheng and Shi Wang, "sTube+: An IoT Communication Sharing Architecture for Smart After-sales Maintenance in Buildings", *ACM Transactions on Sensor Networks (TOSN) - Special Issue on BuildSys'17*, Volume 14, Issue 3-4, December, 2018.

4. **Chuang Hu**, Wei Bao, Dan Wang, Yi Qian, Muqiao Zheng and Shi Wang, "sTube+: An IoT Communication Sharing Architecture for Smart After-sales Maintenance in Buildings", *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Build Environments (BuildSys)*, Delft, The Netherlands, November 8-9, 2017.

5. Qiong Chen, Zimu Zheng, **Chuang Hu**, Dan Wang, and Fangming Liu, "Data-driven Task Allocation for Multi-task Transfer Learning on the Edge", *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, Dallas, Texas, July 7-9, 2019.

6. Dan Wang, Wei Bao, **Chuang Hu**, Yi Qian, Muqiao Zheng and Shi Wang, "sTube: An Architecture for IoT Communication Sharing", *IEEE Communications Magazine*, Volume 56 , Issue 7, Pages 96-101, July, 2018.

7. Zimu Zheng, **Chuang Hu**, and Dan Wang, "Time-aware Chiller Sequencing Control with Data-driven Chiller Performance Profiling (Poster)", *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Build Environments (BuildSys)*, Delft, The Netherlands, November 8-9, 2017.

Acknowledgements

It is hard to believe that this day has finally come. I know I would not have reached this stage if it was not for the help, support, and guidance of many great people who I was lucky to meet.

My Ph.D. supervisor, Dr. Dan Wang, definitely make the top of the list. He was always generous with his time providing me the guidance I needed. I am also very grateful to him for his brilliant comments, critical judgment, scientific advice and many insightful discussions and suggestions which motivate me to sharp my mind and help me get out of some tough days. His endless guidance is hard to forget throughout my life.

I also would like to thank Dr. Wei Bao who always provides useful tips and comments to help me think differently on the problem during our collaboration. His truly scientist intuition and invaluable guidance inspire and enrich my intellectual maturity that I will benefit from, for a long time to come.

My special thanks to Prof. Yi Qian and Dr. Dawei Pan. for the precious support during our collaboration. I am also pleased to say thank you to Dr. Kunfeng Lai, Dr. Abraham Hang-Yat Lam, Dr. Liang Zhang, Mr. Zimu Zheng, Mr. Quanyu Dai, Mr. Qiong Chen, Miss Shi Wang, Mr. Muqiao Zheng, Mr. Fengming Liu, Mr. Kaichen Wei, Mr. Fangqiu Su for their kind help and understanding.

Last but not least, I would like to thank my parents. They gave me enough moral support, encouragement and motivation to accomplish my personal goals. Special thanks to my wife, Yuanyuan Li, who witnesses the joys and sorrows of my PhD study miles away. I am really grateful for her endless love, patience, understanding and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data-driven applications help users search, explore and draw constructive conclusion from data. Data has always played a key role in organizations, business, and industries. The applications providing service for these domains become data driven. For example, the emerging self-driving car [16] in transport, the advancing recommendation system [26] in business, the mature automatic face recognition [9] in security are useful data-driven applications. The data-driven applications become booming and successful in many domains such as industry, security, business, and health.

Data-driven applications extract value from data and transform the value to services. The advance in data science and Internet of Things (IoT) have promoted the booming of data-driven applications. The fast development of data science, especially in machine learning and data mining, has enabled data-driven applications to extract more value from data. There has been tremendous growth in the number of Internet-enabled smart devices. According to CISCO, more than 50 billion devices are expected to be connected to the Internet by 2020 [66]. With numerous devices installed and connected to IoT, the amount of data to be processed has been increasing astronomically. Cisco Systems predicts that cloud traffic is likely to rise nearly fourfold by 2020, increasing 3.9 ZB per year in 2015 to 14.1 ZB per year by 2020 [55]. This means that more data can be used for extracting meaningful information by

data-driven applications.

In the meanwhile, the development of data science and IoT also incurs lots of challenges in communication and computation for data-driven applications. The data-driven applications are typically based on the collection, transmission and processing of large volumes of data. The increasing volume of data imposes heavy burden on the transmission process. The advance in data science, such as deep neural networks (DNN), have increased the computation workload in data-driven applications. In summary, the current data-driven applications are characterized by the large volume of raw data and intensive computation.

The mature cloud computing and emerging edge computing are the two promising choices for data-driven applications. Cloud computing equipped with powerful hardware can provide ample computation resources but suffer from transmitting a large amount of data which may result in high communication cost and long latency. Edge computing which put the computation at the proximity of data source is network-free but limited by the constrained resource of the edge devices.

A single computing paradigm cannot meet the whole requirement of data-driven applications. No matter which paradigm or the collaboration of these two paradigms is employed, a common point is that the data is generated at the edge device. Some work can be done at the source to address or alleviate the problems in the data-driven application. Providing and/or managing the resource at the edge will enable the edge device to spare resources and speed up computation, reduces the transmission costs and data size, and allows using resources it does not possess. In this thesis, we study resource management on the edge to tackle some issues for data-driven applications.

In the following of this chapter, we discuss the pros and cons of edge computing and cloud computing for data-driven applications. In section 1.2, we present the problems we will solve through edge-side resource management strategies for data-driven application in this thesis. Section 1.3 discusses the research framework.

Section 1.4 summarizes the contributions of this thesis. Finally, Section 1.5 gives the outlines of the thesis.

1.1 Data-driven Application with Edge/Cloud Computing

Different data-driven applications have different requirements, which should be fulfilled by the computing platforms amalgamating edge device with their applications.

Cloud Computing has evolved and became an easy-to-use platform for data-driven applications in general to store and process data. The cloud provides ubiquitous and on-demand access to a virtually shared pool of configurable computing and storage resources. Cloud computing is an excellent platform to handle the enormous data generated from the IoT environment due to the cheaper and the larger amount of virtual computing/processing power available at the cloud centre. However, it is not suitable for the applications demanding low-latency, real-time operation and high Quality of Service (QoS).

In the cloud computing paradigm, data-driven applications collect information from the environment and share it to a cloud service for processing as shown in Fig. 1.1(a). The computation occurs on the cloud. The bottleneck is transferring a large amount of data of the things to the cloud through the wide-area network as shown red line in Fig. 1.1(a), which may incur huge transmission cost and latency.

The concept of edge computing, also called fog computing, is receiving important attention to address some of the drawbacks of cloud computing. The main goal of edge computing is to extend cloud computing functions to the edges of the network. Due to proximity to the end-users and geographically distributed deployment, it can support the data-driven applications/services demanding the requirements of low latency, location-awareness, high mobility and high QoS. However, edge computing

units usually do not have enough storage and computing resources in handling the massive amount of data of such kind applications.

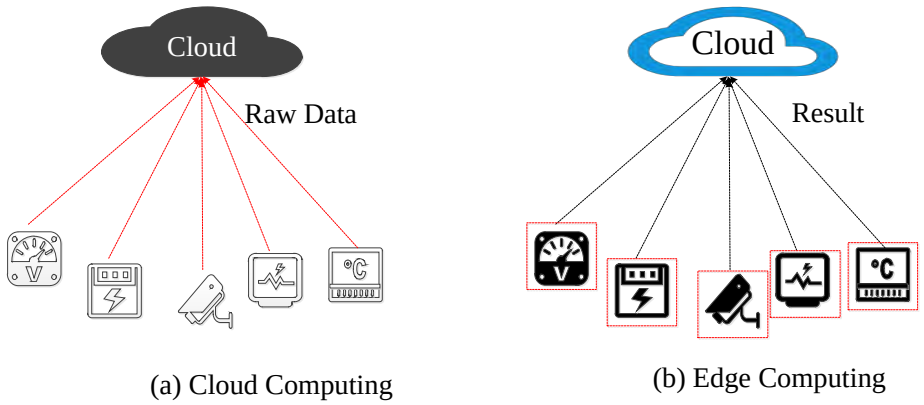
In edge computing paradigm, data-driven applications process data at where the data is generated and sends the processing result to the cloud as shown in Fig. 1.1(b). For example, AWS DeepLens camera can run deep convolutional neural networks (CNNs) to analyze visual imagery [2] and transfers the result to the cloud for further decision making. Only result is transferred to the cloud, which avoids the effect of the network. The bottleneck is executing complicated data-driven applications on the constrained resource of edge device showed in the red line in Fig. 1.1(b). Constrained computation resources impose the main challenge for data-driven applications. Take video analytics application as an example. Video analytics applications have a high-performance requirement. We conducted measurement using Raspberry Pi 3 model B to perform inference on the AlexNet model [45], a small model with moderate accuracy. The execution time is 5 seconds using BVLC benchmark [3].

To address the transmission problems in cloud computing and computation resource in edge computing for data-driven applications, in this thesis, we design a series of edge-side resource management strategies.

1.2 The Problems

Processing data-driven application with edge computing or cloud computing incurs a series of problems, which needs a wide range of research works to solve. In this section, we introduce the problems we will solve through edge-side resource management strategies in this thesis.

1. The first problem is **how to transmit the data from the edge device to the cloud in an easy-to-use and cost-effective way**. In some data-driven application, such as the smart after-sales maintenance, the end device



(a) Cloud Computing (b) Edge Computing

Figure 1.1: Cloud computing and edge computing for data-driven applications. Black box denotes where computation happens. Red line denotes the bottleneck.

(e.g. products in smart after-sales maintenance) are deployed in a wide area, a self-contained solution for data transmission is needed. A dedicated things-to-cloud link (e.g. CAT1, 3G, 4G) from ISPs is a feasible solution. However, it can be expected that the choices from ISPs will not be fine-grained enough to match hundreds or thousands of requirements on different costs and data volumes from the edge devices.

2. The second problem is **how to use the constrained computation capacity of the edge device to meet the requirement of the computation-intensive data-driven applications**. Offloading data-driven application to the remote cloud causes transmission of a large volume of data, edge computing can avoid the effect of network and put the computing at the proximity of data source. Nevertheless, edge devices themselves are limited by their computing capacity.

3. The third problem is **how to meet the increasing requirement on computation and communication resource from the edge device while**

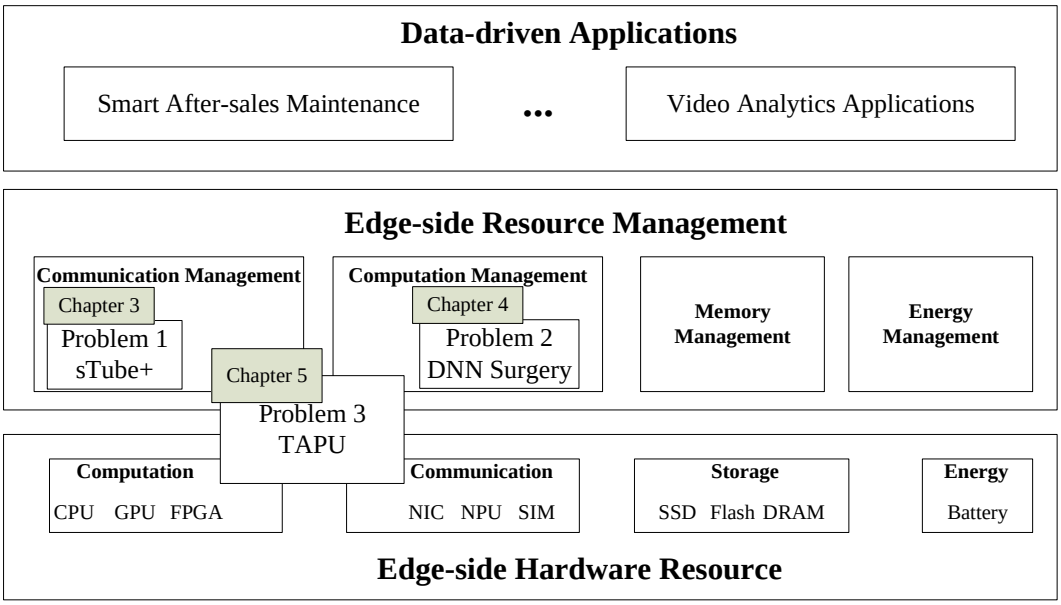


Figure 1.2: The unified research framework.

improving the resource utilization. The increasing performance needs of data-driven applications requires to enhance the computation and communication capability of the edge device, such as the IoT gateway. Adding dedicated resource, such as GPU for computation, NPU for communication, leads to low utilization. New hardware which provides both computation and communication resource is needed.

1.3 Research Framework

In this section, we present the research framework for the proposed techniques. Fig. 1.2 illustrates the sketch of our research framework.

As shown in Fig. 1.2, edge-side resource management runs at the software layers and controls the usage of the resource-constrained hardware resources of edge devices to meet the requirement from the data-driven application in the upper layer. Resource management on the edge device includes computation management, com-

munication management, memory management, and energy management. In this thesis, we mainly focus on communication management and computation management on the edge.

To solve the first problem, for the data-driven applications (e.g. the smart after-sales maintenance) which collects data generated at the end device to the remote cloud, we design a resource sharing strategy. We design **Sharing Tube plus (sTube+)**, a communication sharing architecture, which organizes the IoT devices to transmit their data to the cloud through sharing the limited communication resources (i.e. the choices of things-to-cloud communication links) in a cost-effective and self-contained way.

To tackle the second problem, for the computation-dominant data-driven applications (e.g. the DNN-based video analytic) which have a high requirement on performance, we propose **DNN Surgery**, which allows partitioned DNN processed at both the edge and cloud by considering the wireless bandwidth and the computing resource of the edge device and the cloud. DNN surgery optimally partitions the DNN under different network condition and workload.

To solve the third problem, for the edge device which needs to enhance both computation and communication capability, such as IoT Gateway, we design **Transmission-Analytics Processing Unit (TAPU)**, a novel processing unit which can provide both computation and communication resource. Communication management and computation management strategies of TAPU are also designed to adapt to the runtime variance from the upper application layer.

1.4 Contributions

The contributions of this thesis are summarized as follows.

- We for the first time to clarify the necessity, the scope of IoT communication

sharing for the cloud-based IoT application. In order to bridge the gap between the possible choices of thing-to-cloud communication link from ISPs and the number of requirements on different costs and data volumes the heterogeneous requirements on different costs and data volumes from the cloud-based IoT applications, we propose Sharing Tube plus (sTube+), which organizes a greater number of IoT devices, with heterogeneous data communication requirements to efficiently share fewer choices of TCC links, and transmit their data to the cloud. We demonstrate the cost-efficacy of our approach in a real-world smart after-sales maintenance case.

- For the computation-intensive data-driven application, we design the DNN surgery, which allows partitioned DNN processed at both the edge and cloud while limiting the data transmission. We design a Dynamic Adaptive DNN Surgery (DADS) scheme, which optimally partitions the DNN under different network condition. Under the lightly loaded condition, DNN Surgery Light (DSL) is developed, which minimizes the overall delay to process one frame. In the heavily loaded condition, DNN Surgery Heavy (DSH) is developed, with the objective to maximize throughput.

- We propose a novel Transmission-Analytics Processing Unit (TAPU), a new accelerator using multi-image FPGAs to accelerate data analytics and network functions for data transmission in IoT gateway.

- We implement prototypes with the proposed techniques. We conduct experiments and compare with representative schemes. Experimental results prove the effectiveness of the proposed schemes.

1.5 Thesis Organization

The rest of this thesis is organized as follows.

- **Chapter 2 (Background and Literature Review):** This chapter provides a basic introduction to the data-driven applications using the smart after-sales maintenance and video analytic as the cases. We also study the existing research works in resource management on the edge side.

- **Chapter 3 (sTube+: A Communication Sharing Architecture for Data-driven After-sales Maintenance):** In this chapter, we present the sTube+, the IoT communication architecture to organizes a greater number of IoT devices with heterogeneous data communication requirements to efficiently share fewer choices of things-to-cloud communication links and transmit their data to the cloud.

- **Chapter 4 (DNN Surgery: Accelerating Inference on the Edge):** In this chapter, we propose DNN surgery that can partition DNN inference between the edge device and the cloud at the granularity of neural network layers, according to the dynamic network status and the computation capability of the edge device and the cloud.

- **Chapter 5 (TAPU: a New Processing Unit for Accelerating Multi-type Functions in IoT Gateways):** In this chapter, we discuss the need of hardware accelerator for both computation and communication in IoT gateways and the disadvantages of using dedicated hardware accelerator. We propose transmission-analytic processing unit, a novel accelerator using multi-image FPGA to provide both computation and communication resources.

- **Chapter 6 (Conclusions and Future Directions):** This chapter concludes

the thesis with a summary of our research works and discussions on future research directions.

Chapter 2

Background and Literature Review

In this chapter, we first present a basic introduction to the data-driven applications. We then study the existing research works in resource management on the edge side.

2.1 Data-driven Applications

In this section, we introduce two kinds of data-driven applications: 1) communication dominated data-driven application – smart after-sales maintenance and 2) computation intensive data-driven application – video analytics.

2.1.1 Smart After-sales Maintenance

Smart After-sales Maintenance and Services (SAMS) is a kind of data-driven applications. In SAMS, manufacturers analyze the collected data of their sold-products to improve the operation, maintenance and services of their products. Manufacturers of air conditioners, pumps, elevators, etc., are now transforming their machinery into smart machinery. When sending the data of their products to the cloud, SAMS can operate in a trouble-preventing mode instead of troubleshooting mode.

We use the chiller maintenance as an example to illustrate how SAMS benefits. Current chiller maintenance consists of routine maintenance and emergency repair, and their respective costs are USD $897.12 and USD $5639.94 per time [47]. An

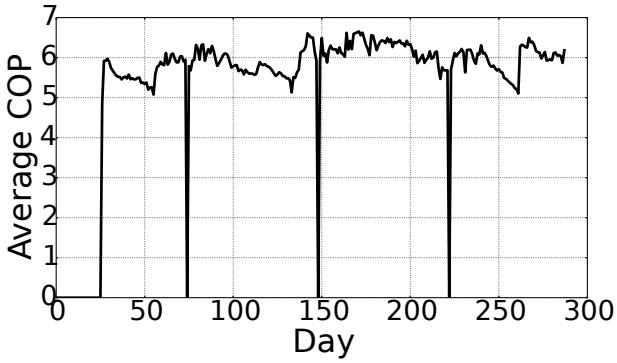


Figure 2.1: Average COP of chillers as a function of day.

optimal maintenance plan is a balance of routine maintenance and emergency repair. This is usually done by analyzing the degradation of chillers. Intuitively, routine maintenance will be more frequent if a certain type of chiller degrades faster. Chiller degradation is affected by many factors, such as its intrinsic reliability and the usage pattern of the chiller (e.g., the Freon level depends on the intensity that the chiller is being used). Though the chiller reliability can be extensively tested in labs, the usage pattern of a chiller is determined by customers and is difficult to know at the time that this chiller is being manufactured. This is one key reason that SAMS can become superior.

The key indicator for the performance (degradation) of a chiller is Coefficient of Performance (COP) [33]. The COP of one chiller measured in our dataset is shown in Fig. 2.1. Maintenance is needed if the COP of a chiller is below a certain threshold.*

To compare the current maintenance plan and SAMS, we conduct an illustrative analysis by using four year data of ten chillers in three buildings. We calculated the optimal plan for current maintenance with a routine maintenance interval of 3.1 months, leading to a cost of USD $4052.64 per chiller per year. For SAMS, we

*A low COP does not mean a direct chiller failure, yet it indicates sensible human comfort down grade and substantial energy usage inefficiency. The current threshold imposed in Country/City Anonymity is 5.7.

can collect the chiller data in real time. The cost reduces to USD $2813.66, with an average maintenance interval of 3.89 months. This leads to a 30.58% saving. Note that this is only a baseline comparison. If we consider joint maintenance of multiple pieces of equipment, a prediction of equipment degradation, and that current maintenance plan has to be conservative (e.g., shorter than 3.1 months), we can expect a much greater gain from SAMS.

The main challenge for SAMS is how to collect data from devices in an easy-to-use and cost-effective way. We solve this problem in Chapter 3 by designing a resource sharing architecture and strategy on the edge side.

2.1.2 Video Analytics

Video analytics is another kind of data-driven applications. Video analytics is the core to realize a wide range of exciting applications ranging from surveillance to self-driving cars.

The advances of deep learning and the massive data collected and labeled make it now possible to analyze images and videos in high accuracy. In deep learning, a neural network model is first trained to recognize objects, or to track targets, etc. As an example, to train a CNN model for self-driving cars, one can use a back-propagation learning algorithm to settle the CNN parameters by translating the road images to the steering wheel angle commands. During video analytics (inference), new data is injected into the trained CNN model. For example, in self-driving cars, each frame of the video is injected to the CNN model using a feed-forward algorithm to determine the steering wheel angle commands that should be performed.

Video analytics applications have a high-performance requirement. To support user experiences, the general principle is that video analytics should match the image recognition speed of a human. For a human being, it takes about 370 ms to recognize a face as familiar [72]. In a ping-pong game, it takes about 250 ms for a novice player

to read the ball and swing, and even shorter for professional players [20]. As a result, video analytics is expected to complete within 100ms.

There are studies trying to reduce computation workloads. The most direct way is reducing model size and sacrifice accuracy. For example, Microsoft and Google developed small-scale DNNs for speech recognition on mobile platforms by sacrificing prediction accuracy [10]. MCDNN [39] proposed generating alternative DNN models to trade-off accuracy and performance/energy. Deep models that are much smaller than normal were proposed [86] for phones.

There are studies that leverage application-specific characteristics to reduce computing workloads. For example, DeepMon [42] exploited the similarity between consecutive frames in first-person-view videos to reduce computation workloads. Glimpse [21] developed a suite of content-aware techniques to sample only a few key frames for processing. The rendering operation, a computation-intensive operation commonly used in 3D game engines, was optimized in [24].

In this thesis, we accelerate video analytics through a series of edge-side resource allocation strategies in Chapter 4 and resource provisioning in Chapter 5.

2.2 Resource Management on the Edge

In this section, we provide an overview of work related to our study of resource management on the edge side. Resource management at the edge can be decomposed into several areas addressing different problems, as shown in the in Fig. 2.2. First, we describe the existing studies on edge-side resource sharing related to our first work sTube+; second, we depict work on edge-side resource scheduling related to our second proposed work namely DNN surgery; and third, report on resource provision on the edge related to our proposed work namely TAPU. We carefully anticipate some of our study to explain the relationship between our work and other discussed

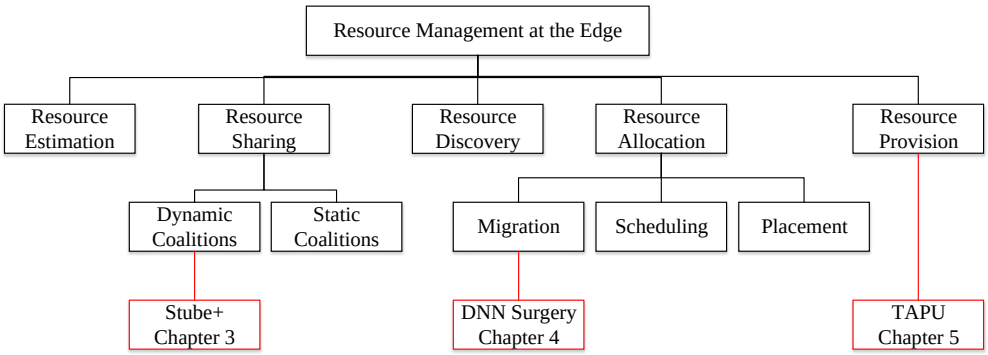


Figure 2.2: A taxonomy of the objective of resource management at the edge.

publications.

2.2.1 Edge-side Resource Sharing

Resources on end devices are heterogeneous and most of the time scarce, and edge devices also have limited resources compared to resources in the cloud. Sharing resources between end devices aims at using other devices resources to get a faster or a lower monetary cost completion of the task. Resource sharing can be classified into dynamic coalitions and static coalitions, according to whether they include how to form the groups of devices that will share resources or if they assume that the formation is already done and focus on the actual sharing.

Sharing computation resources on the edge has been well-studied. Sharing computation resource is typically realized by pooling computation resources in the local vicinity of client devices. For the static coalitions, [56] exploited opportunistic contacts between the devices, creating a computation sharing mechanism to minimize the completion time of the task; [76] consider computation resources shared between two neighbor fog colonies and achieve a 35% reduction of execution time compared to the non-sharing strategy. Moving to the dynamic static, [11] proposed method to create a cluster to share both computation and storage resources of vehicles and

achieved up to 5 times lower computation delay. Some works focused on energy sharing and data sharing on the edge. [98] proposed a data sharing framework which can be used to search for a person with the help of multiple cameras. [59] defined device clusters which can share resource and aimed to share power resources to maximize the network lift time, i.e. saving energy through offloading to another device.

In this thesis, we focus on sharing the communication resource of the edge devices aiming to reduce the communication cost. We discuss the difference between existing communication resource sharing architects and our proposed sTube+.

Smart Building Networks: Modern buildings have building automation systems (BAS) to control building equipment [36]. Traditional BAS is mostly signal-based. An sMap architecture [27] was developed to software-define traditional BAS. In sMap, the IoT devices are organized into a mesh network, and a gateway is used. The target of sMap and BAS is to manage thousands of devices, from different vendors, within a building. The target of sTube+ is to transmit the data of thousands of IoT devices, of the same vendor, spread at hundreds of buildings, to the cloud. sTube+ differs from sMap in the supporting application *context*. The spread of the devices in buildings controlled by different building owners made the gateway approach infeasible since a building-by-building based deployment or agreement is needed.

Mobile Phones as Relays: One recent proposal to transmit IoT data to the cloud is to use mobile phones as relays [96]. The objective is to remove the gateway, which restricts the scalability. An opportunistic network is constructed where IoT devices will search for nearby mobile phones to relay data. sTube+ does not rely on opportunistic data transmissions. sTube+ differs as it is clear on *who* should run the transmission function.

Cellular Network/Edge Routers: Multiplexing data flow of different devices is not new. Cellular base stations and edge routers aggregate data flows. sTube+

differs from them in *where* to multiplex. The location of the multiplexing function of sTube+ is on the IoT devices. Traffic flow multiplexing by base stations/edge routers is controlled by ISPs; yet in sTube+, it is controlled by the vendors.

3G Data Sharing: Data sharing is not new. One example is the hotspot function of mobile phones. 3G hotspot is local to a few phones, and a simple master and slave design is enough. The requirements for sTube+, as represented by SAMS applications, need a scalable architecture that can handle the heterogeneity of the hardware devices, multi-path routing, an overall optimization of the cost of a vendor, etc. The level of *complexity* differs greatly. Another example is represented by family plans, where multiple sim cards are allowed. Yet family plans cannot share *different plans*, and in our scenario, the vendor may register different plans for overall optimization. In addition, it is questionable whether ISPs will provide plans where thousands of sim cards, in particular those with a large amount of small-size flows, can share a single plan. Intrinsically, this means that ISPs take the burden and cut their own profits for the benefit of vendors. As such, we believe that ISPs will impose certain *limit* even if plans with multiple sim cards are developed; making the vendor side sharing still important.

2.2.2 Edge-side Resource Allocation

Resource allocation can be tackled from two different perspectives: where to allocate, and when and how much to allocate. According to allocation approaches, resource allocation can be classified into resource placement, resource scheduling and resource migration.

Load distribution is a common instance of resource placement. [35] proposed an offloading strategy between edge data centers under high loads that show the benefit of having a larger data center as back-up for a small one. [59] proposed a power balancing algorithm in which a device decides whether to offload and to which other

device depending on the energy left in the batteries of the devices. [65] formulated a joint computation and communication resource allocation and optimization problem in a multi-user case focusing on latency and power efficiency. [91] tackled the issue of service placement in a system composed of edge server nodes and traditional cloud nodes.

Resource secluding refers to when and how many resources to allocate for the tasks. There is a huge body of researches forcing on scheduling decisions at the edge level. [15] studied the impact of three different fog scheduling strategies (delay-priority, concurrent and first come-first served) on application QoS. [89] proposed elastic resource allocation for surveillance systems. [90] designed a cost-effective resource scheduling strategy between the mobile cloud and the cloud radio access network.

Considering where the task should be executed, when it comes to services, tasks, and applications, the focus could be on how they can be moved during execution if the new location is better, i.e. on migration. Our proposed DNN surgery belongs to resource migration. Computation offloading is an instance of resource migration. Research efforts focusing on offloading computation from the resource-constrained mobile to the powerful cloud will reduce inference time. Neurosurgeon [44] explored a computation offloading method for DNNs between the mobile device and the cloud server at layer granularity. However, Neurosurgeon is not applicable for the computation partition performed by our proposed DNN surgery for a number of reasons: 1) Neurosurgeon only handles chain-topology DNNs that are much easier to process. 2) Neurosurgeon can only handle one inference task, without considering a sequence of tasks. Needless to say the adaptation to network condition realized by DNN surgery. MAUI [23] is an offloading framework that can determine where to execute functions (edge or cloud) of a program. However, it is not designed specifically for DNN partitioning as the communication data volume between functions is

small. [83] proposed DDNN, a distributed deep neural network architecture that is distributed across computing hierarchies, consisting of the cloud, the edge and end devices. DDNN aims at reducing the communication data size among devices for the given DNN. DNN surgery differs as it handles dynamic network condition to reduce the inference latency (communication and computing latency) rather than communication overhead only.

2.2.3 Edge-side Resource Provisioning

Computation Resource Provisioning. Recently, more powerful computation resources, such as GPU, TPU and FPGA, are provisioned on the edge device to improve the computation capacity. Vanhoucke et al. [85] used fixed point arithmetic and SSSE3/SSE4 instructions on x86 machines to reduce the inference latency. DeepX [48] explored the opportunities to use mobile GPUs to enable real-time deep learning inferences. FPGA has been used for different systems and applications. These studies usually compare the performance of CPU, GPU, FPGA, e.g., in their energy consumption, processing power, etc, e.g., the energy consumption of an FPGA is superior to that of a GPU and the processing power of an FPGA is superior to that of a CPU [63, 46]. These studies then take the advantages of the FPGA to assist/redesign network functions [54], mobile games [53], audio applications [38], sensing apps [64]. [92] proposed an automation tool to generate FPGA-based accelerators for DNN models.

Communication Resource Provisioning. The requirement of network security and the need to reduce the volume of data transmission lead to the network functions running on the edge devices becomes complicated. Software network function is the common method. [28] proposed a scalable software router that parallelisms packet processing with multiple CPU cores. However, software network functions suffer lower performance, the hardware communication resource should be provisioned

to meet the requirement of network functions. To accelerate software packet processing, some studies exploited GPUs. [87] proposed GASPP, a GPU-accelerated packet processing framework. Recent studies implement network functions with FPGA exploiting its programmability and the ability to customize the hardware. [52] accelerated IPsec gateway on FPGA; [8] implemented Gzip on FPGA.

Different from the above works which use dedicated hardware to accelerate computation or communication. Our proposed TAPU provide both computation and communication resource in a single processing unit by exploiting multi-image FPGA. The recent development of multi-image FPGA [6] made it possible to pre-stored images and fast switch among images without reconfiguration. To the best of our knowledge, we are the first to configure and switch the multiple images of an FPGA to provide both communication and communication resource.

Chapter 3

sTube+: A Communication Sharing Architecture for Data-driven After-sales Maintenance

3.1 Introduction

One important value proposition of the Internet of Things (IoT) is the data generated by the IoT devices (a.k.a, things) [62]. When sending such data to the cloud, with state-of-the-art data mining techniques and the computational power of the cloud, the adding value can be significant [88]. For example, it has been shown that big building data (e.g., carbon dioxide (CO2) data from the heating, ventilation and air conditioning (HVAC) systems) can be exploited to predict traffic status of nearby roads [99]. Smart After-sales Maintenance and Services (SAMS), which will become the case study of this chapter, is another example. Manufacturers of air conditioners, pumps, elevators, etc., are now transforming their machinery into smart machinery. When sending the data of their products to the cloud, SAMS can operate in a trouble-preventing mode instead of trouble-shooting mode. This can substantially improve

the quality and reduce the cost of the product maintenance. Moreover, manufacturers can learn the usage patterns of their customers. Thus they can recommend other products and develop top-up services based on such knowledge [51].

To fully realize the aforementioned applications, the things should be accessible anywhere and anytime. One key question remains to be answered: how to transmit the data from the things to the cloud, in an easy-to-use and cost-effective way?

The vendor may develop a WiFi network for the IoT application. However, WiFi needs additional infrastructure, e.g., a gateway that finally relays data to the cloud. This is not suitable for SAMS. For example, a vendor would like to monitor all its air conditioners in a region, installed in a large number of buildings. The WiFi choice needs deployment of WiFi networks on a building-by-building basis. In other words, the vendor is developing a separated network infrastructure. If using existing WiFi networks in the buildings, there will be policy and security concerns. A building can easily have products from tens of vendors. If each vendor wants its equipment to infiltrate the WiFi network of the building, building operators need to bear overwhelming liability. Simply-put, applications such as SAMS are looking for an infrastructure-less solution.

The vendor may rely on the infrastructure of a service provider (ISP) and rent a dedicated wireless communication channel for each IoT device [37] to support the *thing-to-cloud communication (TCC)* links. Current choices for TCC links are very limited. The readily available 3G/4G is over-costly for the majority of IoT devices. The industry has realized this problem and is actively developing less costly wireless communication channels. User Experience-Category (CAT) represents a group of technologies with much smaller data rates and thus costs [60]. CAT1 was released in 2016 and CAT0 is under deployment [74]. Nevertheless, we may expect tens of choices of communication channels with different costs and data rates, yet we will face hundreds, if not thousands, of heterogeneous requirements. In the SAMS example,

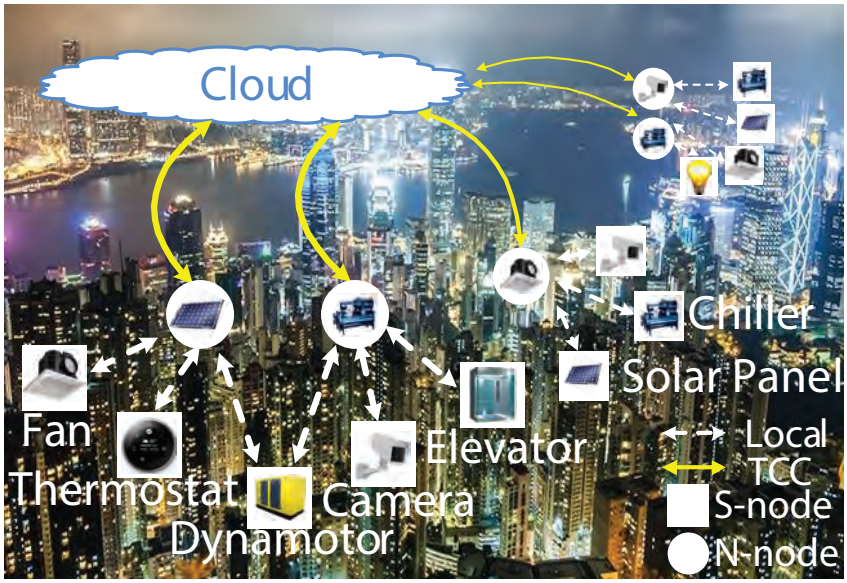


Figure 3.1: Smart After-Sales Maintenance Services (SAMS).

the cost of CAT1 might be justifiable for a chiller, yet it may be too costly for a fan.

We see a clear gap between the possible choices of TCC links, and the number of requirements on different costs and data rates from the IoT applications. To address this issue, we propose Sharing Tube plus (sTube+) for IoT communication sharing. The objective of sTube+ is to organize a greater number of IoT devices, with heterogeneous data communication requirements to efficiently share fewer choices of TCC links, and transmit their data to the cloud. An example SAMS application using sTube+ is shown in Fig. 3.1.

To bring sTube+ into reality, the challenges not only lie in the TCC link sharing optimization, but also that there is currently *no* architecture for IoT communication sharing data delivery. We propose a design approach of centralized price optimization and distributed network control. We architect a layered architecture for data delivery, optimize TCC link sharing, and prototype a functioning sTube+ system. We evaluate sTube+ with experiments and simulations. Finally, we present a SAMS case study. In this case study, we collect data from chillers and pumps, two core components of

a centralized HVAC system, and analyze their performance in the cloud. sTube+ serves as the underlying architecture in this case study.

The rest of this chapter is organized as follows. Section 3.2 presents the motivation. Section 3.3 describes the dedign details of sTube+. We formalize a set of problems for IoT communication sharing optimization, and develop algorithms with provable bounds in Section 3.4. We specially study IoT communication sharing under the pay-as-you-go pricing model in Section 3.5. We prototype a fully functioning system for sTube+ in Section 3.6. We comprehensively evaluate sTube+ in Section 3.7. In particular, we develop a SAMS case study, using sTube+ as the underlying architecture in Section 3.8. We conclude this chapter in Secion 3.9.

3.2 The Motivation

The state-of-the-art wireless communication channels provide a variety of choices that trade off communication range, data rate, and costs for different application needs. Yet the granularity of thing-to-cloud communication choices may not be enough, in the sense that for each IoT device with its own cost and data rate requirement, we cannot find a well-matched thing-to-cloud communication channel.

Readily available self-contained solutions, e.g., 3G/4G [43], are provided by ISPs. 3G/4G are over powerful and expensive for most IoT applications. Alternative solutions include LTE Category 1 (CAT1) released in 2016 and the to-appear LTE Category 0 (CAT0). New choices are being developed, yet the progress can not match the surging requirements. More importantly, there may be requirements that will never be developed by ISPs. For example, CAT1 has a monthly cost at around \$1 for a data volume of 45 MB. Assume that an equipment has a data volume of 50 MB but it can only afford \$1. ISPs will not deliberately develop such plan since it makes CAT1 non-marketable. In a sharing environment, a close-by equipment with

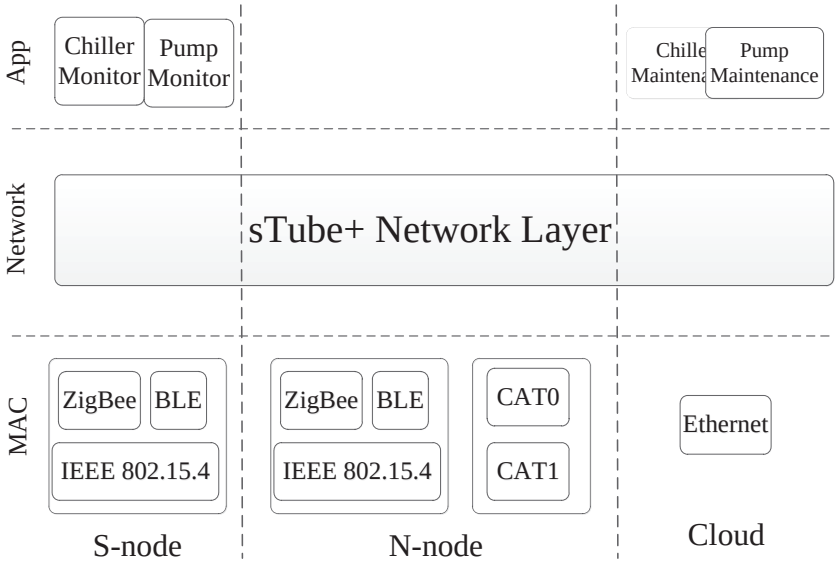


Figure 3.2: A Layered Architecture of sharing tube plus system.

residual data of 5 MB per month can be shared.

There are communication channels that are free but can only form a local *(LOC)* network. Short-range channels include Zigbee, Bluetooth, etc. They are good for device-to-device communication. WiFi, LoRa and SigFox [49] can provide longer-range wireless access. These are not self-contained since gateways are needed to reach the cloud outside. In our design, IoT devices will form LOC networks so as to share the TCC links. This work, however, will not emphasize on the design of the LOC networks.

3.3 The Stube+ Architecture

3.3.1 A Layered Architecture for Data Delivery

An End-to-End Approach: In SAMS, each S-node represents an equipment. Even though in our scenario, all equipment belongs to the same vendor, they differ greatly in operation, maintenance and services. Each of the S-node and its associated cloud application can be individually developed, e.g., by sub-divisions of the vendor, and

there may need possible future function extensions. We thus choose an end-to-end approach and let N-nodes only be responsible for traffic forwarding. From the application's point of view, the S-node talks with the cloud directly, see Fig. 3.2.

Note that the end-to-end approach requires the hardware of the S-nodes to be able to support the IP layer. We believe that this is reasonable since the accumulated value of the collected data in long-term should outweigh such one-time hardware overhead.

Network Topology Control: With an end-to-end design, the cloud, N-nodes and S-nodes are all involved in the network layer. We now study how the topology/nexthops should be managed.

For a very small scale network, the cloud can adopt a centralized design, where it computes all connections and broadcasts the peering results. For a general network, the cloud should not be triggered by micro-level dynamics, i.e., the peering among N-nodes and S-nodes. We choose to let the cloud only manage and monitor the *data budgets* of N-nodes, i.e., the data volume allocated to an N-node for its TCC link in a period of time. Note that the data budgets of the N-nodes in a sub-area may be exhausted because certain S-nodes have unexpected traffic, other N-nodes fail, new S-node joins, etc. Nevertheless, the number of N-nodes is much smaller than the number of equipment in the system and the frequency of budget allocation and updates is low.

The nexthop of an N-node is the cloud directly.* In this work, we also do not consider multiple wireless hops where an S-node uses other S-nodes to relay its data. As such, the remaining issue is to settle the peering between S-nodes and N-nodes.

Our objective is to minimize the complexity. We make two choices. First, we choose to let S-nodes take the initiative to manage the peering with the N-nodes.

*Theoretically, an N-node can route from other N-nodes; yet this increases the complexity and is not necessary for common cases.

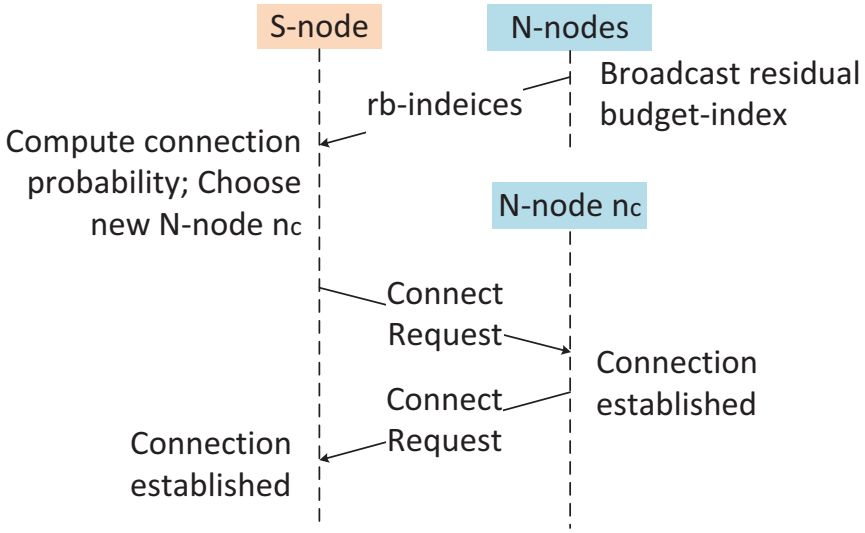


Figure 3.3: Illustration of nodes interaction when periodically choosing N-node.

In particular, an S-node may need to use the data budget of multiple N-nodes. Therefore, letting S-nodes, rather than N-nodes, take the initiative has much fewer overheads. Second, we develop an N-node peering algorithm, where each S-node makes independent decisions, yet the joint force collectively adapts to various network and data budget dynamics.

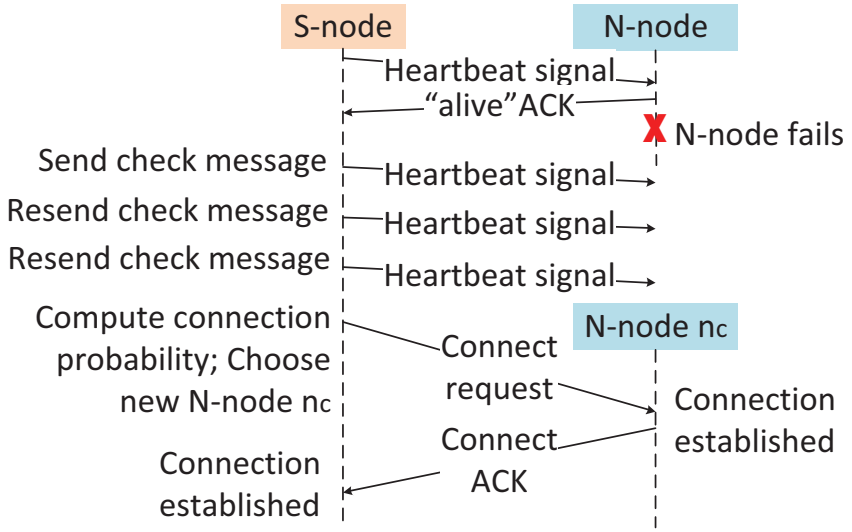


Figure 3.4: Illustration of nodes interaction when the connected N-node is failure.

The N-node peering algorithm (N-peering) of the S-nodes: Each S-node maintains a set of neighboring N-nodes. Each N-node periodically broadcasts its *residual budget-index (rb-index)* to all its neighboring S-nodes. This rb-index is designed as an increasing function of its remaining data budget. Periodically, S-node will select to connect to one N-node based on these rb-indices sent from its neighboring N-nodes. Specifically, let $\mathcal{N}$ be the set of neighboring N-nodes of an S-node and I_i denote N-node i's rb-index. The S-node computes connection probabilities to each neighboring N-node as $p_i = \frac{I_i}{\sum_{j\in\mathcal{N}} I_j}$, and connects to one of them according to these probabilities. As a consequence, the N-node with a greater remaining budget has a greater probability to be selected. We show the nodes interact of the N-node peeing process in Fig. 3.3.

Neighbor maintenance of the S-nodes: Each S-node periodically sends heartbeat signals to check the availability/failure of its neighboring N-nodes, and updates neighbor if the original neighbor fails. The process is shown in Fig. 3.4. Specifically, the S-node periodically sends heartbeat signal to the connected N-node and waits "alive" ACK from it. If the S-node does not receive respond from the N-node in time t, it resends heartbeat signal to the connected N-node. If the S-node does not receive any respond after three heart signal, the S-node regards the N-node as failure and updates to a new neighbor. To minimize possible data loss, the S-nodes with a higher data rates will have a shorter checking period. Let T_{ci} be the *checking period* of S-node i. Let D_i be the successive data loss that can be tolerated. Let r_i be the data rate of S-node i. Let T_u be the period needed to connect to a new neighbor. The S-node i sets $T_{ci} = \frac{D_i}{r_i} - 3t - T_u$. Due to the reliable transmission provided by CoAP (described in Section. 3.6.1), the data loss of an S-nodes occurs only when the connected N-nodes fails.

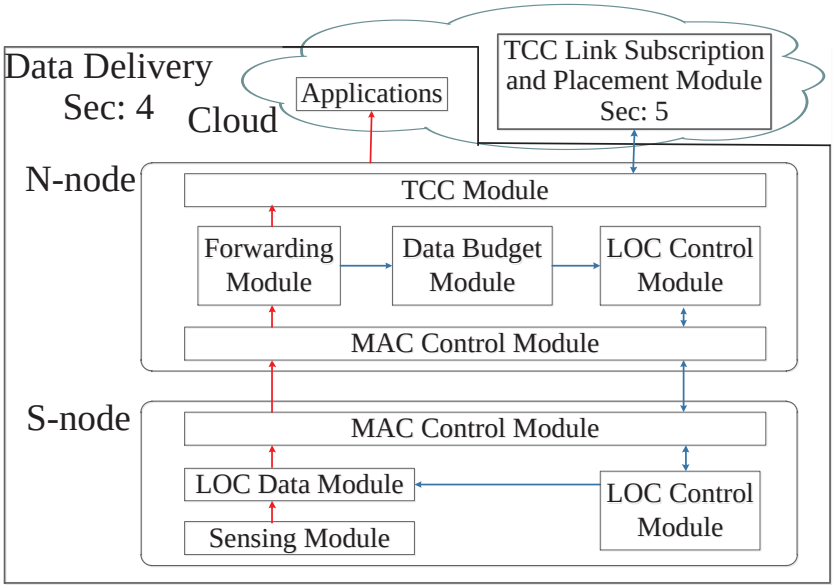


Figure 3.5: sTube+ module design.

3.3.2 Detailed Modules for a Functioning System

A functioning system has modules of all layers, see Fig. 3.5.

S-nodes have four modules. The *sensing* module connects to the equipment and collects sensing data. The *MAC control* module maintains the data link level connection between itself and the N-nodes within its communication range. The *LOC control* module maintains the network topology. The *LOC data* module transmits the data to the N-node.

N-nodes have five modules. *MAC control* module maintains the data link level connection between itself and the S-nodes. The *forwarding* module relays the data received from its MAC layer by forwarding the packet. The *TCC* module maintains the data link level connection between the N-node and the clouds. The LOC control module answers network layer queries from S-nodes. The *data budget* module maintains its data usage and accepts recharging from the cloud if necessary.

The cloud runs applications. The cloud has a centralized *TCC link subscription and placement* module. It computes data budgets (details in Section 5) of N-nodes.

3.3.3 Security Concerns

IoT systems face various security problems. Common problems and solutions can be found in [12, 61]. A specific security concern for SAMS is that a vendor does not want its data captured by other vendors. For example, an attacker may eavesdrop the data transmission from an S-node, or fake the identity of an N-node to conduct a man-in-the-middle attack. Here, the challenge in sTube+ is that S-nodes cannot connect to the Internet directly. As such, we need to maintain the integrity of N-nodes.

We address this problem by a simple authentication design. First, since each N-node is able to connect to the Internet, the communications between an N-node and the cloud can be safely established by using standard Transport Layer Security (TLS) protocols. Second, an S-node and N-node should also be able to verify each other and establish a safe communication link. This can be achieved via exchanging their public keys. The main issue here is that how the S-node and N-node can verify each other's public key when S-node is disconnected from the Internet. In our scenario, since S-nodes and N-nodes are produced by the same manufacturer, the manufacturer can hard code the certificate (derived from the manufacture's private key) when the node is produced, i.e., certificate pinning. The manufacture's public key is also pinned to the node. As a result, the two parties are able to verify each other even if they are disconnected from the Internet.

3.4 IoT Communication Sharing Optimization

The IoT Communication Sharing (ICS) problem is to answer which N-node (location) should reserve a TCC link from the ISP and how much budget they should reserve so as to minimize the overall monetary cost. In this section, we first present the network settings. Then we formulate the ICS problem as a cost minimization problem, analyze the problem complexity, then we design algorithm for ICS problem.

3.4.1 Network Topology

The considered network includes m S-nodes and q N-nodes. Let $\mathcal{N} = \{n_1, n_2, ..., n_q\}$ denote the set of N-nodes. An N-node can be either *installed* or *vacant*. Let f_j denote the indicator, i.e., $f_j = 1$ if installed; $f_j = 0$ if vacant. We define $\mathbf{f} \triangleq (f_1, f_2, \ldots, f_q)$. $\mathbf{f}$ is a decision variable to be optimized.

Let $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ be the set of S-nodes. Let $\mathcal{S}_j$ denote the subset of S-nodes, which can reach N-node n_j. We define $\mathbb{S} \triangleq \{\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_q\}$. Here, the term "reach" means that it is possible for the S-node to deliver its data to the N-node through some LOC links. We assume that $\bigcup_{j=1}^{q} \mathcal{S}_j = \mathcal{S}$, i.e., each S-node can reach at least one N-node. One of our design aims is to install a subset of N-nodes to cover all S-nodes, i.e.,

$$\bigcup_{j:f_j=1} \mathcal{S}_j = \mathcal{S}. \tag{3.1}$$

In the network, the thing-to-cloud communication (TCC) links connect N-nodes and the cloud, which are charged by ISPs. LOC links connect S-node and N-nodes, which are free.

3.4.2 Load Constraint Modeling

In this subsection, we discuss how each N-node is able to accommodate the data usage from its connected S-nodes. In each billing cycle (e.g., one month), S-node s_i requires to upload a data volume of u_i to the cloud. The S-node's data volume is split to be transferred via one or more N-nodes. We define $\mathbf{u} \triangleq (u_1, u_2, \ldots, u_m)$. Let v_{ij} be the split data volume of s_i transferred via n_j. We define $\mathbf{v} \triangleq (v_{ij} : \forall i = 1, 2, \ldots m, \forall j = 1, 2, \ldots, q)$. In this work, $\mathbf{u}$ is given as a priori, and $\mathbf{v}$ is to be optimized. We assume that u_i and v_{ij} values are integers (e.g., in kilobytes).

For the given u_i for S-node s_i, v_{ij} are decision variables to be designed. Since s_i's

data are transferred via its connected installed N-nodes, we have

$$\sum_{j:f_j=1} v_{ij} = u_i, \forall i = 1, 2, \ldots, m. \tag{3.2}$$

The data of s_i cannot be uploaded via N-nodes that are vacant or out of its communication range. Therefore, we have

$$v_{ij} = 0, \forall i, j : f_j = 0 \vee s_i \notin \mathcal{S}_j. \tag{3.3}$$

In each billing cycle, by paying to the ISP, each N-node could purchase a data volume allowed to upload to the cloud via its TCC link. Let d_j denote the data volume purchased by n_j. We define $\mathbf{d} \triangleq (d_1, d_2, \ldots, d_q)$. $\mathbf{d}$ is a decision variable. If n_j is installed, the load w_j at the TCC link of n_j is the accumulated data amount uploaded by its connected S-nodes. Therefore, we have $w_j = \sum_{i:s_i \in \mathcal{S}_j} v_{ij}$. $w_j = 0$ if n_j is vacant. The load w_j at the TCC link of node n_j cannot exceed the purchased data volume d_j. Therefore, we have

$$\sum_{i:s_i \in \mathcal{S}_j} v_{ij} \leqslant d_j, \forall j = 1, 2, \ldots, q. \tag{3.4}$$

3.4.3 The Cost of TCC Sharing

For TCC links, let $\mathcal{C} = \{c_1, c_2, \ldots, c_l\}$ be the set of available plans provided by ISPs, where k denotes the number of plans. The monetary cost for data volume x of plan c_i $(i = 1, 2, \ldots, l)$ can be presented as a function $c_i(x)$.[†]

Let $c'_j \in \mathcal{C}$ denote the plan adopted by N-node n_j. We define $\mathbf{c'} \triangleq (c'_1, c'_2, \ldots, c'_q)$. Once a plan is selected, it cannot be changed within the billing cycle. For the N-node

[†]For different pricing models, the form of cost functions are different. For example, the monthly plan (MP) pricing model, ISPs provide a set of monthly data plans. For data plan m_i, let t_i denote the price charged for the fixed amount of cap usage(denoted as k_i). If the data usage hits this cap then a higher price c is charged for each per data usage unit above the cap, the cost function can be presented as

$$c_{m_i}(x) = \begin{cases} t_i & , x \leqslant k_i \\ t_i + c(x - k_i) & , x > k_i. \end{cases}$$

The pay-as-you-go pricing model is discussed in Section 3.5.

n_j with purchased data volume d_j, the communication cost of n_j can be present as $c'_j(d_j)$. Thus, the total communication cost for q N-node locations is given as:

$$T_{total} = \sum_{j=1}^{q} c'_j\left(d_j\right). \tag{3.5}$$

3.4.4 IoT Communication Sharing Problem Formulation

The goal of the IoT Communication Sharing (ICS) problem in this work is to minimize the overall monetary cost of the TCC links, given a set of available plans, the network topology, and the possible locations for S-nodes and N-nodes. Hence, the overall monetary cost of the TCC links is the objective function. The decision variables are the installation indicators $\mathbf{f}$, the adopted cost function $\mathbf{c}'$, the subscribed data volume $\mathbf{d}$ and the data volume $\mathbf{v}$ from each S-node s_i uploaded from each N-node n_j. The constraints are shown in Sections 3.4.1 and 3.4.2. In summary, we have the following optimization problem:

Problem 3.1. *(ICS) Given $\mathcal{S}$, $\mathcal{N}$, $\mathbf{u}$ and $\mathcal{C}$, determine $\mathbf{f}$, $\mathbf{c}'$, $\mathbf{d}$ and $\mathbf{v}$, subject to constraints (3.1), (3.2), (3.3) and (3.4), to minimize $T_{\text{total}} = \sum_{j=1}^{q} c'_j\left(d_j\right)$.*

3.4.5 Problem analysis

Theorem 3.1. *Problem ICS is NP-complete.*

Proof. We prove this theorem by transforming the problem into the minimum set cover problem. Consider a special case that $u_i = 1, \forall i = 1, 2, \ldots, m$. Let the monetary cost at each N-node be a if it is installed, be 0 if vacant. Therefore, we aim to minimize the number of installed N-nodes. As a result, Problem 3.1 is equivalent to an optimal set cover problem: to select a minimum number of sets from $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_q\}$ that covers all elements in the $\mathcal{S}$. □

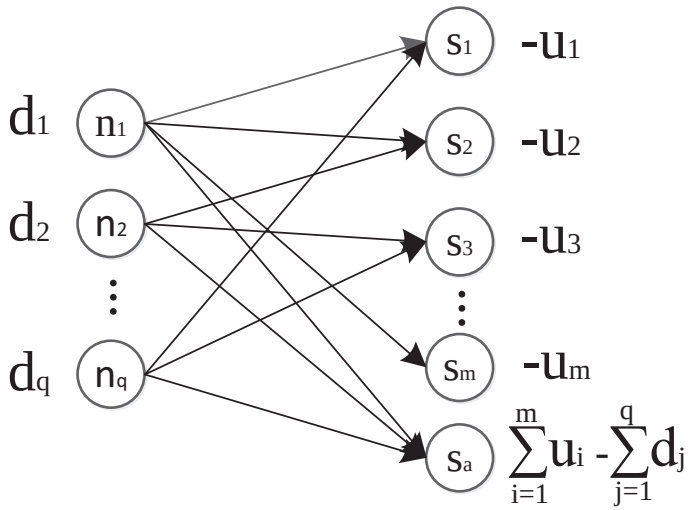


Figure 3.6: Illustration of conversion to minimum cost flow problem.

3.4.6 The ICS Algorithm

In this section, we solve the ICS problem. However, ICS problem is NP-complete, it is unrealistic to find a globally optimal solution within polynomial time. We design Minimize Communication Cost algorithm (denoted as MCC) which achieves a locally optimal solution. The rationale to develop MCC is as follows. We need to determine 1) the purchasing strategy, i.e., $\mathbf{c}'$ and $\mathbf{d}$ values, and 2) the data upload scheme $\mathbf{v}$. Accordingly, we develop two sub-functions: `best-Plan()` and `best-Upload()`. Given the data upload scheme $\mathbf{v}$, `best-Plan()` will search for the minimized cost purchasing strategy i.e., $\mathbf{c}$ and $\mathbf{d}$. Given $\mathbf{c}$ and $\mathbf{d}$, `best-Upload()` will find an even better data upload scheme $\mathbf{v}$. `best-Plan()` and `best-Upload()` are then conducted alternatively to gradually improve the overall cost.

Given the data upload scheme $\mathbf{v}$, finding the best purchasing strategy (i.e, `best-Plan()`) can be optimally solved. `best-Plan()` first computes the data usage of each N-node according to $\mathbf{v}$. Then, knowing the data usage, it can find the minimized cost purchased strategy for each N-node by simply searching within all plans in $\mathcal{C}$.

Given the purchasing strategy (i.e., $\mathbf{c}'$ and $\mathbf{d}$), finding the best data upload scheme

$\mathbf{v}$ (i.e., `best-Upload()`) can be converted to the minimum cost flow problem. We illustrate the conversion in Fig. 3.6. We construct a graph which contains q N-nodes, m S-nodes and one auxiliary S-node as the vertices. If $s_i \in \mathcal{S}_j$, then an edge is added between s_i and n_j. For the auxiliary S-node, we add edges between it and all N-nodes. For N-node n_j, we attach the (positive) purchased data volume d_j to it. For each S-node s_i, we attach the (negative) data usage $-u_i$ to it. For the auxiliary S-node s_a, we attach the negative data usage $\sum_{i=1}^{m} u_i - \sum_{j=1}^{q} d_j$ to it. For the edges connected to n_j, the attached unit delivery costs are $\frac{c'_j(d_j)}{d_j}$. Our problem now is equivalent to obtaining the minimum cost flow through the network. The N-nodes are "sources" for the flow entering the system and the S-nodes are "sinks" where flow leaves the system. Brenner's algorithm [19] is used in `best-Upload()` to solve the minimum cost flow problem with the computational complexity of $\mathcal{O}(q(\log q)^2(m+1)^2)$.

Algorithm 3.1: Minimize Communication Cost, `MCC()`.

Input: $\mathcal{S}, \mathcal{N}, \mathbb{S}, \mathbf{u}, \mathcal{C}$
Output: $\mathbf{c}, \mathbf{d}, \mathbf{f}, \mathbf{v}$

```
1 v ← 0; f ← 0; c' ← 0; d ← 0; ṽ ← 0;
2 v ← init-Upload(S, N, 𝕊, u);
3 repeat
4 |   ṽ ← v;
5 |   [c', d] ← best-Plan(C, v);
6 |   v ← best-Upload(S, N, 𝕊, u, c', d);
7 until ṽ == v;
8 f ← compute-Indicator(d);
9 return c, d, f, v;
```

The overall algorithm `MCC()` is an iterative algorithm shown in Algorithm 3.1. The overall algorithm first calls `init-Upload()` (line 2) to initialize each $v_{ij} \in \mathbf{v}$ to u_i and then it calls `best-Plan()` (line 5) to determine the purchasing strategy $\mathbf{c}$ and $\mathbf{d}$. Such $\mathbf{c}$ and $\mathbf{d}$ are given to `best-Upload()` (line 6). `best-Upload()` will adjust the data upload scheme $\mathbf{v}$ according to the purchasing strategy. Such $\mathbf{v}$ is returned to `best-Plan()`. The termination condition for iteration is that there is no change

in the data upload scheme $\mathbf{v}$. Then `MCC()` calls `compute-Indicator()` to compute the installed/vacant indicator of N-nodes (line 8) according to $\mathbf{d}$ (if $d_j > 0$, then $f_j = 1$. Otherwise $f_j = 0$).

We now analyze the convergence of `MCC()`. Let $\mathbf{d}_y$ denote the purchased data volume $\mathbf{d}$ result of the `best-Plan()` in the y-th round iteration. The maximum value of each entry of $\mathbf{d}$ is limited to $d_{\max}$. According to Brenner's algorithm, $\mathbf{d}_{y+1} \preceq \mathbf{d}_y$ when `best-Plan()` is called. Since $\mathbf{d}$ is bounded (i.e., $\mathbf{0} \preceq \mathbf{d}$), $\mathbf{d}_y$ will converge after a finite number of iterations. In each iteration at least one entry of $\mathbf{d}$ is decreased by 1. Therefore, the algorithm will converge at most in r rounds, to a local optimal, where $r = qd_{\max}$. `MCC()` is a polynomial-time algorithm with the computational complexity of $\mathcal{O}(rq(\log q)^2(m+1)^2 + rql)$.

3.5 ICS in the Pay-As-You-Go Pricing Model

We now specifically consider the pay-as-you-go (PAYG) pricing model. This is because PAYG is likely to be the primary pricing model for IoT communication for this moment when the IoT industry is still in its early stage. Looking into the history, PAYG is always the pricing model in early stages of a new business, e.g., pay per call, pay per megabyte of data. Monthly plan (MP) emerges when the business becomes mature and as a mean of price reduction when facing competition [22]. As a matter of fact, in our experiment, the CAT1's pricing model is PAYG.

3.5.1 Problems

For the PAYG pricing model, $\mathcal{C} = \{c_1\}$, i.e., there is only one plan for PAYG. The cost function is represented by Eq. (3.6). This is a staircase function. Here x is the data usage; L is an integer to denote the step size of pricing model. Let p_i be the price for the i-th step of L data volume. In practice, p_i decreases as the price step

increases [41] and $\lim_{i\to+\infty} p_i = p_{\min}$, where $p_{\min}$ is positive.

$$c_1(x) = \sum_{i=1}^{\lceil \frac{x}{L} \rceil} p_i. \tag{3.6}$$

The overall cost using PAYG is $\sum_{j=1}^{q} c_1 \left(\sum_{i:s_i \in \mathcal{S}_j} v_{ij} \right)$. Thus, we arrive the following problem:

Problem 3.2 (ICS-PAYG)**.** *Given $\mathcal{S}$, $\mathcal{N}$, $\mathbb{S}$, $\mathbf{u}$ and c_1, determine $\mathbf{f}$, $\mathbf{d}$ and $\mathbf{v}$, subject to constraints (3.1), (3.2), (3.3) and (3.4), to minimize $T_{\text{total}} = \sum_{j=1}^{q} c_1 \left(\sum_{i:s_i \in \mathcal{S}_j} v_{ij} \right)$.*

In reality, we notice that many S-nodes can reach a limited number of N-nodes. We therefore consider a case that the degree of an S-node (i.e., the number of N-nodes an S-nodes can reach) is limited to D. We have the following problem:

Problem 3.3 (ICS-D-PAYG)**.** *Given $\mathcal{S}$, $\mathcal{N}$, $\mathbb{S}$, $\mathbf{u}$, D and c_1, determine $\mathbf{f}$, $\mathbf{d}$ and $\mathbf{v}$, subject to constraints (3.1), (3.2), (3.3) and (3.4), to minimize $T_{\text{total}} = \sum_{j=1}^{q} c_1 \left(\sum_{i:s_i \in \mathcal{S}_j} v_{ij} \right)$.*

Theorem 3.2. *Problems ICS-PAYG and ICS-D-PAYG are both NP-complete.*

The proof is similar to the proof of Theorem 3.1. Intrinsically, the complexity comes from N-nodes covering S-nodes, rather than the pricing model; thus, the complexity of NP-completeness holds.

3.5.2 Algorithms

We develop Fast N-node Deployment (FND) algorithm for the ICS-PAYG problem, and Layering N-node Deployment (LND) algorithm for the ICS-D-PAYG problem.

The problem ICS-PAYG and ICS-D-PAYG can be divided into three subproblems: N-node placement to cover all S-nodes (i.e. to find $\mathbf{f}$), the upload scheme (i.e. to find $\mathbf{v}$) and data volume subscription at each installed N-node (i.e. to find $\mathbf{d}$).

FND and LND first solve the N-node placement to cover all S-nodes and the upload scheme. The TCC link placement to cover all S-nodes is a set cover problem. For the ICS-PAYG problem, FND adopts the greedy set cover algorithm in [77]. For the ICS-D-PAYG problem, LND employs the layering set cover algorithm in [22] which takes advantage of the degree information of S-nodes. In this way, the greedy set cover algorithm and the layering set cover algorithm select N-node one by one. Whenever an N-node is selected, the newly covered S-nodes will upload all their data volume via this N-node.

After the above steps to determine $\mathbf{f}$ and $\mathbf{v}$ for both FND and LND, each installed N-node subscribes the closest data cap that is greater than the data volume needed to be transferred via it, so that $\mathbf{d}$ is determined.

Theorem 3.3. *The approximation ratio of the algorithm FND for ICS-PAYG is* $\frac{p_1}{p_{\min}}(\ln m + 2)$.

Proof. Let the cost of FND be r_{f}, and the optimal cost of ICS-PAYG (denoted as OPT) be r_{o}. Directly proving $\frac{r_{\mathrm{f}}}{r_{\mathrm{o}}} \leqslant \frac{p_1}{p_{\min}}(\ln m + 2)$ is hard, we prove this by divide and conquer. As the cost is related to the number of installed N-node and the number of purchased data volume steps under the PAYG pricing model, we first prove the installed N-node number of FND (denoted as k_{f}) and OPT (denoted as k_{o}) meets $\frac{k_{\mathrm{f}}}{k_{\mathrm{o}}} \leqslant \ln m + 1$, then we prove the purchased steps of FND (denoted as t_{f}) and OPT (denoted as t_{o}) meets $t_{\mathrm{f}} \leqslant t_{\mathrm{o}} + k_{\mathrm{f}}$. Based on these, we can prove $\frac{r_{\mathrm{f}}}{r_{\mathrm{o}}} \leqslant \frac{p_1}{p_{\min}}(\ln m + 2)$.

We first prove $\frac{k_{\mathrm{f}}}{k_{\mathrm{o}}} \leqslant \ln m + 1$. Let $k_{\min}$ be minimum number of N-nodes that can cover all S-nodes. We have $\frac{k_{\min}}{k_{\mathrm{o}}} \leqslant 1$ and $\frac{k_{\mathrm{f}}}{k_{\min}} \leqslant \ln m + 1$ (by the approximation ratio of greedy set cover algorithm). Then we have $\frac{k_{\mathrm{f}}}{k_{\mathrm{o}}} \leqslant \ln m + 1$.

We then prove $t_{\mathrm{f}} \leqslant t_{\mathrm{o}} + k_{\mathrm{f}}$. Let X denote the total data usage of all S-nodes, and $x = Lq - r$, where $q \in \mathbb{N}, 0 \leqslant r < L$. $q \leqslant t_f$, $q \leqslant t_o$ (otherwise the purchased data volume of FND and OPT is smaller than total data usage x). Let x_j $(j = 1, 2, \ldots, k_{\mathrm{f}})$

denote the data usage of the installed N-node j of FND, and $x_j = Le_j - r_j$, where $e_j \in \mathbb{N}, 0 \leqslant r_j < L$. e_j is the steps purchased by the installed N-node j, thus $t_{\mathrm{f}} = \sum_{j=1}^{k_{\mathrm{f}}} e_j$, $x = qL - r = L\sum_{j=1}^{k_{\mathrm{f}}} e_j - \sum_{j=1}^{k_{\mathrm{f}}} r_j$, we have $t_{\mathrm{f}} = q - \frac{r}{L} + \frac{\sum_{j=1}^{k_{\mathrm{f}}} r_j}{L}$, as $0 \leqslant r_j < L$, we have $t_{\mathrm{f}} \leqslant q + \frac{k_{\mathrm{f}} L}{L} = q + k_{\mathrm{f}}$.

At last, we prove $\frac{r_{\mathrm{f}}}{r_{\mathrm{o}}} \leqslant \frac{P_1}{P_{\min}}(\ln m + 2)$ based on the above proof. The price of each step ranges from p_1 to $p_{\min}$, thus $r_{\mathrm{f}} \leqslant p_1 t_{\mathrm{f}}$ and $r_{\mathrm{o}} \geqslant p_{min} t_{\mathrm{o}}$. $\frac{r_{\mathrm{f}}}{r_{\mathrm{o}}} \leqslant \frac{p_1 t_{\mathrm{f}}}{p_{\min} t_{\mathrm{o}}} \leqslant \frac{p_1}{p_{\min}}(\frac{t_o}{t_o} + \frac{k_f}{t_o})$, each installed N-node must purchase at least one step, thus $t_o \leqslant k_o$, so $\frac{r_{\mathrm{f}}}{r_{\mathrm{o}}} \leqslant \frac{p_1}{p_{\min}}(1 + \frac{k_{\mathrm{f}}}{k_{\mathrm{o}}}) \leqslant \frac{p_1}{p_{\min}}(\ln m + 2)$. □

Theorem 3.4. *The approximation ratio of the algorithm LND for ICS-D-PAYG is* $\frac{p_1}{p_{min}}(D + 1)$.

The difference between ICS-PAYG and ICS-D-PAYG comes from TCC link placement to cover all S-nodes which is equivalent to the minimum set cover problem. Different with the ICS-PAYG problem, the maximum degree of S-nodes D is given in the ICS-D-PAYG problem, thus the information about the degree of S-nodes can be utilized to improve N-node placement. Using the degree information, the number of installed N-nodes selected by LND is at most D times to the minimum set cover, while this ratio is $\ln m + 1$ in FND for ICS-PAYG problem.

FND and LND are based on greedy algorithms. Though they have bounded performance, they may not perform well in some special cases. If that happens, we also develop a heuristic `MCC-PAYG()` based on `MCC()`. As the pricing model is determined, there is no need to search plans for N-nodes. Thus, line 5 of `MCC()` can be replaced by computing the subscription data volume directly. The complexity of `MCC-PAYG()` is $\mathcal{O}(rq(\log q)^2(m+1)^2)$.

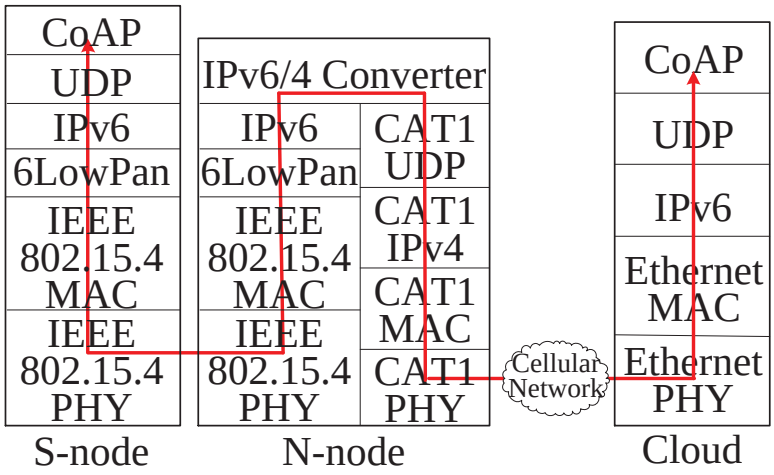


Figure 3.7: End-to-End Communication.

3.6 Implementation

We present an implementation of the sTube+ architecture. This includes the MAC layer, network layer, and application layer. We present a case study of a SAMS in Section 3.8, where we develop the sensing module to collect data from real equipment and conduct data analytics in the cloud.

3.6.1 The Network Stack

MAC layer: We implement IEEE 802.15.4, ZigBee, and Bluetooth as the MAC layer for the LOC network. We choose CAT1 as the MAC layer for the TCC link.

Network layer: We choose 6LoWPan (IPv6) as the networking layer protocol. There are two special challenges.

The first is that our CAT1 only supports IPv4. Moreover, it only provides application layer interfaces. Thus, we develop an IPv6-IPv4 converter. It locates in the application layer of the N-node (see Fig. 3.7), yet it emulates the network layer. It has two functions: packet format transformation and IPv6-IPv4 address mapping.

For packet format transformation, the packet we get from the LOC network is an

IPv6 packet. We remove all headers to get the application packet. Then we put such packet to the CAT1 interface. The address mapping is done by mapping a group of IPv6 address to an IPv4 address (the address of CAT1) and a port. Every N-node establishes a table of the mapping. Each entry in this table is automatically inserted when the first packet from the S-node reaches the N-node, i.e., N-node allocates each S-node connected to it a universal port with the CAT1's IPv4 address.

The second challenge is that in practice, an S-node should have a fixed IP address. Yet in our implementation, each S-node gets its IPv6 address from N-node using the uIP library from Contiki, making the IP address dynamic. Since the interaction between an S-node and the cloud is bi-directional, the dynamic IP address can break the interaction. To this end, in the application layer, we develop a notification mechanism such that if the IP address of the S-node changes, the S-node will notify the cloud.

Application layer: We use CoAP and UDP for application layer protocols. sTube+ chooses the optional reliable transmission model of CoAP. Specifically, reliable transmission in CoAP is achieved by marking individual messages with the confirmable flag. When the cloud receives a confirmable message, it responds with an acknowledgment message to let the S-node know the message arrived. The S-node will automatically retransmit a confirmable message if an acknowledgment message is not received in the timeout interval.

3.6.2 The Routing Choice

In our IoT application context, data are routed from the S-nodes to the cloud. We choose RPL [93] for routing. RPL is a gradient routing technique that organizes nodes as a Direct Acyclic Graph (DAG) rooted at the sink. RPL has an objective function. The goal is to minimize the cost to reach the sink from any node. This function has to be customized. Recall that in our algorithm, we compute the amount of traffic an

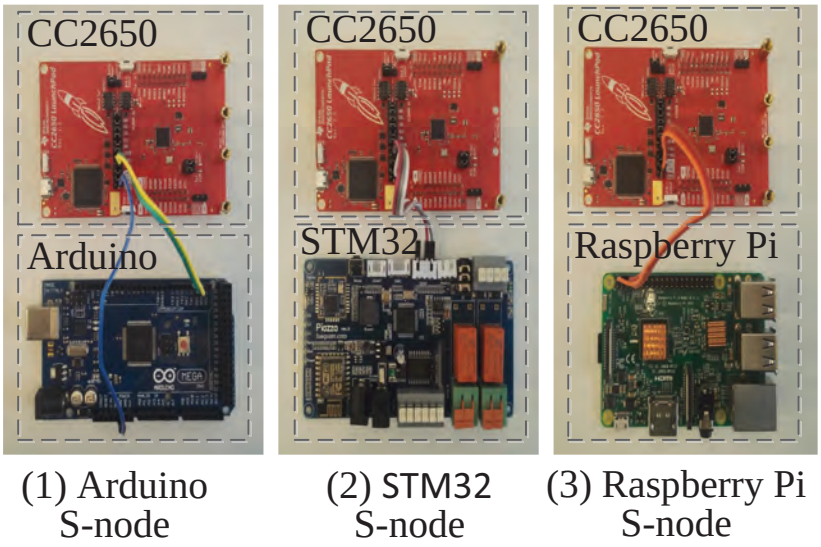


Figure 3.8: The S-node.

S-node sends to each peering N-node. In our implementation, the objective function maintains a "volume-N-node" table. The table records the residual data volume of the S-node can be transmitted through its peering N-node. The objective function chooses the N-node with residual data volume in a round-robin fashion.

3.6.3 Hardware Choices

The S-nodes: We use Arduino MEGA 2560, STM32 and Raspberry Pi 3 Model B as the S-node hardware board (Fig. 3.8) by considering the different requirements of the capability of hardware board from the equipment and the hardware cost. For example, for the Fan, only the speed of fan should be sensing and the cheap Arduino board can meet its requirement; While for the chiller, the S-node gains the sensing data from the chiller control interface and Modbus RTU protocol should be run on hardware board, thus the more powerful and expensive raspberry pi should be adopted. For the LOC module, we use a Texas Instruments CC2560 SimpleLink^{TM} Wireless MCU for the 802.15.4 radio interface.

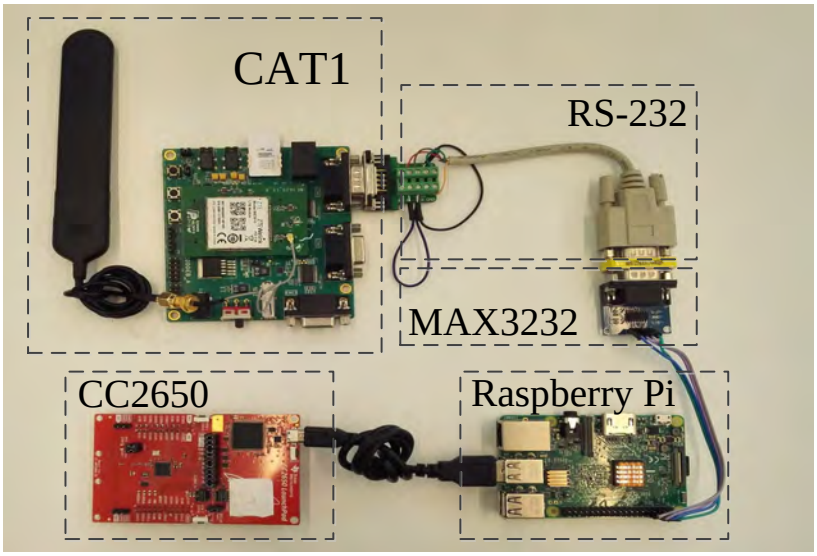


Figure 3.9: The N-node.

The N-nodes: We use a Raspberry Pi 3 Model B as the N-node platform (Fig. 3.9). For LOC side, we use a Texas Instruments CC 2560 SimpleLink^{TM} Wireless MCU for the 802.15.4 radio interface. Then this module is connected to Raspberry Pi using a USB-to-serial cable. For the TCC side, As the interface of Raspberry Pi is TTL, while the interface provided by CAT1 is RS-232, we use the MAX3232 as a converter. The baud rate of the serial port is 19200 bits per second, i.e. 2400 bytes per second. The CAT1 module supports speeds of 5 Mbps upload and 10 Mbps download. Thus, the maximum sample rate the proposed approach can adapt is 2400 bytes per second. We rent CAT1 data plans from Telecom Anonymity.

The Cloud: We rent a server in Cloud Anonymity with 8 cores of 2.5 GHz, and a total memory of 128GB. The data in the cloud are stored in XML format.

3.7 Performance Evaluation

3.7.1 Evaluation by Experiments

1) System Setup:

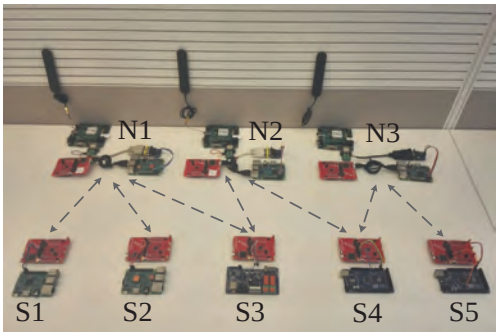


Figure 3.10: The network topology of the experiments.

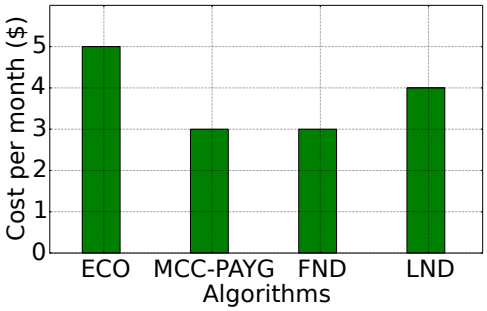


Figure 3.11: The monthly cost of different algorithms.

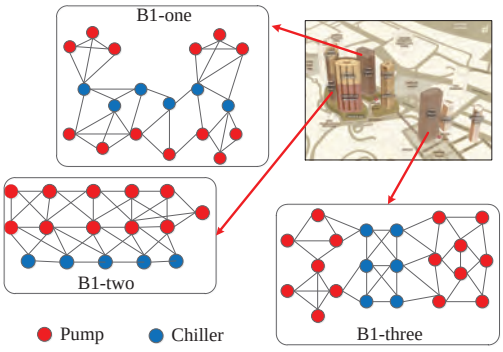


Figure 3.12: Illustration of the topology of B1.

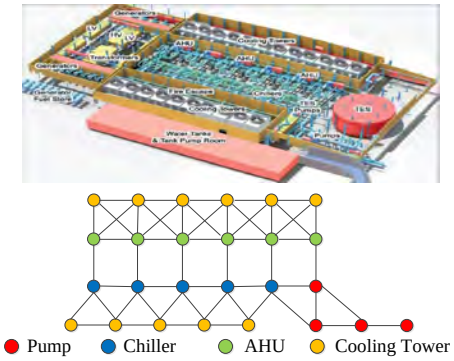


Figure 3.13: Illustration of the topology of B2.

The network topology is shown in Fig. 3.10. There are three N-nodes and five S-nodes. The links are configured as in the figure. We set the data traffic for S_1 and S_2 to be 200 bytes every three minutes, and the data traffic for S_3, S_4 and S_5 to be 600 bytes every minute.[‡] We adopt the PAYG model from China TeleCom. Each 40 MB costs \$1, i.e., $L = 40$, $p_1 = p_2 \ldots = p_{\min} = 1$ in Eq. (3.6).

We compare three IoT communication sharing algorithms MCC-PAYG, FND and LND with the exclusive channel occupation (ECO) algorithm, i.e., each device transmits its data directly to the cloud via its dedicated purchased TCC link.

[‡]Our S-nodes and N-nodes do not connect to equipment, since 1) our IoT communication sharing is general for all types of equipment, and 2) we admit that we do not have enough Fans/Chillers for an eight-node experiment.

Table 3.1: Monthly data usage of different equipment.

	Chiller	Pump	AHU	Tower
B1	6.62 MB	4.25 MB	-	-
B2	9.26 MB	3.65 MB	12.45 MB	3.90 MB

Equipment number	B1			B2
	one	two	three	
Chiller	6	5	6	5
Pump	13	11	15	4
AHU	-	-	-	6
Tower	-	-	-	11

Table 3.2: The number of equipment at B1 and B2.

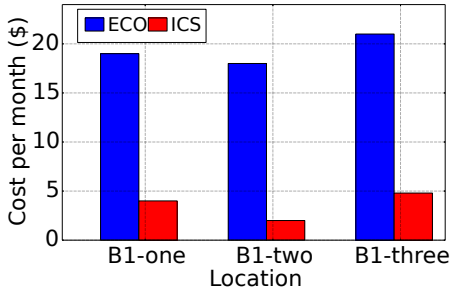


Figure 3.14: The monthly cost at B1 under the PAYG pricing model.

2) Experiment Results: The system is turned on for 6 hours and the overall data usage is scaled to one month. We derive the overall monthly cost of different algorithms. The results are shown in Fig. 3.11. We can observe that under the PAYG model, MCC-PAYG, FND and LND lead to a cost saving of 20%–40% as compared with ECO. This matches our expectation since IoT communication sharing will bring significant cost reductions. Next, we will evaluate a variety set of network configurations by trace-driven simulations, and we will see that the saving is more significant when the network is larger.

3.7.2 Evaluation by Trace-driven Simulations

We now use trace-driven simulations to evaluate ICS. We first present two real-world cases and price models employed in the evaluation. We then show the result of the cost reduction introduced by ICS. We also show how ICS effectively makes use of the purchased data volume of TCC link, and the performance of algorithms for the PAYG pricing model.

1) Simulation Setup: We evaluate ICS and algorithms by simulation using two

Table 3.3: The monthly data plans.

Type	1	2	3	4	5
Price	2 $	5 $	7 $	15 $	25 $
Volume	30 MB	100 MB	200 MB	500 MB	1 GB

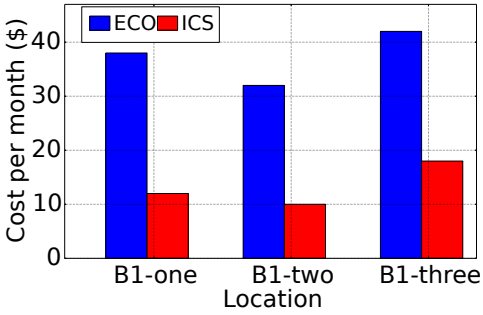


Figure 3.15: The monthly cost at B1 under the MP pricing model.

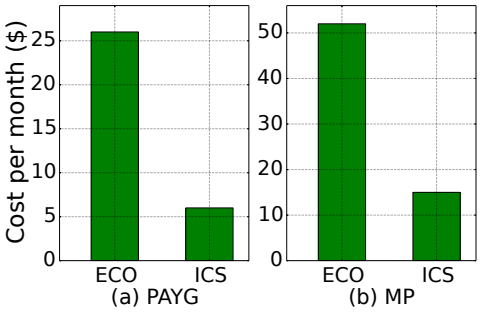


Figure 3.16: The monthly cost at B2 under different pricing model.

real-world cases.

Case 1: We work on a SAMS application and we collect data of building group belong to Anonymity property Ltd (denoted as B1). It consists of three buildings (denoted as B1-one, B1-two and B1-three). We collect the data of chillers and pumps which belong to a same vendor Anonymity. The chillers and the pumps of each building are located in the plant room on the top floor. The number of the chillers and the pumps of each building of B1 is showed in Table 3.2. Each equipment connects to an S-node and can be regarded as a possible location of N-node. The network topology of B1 is shown in Fig. 3.12.

We collected four types of data for the chiller to compute COP, such as the supplying/returning chilled water temperature. We collect data of the power input and the heat transfer to circulating water for the pump to compute Water Transfer Coefficient (WTC) [§] The data of the chillers and the pumps were collected at 30 minute intervals. The monthly data volume collected from each chiller and each

[§] WTC is a performance index of the pump. The pump should be maintained before the WTC under a threshold.

pump are showed in Table 3.1.

Case 2: We also employ the publicly available data. We use the data of a building (denoted as B2) located at Kuwait University [71]. The data are from chillers, pumps, air handling units (AHU) and cooling towers of B2. The number of each kind of equipment is showed in Table 3.2. Each equipment can be regarded as an S-node and a possible location of N-node. The network topology of B1 is shown in Fig. 3.13.

For the traffic of chiller, pump and cooling tower, we use the real traffic collected in [14]. The data of chiller, pump and cooling tower were collected at one minute intervals. The data of chillers includes 8 types of data, such as work load, supply temperature and return temperature. For the traffic of AHU, we use the real traffic pattern described in [95], which also collects data at one minute intervals. The data of AHU includes 14 features including air mass flow rate, room air temperature, etc. The monthly data volume of the four types of equipment is shown in Table 3.1.

The pricing models: We study two pricing models: 1) PAYG, the first 40 MB costs \$1, i.e., $L = 40, p_1 = 1$, the prices of the following 40 MB steps are \$0.8, i.e., $p_2 = p_3, \ldots = p_{\min} = 0.8$ in Eq. (3.6); 2) MP, the monthly data plans are shown in Table 3.3 with \$0.6 charged for each 1 MB over the cap.

Evaluation Criteria: We evaluate ICS employing MCC algorithm and ECO with the settings of in Case1 and Case2 under the two pricing models. We also run MCC-PAYG, FND and LND algorithms for PAYG pricing model.

We evaluate the data transmission cost of ICS and ECO. We introduce *underutilized ratio* for TCC link. The underutilized ratio of TCC link is the ratio between the unused purchased data volume and the purchased data volume of TCC link. The underutilized ratio can indicate how effectively the purchased data volume of TCC link has been used. We evaluate the underutilized ratio of TCC link of ICS and ECO. We also evaluate the data transmission cost of MCC-PAYG, FND and LND

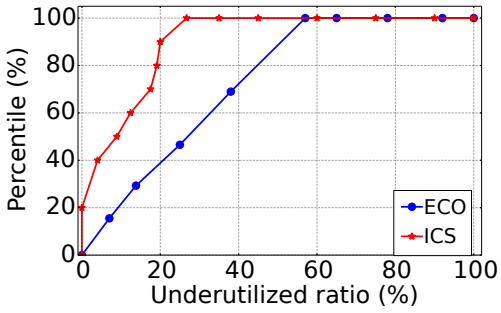


Figure 3.17: The CDF of the underutilized ratio of TCC links at B1.

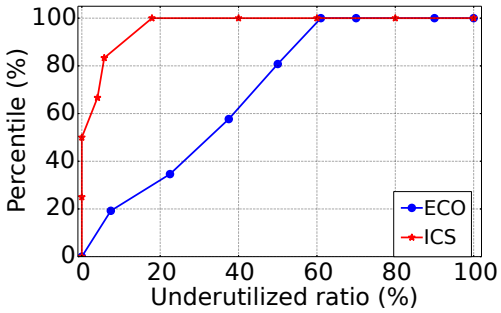


Figure 3.18: The CDF of the underutilized ratio of TCC links at B2.

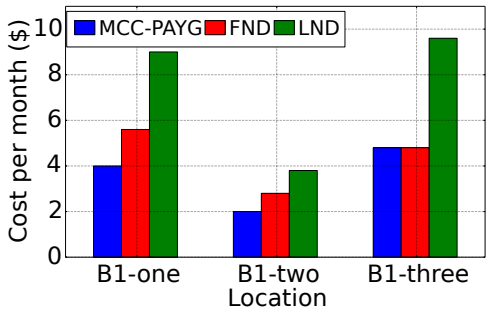


Figure 3.19: The monthly cost of different algorithms for PAYG at B1.

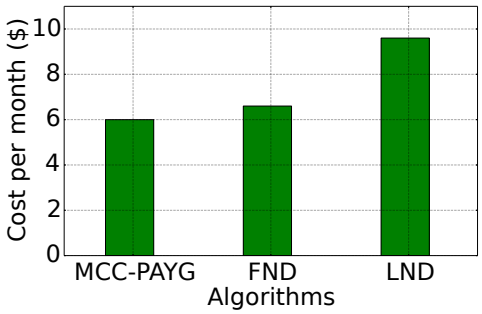


Figure 3.20: The monthly cost of different algorithms for PAYG at B2.

under PAYG pricing model.

2) Results:

Data Transmission Cost Reduction: We first compare ECO and ICS under two pricing models at B1 in Figs. 3.14–3.15. We see that ICS shows a higher cost saving compared with experimental results. This matches our expectation since the advantage of sharing becomes more significant when there are more S-nodes to share. For the PAYG pricing model, the cost of ECO is 4.8 times, 8 times and 4.4 times to that of ICS at B1-one, B1-two and B1-three respectively. For the MP pricing model, the cost of ECO is 3.2 times, 3.2 times and 2.4 times to that of ICS at B1-one, B1-two and B1-three respectively, a slightly less than that of PAYG. This is because, in MP pricing model, the cost gap between two adjacent plans is bigger, thus if the data

volume of one monthly data plan cannot meet the requirement of an N-node, the N-node has to purchase the other one whose price is much higher so that purchased data are underutilized, while in the PAYG model, the TCC link can purchase steps which are cheaper one by one.

In Fig. 3.16, we also compare the cost of ECO and ICS under the PAYG and MP pricing models at B2. We see cost savings of 78% and 71% on PAYG and MP respectively. This further confirms that ICS significantly outperform ECO.

The underutilized ratio of TCC link: We compare the underutilized ratio of TCC links under ECO and ICS. Please note that the higher underutilized ratio indicates the customer waste more data volume which has been paid. High underutilized data volume ratio discourages customers.

In Figs. 3.17–3.18, we show the cumulative distribution function (CDF) of TCC links' underutilized ratio of ECO and ICS at B1 and B2 under the PAYG pricing model where $L = 10$ MB. In Fig. 3.17, we can observe that 20% TCC link's underutilized ratio of ICS at B1 is 0% which means the data volume of these TCC links has been used up without waste. No TCC links' purchase can be fully used under ECO. We can also observe that the underutilized ratio of all TCC links of ICS is under 23%. For the ECO, the underutilized ratios of TCC links could arrive 57%.

The underutilized ratio gap between ECO and ICS is even greater at B2, shown in Fig. 3.18. We can see that 50% TCC links' underutilized ratio is 0%, while this value is still 0 for ECO. We can also observe that the underutilized ratio of all TCC links of ICS is under 18%. For the ECO, the underutilized ratio of TCC links could be as much as 61%. This illustrates that through IoT communication sharing, the purchased data volume of TCC link can be made better use of compared with ECO. This is also the reason why ICS can lead to a substantial cost reduction compared with ECO.

The performance of algorithms for PAYG: We compare our MCC-PAYG

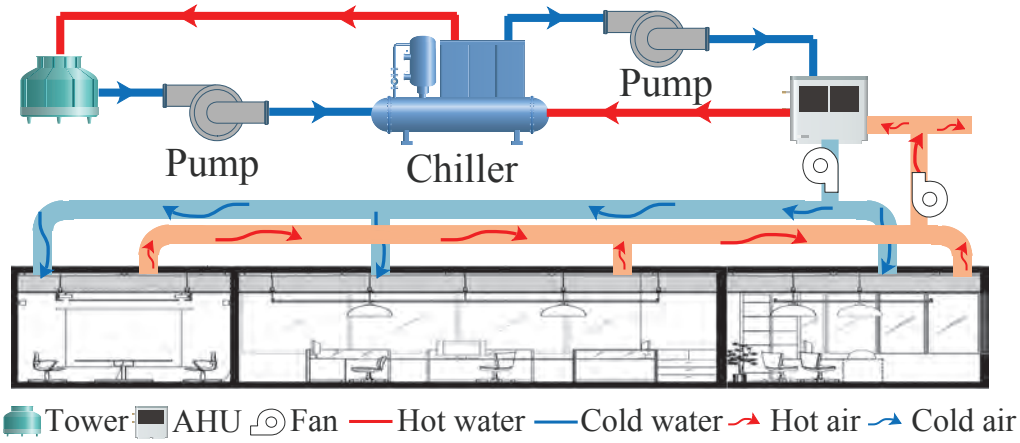


Figure 3.21: A typical centralized HVAC system.

algorithm with FND and LND algorithms for the PAYG pricing model.

In Figs. 3.19–3.20, we show the monthly cost of MCC-PAYG, FND and LND at B1 and B2 respectively. We notice that MCC-PAYG, FND and LND have a cost saving of 76%–88% compared with ECO (the costs of ECO at three building of B1 shown in Fig. 3.14 are \$19, \$16 and \$21 respectively, and \$26 at B2 shown in Fig. 3.16). These results illustrate that MCC-PAYG, FND and LND work effectively compared with ECO under the PAYG pricing model.

We see that MCC-PAYG outperforms FND and LND in these two situations. We also see that MCC-PAYG and FND outperform LND. Compared with LND, FND has cost saving from 26% to 50%. This is because, at B1 and B2, the maximum degree of S-nodes range from 5 to 8, and LND is suitable for the scenario where the degree of S-nodes is small.

3.8 A Case Study

We are developing a SAMS for a centralized air-conditioning system (Fig. 3.21 is an illustration of a centralized air-conditioning system with water tower, chillers to cool down the water, pumps to push water circulation, air handling unit (AHU) to use

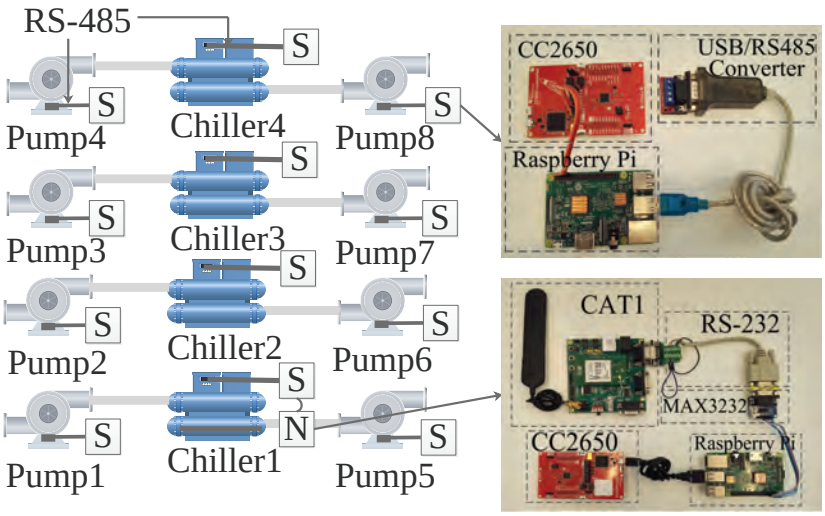


Figure 3.22: SAMS supported by the sTube+ architecture.

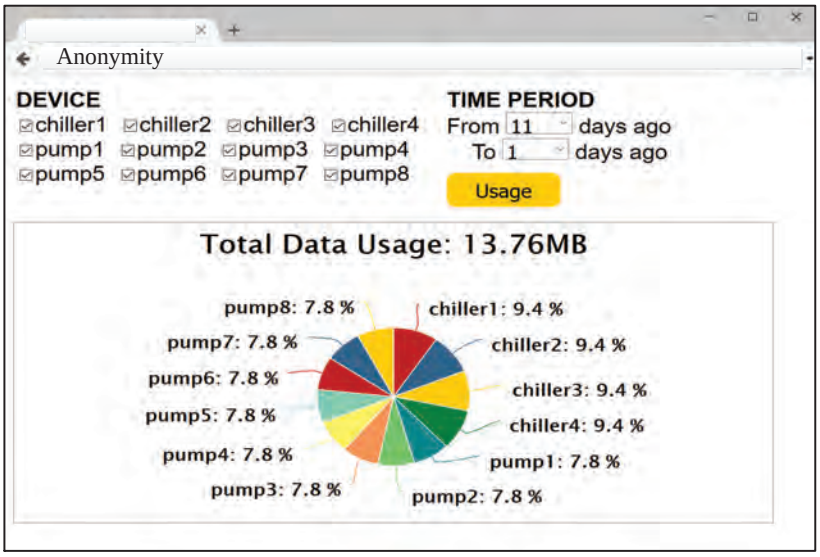


Figure 3.23: The data usage of the 4 chillers and 8 pumps.

Para.	Description
F_r	Condenser flow rate (m^3/h)
T_r	The returning chilled water temperature ($^\circ C$)
T_s	The supplying chilled water temperature ($^\circ C$)
W_c	Chiller power input (kWh)
Q	Heat transfer to circulating water (kJ)
W_p	Pump power input (kWh)

Table 3.4: The parameters for computing COP and WTC.

cold water to cool down the air, and fans to push air circulation. Finally, cold air will air-condition the offices and the temperature is controlled by the amount/speed of cold air allowed into an office).

We compute the performance of a chiller by Coefficient of Performance (COP, COP = $\frac{4.181 \times Fr \times (T_r - T_s)}{W_c}$) and the performance of a pump by Water Transfer Coefficient (WTC, WTC = $\frac{Q}{W_p}$) [57].

We develop the sensing module on Raspberry Pi to collect the raw data in Table 3.4. Chillers and pumps have standard APIs to output data from their embedded sensors. Using chiller as an example, a chiller controller uses a ModBus RTU protocol with an RS-485 interface. Modbus RTU protocol is a query-response protocol. We implement an application in Raspberry Pi using the standard library libmodbus to query the chiller through Modbus RTU protocol. The communication between USB port of Raspberry Pi and RS-485 need a USB/RS485 Converter module as the electrical level difference. Our hardware is shown in Fig. 3.22.

We deployed one N-node on a chiller and 12 S-nodes, four chillers and eight pumps (Fig. 3.22). All our nodes are powered by AC and we ran our system for 12 consecutive days. Our cloud monitored the data consumed by each S-node (Fig. 3.23). Our system can lead to a great cost reduction. In our case, We employ the PAYG-E pricing model provided by China TeleCom. The total data traffic of four chillers and eight pumps in 10 days was 13.76 MB. The monthly communication

cost of our system is $2 . If adopting the ECO method, the cost is $12 which is six times to our method. Note that, $2 is also the optimal cost we can get under the real pricing model.

3.9 Chapter Summary

One core value of the Internet-of-Things is the data of the things (i.e., IoT devices). Yet, transmitting the data to the cloud is still not pervasively achievable. The industry is actively developing various communication choices to support the diverse requirements of IoT data transmission. We demonstrated in this chapter, that the number of IoT communication choices may not easily catch up the requirements. We carefully analyzed example application scenarios. We proposed a solution of sTube+ on IoT communication sharing. The design of sTube+ includes a layered data delivery architecture, algorithms for cost optimization, and a prototype of a fully functioning system. We further develop a case study of chiller and pump maintenance, where sTube+ acts as the underlying architecture.

Chapter 4

DNN Surgery: Accelerating Inference on the Edge

4.1 Introduction

Recent advances in deep neural networks (DNN) have substantially improve the accuracy and speed of computer vision and video analytics, which creates new avenues for a new generation of smart applications. The maturity of cloud computing, equipped with powerful hardware such as TPU and GPU, becomes a typical choice for such kind computation intensive DNN tasks. For example, in a self-driving car application, cameras continuously monitor and stream surrounding scene to servers, which then conduct video analytic and feed back control signals to pedals and steering wheels. In an augmented reality application, a smart glass continuously records its current view and streams the information to the cloud servers, while the cloud servers perform object recognition and send back contextual augmentation labels, to be seamlessly displayed overlaying the actual scenery.

One obstacle to realizing smart applications is the large amount of data volume of video streaming. For example, Google's self-driving car can generate up to 750 megabytes of sensor data per second [84], but the average uplink rate of 4G, fastest existing solution, is only 5.85Mbps [7]. The data rate is substantially decreased when

the user is fast moving or the network is heavily loaded. In order to avoid the effect of network and put the computing at the proximity of data source, edge computing emerges. As a network-free approach, it provides anywhere and anytime available computing resources. For example, AWS DeepLens camera can run deep convolutional neural networks (CNNs) to analyze visual imagery [2]. Nevertheless, edge computer themselves are limited by their computing capacity and energy constraints, which cannot fully replace cloud computing.

From Fig. 4.1, we observe that, for the DNN, the amount of some intermediate results (the output of intermediate layers) are significantly smaller than that of raw input data. For example, the input data size of tiny YOLOv2 [73] is 0.95MB, while the output data size of intermediate layer max5 is 0.08MB with a reduction of 93%. This provides the opportunity for us to take the advantages of the powerful computation capacity of the cloud computing and the proximity of the edge computing. More specifically, we can compute a part of DNN on the edge side, transfer a small number of intermediate results to the cloud, and compute the left part on the cloud side. The partition of DNN constitutes a tradeoff between computation and transmission. As shown in Fig. 4.2, partition at different layers will cause different computation time and transmission time. So, an optimal partition is desirable.

Unfortunately, the decision on how to split the DNN layers heavily depends on the network conditions. In a LTE network, the throughput can decrease by 10.33 times during peak hours [70], and this value could reach 18.65 for a WiFi hotspot [34]. Under a high-throughput network condition, computing delay dominates and it is more desirable to offload the DNNs as early as possible. However, if the network condition degrades severely, we should prudently determine the DNN cut so to decrease the volume of data transmission. For example, Fig. 4.3 shows that when the network capacity is as high as 18Mbps, the optimal cut is at input layer and the overall processing delay is 0.59s. However, when the network capacity is lowered to

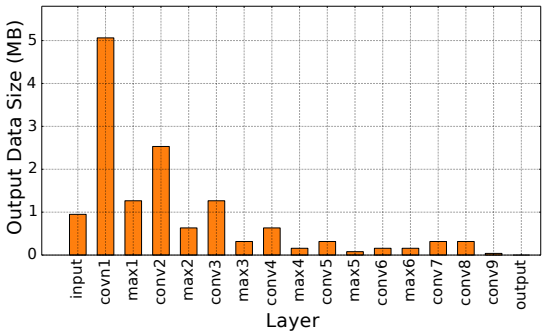


Figure 4.1: The output data size of each layer of YOLOv2.

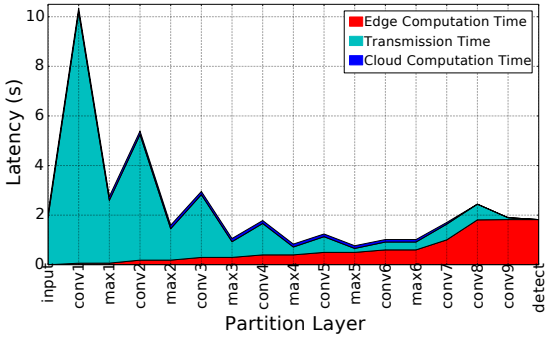


Figure 4.2: Latency constitution when partition at the different layers of tiny YOLOv2. Bandwidth is 4Mbps.

4Mbps, cutting at input layer is no longer valid as the communication delay increases substantially. Under this scenario, cutting at max5 is optimal, with a delay reduction of 62%.

Another challenge in the partition is that the recent advances of DNN show that DNNs are no longer limited to a chain topology, DAG topologies gain popularity. For example, GoogleNet [80] and ResNet [40], the champion of ImageNet Challenge 2014 and 2015 respectively, are DAGs. Obviously, partitioning DAG instead of chain involves much more complicated graph theoretic analysis, which may lead to NP-hardness in performance optimization.

To this end, in this chapter, we investigate the DNN partition problem, in order to find the optimal DNN partitioning in an integrated edge and cloud computing environment with dynamic network conditions. We design a Dynamic Adaptive DNN Surgery (DADS) scheme, which optimally partitions the DNN network by continually monitoring the network condition. The key design of DADS is as follows. DADS keeps monitoring the network condition and determines if the system is operated in the lightly loaded condition or heavily loaded condition. Under the lightly loaded condition, DNN Surgery Light (DSL) is developed, which minimizes the overall delay to process one frame. In this part, in order to solve the delay minimization problem,

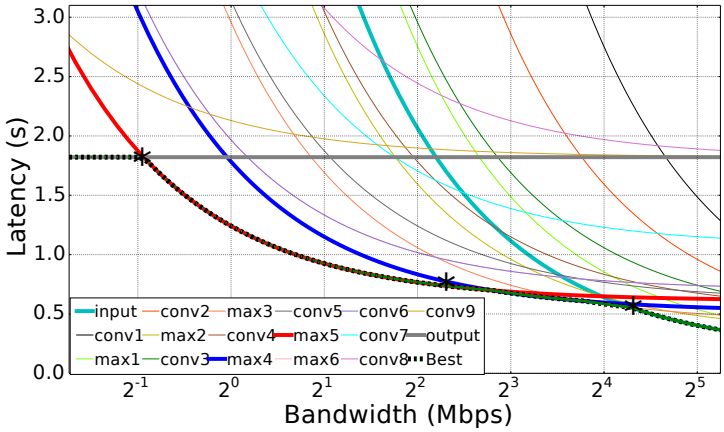


Figure 4.3: The latency of partition at different layers of YOLOv2 as a function of bandwidth.

we convert the original problem to an equivalent min-cut problem so that the globally optimal solution can be found. In the heavily loaded condition, DNN Surgery Heavy (DSH) is developed, which maximizes the throughput, i.e. the number of frames can be handled per unit time. However, we prove such optimization problem is NP-hard, which cannot be solved within polynomial computational complexity. DSH resorts an approximation approach, which achieves an approximation ratio of 3.

Finally, we develop a real-world testbed to validate our proposed DADS scheme. The testbed is based on the self-driving car video dataset and real traces of wireless network. We test 5 DNN models. We observe that compared with executing entire DNNs on the cloud and on the edge, DADS can reduce execution latency up to 6.45 times and 8.08 times respectively, and improve throughput up to 8.31 times and 14.01 times respectively.

4.2 An Edge-Cloud DNN Inference (ECDI) Model

4.2.1 Background

Video analytics is the core to realize a wide range of exciting applications ranging from surveillance and self-driving cars, to personal digital assistants and automatic

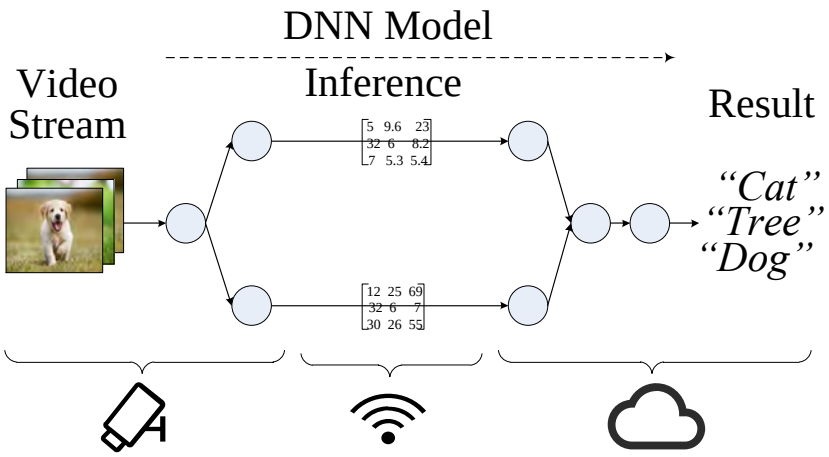


Figure 4.4: A 7-layer DNN model classifies frames of video.

drone controls. The current state-of-the-art approach is to use a deep neural network (DNN) where the video frames are processed by a well-trained constitutional neural network (CNN) or recurrent neural network (RNN). Video analytics use DNNs to extract features from input frames of the video and classify the objects in the frames into one of the predefined classes.

DNN network consists of quite a few layers which can be organized in a directed acyclic graph (DAG). Fig. 4.4 shows a 7-layer DNN model. Inference for video is performed with a DNN using a feed-forward algorithm that operates on each frame separately. The algorithm begins at the input layer and progressively moves forward layer by layer. Each layer receives the output of prior layers as the input, performs a series of computation on the input data to get the output, and feeds its output to the successor layers. This process terminates once the computation of output layer is finished.

The video is generated at the edge side and the frames of the video are fed into the DNN as input. The computation of each layer in DNN can be performed at the edge or at the cloud. Computing layers at edge devices does not require to transmit data to the cloud but incurs more computation due to resource-constrained device.

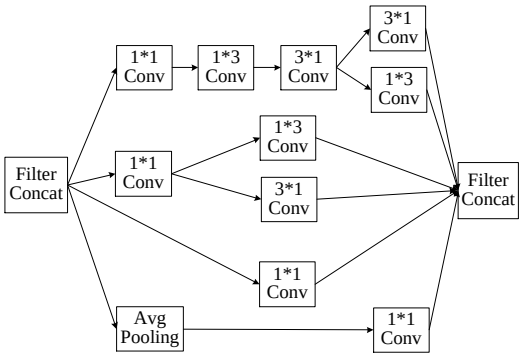


Figure 4.5: The inception v4 network represented in layer form.

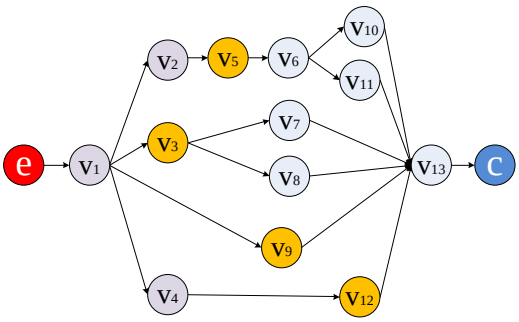


Figure 4.6: Graph representation of inception v4 network.

Computing layers at the cloud leads to less computation but incurs transmission latency for transmitting data from edge devices to the cloud.

4.2.2 The ECDI Model

In this subsection. We formally present the ECDI model.

1) Video Frame: A video consists of a sequence of frames (pictures) to be processed, with a sampling rate Q frames/second. Each sampled frame is fed to a predetermined DNN for inference. Please note that the sampling rate is not the frame rate of the video. It indicates how many frames/pictures are processed each unit time [97].

2) DNN as a Graph: A DNN is modeled as a directed acyclic graph (DAG). Each vertex represents one layer of the neural network. A layer is indivisible and must be processed on either the edge side or the cloud side. We add an virtual entry vertex and an exit vertex to represent the starting point and the ending point of DNN respectively. The links*represent communication and dependency among layers.

Let $\mathcal{G} = (\mathcal{V} \bigcup \{e, c\}, \mathcal{L})$ denote the DAG of DNN, where $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ is

*Please note that to avoid misunderstanding, throughout this work, we use the term "link" to represent "edge of a graph." This is because "edge" in this work has already represented "edge computing."

the set of vertices representing the layers of the DNN (specially, v_1 and v_n represent the input layer and output layer respectively). e and c denote virtual entry and exit vertices (to facilitate the subsequent analysis). $\mathcal{L}$ is the set of links. A link $(v_i, v_j) \in \mathcal{L}$ represents that v_i has to be processed before v_j, and v_i feeds its output to v_j. Fig. 4.6 shows the DAG of the pure inception v4 network [79] in Fig. 4.5.

Since each layer can be processed on either the edge or cloud side, its processing time depends on where it is processed (i.e. on the edge or on the cloud). Let t_i^e and t_i^c be the time needed to process v_i one edge and cloud respectively. Let d_i and t_i^t denote the output data size and the transmission time of v_i. We define $\mathbf{D}_t = \{d_1, d_2, \cdots, d_n\}$. Let B be the network bandwidth, we have $t_i^t = \frac{d_i}{B}$. Please note that B can be dynamically changed and we need to adapt such changes. We define $\mathbf{F}_e = \{t_1^e, t_2^e, \cdots, t_n^e\}$, $\mathbf{F}_c = \{t_1^c, t_2^c, \cdots, t_n^c\}$, $\mathbf{F}_t = \{t_1^t, t_2^t, \cdots, t_n^t\}$. They denote the *three key delays*: processing delay at the edge, transmission delay, and processing delay at the cloud of each layer.

3) DNN Partitioning: Our objective is to partition DNN into two parts so the one part is processed at the edge and the other is processed at the cloud. Mathematically, we should find a set of vertices $\mathcal{V}_S$ as a subset of $\mathcal{V}$ such that removing $\mathcal{V}_S$ causes that the rest of $\mathcal{G}$ becomes two disconnected components. One component contains e, denoted by $\mathcal{V}'_E$ and the other component contains c, denoted by $\mathcal{V}_C$. $\mathcal{V}_S$ is the cut so that all down-streaming layers are processed at the cloud. $\mathcal{V}'_E$ and $\mathcal{V}_S$ are processed at the edge and $\mathcal{V}_C$ are processed at the cloud. We define $\mathcal{V}_E = \mathcal{V}'_E \bigcup \mathcal{V}_S$. The output data of vertices in $\mathcal{V}_S$ will be transmitted from the edge side to the cloud. $\mathcal{V}_E$, including $\mathcal{V}'_E$ and $\mathcal{V}_S$ will generate processing delay at the edge. $\mathcal{V}_S$ will generate transmission delay. $\mathcal{V}_C$ will generate processing delay at the cloud. Our aim is to determine best cut $\mathcal{V}_S$ so that the overall delay is minimized.

As shown in Fig. 4.6, we cut at $\mathcal{V}_S = \{v_3, v_5, v_9, v_{12}\}$ so that the $\mathcal{V}'_E = \{e, v_1, v_2, v_4\}$, $\mathcal{V}_E = \{e, v_1, v_2, v_3, v_4, v_5, v_9, v_{12}\}$, and $\mathcal{V}_C = \{v_6, v_7, v_8, v_{10}, v_{11}, v_{13}, c\}$. The overall de-

lay is the processing delay of $\mathcal{V}_E$ on the edge and $\mathcal{V}_C$ on the cloud plus the communication delay of the output data of layer in $\mathcal{V}_S$.

4) Delay Components: Once the partition is made, each frame is processed at the edge, and then sent from the edge to the cloud, and then processed at the cloud. Since there are multiple frames to be processed, we assume that the three stages are conducted in *pipeline*. In order words, when frame 1 is being processed at the cloud, frame 2 can be transmitted and frame 3 can be processed at the edge.

The delays of the three stages are characterized as follows. In the edge-computing stage

$$T_e = \sum_{v_i \in \mathcal{V}_E} t_i^e. \tag{4.1}$$

In the cloud-computing stage

$$T_c = \sum_{v_i \in \mathcal{V}_C} t_i^c. \tag{4.2}$$

In the communication stage

$$T_t = \sum_{v_i \in \mathcal{V}_S} t_i^t. \tag{4.3}$$

For each frame, T_e, T_c, and T_t are spent for each stage. Frames are processed in pipeline every $\frac{1}{Q}$. As a consequence, the Gantt chart (scheduling chart) of frames can be shown in Fig. 4.8. T_e, T_c, and T_l cannot exceed $\frac{1}{Q}$. Otherwise, the incoming rate is greater than the completion rate, leading to system congestion. **Our aim is to smartly partition the DNN so that the overall delay to process frames is minimized and the system is not congested.**

4.2.3 Parameter Estimation for ECDI

In this subsection, we discuss how to derive the input parameters. The first class of parameters is called DNN profile, including DNN topology $\mathcal{G}$, processing delays

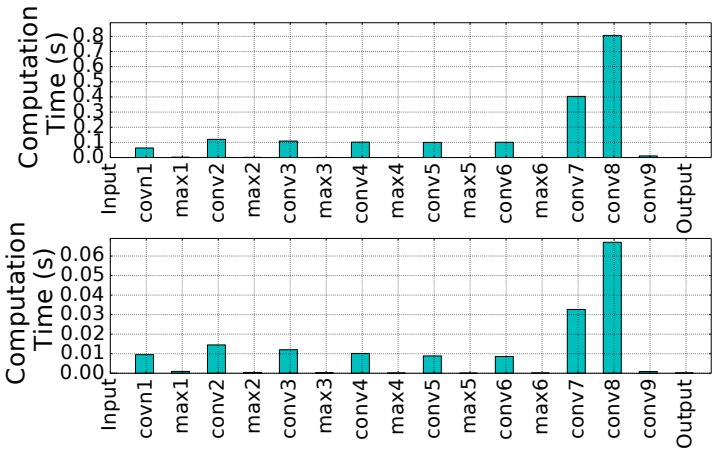


Figure 4.7: The computation latency of YOLOv2's layers on the edge (top) and cloud (bottom) respectively.

of each layer at the edge and the cloud $\mathbf{F}_e$, $\mathbf{F}_c$, data size of each layer $\mathbf{D}_t$. These parameters can be well derived in advance. $\mathcal{G}$ and $\mathbf{D}_t$ can be directly derived given the DNN definition. $\mathbf{F}_e$ and $\mathbf{F}_c$ can be measured beforehand. For example, we derive $\mathbf{D}_t$ of tiny YOLOv2 model and measure $\mathbf{F}_e$ of tiny YOLOv2 model processed on Raspberry Pi 3 model B and Ali Cloud respectively. We show the results in Fig. 4.1 and Fig. 4.7 respectively.

The value B is dynamic and should be measured during the process of DNN inference. This can be realized by a method similar to HTTP DASH [78]. We use the tool "ping" at edge to send two different size data consecutively to the cloud, and measure the response times. The bandwidth equals to the ratio between the difference of data size and the difference of response times.

The value Q is user-specific. The user lets the system know Q when the inference starts. The system does nothing unless Q is too large for the system to handle (See Section 4.3.4).

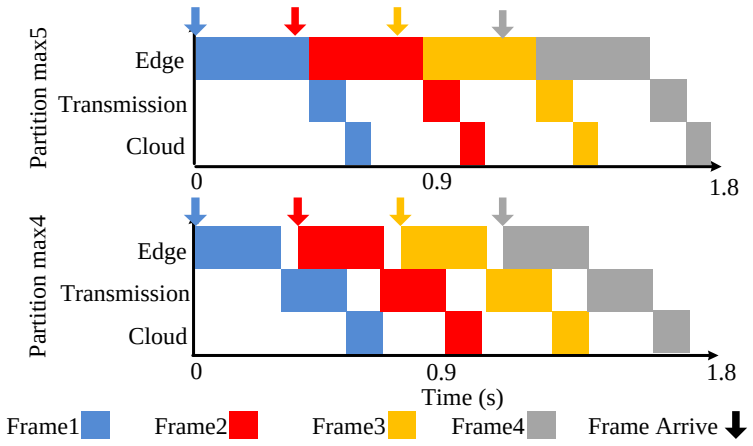


Figure 4.8: Gantt charts for three stages.

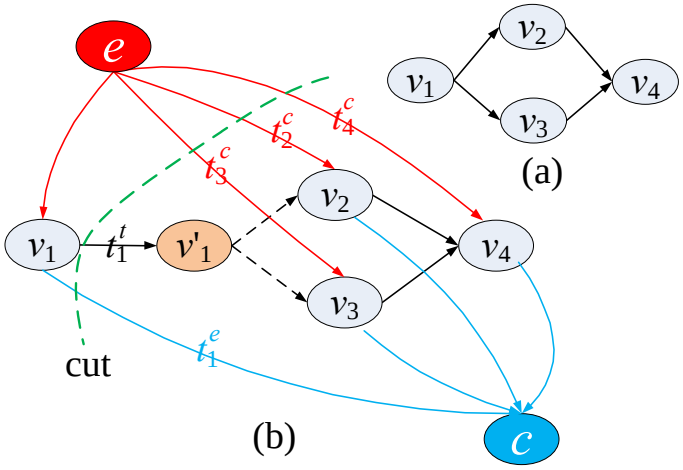


Figure 4.9: Illustration of conversion to the minimum s-t cut problem.

4.3 ECDI Partitioning Optimization

4.3.1 The Impact of DNN Inference Workloads

Our first objective is to minimize the overall delay to process each frame. This is true under the *light workload*: for each stage, the current frame is completed before the next frame arrives. Mathematically $\max\{T_e, T_t, T_c\} < \frac{1}{Q}$ so that the Gantt chart is shown as the bottom one of Fig. 4.8. In this case, we just need to complete every frame *as soon as possible*, i.e., minimize $T_c + T_t + T_e$.

However, if the system is heavily loaded, minimizing $T_e + T_t + T_c$ may lead to system congestion as $\max\{T_e, T_t, T_c\} \geqslant \frac{1}{Q}$. For example, in Fig. 4.8 (top), $T_e > \frac{1}{Q}$ so that the next frame arrives before the current frame is completed at the edge. Therefore, under this situation, we need to maximize the throughput of the system, i.e. how many frames at most the system can handle per unit time. Our objective is to minimize $\max\{T_e, T_t, T_c\}$ as the system throughput is $\frac{1}{\max\{T_e, T_t, T_c\}}$. For presentation convenience, $\max\{T_e, T_t, T_c\}$ is referred to as the **max stage time**.

Please note that in Section 4.3.4, we will further discuss how to judge if the system is lightly loaded or heavily loaded. There, we also need to consider that if the sampling rate is greater than $\frac{1}{\min\max\{T_e, T_t, T_c\}}$ so that the system will be congested eventually. The system has to force the sender/user to reduce sampling rate.

4.3.2 The Light Workload Partitioning Algorithm

In this subsection, we study **Edge Cloud DNN Inference for Light Workload (ECDI-L)** problem. Our goal is to minimize the overall delay of one frame, under a given the network condition B. In summary, we have the following optimization problem:

Problem 4.1. *(ECDI-L) Given $\mathcal{G}$, $[\mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t]$, and B, determine $\mathcal{V}_E$, $\mathcal{V}_S$ and $\mathcal{V}_C$, to minimize $T_{inf} = T_e + T_t + T_c$.*

Proposition 4.1. *Problem ECDI-L can be solved in polynomial time.*

One challenge to solve ECDI-L problem directly is that each vertex in $\mathcal{G}$ contains three delay values $t_i^e, t_i^c, t_i^t = \frac{d_i}{B}$. The delay that contributes to the overall delay depends on where the vertex is processed. To this end, we construct a new graph $\mathcal{G}'$ so that each edge only captures *a single* delay value. By doing so, we convert ECDI-L problem to the minimum weighted s-t cut problem of $\mathcal{G}'$.

We first illustrate how to construct $\mathcal{G}'$ based on $\mathcal{G}$.

Cloud Computing Delay Based on $\mathcal{G}$, we add links between e and each vertex $v \in \mathcal{V}$, referred to as "red links," to capture the cloud-computing delay of v.

Edge Computing Delay Similarly, we add links between vertex $v \in \mathcal{V}$ and c, referred to as "blue links," to capture the edge-computing delay of v.

Communication Delay All the other links correspond to communication delays. A link from v to u should capture the communication delay of v. However, this is insufficient as one vertex may have multiple successors and its communication delay is counted multiple times. For example, v_1 in Fig. 4.6 has 4 outgoing links but the communication delay of v_1 has to be counted at most once. To this end, we introduce *axillary vertices* into graph $\mathcal{G}'$. That is, for any vertex $v_k \in \mathcal{V}$ whose outdegree is greater than one, we add an auxiliary vertex v'_k and link (v_k, v'_k). The links from v_k to successors of v_k are now re-placed from v'_k to successors of v_k. For example, a 4-layer DNN is shown in Fig. 4.9(a). The outdegree of vertex v_1 is greater than one, we thus add an auxiliary vertex v'_1 and link (v_1, v'_1) shown in Fig. 4.9(b). The links (v_1, v_2) and (v_1, v_3) are re-placed by links (v'_1, v_2) and (v'_1, v_3) respectively. We define $\mathcal{V}_D$ to be the set of axillary vertices.

Now, without considering e and c, if a vertex v has one successor, the link starting from v corresponds to its communication delay, which is referred to as "black link." If v has multiple successors, then all the links starting from v are referred to as "dashed links" and should not be considered since the communication delay has already been considered from v to v'.

Links are assigned costs. The costs assigned to red, blue, black links are cloud-computing, edge-computing, and communication delays. Dashed links are assigned

infinity.

$$c(v_i, v_j) = \begin{cases} t_i^e, & \text{if } v_i \in \mathcal{V}, v_j = c. \\ t_i^t, & \text{if } v_i \in \mathcal{V}, v_j \in \mathcal{V} \bigcup \mathcal{V}_D. \\ t_i^c, & \text{if } v_i = e, v_j \in \mathcal{V}. \\ +\infty, & \text{others.} \end{cases} \tag{4.4}$$

At this stage, we can convert ECDI-L problem to the minimum weighted s–t cut problem of $\mathcal{G}'$.

A cut is a partition of the vertices of a DAG into two disjoint subsets. The s–t cut of $\mathcal{G}'$ is a cut that requires source s and sink t to be in different subsets, and its *cut-set only consists of links going from the source's side to sink's side.* The value of a cut is defined as the sum of the cost of each link in the cut-set. Problem ECDI-L is equivalent to the minimum e–c cut of $\mathcal{G}'$. If cutting on link from e to $v_i \in \mathcal{V}$ (red link shown in Fig. 4.9(b)), then v_i will be processed on the cloud, i.e $v_i \in \mathcal{V}_C$. If cutting on link from $v_j \in \mathcal{V}$ to c (blue link show in Fig. 4.9(b)), then v_j will be processed on the edge, i.e. $v_j \in \mathcal{V}_E$. If cutting on link from $v_i \in \mathcal{V}$ to $v_j \in \mathcal{V} \bigcup \mathcal{V}_D$ (black link show in Fig. 4.9(b)), then the data of v_i will be transmitted to the cloud, i.e $v_i \in \mathcal{V}_S$. It is impossible to cut on link from $v_i \in \mathcal{V}_D$ to $v_j \in \mathcal{V}$ (dashed links), because otherwise it will lead to infinite cost (but finite cost exists). The total cost of cut on red links equals to cloud computation time T_c. The total cost of cut on blue links equals to edge computation time T_e. The total cost of cut on black links equals to transmission time without network latency T_t. If the e–c cut of $\mathcal{G}'$ is minimum, then the inference latency on a single frame is minimum. For example, in Fig 4.9(b), the cut is at (e, v_2), (e, v_3), (e, v_4), (v_1, v_1') and (v_1, c). v_1 is processed at the edge so that t_1^e is counted in the blue link. v_2, v_3 and v_4 are processed at the cloud so that t_2^c t_3^c and t_4^c are counted in the red links. The communication delay t_1^t is counted in the black link.

We develop DNN Surgery Light (denoted as DSL) algorithm for ECDI-L prob-

lem. The overall algorithm `DSL()` is shown in Algorithm 4.1. The algorithm first calls `compute-net()` to compute $\mathbf{F}_t$. Then it calls `graph-construct()` (line 2) to construct $\mathcal{G}'$ based on $\mathcal{G}$ with the computation complexity of $\mathcal{O}(n+m)$, where n is the number of layers $|\mathcal{V}|$, m is the number of links $|\mathcal{L}|$, and then it calls `min-cut()` (line 3) to find minimum e–c cut of $\mathcal{G}'$ which outputs the partition strategy (i.e. $\mathcal{V}_E, \mathcal{V}_S$ and $\mathcal{V}_C$). Boykov's algorithm [18] is used in `min-cut()` to solve the minimum e–c cut problem with the computational complexity of $\mathcal{O}((m+n)n^2)$. `DSL()` is a polynomial-time algorithm with the computational complexity of $\mathcal{O}((m+n)n^2)$.

Algorithm 4.1: DSL Algorithm `DSL()`.

Input: $\mathcal{G}, \mathbf{F}_e, \mathbf{F}_c\mathbf{D}_t, B$
Output: $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c$
1 $\mathbf{F}_t \leftarrow$`compute-net(`$\mathbf{D}_t, B$`);`
2 $\mathcal{G}' \leftarrow$`graph-construct(`$\mathcal{G}, \mathbf{F}_e, \mathbf{F}_c, \mathbf{F}_t$`);`
3 $[\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c] \leftarrow$ `min-cut(`$\mathcal{G}'$`);`
4 return $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c$;

4.3.3 The Heavy Workload Partitioning Algorithms

As discussed in Section 4.3.1, we formulate the **Edge Cloud DNN Inference for Heavy Workload (ECDI-H)** problem, to minimize $\max\{T_e, T_t, T_c\}$. The decision variables are $\mathcal{V}_E$, $\mathcal{V}_S$ and $\mathcal{V}_C$. In summary, we have the following optimization problem:

Problem 4.2. *(ECDI-H) Given* $\mathcal{G}$, $[\mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t]$, *and* B, *determine* $\mathcal{V}_E$, $\mathcal{V}_S$ *and* $\mathcal{V}_C$, *to minimize* $\max\{T_e, T_t, T_c\}$ *(i.e. maximize throughput).*

ECDI-H Problem is NP-hard. We provide the sketch of the proof. We prove it by reducing from the smallest component of the most balanced minimum st-vertex cut problem (MBMVC-SC), which is known to be NP-complete [17]. We consider the following MBMVC-SC problem on $\mathcal{G}_A = (\mathcal{V}_A \bigcup \{s,t\}, \mathcal{L}_A)$, the goal is to find a

vertex cut set $\mathcal{V}_C$ to partition the graph into two disjoint components $(\mathcal{V}_1, \mathcal{V}_2)$, for which $s \in \mathcal{V}_1$, $t \in \mathcal{V}_2$ and the largest components among $\{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_C\}$ is minimum. Any instance of the above problem is equivalent to an instance in ECDI-H problem. Due to page limitation, detailed explanations are omitted.

ECDI-H is NP-hard. It is unrealistic to find a globally optimal solution within polynomial time. We design DNN Surgery Heavy (denoted as DSH) algorithm which achieves a locally optimal solution. In addition, its approximation ratio is 3.

The rationale to develop DSH is as follows. We modify $\mathcal{G}'$ by changing the costs of links as follow:

$$c(v_i, v_j) = \begin{cases} \alpha t_i^e, & \text{if} \quad v_i \in \mathcal{V}, v_j = c. \\ \beta t_i^t, & \text{if} \quad v_i \in \mathcal{V}, v_j \in \mathcal{V} \bigcup \mathcal{V}_D. \\ \gamma t_i^c, & \text{if} \quad v_i = e, v_j \in \mathcal{V}. \\ +\infty, & \text{others.} \end{cases} \tag{4.5}$$

Here α, β and γ are non-negative variables. The approach is to run `DSL()` with several different α, β and γ values. By this way, a solution is generated to optimize ECDI-L with a specific α, β, γ tuple. Then we test if this solution is also good enough for ECDI-H. If it is better than all existing solutions, it is regarded as a new solution to ECDI-H. We repeat the above procedure for a wide range of α, β, γ tuples.

Here, the result of `DSL()` is determined by the ratio of the three parameters, instead of their absolute values. Therefore, we can fix one of the three, for example, $\beta = 1$, and only vary the other two. Thus, we have a two-dimensional search space for α and γ. We first search in the two-dimensional plane with a coarse granularity to find the best solution. Then we use a finer granularity search in the neighborhood of the best solution for further improvement. We repeat the steps until the improved performance is smaller than a threshold ϵ.

The overall algorithm `DSH()` is shown in Algorithm 4.2. A function `search()` (line 11–19) is designed to search for the best solution in a given space $\mathbf{S} \triangleq [\alpha_l, \gamma_l, \alpha_h, \gamma_h]$,

meaning that $\alpha_l \leqslant \alpha \leqslant \alpha_h, \gamma_l \leqslant \gamma \leqslant \gamma_h$, and a granularity δ (line 13–14), i.e. the step size of changing α and γ is δ each time. For each α and γ, `search()` calls `DSL()` to compute the vertex cut and calls `max-time()` to compute the $\max[T_c, T_e, T_t]$. Lines 17–18 guarantee $\max[T_c, T_e, T_t]$ derived is non-increasing.

The overall algorithm first initializes the search granularity δ to be 1 (line 2) and the search space large enough (line 3–4). It calls `search()` (line 8) to search on the given space $\mathbf{S}$ with a granularity δ, and returns the best α and γ found currently. Then `DSH()` narrows down the search space $\mathbf{S}$ (line 8) to the neighborhood of the best α and γ for the current iteration, and adjusts δ to a finer granularity (line 9). Such space $\mathbf{S}$ and granularity δ is returned to `search()`. The termination condition for the loop is that the improved performance is smaller than a threshold ϵ (line 5). Finally, it returns the vertex cut with the best-found performance (line 10). Obviously, we can achieve a local optimal result with respect to the neighborhood of the final α and γ.

Theorem 4.2. *The approximation ratio of the algorithm DSH for ECDI-H is* 3.

Proof. Let the max stage time of DHL be t_{DSH}. Let the optimal max stage time of ECDI-H be t^*. We prove $\frac{t_{DSH}}{t^*} \leqslant 3$. Let T^* denote the minimum inference latency for one frame. Let T_o denote the inference latency of a single frame when achieving the optimal max stage time. We have $T^* \leqslant T_o$. Because there are three stages, we have $T_o \leqslant 3t^*$, thus $T^* \leqslant 3t_o$.

As shown in Algorithm 4.2, when $\delta = 1$, `Search()` will calls `DSL()` using $\alpha = 1$ and $\gamma = 1$ as the parameter. When $\alpha = 1$ and $\gamma = 1$, `DSL()` achieves the minimum inference time T^* for one frame. Let t_1, t_2 and t_3 be the edge computation time, the transmission time and the cloud computation time respectively when achieving the minimum inference time. We have $T^* = t_1 + t_2 + t_3$. `DSH()` guarantees the searched max stage time is non-increasing, thus $t_m \leqslant \max\{t_1, t_2, t_3\}$, combined with

Algorithm 4.2: DSH Algorithm `DSH()`.

Input: $\mathcal{G}, \mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t, B, \epsilon, K$
Output: $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_{max}$

1 $\mathbf{F}_t \leftarrow$`compute-net(`$\mathbf{D}_t, B$`)`;
2 $T_{max} \leftarrow +\infty$; $T'_{max} \leftarrow 0$; $\delta \leftarrow 1$;
3 $\alpha_l \leftarrow 0$; $\gamma_l \leftarrow 0$; $\alpha_u \leftarrow \frac{\sum(\mathbf{F}_e)}{\min(\mathbf{F}_t)}$; $\gamma_u \leftarrow \frac{\sum(\mathbf{F}_c)}{\min(\mathbf{F}_t)}$;
4 $\mathbf{S} \leftarrow [\alpha_l, \gamma_l, \alpha_u, \gamma_u]$;
5 **while** $|T'_{max} - T_{max}| \geqslant \epsilon$ **do**
6 $\quad T'_{max} \leftarrow T_{max}$;
7 $\quad [\alpha, \gamma, \mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_{max}] \leftarrow$`Search(`$\mathbf{S}, \delta, T_{max}$`)`;
8 $\quad \alpha_l \leftarrow \alpha - \delta$; $\alpha_u \leftarrow \alpha + \delta$;$\gamma_l \leftarrow \gamma - \delta$; $\gamma_u \leftarrow \gamma + \delta$;
9 $\quad \delta \leftarrow \delta / K$;
10 return $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_{max}$;
11 **function** *`Search`*$([\alpha_l, \gamma_l, \alpha_u, \gamma_u], \delta, T^*_{max})$
12 $\quad T_{max} \leftarrow +\infty$;
13 $\quad$ **for** $\alpha \leftarrow \alpha_l; \alpha \leqslant \alpha_u; \alpha \leftarrow \alpha + \delta$ **do**
14 $\quad\quad$ **for** $\gamma \leftarrow \gamma_l; \gamma \leqslant \gamma_u; \gamma \leftarrow \gamma + \delta$ **do**
15 $\quad\quad\quad [\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c] \leftarrow$`DSL(`$\mathcal{G}, \alpha\mathbf{F}_e, \gamma\mathbf{F}_c, \mathbf{D}_t, B$`)`
16 $\quad\quad\quad T_{max} \leftarrow$`max-time(`$T_e, T_t, T_c$`)`;
17 $\quad\quad\quad$ **if** $T_{max} \leqslant T^*_{max}$ **then**
18 $\quad\quad\quad\quad \alpha^* \leftarrow \alpha$; $\gamma^* \leftarrow \gamma$; $T^*_{max} \leftarrow T_{max}$;
19 $\quad$ return $\alpha^*, \gamma^*, \mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T^*_{max}$;

$T^* = t_1 + t_2 + t_3$, we have $t_m \leqslant T_{min}$. As $t_m \leqslant T^*$ and $T^* \leqslant 3t^*$, we prove $\frac{t_{DSH}}{t^*} \leqslant 3$. □

4.3.4 The Dynamic Partitioning Algorithm

We now consider network dynamics. In practice, the network status B varies. This will affect the workload mode selection and the partition decision dynamically. We design Dynamic Adaptive DNN Surgery (DADS) scheme to adapt network dynamics.

It is shown in Algorithm 4.3. `monitor-task()` monitors whether the video is active (line 2). This can be realized by tool "iperf." Detailed implementation can be

found in Section 4.4. The real-time network bandwidth is derived by `monitor-net()` (line 3). Then `DSL()` is called to compute the partition strategy (line 4). In this case, if it satisfies the sampling rate $\frac{1}{Q}$, i.e.$\max\{T_e, T_t, T_c) < \frac{1}{Q}$, we can confirm that the system is in the light workload mode and the partition by DSL is accepted.

Otherwise, the system is in the heavy workload mode and calls `DSH()` to adjust the partition strategy to minimize the max delay (line 6). However, if the completing rate is still smaller than the sampling rate, it means that the sampling rate if too large so that even `DSH()` still cannot satisfy the sampling rate. The system will be congested. It calls the user to decreases the sampling rate (line 7–8).

Algorithm 4.3: DADS Algorithm `DADS()`

```
1 while true do
2     if monitor-task()==true then
3         B ← monitor-net();
4         [V_E, V_S, V_C, T_e, T_t, T_c] ← DSL(G, F_e, F_c, D_t, B);
5         if max{T_e, T_t, T_c} > 1/Q then
6             [V_E, V_S, V_C, T_max] ← DSH(G, F_e, F_c, D_t, B, ϵ, K);
7             if T_max > 1/Q then
8                 inform-decrease();
9         execute(V_E, V_S, V_C);
```

4.4 Implementation

We implement a DADS prototype system. We use the Raspberry Pi 3 model B as the edge device, integrated with a Logitech BRIO camera. We rent a server in Cloud Ali with 8 cores of 2.5 GHz and a total memory of 128 GB. We employ WiFi as the communication link between the edge device and the cloud. The wired link from the edge router and the cloud is sufficiently large. We implement our client-server interface using GRPC, an open source flexible remote procedure call (RPC) interface for inter-process communication.

The edge device. The duty of the edge device is to 1) extract video from the camera and to sample frames from video, 2) make partition decision, 3) process the layers allocated to the edge device, and 4) inform the cloud the partition decision and transfer the intermediate results to the cloud.

For video extraction, we extract videos from camera logitech BRIO using the provided API `video_capture()`. The camera transfers the captured video to Raspberry Pi through the USB-to-serial cable.

For partition decision making, we implement a process that monitors the generated frame by the camera, and runs DADS scheme. DADS requires to estimate the real-time network bandwidth. We use the command "iperf" provided by the operation system Raspbian on Raspberry Pi. This command feeds back the real-time network bandwidth between the Raspberry Pi and the cloud.

For processing allocated layers on the edge, we install a modified instance of Caffe and store a full DNN model on the edge device. The challenge is to control Caffe to stop execution at partitioned layers (e.g., $\mathcal{V}_S$). In Caffe, there is a "prototxt" file recording the DNN structure. Layers are processed according to this file. To solve the challenge, we modify the model structure file "prototxt" by inserting a "stop layer" after each partitioned layer. The instance of Caffe will stop processing at the desired places.

For the intermediate results and partition decision transmission, the edge device calls the RPC function `receiveRPC()` provided by the cloud to transmit the data to the cloud.

The cloud. The duty of the cloud is to execute the DNN layers allocated to the cloud. There are two jobs: 1) to receive the partition decision and the intermediate results from the edge device, and 2) to execute the layers allocated to the cloud.

For the first job, we expose an API `receiveRPC()` to the edge device. After completing processing layers allocated to the edge, the edge device calls this RPC

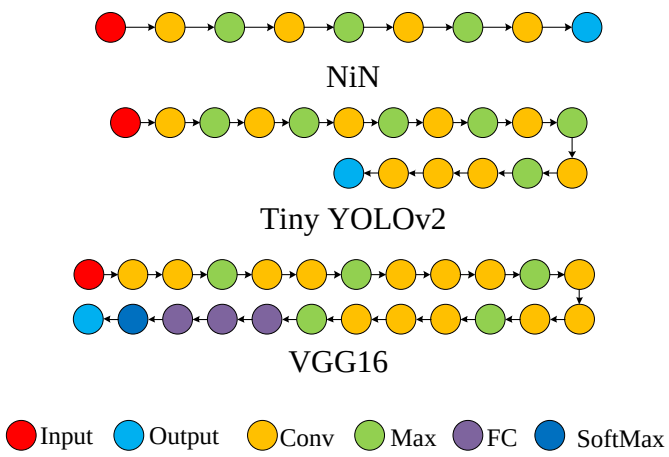


Figure 4.10: The chain-topology DNN models.

function to transmit the intermediate results packed with the partition decision to the cloud.

For the second job, we implement a modified instance of Caffe and store a full DNN model. The challenge is to execute only the layers allocated to the cloud. To this end, after receiving the partition decision and intermediate results, the layers allocated to the edge are deleted before the marked place in "prototxt," and the intermediate results are forwarded to the corresponding layers as input. By this way, only layers allocated to the cloud will be executed.

4.5 Performance Evaluation

We evaluate the DADS prototype (Section 4.4) using real-trace driven simulations.

4.5.1 Setup

Video Datasets. We employ the publicly available BDD100K self-driving dataset. The videos of this dataset are obtained from the camera on the self-driving car. Each video is about 40 seconds long and is viewed in 720p at 30 FPS.

Workload Setting. We divide the inference task into low workload mode and

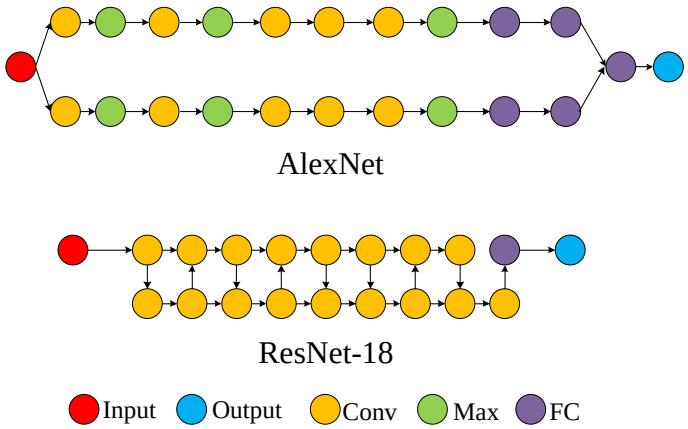


Figure 4.11: The DAG-topology DNN models.

Table 4.1: DNN Benchmark Specifications

	CAT1	3G	4G	WiFi
Uplink rate (Mpbs)	0.13	1.1	5.85	18.88

heavy workload mode. Accordingly, We transform the video into different sampling rates to produce different workload. We set a low sampling rate to 0.1 frame per second when evaluating light workload mode, and 20 frames per second for heavy workload mode. The default resolution is 224p. Each inference task consists of processing 100 frames using the given DNN benchmarks.

Communication Network Parameters. To model the communication between edge and cloud, we used the average uplink rate of mobile Internet for different wireless networks, i.e. CAT1, 3G, 4G and WiFi as shown in Table 4.1.

DNN Benchmarks. DADS can make partition not only on chain topology DNN but also on the DAG topology. We evaluate the performance of DADS for

Table 4.2: DNN Benchmark Specifications

Type	Chain			DAG	
Model	NiN	YOLOv2	VGG16	Alexnet	ResNet18
Layers	9	17	24	23	20

both topologies. For the chain topology, NiN, tiny YOlOv2 and VGG16, are well-known models used as benchmarks in this evaluation shown in Fig. 4.10. For the DAG topology, we employ AlexNet and ResNet-18 as the benchmarks shown in Fig. 4.11.

Evaluation Criteria: We compare DADS against Edge-Only (i.e. executing the entire DNN on the edge), Cloud-Only (i.e. executing the entire DNN on the cloud), and a variant Neurosurgeon which is a partition strategy for chain-topology DNN. To evaluation Neurosurgeon's performance for DAG, we consider a variant Neurosurgeon, which first employs topological sorting method to transform the DAG topology to the chain topology, and then uses the original partition method. We use the Edge-Only method as the baseline, i.e. the performance is normalized to Edge-Only method.

We evaluate the latency and throughput of DADS compared with Edge-Only, Cloud-Only and Neurosurgeon in Section 4.5.2. We also evaluate the impact of different types of wireless network to DADS, and the impact of bandwidth on the selection of workload mode in Section 4.5.3.

4.5.2 Performance Comparison

We first compare our DADS with Edge-Only, Cloud-Only and Neurosurgeon under light workload mode and heavy workload mode across the 5 DNN benchmarks in Fig. 4.12–4.14. The results are normalized to Edge-Only method. We see that DADS achieves a higher latency speedup and throughput gain compared with other methods.

Comparing DADS with Edge-Only and Cloud-Only: DADS has a latency speedup of 1.91–6.45 times, 1.35–8.08 times compared with Edge-Only and Cloud-Only methods respectively under the light workload mode shown in the bottom graph of Fig. 4.12. DADS has a throughput gain of 3.45–8.31 times, 1.46–11.13 times compared with Edge-Only and Cloud-Only methods respectively under the light

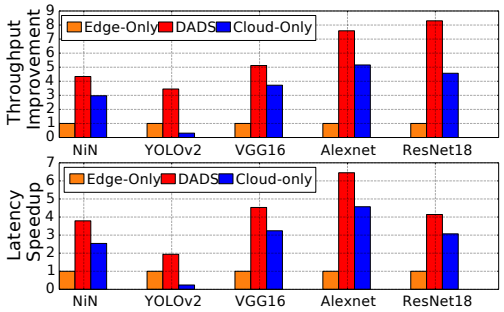


Figure 4.12: Latency speedup and throughput gain achieved by DADS under light workload mode.

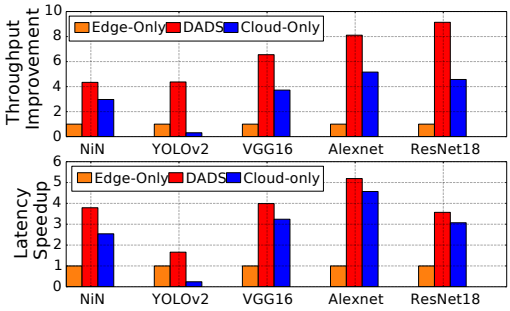


Figure 4.13: Latency speedup and throughput gain achieved by DADS under heavy workload mode.

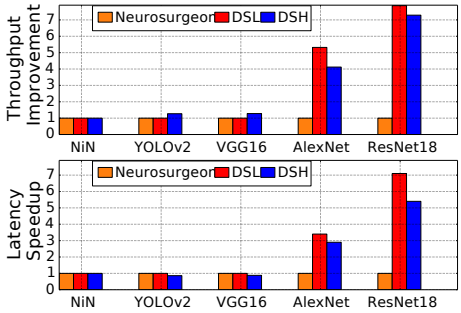


Figure 4.14: Latency and throughput speedup achieved by DADS vs. Neurosurgeon under light and heavy workload modes.

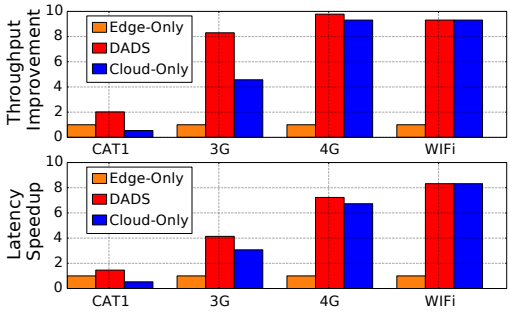


Figure 4.15: Latency speedup and throughput gain achieved by DADS of different networks under light workload mode.

workload mode shown in the upper graph of Fig. 4.12. This is because, Edge-Only method executes the entire DNN on the edge side, it avoids data transmission and ignores the weak computation capacity of edge side. Cloud-Only method ignores the effect of the transmission time. DADS considers both computation and transmission, and it makes a good tradeoff between them.

From Fig. 4.16, we can see that, for the heavy workload mode, DADS outperforms Edge-Only and Cloud-Only 1.66–5.19 times and 1.07–6.92 times respectively in latency reduction, and DADS outperforms Edge-Only and Cloud-Only 4.34–9.14 times and 1.46–14.10 times respectively in throughput gain. This further confirms that DADS significantly outperforms Edge-Only and Cloud-Only methods.

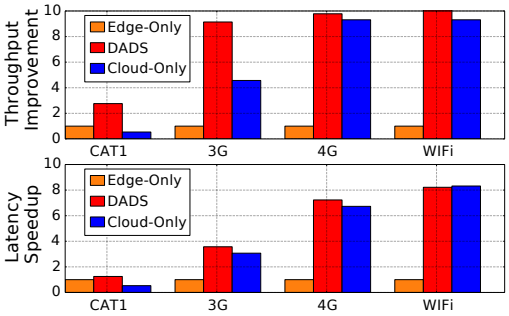


Figure 4.16: Latency speedup and throughput gain achieved by DADS of different networks under heavy workload mode.

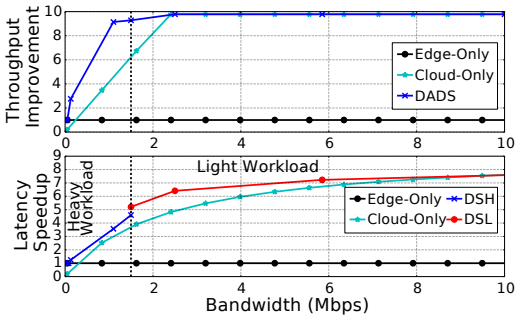


Figure 4.17: Latency speedup and throughput gain achieved by DADS as a function of bandwidth.

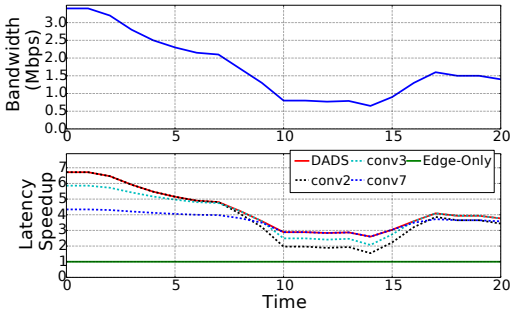


Figure 4.18: The impact of network variance on DADS partition decision using Edge-Only as the baseline.

Comparing DADS with Neurosurgeon: Neurosurgeon can automatically partition DNN between the edge device and cloud at granularity of neural network layers, but it is only effective for chain topology.

From Fig. 4.14, we can see that, for the chain topology models, DADS and Neurosurgeon have the similar performance in latency and throughput for the light workload. While for the heavy workload, Neurosurgeon has a latency reduction of 16.28% and 13.64% than that of DADS for YOLOv2 and VGG16, however the throughput gain of DADS is 1.26 times and 1.27 times than that of Neurosurgeon under these two DNN models. This is because, for the heavy workload, the higher throughput is prior for DADS. We also can see that, for the heavy workload and

NiN model, the latency and the throughput of Neurosurgeon and DADS are both the same. This is because for NiN model, DADS achieves the minimum max stage time when the latency is minimum.

For the DAG topology, we can observe that DADS outperforms Neurosurgeon significantly. For DAG topology models, DADS has a latency speedup 66%–86% and throughput gain of 76%–87% compared with Neurosurgeon. This observation validates the usefulness of DADS for DAG topology.

4.5.3 Network Variation

In this section, we evaluate how transmission network affects the performance of DADS using ResNet18 model. The sampling rate is 1 frame per second.

The Impact of Transmission Network Type: We first evaluate the performance of DADS, Edge-Only and Cloud-Only for ResNet18 model when using Cat1, 3G, 4G and WiFi as the communication network.

In Figs. 4.15–4.16, we show the latency speedup and the throughput gain achieved by DADS and Cloud-Only normalized to Edge-Only when using Cat1, 3G, 4G and WiFi for light and heavy workload respectively.

Shown in Fig. 4.15, when the workload is light and the edge device communicates with the cloud through Cat1, DADS achieves 1.46 times latency reduction and 2.03 times throughput gain compared with Edge-Only. When the network changes to 3G, 4G and 5G the latency reduction and the throughput gain becomes more significant: 4.14 times and 8.3 times for 3G, 7.23 times and 9.78 times for 4G, 8.32 times and 9.31 times for WiFi respectively. When the communication link provides more bandwidth, DADS pushes larger portions of layers to the cloud to achieve better performance. We can also see that, compared with Cloud-Only, DADS achieves latency reduction of 64% for CAT1, 26% for 3G and 7% for 4G receptively, and throughput gain of 73% for CAT1, 45% for 3G and 4% for 4G. For WiFi, the performance of Cloud-Only

is good enough, it has the same performance with DADS.

Edge-Only is only good for low data rate. Cloud-Only is only good for high data rate, DADS can be adaptive to a wide range of network setting.

The Impact of Bandwidth on Workload Mode Selection: In Fig. 4.17, we show the workload mode switch of DADS under different network bandwidth. We can see that when the available bandwidth is smaller than 1.51Mbps, DADS works at heavy workload mode, and the achieved latency speedup and throughput gain increase compared with Edge-Only. When the bandwidth is greater than 1.51Mbps, DADS works at light workload mode.

We also evaluate DADS's resilience to real-world measured wireless network variations. In Fig. 4.18, the top graph shows measured wireless bandwidth over a period of time. The bottom graph shows the latency speedup of DADS normalized to Edge-Only for ResNet18 model. We can see that DADS adjusts the partition strategy according to the bandwidth variance successfully. For example, when the bandwidth drops from 3.41Mbps to 2.15Mbps, DADS changes the partition from conv2 layer to conv3 layer. DADS changes the partition from conv3 layer to conv7 layer when bandwidth is smaller than 1.72Mbps.

4.6 Chapter Summary

In this chapter, we study DNN inference acceleration through collaborative edge-cloud computation. We propose Dynamic Adaptive DNN surgery (DADS) scheme that can partition DNN inference between the edge device and the cloud at the granularity of neural network layers, according to the dynamic network status. We present a comprehensive study of the partition problem under the lightly loaded condition and the heavily loaded condition. We also develop an optimal solution to the lightly loaded condition by converting it to min-cut problem, and design a

3-approximation ratio algorithm under the heavily loaded condition as the problem is NP-hard. We then implement a fully functioning system. Evaluations show that DADS can effectively improve latency and throughput in an order compared with executing the entire DNN on the edge or on the cloud.

Chapter 5

TAPU: a New Processing Unit for Accelerating Multi-type Functions in IoT Gateways

5.1 Introduction

Internet of Things (IoT) Gateway is emerging as a key element of bringing legacy and next-gen devices to the IoT. Nearly every IoT system needs some way to connect devices to the cloud so that data can be sent back-and-forth between them. IoT gateways can be essential in making this connection possible because gateways act as bridges between devices and the cloud. They integrate protocols for networking, help manage storage and edge analytic on the data, and facilitate data flow securely between edge devices and the cloud.

With the exponential growth in the number of connected devices (an installed base of 15.4 billion devices in 2015 to 30.7 billion in 2020 [94]), the booming of data and the increasing requirement on data security, the functions an IoT gateway need to support keeps increasing nowadays. The first example is to handle network lea protocols for secure transmissions of critical data. The second example is to support video analytics. For example, in-situ cameras have been installed at smart home to detect and prevent the elderly from falling [30]. The video data is aggregated to the

IoT gateway. The IoT gateway can relay all videos to the cloud, yet this leads to huge cellular costs, transmission latency and privacy problem. As such, there is a need to execute video analytic within the IoT gateway.

As the functions of the IoT gateway increase, the current generation of IoT gateway face the pressure to upgrade their hardware. A CPU can handle the functionality of the first generation IoT gateways which are designed to facilitate communication protocol compatibility and device management functions. For the second generation IoT gateways which support regular network functions and video analytic, performing software network functions (i.e. processing on CPU) and adding a GPU as an accelerated for video analytic can meet the requirement on computation as shown in Fig. 5.1(a). For example, Dell Edge Gateways for IoT [4] and ADLINK IoT gateway products [1] provide the configuration choices of GPU. When comes to the current generation of IoT gateway with the advanced network functions (e.g. data encryption and decryption, data compression), such method results in CPU overloading and low utilization of GPU as shown in Fig. 5.1(b). More specifically, performing the advanced software network functions consumes the most of CPU resources so that few or none CPU resources for data pre-processing of video analytics, which further leads to low utilization of GPU.

To overcome the above limitations, a feasible solution is offloading both video analytic and the advanced network functions to the hardware accelerator as shown in Fig. 5.1(c). Video analytics need accelerator with parallel processing power such as GPU, while network functions need specialized hardware accelerator with programmability and the ability to customize the hardware. Customizing a dedicated hardware accelerator for each function leads to low utilization, as the accelerator only operates when the function is called and is idle at the other time. It also leads to expensive cost and bloated size, while the IoT gateway usually has stringent constraints in cost and size. In a short summary, IoT gateways are seeking

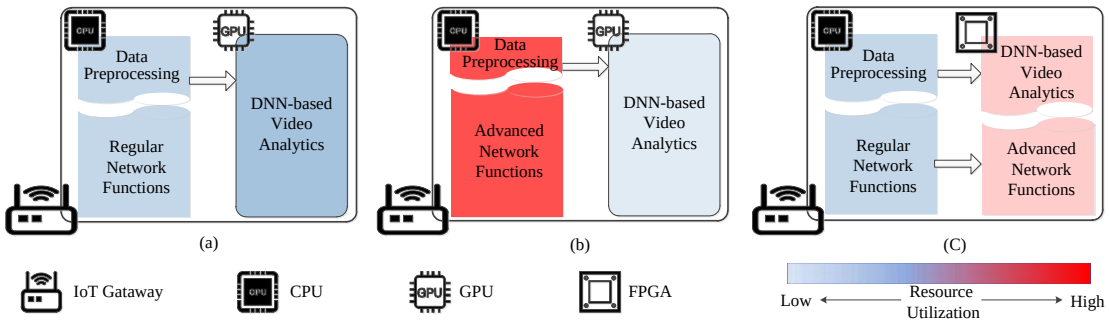


Figure 5.1: IoT gateways and its functions. (a) IoT gateway with a GPU for data analytic and regular network function on CPU. (b) The current IoT gateway with a GPU for video analytics and advanced network functions on CPU. (c) IoT gateway with offloading both advanced network functions and data analytic to the hardware accelerator.

for a multiplex-enable accelerator for video analytic and transmission (i.e. network functions) with the programmability and the ability to customize.

In this chapter, we propose a novel Transmission-Analytics Processing Unit (TAPU), a new accelerator using multi-image FPGAs to accelerate data analytic and network functions for data transmission in IoT gateway. The architecture of the multi-image FPGA is shown in Fig. 5.2. A multi-image FPGA can pre-store multiple images in

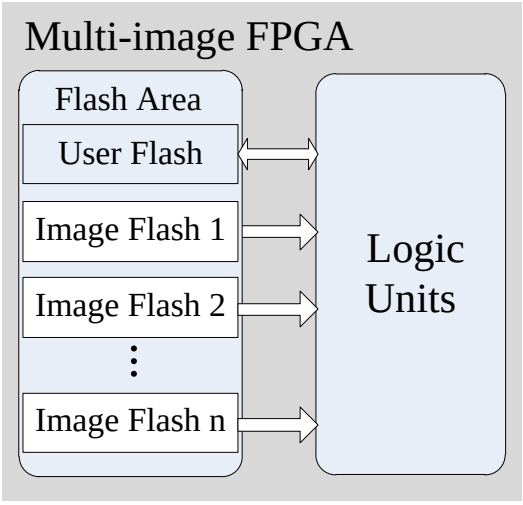


Figure 5.2: The multi-image FPGA.

the FPGA image flash and fast switch between images; an example of such FPGA is the newly developed off-the-shelf Intel Max10, a dual-image FPGA. We can then pre-configure one image for analytic and the other images for network functions. Thus, we can multiplex the accelerator for both analytic and network functions by switching images. FPGA naturally has the programmability and the ability to customize the hardware.

To bring TAPU into reality, we face two challenges: as the very first study, it is not clear what modules should be added or modified when incorporating TAPU into IoT gateway and how to offload the floating *workload* of video analytic and network processing to TAPU so that the computation capacity can be maximally exploited.

In this chapter, we first present the system design of TAPU from the hardware to the software. For the hardware design, we discuss the FPGA choice and abstract the accelerator modules as hardware functions that the developers can call just like calling a software function. For the software of TAPU, we provide offline manager that determining how to pre-configure the two images of the FPGA and runtime manager that determining how to switch images adapting to runtime variations. We choose to use TAPU process network packet first and use the residual computation capacity to accelerate analytic in terms of the needed executing time. To maximally exploit the computation capacity of TAPU, we first develop two schemes to estimate the residual computation capacity of TAPU for non-preemption case and general case respectively. We then develop an inference task offloading algorithm to offload and run in parallel the inference tasks in the FPGA, with the bojective to minimize the video analytics job.

We develop a fully functioning system, using Raspberry Pi as the IoT gateway, connected with a Logitech BRIO camera, and a TAPU using off-the-shelf Max 10, a dual-image FPGA. Currently, Max10 cannot auto-switch between images. We redesign the hardware switch of Max10.

Finally, we evaluate TAPU through trace-driven experiments. We show that TAPU can improve the performance of data analytic and transmission for 1.49 times and 2.33 times respectively compared with the current approach and the utilization of TAPU can reach up to 92.51%.

5.2 System Design

Here we present the system design of TAPU as shown in Fig. 5.3. To make TAPU as an efficient and easy-use accelerator for transmission and video analytics, we consider design from both the hardware and the software.

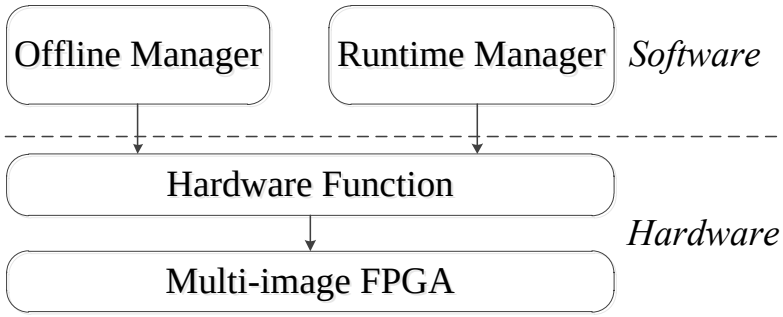


Figure 5.3: Overview of the system design of TAPU.

5.2.1 Hardware Design

Owning to its performance and programmability, FPGA is employed as the hardware by TAPU. Currently, there are many types of FPGA. TAPU employs the multi-image FPGA. In this subsection, we illustrate the reason for this FPGA choice, and design hardware functions to easily use the multi-image FPGA.

FPGA Choice. One term in FPGA is *image* which is an FPGA bitstream that contains the programming information for an FPGA. An FPGA device must be programmed using an image in order for it to implement the specified functionalities. For the traditional single-image FPGA, it usually takes minutes to program FPGA

according to a new image. Recently emerging multi-image FPGA can pre-store multiple images in the FPGA flash and fast switch between images, e.g., the newly developed off-the-shelf Max10, a dual-image FPGA. TAPU chooses the multi-image FPGA considering from the sharing pattern aspect.

TAPU accelerates both network functions and video analytics. This means network functions and video analytic share the limited FPGA resource. There are two ways to share FPGA: 1) sharing on *space* dimension, i.e. a part of the programmable logic units of FPGA is configured to implement network functions, the other is configured to implement video analytics; 2) sharing on *time* dimension, i.e. one image is configured for network functions, one image is configured for video analytic, and configuring one of these two images to logic units at a time.

TAPU chooses to share on time dimension for the following reasons. First, the programmable logic units (look-up tables, registers, and block RAMs) of an FPGA are limited. Even designed carefully, it is hard to put everything into FPGA simultaneously as each function consumes dedicated logic units. For example, implementing medium DNN model GoogLenet [69] on Intel Max10 FPGA consumes almost 90% logic units. This makes it unpractical to share FPGA on the space dimension. Indeed, we can use a more advanced FPGA which contains more resources, but the price of high-end FPGA is very high. Second, sharing on time is more flexible compared to sharing on space in terms of resource allocation. For sharing on space dimension, the resource occupying rate of each function is fixed. If the system wants to change the resource allocation for a function, modifying the image and reprogramming FPGA is needed, which usually takes hours. However, sharing on time can adapt the FPGA resource allocation by simply increasing or decreasing the time slots for each function.

Sharing on time dimension requires to fast switch between images, which can be fulfilled by the multi-image FPGA. For example, it cost near-zero time (less than 9

Table 5.1: Hardware APIs

Functions	Description
`Img_load()`	load the user specified image to the user specified configuration flash area.
`Img_program()`	program the image which has been pre-loaded in the specified configuration flash area into the logic units.
`Img_search()`	query the desired image pre-loaded in which area
`Img_programmed()`	query which image has been programmed into the logic units.
`Task_process()`	get the raw data to FPGA as input, and run the image which has been programmed to the logic units.

ms) for Intel Max10 dual-image FPGA to switch between two images. Thus TAPU chooses the multi-image FPGA as the hardware rather than traditional single-image FPGA.

Listing 5.1: Code of offloading a video analytics task to FPGA using hardware function calls

```
1 //program the desired image into logic units
2 If (Desired_Image!=Image_programmed()) {
3     image_area=Image_search(Desired_Image);
4     Image_program(image_area);
5 }
6
7 //Process the video analytic task
8 result=Task_process(frame);
```

Hardware Function Abstractions. TAPU offloads the workload of multi-type functions to multi-image FPGA and allocates FPGA resource to functions by controlling the time for the images. However, it is difficult and time-consuming for the upper layer to write code that covers specifics of FPGA offloading and carefully handles vendor-specific details. To this end, we design *hardware function abstractions* to enable upper layer using the multi-image FPGA in the form of a function call.

The hardware function abstractions provide the upper layer a programming model that fully encapsulates the low-level specifics of the multi-image FPGA. The APIs for

the upper layer to interact with FPGA are shown in Table 5.1. Using the hardware APIs, we can use multi-image FPGA easily in a way of a function call, without knowledge about the low-leave specifics. In the following, we explain the process of using the hardware APIs to offload a video analytic task to the multi-image FPGA shown in Listing 5.1.

From Listing 5.1, we can see that function `Image_programmed()` is first called to check whether the programmed image in the logic units is the desired image used to process the video. If not, function `Image_search()` is called to find which configuration area stores the desired image, and program the image into logic units by calling function `Image_program()`. Finally, function `Task_process()` is called to use the frame of the video as input, and run the image to get the analytic result.

5.2.2 Software Design

To ease application development and maximally utilize the FPGA resource, we design a series of software. As shown in Fig. 5.3, the software of TAPU includes two components: 1) TAPU offline, to determine what functions should be offloaded to FPGA and how to pre-configure the images of FPGA; and 2) TAPU runtime, to determine how to switch images in runtime. Runtime switching adapts to runtime variations, and maximally utilized the TAPU resource. We develop *Offline Manager* and *Runtime Manager* respectively.

Offline Manager. The gist of the offline manager is to decouple the developers from the time-consuming FPGA programming and the complicated decision on images of FPGA. To achieve the former goal, offline manager design a database called *Image Lib* to store the pre-configured images. For the later one, the offline manager designs a *Network Functions (NF) Image Decision* module to determine what network functions are offloaded to FPGA and a *Video Analytics (VA) Image Decision* module to determine what DNN model Image is used for video analytic. Fig. 5.4

illustrates the modules in the offline manager.

Image Lib: Developing and programming FPGA is time-consuming. It usually takes hours to synthesize and implement HDL (Hardware Description Language) source code to the final image. Developing after demand presents a significant barrier to fast development. To solve this, TAPU implements image library (Image Lib) which consists of a network function library (called NFLib in short) and a video analytic models library (called VALib in short). Image Lib provides images in a "ready for using" way.

NFLib contains images that implement commonly used network functions, e.g. IPsec (a secure network protocol suite that can encrypt and authenticate data packets), Gzip (a function used for file compression/decompression.). VALib contains images that implement commonly used video analytic models, e.g. the light-weight DNN model with moderate predict accuracy Yolo V2, the complex DNN model with high predict accuracy VGG.

Clearly, Image Lib cannot contain all network functions and video vision models in advance. TAPU designs an API for developers to add own images into the respective image library.

NF Image Decision: In the IoT Gateway, there are tens of network functions, while the number of FPGA images is limited. We can see a clear gap between the number of FPGA images and the number of network functions. It is impossible to offload all network functions to FPGA. The problem we face here is that determining which network functions to offload to FPGA. NF image decision module makes the choice based on the the performance of processing the network functions on different hardware, and the "stability" of these network functions.

The first consideration is the performance of processing the network functions on hardware. Operating Network function on FPGA is not always perfect, and operating network function on CPU is not always that bad. There are two types of processing in

data plane: *shallow packet processing*, i.e. executing operations based on the packet header, e.g. NAT, firewall; *deep packet processing*, i.e. executing operations on the whole packet data, such as IPsec gateway, flow compression, etc. For shallow packet processing, as the packet headers follow the unified protocol specification, it usually does not need too much computation efforts. It is fairly fast for CPU to perform shallow packet processing. In terms of deep packet processing, it usually needs much more compute resources since data of higher layers has no regular patterns take more CPU cycles to complete the processing [54]. Thus, NF image decision module gives priority to offload deep packet processing type of network function to FPGA, and if there are available image left, it offloads shallow packet processing type of network function.

The second is the stability of network functions. Since debugging and modifying the design of network functions on FPGA is time-consuming, NF image decision module perfects to offload the stable network functions to FPGA. It is easy to count the changing frequency of a network function based on the development log. NF image decision module uses the changing frequency as the parameter of stability.

In short summary, NF image decision module makes the decision based on the types (shallow/deep packet processing) and then the stability of network functions.

VA Image Decision: For the video analytic in the IoT gateway, unlike network functions, only one DNN model is needed. However, different video analytic models result in varied computation workload and inference quality.

The problem we face here is given the requirements on inference quality, select the DNN model that can meet the requirements and minimize the video analytics execution time. To solve this, we first profiles the relationships among the DNN model,inference quality, and video analytics execution time. These parameters can be well derived in advance. and then we select the model which meets the requirements of video analytic while the execution time is minimum.

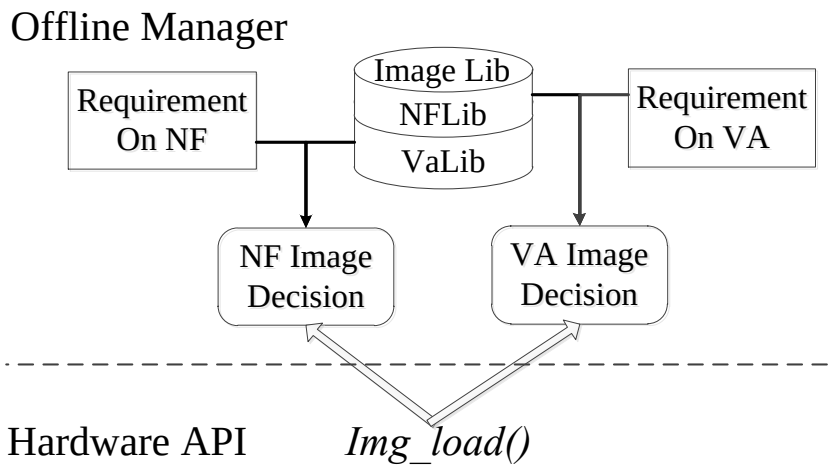


Figure 5.4: Offline Manager

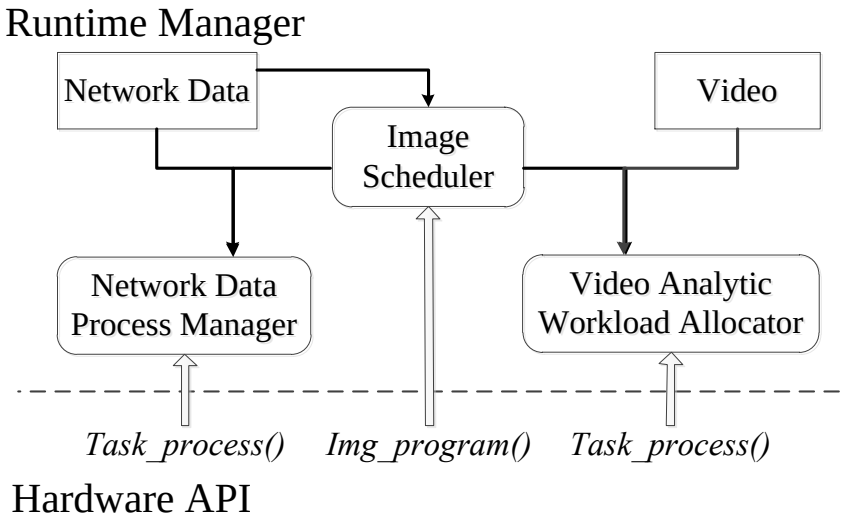


Figure 5.5: Runtime Manager

Runtime Manager. Runtime manager handles the FPGA resource allocation. More specifically, the runtime manager controls the switch of the FPGA images to allocate FPGA resource to network functions and video analytics. The objective of the runtime manager is to maximize the FPGA resource utilization. Runtime manager consists of the following modules shown in Fig. 5.5.

Image Scheduler is the core module of runtime manage and is design to allocate FPGA resource to network functions and video analytic by controlling the switch of

FPGA image.

Image scheduler first allocate resource for network functions, and then use the residual computation capacity to accelerate video analytics. Image scheduler give priority to network function when allocating FPGA resource. This is because that small packet size that an Ethernet packet varies from 64 bytes to 1500 bytes, and FPGA-based accelerating solutions for network functions processing such small size data only requires ultra-low microsecond-level latency, while the data size of video varies from tens to hundreds of megabytes with millisecond-level latency requirement. The processing latency of network function is much smaller than that of video analytics.

Image scheduler estimates the network traffic, and based on the estimation result allocates enough FPGA resource for network functions (i.e. switch to network function image with enough time in this work) and residual FPGA resource to video analytics. The network traffic estimation and the computation of residual resource are described in Sec. 5.3 Image scheduler sends the allocation result to network data process manager and video analytics workload allocator (described in below), and calls hardware API `Img_program()` to switch corresponding image when resource is allocated to that function.

Network Data Process Manager is designed to schedule the network packet processing when receives the resource allocation result from image scheduler. When the resource is allocated to network functions, this module calls hardware API `Task_process()` to operate network functions on all the unprocessed network packets.

When TAPU employs trace-driven estimation scheme when estimating the network traffic (see Sec. 5.3.2), this module has another function. It sends a preemption signal to image scheduler module when existing network packets will validate the delay tolerance. When image scheduler module receives preemption signal, it switches

FPGA to network function image immediately.

Video Analytic Workload Allocator schedules the processing of the *video analytics tasks* when FPGA resource is allocated for video analytics. A video analytics task is processing a frame of video using the given DNN model.

The video analytic tasks can be executed on the local CPU of or the FPGA of TAPU. We design a Video analytic workload allocator to determine where the inference tasks are executed. It allocates the inference task to CPU one by one due to the limited computation power of CPU. As the performance of executing video analytic tasks on TAPU is affected by the batch size, the resolution of the input frames, tt also computes the resoluion of input frames and the number of inference tasks in a bathc allocated to FPGA when receiving the residual computation capacity from the residual capacity estimation module. Video analytic workload allocator runs the inference task offloading optimization algorithm (described in Sec. 5.4) to minimize the overall delay to process video analytic tasks using the allocated FPGA resource.

5.3 Residual Computation Capacity Estimation

We now study residual computation capacity estimation. As the FPGA is shared in the time dimension, we use the amount of time that can be allocated for video analytic as the metric to represent the residual computation capacity. We divide time into *periods*. Let T_p be the time length for a period. Each T_p consists of a sub-period T_n for the network functions, and a sub-period T_r for video analytics.

The length of the sub-period of network functions images is determined by the traffic load from the application. Estimating application traffic load heavily depends on whether the system is developed for single applications or general purposes. Single application examples include the traffic surveillance camera which transmits the

video of traffic to the remote control center and recognizes car license appearing in the video, the Google Xbox which relays the game data between the players. A general-purpose engine is to accelerate video analytic for a set of upper layer applications, e.g., the mobile phone with functions such as speech recognition, fingerprint authentication and face recognition. Our framework can be used for both.

Clearly, if the system is developed for a single application, we can develop an in-depth traffic load model that captures the application characteristics. There are many existing studies. For example, [81] use a Markov modulated gamma process to model 3D video traffic, [67] proposes a probability density function to model the network traffic for games. For general application traffic estimation, there are mainly three methods to model traffic. The first one assumes sources continually send data at a constant rate to the network [29]. The second method employs probabilistic functions to model traffic behavior [50]. This method always observes the application traffic follows a specific distribution and adjust the parameter of the distribution to model the network traffic. The third method employs traffic traces to evaluate network performance [32].

The uniqueness of our case is on *preemption*, i.e., network packet processing is the first priority and video analytic acceleration can be preempted when necessary. We first study an extreme that application traffic estimation should not incur any preemption, we call it as *non-preemption estimation scheme (NPES)*. We then study a general case where preemption is allowed and we estimate application traffic using the traffic load of previous periods, we call it as *trace-driven estimation scheme (TDES)*.This is a simple scheme which can be most widely applied.

5.3.1 Non-Preemption Estimation Scheme

In this section, we design a non-preemption estimation scheme (NPES) to estimate the residual computation capacity and guarantee no preemption. The objective of

NPES is to guarantee no preemption and to maximize the residual computation capacity T_r in each period.

The residual computation capacity in a period depends on: 1) the length of a period, 2) the number of packets processed in a period, and 3)the network packet processing throughput of FPGA. In order to compute the exactly residual computation capacity of each period, We should have the knowledge of the number of packets processed in the period. Thus for period i, NPES chooses to buffer all network packets and process these packets at the beginning of next time period $i + 1$.

The first objective of NPES is to guarantee no preemption. As said, for each packet, there is a delay tolerance d, which is determined by the application. The packet should be processed before validating the tolerated delay, otherwise, a preemption will be incurred. We make the following assumption that the network packet processing throughput w of FPGA is bigger than the maximum generation speed of the network packets v_{max}, i.e. $s > v_{max}$.

Theorem 5.1. *If the network packet processing throughput w of FPGA is bigger than the maximum generation speed of network packets v_{max}, and the period $T_p \leqslant \frac{d}{2}$, no preemption will be incurred in NPES.*

Proof. No preemption means all packets can be processed before the tolerance delay d. Thus we prove that if $s > v_{max}$ and $T_p \leqslant \frac{d}{2}$, all packets can be processed before validating the tolerated delay. Let T_w denote the time a packet p_j needed to wait for being processed, we prove that $T_w \leqslant d$.

T_w includes two parts: 1) the time in which the packet is buffered in its generation period i, denoted as T_b, and 2) the time needed to wait for NIC to process the unprocessed packets generated before p_j in the period $i + 1$, denoted as T_c. It is obvious that $T_b \leqslant T_p$. Let N denote the number of unprocessed packets generated before p_j, we have $T_c = \frac{N}{s}$. As the maximum generation speed of the network packets

is v_{max}, thus $N \leqslant T_p v_{max}$, combined with $T_c = \frac{N}{s}$ and $v_{max} \leqslant s$, we have $T_c \leqslant T_p$. Therefore, $T_w = T_b + T_c \leqslant 2T_p$. If $T_p \leqslant \frac{d}{2}$, we have $T_w \leqslant d$.

□

As the network packet processing throughput of FPGA is faster than the generation speed of network packets, thus the longer period will lead to more residual computation capacity. Therefore, the length of the period is set to be $\frac{d}{2}$, i.e. $T_p = \frac{d}{2}$.

Let v be the number of packets generated in the previous period. Let s be the network packet processing throughput of FPGA. We can compute $T_r = \frac{d}{2} - \frac{v}{s}$. Note that, v is the number of packets generated in the previous period, it is a known value when we compute the residual computation capacity.

5.3.2 Trace-Driven Estimation Scheme

We develop a trace-driven estimation Scheme TDES for a general case which allows preemption to estimate the residual computation capacity. The objective of TDES is to maximize the residual computation capacity T_r in each period. Our idea is to predict the number of the packet that will be generated in the period. Obviously, if we can exactly predict the number of packets denoted as v_p, we can compute the residual computation capacity $T_r = T_p - \frac{v_p}{s}$, where s is the network packet processing throughput of FPGA.

We choose to use network traffic traces to estimate application traffic rather than using a constant injection rate and probabilistic function. This is because, in general, the prediction accuracy of the trace-based method is superior to that of constant injection rate and probabilistic function [82]. We employ auto-regressive (AR) predictor [75] to predict the traffic load in the next period based on the traffic load of previous periods.

Let K denote the number of periods whose network information is used to es-

timate the network traffic of the next period P_p. Let P_i denote the i-th period before P_p, where $i = 1, 2, \cdots, K$. Let w_i and v_i denote the actual and predicted number of network packets generated in period P_i respectively. Let v_p denote the number of network packets generated in period P_p. The AR model can be present as $v_p = \eta_1 w_1 + \eta_2 w_2 + \cdots + \eta_K w_K + \alpha$, where α is the white noise, η_i $(1 \leqslant i \leqslant K)$ is the smoothing value which reflect the periodic characteristics of the network packet. We adopt the following equation to compute the mean square error (MSE) between the actual packet quantity w_i and the periodic network packets q_i:

$$MSE = \frac{1}{K} \sum_{i=1}^{K} (w_i - v_i)^2. \tag{5.1}$$

The smaller MSE can better reflect the periodic characteristics, thus we carefully choose K and η_i $(1 \leqslant i \leqslant K)$ to make MSE smallest.

Now we consider the length of T_p which also affects the residual computation capacity. As each packet has a delay tolerance d, thus it can be buffered at a most d time, otherwise, a preemption is incurred. As it is hard to predict the exactly generation time of packets, we consider the worse case that a packet is generated at the beginning of the period. Thus, we set the length of the period to be d, i.e. $T_p = d$. Combined with the predicted number of packets generated in the next period v_p and the packet processing throughput s, we can compute the residual computation power $T_r = d - \frac{v_p}{s}$.

5.4 Inference Task Offloading

The objective of this section is to study how to assign *inference tasks* to minimize the execution time of *video analytic job*. A video inference task is to make an inference on a frame of video using the given DNN model. A video analytic job consists of

multiple inference tasks. In our system, the inference tasks can be executed on the CPU or on the TAPU. The CPU on the edge deivce is always available for executing inference task while the TAPU can be used for executing inference task only when it has residual computation capacity, thus our objective is equivalent to maximize the number of inference tasks completed in FPGA in the limited residual computation capacity.

In FPGA processing, to maximally exploit the FPGA capability, inference tasks should be processed in parallel. In other words, we can group inference tasks into a batch. Let $\mathcal{B} = \{b_1, b_2, \cdots, b_n\}$ denote the set of possible batch size. The execution time of batch depends on the resolution of the input frame and the batch size. We define execution time function $t(b_i, r)$ to present the execution time of batch with batch size b_i when the resolution of the input frames is r. Note that the residual computation capacity is T_r. In this time slot, multiple batches can be executed. Let q denote the number of batches assigned to FPGA. Let $\mathcal{P} = \{p_1, p_2, \cdots, p_q\}$ be the set of the batch size of batches, where $p_i \in \mathcal{B}$. Let R denote the resolution of input frames. The total execution time of such q batches cannot exceed the residual computation capacity T_a, thus we have $\sum_{i=1}^{q} t(p_i, R) \leqslant T_r$.

On the application side, the application has requirement on *inference quality*. The inference quality is an indicator of the prediction accuracy. Higher inference quality indicates higher prediction accuracy. F1 score [97] is employed as the criterion for inference quality (details in Section 5.4.2). F1 score depends on the resolution of input frames. We define the inference quality function $f(r)$ to represent the F1 score when the resolution of input frame is r. Let F denote the F1 score required by the application. The inference quality should meet the application's requirement, thus we have $f(R) \geqslant F$.

The **inference task offloading (ITO) problem** is given the inference quality requirement F, the residual computation capacity T_a, determine the resolution of

		Resolution					
		240p	360p	480p	576p	720p	1080p
Batch Size	1	117.18	126.05	219.15	263.09	476.18	894.51
	2	168.07	168.07	338.29	463.09	907.80	1713.43
	4	231.12	188.98	475.61	632.60	1178.25	2024.32
	8	434.24	329.47	-	-	-	-

Table 5.2: Batch execution time profile of tiny YOLOv2 DNN model on Intel Max 10 FPGA

video injected R, the number of batches q, and the appropriate batch size $\mathcal{P}$, so that the execution time FPGA needs to process the total batches does not exceed the computation capacity, i.e. $\sum_{i=1}^{q} t(p_i, R) \leqslant T_r$, and meet the inference quality requirement, i.e. $f(R) \geqslant F$, to maximize the number of inference tasks completed on FPGA, i.e. $\sum_{i=1}^{q} p_i$.

There are three challenges to solve ITO Problem: 1) Batch execution time profiling: the execution time needed to process inference tasks in batch on FPGA is affected by the resolution of the input frames and the batch size. Therefore we need to estimate execution time needed to inference tasks in different batch sizes and resolutions, i.e. computing the execution time function $t(b_i, r)$; 2) inference quality profiling: the inference quality depends on the resolution of the input frames. Therefore we need to estimate the inference quality under different resolutions, i.e. computing the inference quality function $f(r)$; and 3) to determine the number of batches q, the appropriate batch size set $\mathcal{P}$ and the resolution R. In the following of this section, we give the solutions to the above three challenges.

5.4.1 Batch Execution Time Profiling

The execution time needed to process inference tasks in batch on FPGA is affected by the resolution of input frames and the batch size.

We estimate the latency of inference tasks executed in batching through mea-

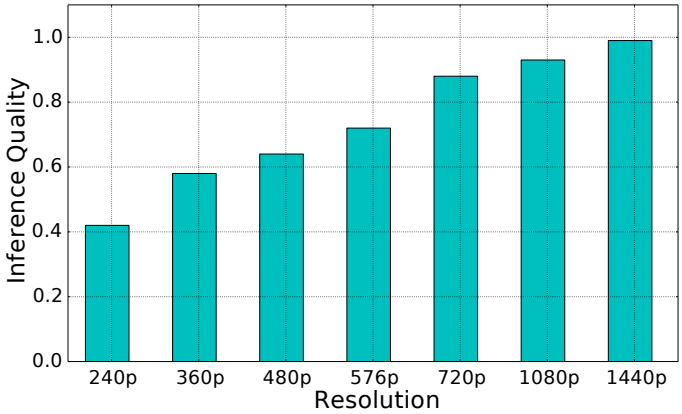


Figure 5.6: Inference quality profile of tiny YOLOv2 DNN model.

surement. More specifically, we run the inference tasks on all combination of the two parameters (i.e. the resolution of input frames, and the batch size), and record the execution latency. We admit that profiling through measurement is expensive, but it can achieve high accuracy compared with other profiling methods such as statistical modeling [31], analytical modeling [68].

We show the batch execute time profile in Tab. 5.2 when the platform is Max 10 FPGA, the DNN model is tiny YOLOv2 . It shows that higher resolution leads to higher execution latency. It also shows that a bigger batch size leads to higher total execution time but lower average processing time for each inference task.

5.4.2 Inference Quality Profiling

Inference quality is defined as the F1 score. The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

The inference quality depends on the employed DNN model and the resolution of the video. Using datasets where these data are labeled, we can empirically measure precision and recall of the inference result different resolutions of input frames. Using the precision and recall, we can compute the F1 score, i.e the inference quality. We

show the inference quality profiles in Fig. 5.6 when the DNN model is tiny YOLOv2. We can see that, higher resolution leads higher inference quality.

5.4.3 The Inference Task Offloading Algorithm

In this section, we first analyze the complexity of inference task offloading problem, and then design an algorithm to solve the inference task offloading problem.

Theorem 5.2. *Problem ITO is NP-complete.*

Proof. We prove this theorem by transforming the problem into the unbounded knapsack problem. The unbounded knapsack problem is given a set of items, each with a weight and a value, determine the number of each item to jinclude in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. For the problem ITO, the batch can be regarded as the item, the latency of inference tasks in each batch and the batch size can be regarded as the weight and the value respectively. The given limit weight is the available inference time. As a result, Problem ITO is equivalent to an optimal unbounded knapsack problem. □

Problem ITO is NP-complete, it is unrealistic to find a globally optimal solution within polynomial time. We develop Maximize Inference Task Offloading (denoted as MITO) approximation algorithm to solve ITO problem.

The problem ITO can be divided into two subproblems: resolution selection to meet the inference quality requirement (i.e. to find R) and batch selection (i.e. to find q and $\mathcal{P}$).

MITO first solve the resolution selection to meet the inference quality requirement. Higher resolution leads to higher inference quality and higher latency for inference tasks given the DNN model, the execution platform and batch size. Thus

we aim to select the resolution just to satisfy the quality requirement in order to reduce latency. This can be optimally solved by binary search the quality profile.

After the above step to determine r, the task assignment can be converted to the unbounded knapsack problem. The batch can be regarded as the item, the latency of inference tasks in each batch is the weight, the batch size is the value, the residual computation capacity is the capacity of the knapsack. MITO employs the greedy unbounded knapsack algorithm in [25]. The greedy unbounded knapsack algorithm selects the batch size of each batch one by one. Whenever a batch size is selected, such number of inference tasks will be assigned to FPGA and be executed as a batch. In this way, q and $\mathcal{B}$ are determined.

Theorem 5.3. *The approximation ratio of algorithm MITO for ITO problem is 2.*

The MIT problem is equivalent to the unbounded knapsack problem. The approximation ratio of 2 comes from the greedy unbounded knapsack algorithm we employ in MITO. Due to space limitation, detailed proofs are omitted.

5.5 Implementation

We implement a prototype of TAPU and integrate it into IoT gateway, see Fig. 5.7. Here we use a Raspberry Pi to simulate IoT gateway, integrated with a Logitech BRIO camera capturing video. We establish a cloud as the data receiving end to communicate with the Raspberry Pi, and use a laptop computer to display the cloud end. Two deep packet processing network functions the data encryption standard (DES) algorithm and the high-speed data compression (Gzip), and video analytics are offloaded to TAPU for accelerating. In the following subsection, we present the implementation details.

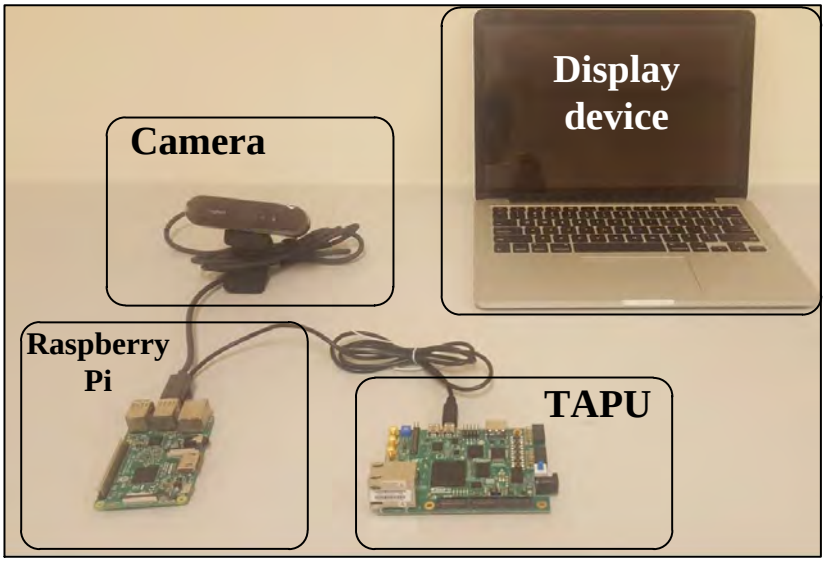


Figure 5.7: The prototype implementation.

5.5.1 Hardware Implementation

We choose Intel Max10 [5], a dual-image FPGA as the hardware board for TAPU.

Integration of TAPU and Raspberry Pi. We connect Max10 to the Raspberry Pi. Max 10 has a USB Blaster that can be used to connect the USB port of the Raspberry Pi using a USB-to-serial cable. We implement the communication between TAPU and Raspberry Pi using Thrift, an open source flexible RPC interface. The RPC message is transferred through the USB-to-serial cable. We also offload the workload of processing network packet in MAC and PHY layer to Max10, i.e. TAPU also serves as a network interface card (NIC), so that aft processing, network packets can be directly transferred to cloud without wasting time for returning processing result to NIC. We block the other NIC on the Raspberry Pi by editing the file "ifcfg-eth0" in Linux.*

FPGA Image Switch Control. TAPU provides the API `Img_program()` which is used to switch programmed images in logic units. One special challenge here is that

*Intuitively, Raspberry Pi has two NICs and we only use the Max 10 NIC.

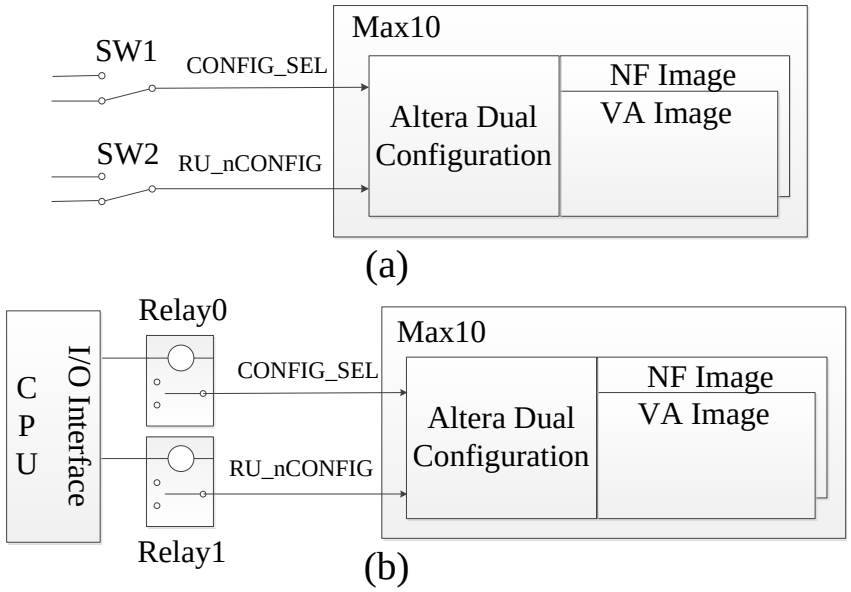


Figure 5.8: Illustration of using relays to replace switches.

the switching between images of FPGA should be done automatically by instructions. However, off-the-shelf Max10 only provides manual switching on the switch $SW1$ and the switch $SW2$ shown in Fig 5.8(a). One switch is used to select the image by $CONFIG\_SEL$ pin, and the other switch is used to trigger reconfiguration by $RU\_nCONFIG$ pin. We conduct a hardware redesign to replace the hand switch $SW1$ and $SW2$ by relays shown in Fig 5.8(a). We remove the switch $SW1$ and $SW2$, and solider a relay at the original place of $SW1$ and $SW2$ respectively. We show the code of API `Img_program()` in Listing 5.2. We can see that when switch to the image stored in flash area one, TAPU switches $CONFIG\_SEL$ pin by Relay0, then triggers the reconfiguration by pulling the $RU\_nCONFIG$ pins down by Relay1, and vice versa.

5.5.2 Network Functions Implementation

We implement network functions DES and Gzip in a single image. This is because only one image can be used for network functions in Max10 (the left one is allocated

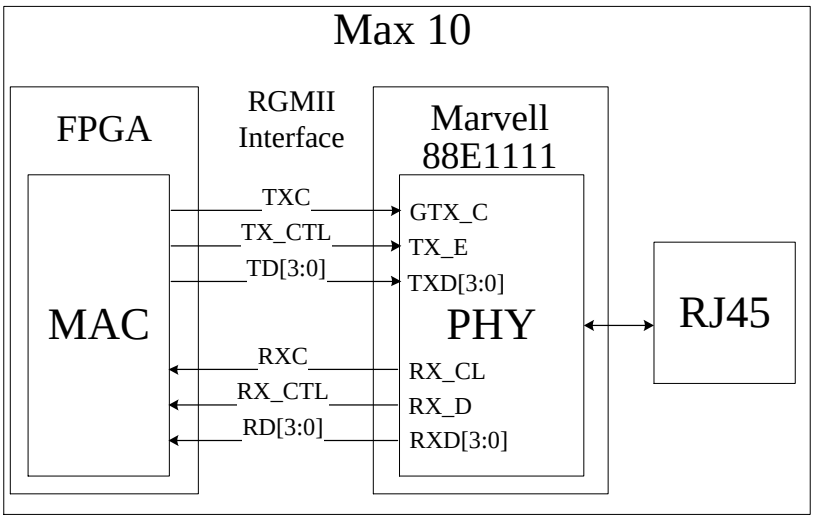


Figure 5.9: Implementation of processing network packet in MAC and PHY layer.

Listing 5.2: Code of API `Img_program(Desired_area)`

```
1 if (Desired_area == Configuration_Area_1){
2     Relay0 = 1;
3 else if (Desired_area == Configuration_Area_2)
4     Relay0 = 0;
5 Relay1 = 1;
```

for video analytic), and the logic units of Max10 FPGA is large enough for implementing these two functions together. We adopt the Partial Reconfiguration (PR) technology to implement the two functions in one image. The kernel architecture is based on a Verilog implementation by IBM presented in [58].

We also process network packet in MAC layer and PHY layer. The MAC layer constructs Ethernet frames, which includes realigning the payload, modifying the source address, calculating and appending the CRC-32 field, and inserting inter packet gap bytes. The PHY converts digital signal to analog signal. The challenges are that processing packet in MAC layer needs certain computation capability, and processing packet in PHY layer requires a PHY chip. Thus we implement the MAC layer in the FPGA area of Ma10 which has enough computation capacity and im-

plement the PHY layer in Marvell 88E111 chips of the Max 10 development board which includes a PHY chip shown in Fig. 5.9 MAC communicates with PHY using RGMII interfaces. PHY connects the Ethernet cable through the modular connector RJ45 to send the packets.

5.5.3 Video Analytic Implementation

We choose to implement the video analytic image through PipeCNN, an efficient OpenCL-based CNN accelerator on FPGAs, which provides faster hardware development cycle and software-friendly program interfaces. Using PipeCNN, we conduct design space exploration to find the optimal design that maximizes the throughput or minimizes the execution time. This reduces the development cycle significantly. We implement Yolov2 as the DNN model for video analytics.

5.6 Performance Evaluation

We use experiment based on real-world data to evaluate the implemented prototype.

5.6.1 Experiment Setup

Network Traffic Datasets: For the data transferred from IoT gateway (i.e. Raspberry Pi in our prototype) to the cloud, we use a real-world smart grid dataset released in [13], which contains several types of sensing data with timestamps. All types of data are transferred to the cloud periodically. For example, the value of real-time voltage whose sensing rate is 1 time per second is transferred to the cloud in a period of 20 seconds.

Video Analytic Setting: We configure the camera to capture 15 images per second, i.e. the frame rate is 15 FPS. The resolution of the captured image is 720 p. We employ YOLOv2 to detect animations in the video.

Evaluation Criteria: The network traffic of IoT gateway is regular, thus TAPU-based IoT gateway uses non-preemption estimation scheme by default.

We compare TAPU with the following two approaches. The first approach is offloading only video analytics to Intel Max10 FPGA and implementing all network function is software, .i.e processing network packets only in CPU. We call this approach as **Only-VA-FPGA** in short. The second approach is offloading video analytic and network function to two dedicated Intel Max10 FPGA without sharing. We call this approach as **Dedicated-FPGA**.

We first evaluate the throughput on video analytic and packet processing of TAPU, Only-VA-FPGA and Dedicated-FPGA. Then, we introduce utilized ratio of FPGA. THe utilized ration of FPGA is the ratio between the actually used FPGA resource and the total amount of FPGA resource. The utilized ratio of FPGA can indicate how effectively the resource of FPGA has been used. We evaluate the utilized ratio of the above three approaches.

5.6.2 Experiment Results

Throughput on video analytic. We compare the throughput on video analytic of TAPU, Only-VA-FPGA and Dedicated-FPGA in Fig. 5.10. We see that our TAPU and Dedicated-FPGA outperform Only-VA-FPGA significantly. The video analytic throughput of TAPU and Dedicated-FPGA are 1.49 times and 1.52 time to that of Only-VA-FPGA respectively. This is because that advanced network functions occupant most of CPU resource, and only a little of CPU resource is allocated for pre-processing of video analytic. Even though a dedicated FPGA can be used for accelerating video analytic in Only-VA-FPGA, the pre-processing of video analytic on CPU becomes the bottleneck. While for TAPU and Dedicated-FPGA, advanced network functions are processed on FPGA, thus enough CPU resource can be used for pre-processing.

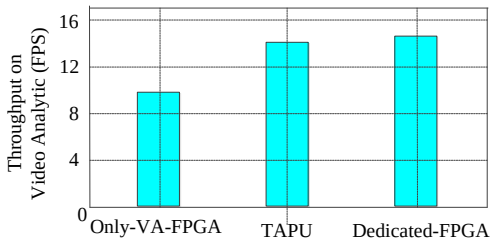


Figure 5.10: The throughput on video analytics.

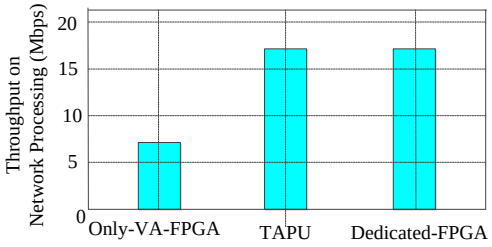


Figure 5.11: The throughput on network processing.

We also observe that the throughput of TAPU (14.71 FPS) is a little smaller than that of Dedicated-FPGA (15 FPS), however, this gap is small. In TAPU, FPGA is shared by video analytic and network functions, which leads such a small gap.

Throughput on network processing. We also show the network processing throughput of TAPU, Only-VA-FPGA and Dedicated-FPGA in Fig. 5.11. Similar to the result of throughput on video analytic, TAPU and Dedicated-FPGA outperform Only-VA-FPGA significantly in terms of throughput on network processing. TAPU and Dedicated-FPGA can process network packet at the speed of 16.8 Mbps which is 2.33 times to that of Only-VA-FPGA. Though most CPU resources are allocated for network functions, the speed of software-based processing cannot catch up that of hardware acceleration.

Different to the result of throughput on video analytic, even through network function and video share the FPGA resource, the throughput on network processing of TAPU is the same as that of Dedicated-FPGA. This is because the priority of network processing is higher than that of video analytic in TAPU. More specifically, FPGA resource is allocated for accelerating video analytic only when existing residual computation capacity after processing network packets.

The utilized ratio of FPGA. We also compare the FPGA utilization of TAPU, Only-VA-FPGA and Dedicated-FPGA. We show the Gantt chart of FPGA working

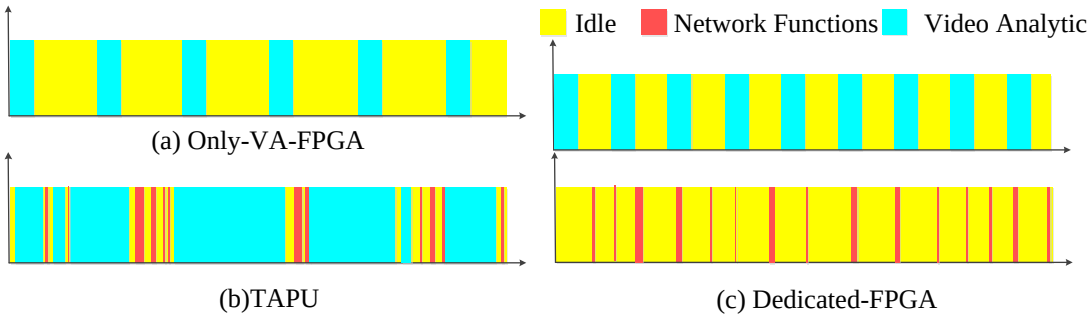


Figure 5.12: The working Status of FPGA. (The yellow box represents that FPGA is idle; the red box represents that FPGA is used for processing network packets; the blue box represents that FPGA is used for video analytics.)

Table 5.3: The average utilized ratio of FPGA.

Approach	Utilized Ratio
Only-VA-FPGA	37.29%
TAPU	92.51%
Dedicated-FPGA	36.32%

status of these three approaches in Fig. 5.12. We ignore the image switch time as it is too small (around 9 ms). From Fig. 5.12(a) and Fig. 5.12(c), we can see that in Only-VA-FPGA and Dedicated-FPGA, yellow box occupant a high ratio, which represents the FPGA is idle at the most of the time. Most of FPGA resource wastes in Only-VA-FPGA and Dedicated-FPGA. While for TAPU, from Fig. 5.12(c), We can see that the idle time (yellow box) of FPGA in TAPU is significantly less than that of Only-VA-FPGA and Dedicated FPGA. For TAPU, most of the resources are used to accelerate video analytic and network processing. This illustrates that TAPU can fully utilize FPGA resource.

We also show the utilized ratio of FPGA in Tab. 5.3. The FPGA utilized ratio in TAPU can reach up to 92.51%. It is 2.48 times compared to Only-VA-FPGA. When compared to Dedicated-FPGA, the gap becomes more significant, which is 2.55 times. Though Dedicated-FPGA has the best performance in throughput which gains from over provisioning, it has the lowest utilized ratio. TAPU has almost the

same performance in throughput with Dedicated-FPGA and much higher utilization without over-provisioning.

5.7 Chapter Summary

In this chapter, we first review the current status of exacting video analytic and network functions in the IoT gateway. Both the computation-intensive video analytic and the increasingly advanced network functions need the hardware accelerators to meet the performance requirement. The dedicated accelerator for each function results in low utilization and the IoT gateways usually have stringent constraints in size and cost. To this end, we propose TAPU, a new accelerator using multi-image FPGA to accelerate both video analtyics and network functions. As a very first study, we design from hardware to software which clarifies the necessary modules. To maximally exploit the computation capacity of TAPU with minimal influence of the network processing, we developed algorithms for residual computation capacity estimation and efficient algorithms for the video analytic task offloading to TAPU. We implement a fully functioning system. Our evaluation demonstrated that the average utilization of TAPU can reach up to 92%, and TAPU can achieve throughput on video analytic and network functions up to 1.49 times and 2.33 times.

Chapter 6

Conclusions and Future Directions

This chapter concludes this thesis by summarizing our original contributions in Section 6.1 and by pointing towards the possible future directions of furthering our research in Section 6.2.

6.1 Conclusions

The past decade has created tremendous expectations on IoT changing the landscape of data-driven services with benefits for multiple societal sectors. Many researchers have contributed to the development of technologies and addressed challenges that come with resource scarcity in the end devices. Other researchers with a background in cloud computing have looked at how to carry the data generated by the massive IoT deployments and how to efficiently use the cloud resources. The area of edge computing brings these two ends of the same service together in an emerging ecosystem and creates a means to discuss resource adequacy from an end-to-end perspective. In this thesis, we have tried to study resource management, not from a cloud perspective or an IoT device perspective, but with a focus on edge-side resource management.

In this thesis, we study three schemes of resource management on the edge from the perspective of communication resource sharing, the combination of computation

and communication resource optimization, and computation and communication resource provision respectively:

First, we carefully analyze the emerging IoT applications such as smart after-sales maintenance and services. We show that a separate IoT network is needed to serve their requirement of sending the data to the cloud. A core obstacle is the high costs of communication choices. We propose a solution of sTube+ on IoT communication sharing. The design of sTube+ includes a layered data delivery architecture, algorithms for cost optimization and incremental development of devices, and a prototype of a fully functioning system. Our evaluation results show that sTube+ can lead to a cost reduction of five times and eight times respectively for the two real-world cases. We further develop a case study of chiller and pump maintenance, where sTube+ acts as the underlying architecture.

Second, we study DNN inference acceleration through collaborative edge-cloud computation. We propose Dynamic Adaptive DNN surgery (DADS) scheme that can partition DNN inference between the edge device and the cloud at the granularity of neural network layers, according to the dynamic network status and the computation capacities of the edge device and the cloud. We present a comprehensive study of the partition problem under the lightly loaded condition and the heavily loaded condition. We also develop an optimal solution to the lightly loaded condition by converting it to a min-cut problem, and design a 3-approximation ratio algorithm under the heavily loaded condition as the problem is NP-hard. Real-world prototype based on self-driving car video dataset is implemented, showing that compared with executing entire the DNN on the edge and cloud, DADS can improve latency up to 6.45 and 8.08 times respectively, and improve throughput up to 8.31 and 14.01 times respectively.

Third, we review the current status of existing data-driven applications and network functions in the IoT gateway. Both the computation-intensive data-driven

applications and the increasingly advanced network functions need the hardware resources to meet the performance requirement. We propose a novel transmission-analytics processing unit, a new accelerator using multi-image FPGAs to provide both computation and communication resource for data-driven applications in IoT gateway. We evaluate TAPU through real trace-driven experiment. We see that the utilization of TAPU can reach up to 92%, and brings significantly higher throughput on video analytic and network processing over the current approach.

6.2 Future Directions

We close this thesis with our comments and suggestions on the ways in which the current research can be advanced.

First, with our proposed sTube+ data delivery architecture, there is a large space for research in price optimization for according to different pricing models and application scenarios. We plan to develop a comprehensive SAMS for the energy systems of buildings. The proposed sTube+ organizes a greater number of IoT devices belonging to the same vendor, with heterogeneous data communication requirements, to share fewer choices of TCC links and transmit their data to the cloud, where the IoT devices belong to the same vendor. The results of our evaluation show that the larger scale sharing can lead to greater communication cost reduction. If the devices belonging to different vendors share the communication resource, it will lead to a larger-scale sharing thus more monetary cost will be reduced. Privacy and a fair price for the data trade between different vendors are the challenges for this large scale sharing. Sharing the communication resource in multi-vendors can be a future direction for us to explore.

Second, when we design DNN surgery, we assume that the computation resource of the edge device and the cloud are always available for the data-driven application.

In some scenarios, the computation resources are shared by multiple applications. We will explore how to partition the DNN inference under the limited computation resources of the edge devices and the cloud. The proposed DNN surgery focuses on optimizing deep learning inference. We will explore whether it is beneficial to partition the deep learning training and how to partition the training process.

Finally, as the first exploration of hardware providing both computation and communication resource, the resource allocation of TAPU for computation and communication is straightforward. In the future, we will extend the resource management of the proposed TAPU to make it more efficient.

From the edge-side resource management prospective, we noted that the resource objectives allocation and optimization were well studied. Moreover, computation and communication resources are the most commonly addressed, typically being stationary and located within a single level. Therefore, research is less prevalent on data, storage, and energy as a resource and less extensive towards the estimation, discovery, and sharing objectives. Furthermore, new works should consider mobility and multilevel locality on the supply side. Elaborating on mobility, the new phenomenon at the edge is that the supply side can also be mobile and not only the demand side as it was the case in classic clouds. Indeed, edge systems will have to deal with a greater variety of mobility with end devices that are often mobile. It is not obvious that the mobility patterns of all those devices. More work is needed on collecting mobility traces from the different edge applications to see if present patterns can in a generic way be used to create pertinent edge mobility models. will be similar

Bibliography

[1] Adlink iot gateway. https://www.adlinktech.com/en/Industrial_IoT_and_Cloud_solutions_IoT_Gateway.aspx. Accessed May, 2019.

[2] AWS DeepLens. https://aws.amazon.com/deeplens. Accessed June, 2018.

[3] Berkeley vision and learning center. https://github.com/BVLC/caffe. Accessed Oct., 2018.

[4] Dell edge gateways for iot. https://www.dell.com/en-us/work/shop/cty/sf/edge-gateway. Accessed May, 2019.

[5] Intel max 10 fpga development kit. https://www.altera.com/products/boards_and_kits/dev-kits/altera/max-10-fpga-development-kit.html. Accessed Jun., 2018.

[6] Revolutionizing non-volatile integration. https://www.altera.com.cn/products/fpga/max-series/max-10. Accessed May, 2018.

[7] State of Mobile Networks: USA. https://opensignal.com/reports/2017/08/usa/state-of-the-mobile-network. Accessed June, 2018.

[8] Mohamed S Abdelfattah, Andrei Hagiescu, and Deshanand Singh. Gzip on a chip: High performance lossless data compression on fpgas using opencl. In *Proc. ACM IWOCL'14*, Bristol, UK, May 2014.

[9] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (12):2037–2041, 2006.

[10] Petar Aleksic, Mohammadreza Ghodsi, Assaf Michaely, Cyril Allauzen, Brian Hall, David Rybach, and Pedro Moreno. Bringing contextual information to google speech recognition. In *Proc. INTERSPEECH'15*, Dresden, Germany, Sep. 2015.

[11] Hamid Reza Arkian, Reza Ebrahimi Atani, Abolfazl Diyanat, and Atefe Pourkhalili. A cluster-based vehicular cloud architecture with learning-based resource management. *The Journal of Supercomputing*, 71(4):1401–1426, 2015.

[12] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[13] Sean Barker, Aditya Mishra, David Irwin, Emmanuel Cecchet, Prashant Shenoy, and Jeannie Albrecht. Smart*: An open data set and tools for enabling research in sustainable homes. *SustKDD, August*, 111(112):108, 2012.

[14] Abdlmonem H Beitelmal, Chandrakant Patel, et al. A steady-state model for the design and optimization of a centralized cooling system. *International journal of energy research*, 34(14):1239–1248, 2010.

[15] Luiz F Bittencourt, Javier Diaz-Montes, Rajkumar Buyya, Omer F Rana, and Manish Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35, 2017.

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[17] Paul Bonsma. Most balanced minimum cuts. *Discrete Applied Mathematics*, 158(4):261–276, 2010.

[18] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[19] Ulrich Brenner. A faster polynomial algorithm for the unbalanced hitchcock transportation problem. *Operations Research Letters*, 36(4):408–413, 2008.

[20] Sema Can, Bülent Kilit, Erşan Arslan, and Salih Suveren. The comparison of reaction time of male tennis players, table tennis players and the ones who don't exercise at all in 10 to 12 age groups. *Journal of Physical Education & Sports Science/Beden Egitimi ve Spor Bilimleri Dergisi*, 8(2), 2014.

[21] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168. ACM, 2015.

[22] Costas Courcoubetis and Richard Weber. *Pricing communication networks: economics, technology and modelling*. John Wiley & Sons, 2003.

[23] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proc. ACM MobiSys'10*, San Francisco, CA, Jun. 2010.

[24] Eduardo Cuervo, Alec Wolman, Landon P Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu, and Madanlal Musuvathi. Kahawai: High-quality mobile gaming using gpu offload. In *Proc. ACM MobiSys'15*, Florence, Italy, May 2015.

[25] George B Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[26] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[27] Stephen Dawson-Haggerty, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler. sMAP: a simple measurement and actuation profile for physical information. In *Proc. ACM SenSys'10*, Zurich, Switzerland, Nov. 2010.

[28] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 15–28. ACM, 2009.

[29] Masoumeh Ebrahimi and Masoud Daneshtalab. EbDa: A new theory on design and verification of deadlock-free interconnection networks. *ACM SIGARCH Computer Architecture News*, 45(2):703–715, 2017.

[30] Nashwa El-Bendary, Qing Tan, Frédérique C Pivot, and Anthony Lam. Fall detection and prevention for the elderly: A review of trends and challenges. *International Journal on Smart Sensing & Intelligent Systems*, 6(3), 2013.

[31] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *arXiv preprint arXiv:1801.08618*, 2018.

[32] WeiDong Feng, Yong Sun, Zheng Zhou, Qiang Rao, Di Chen, Linhui Yang, and Yawei Wang. Study on multi-network traffic modeling in distribution communication network access service. In *Proc. IEEE ICACT'18*, Mumbai, India, Feb. 2018.

[33] Nofirman Firdaus and Bambang Teguh Prasetyo. Chiller: Performance deterioration and maintenance. *Energy Engineering*, 113(4):55–80, 2016.

[34] Mirko Franceschinis, Marco Mellia, Michela Meo, and Maurizio Munafo. Measuring TCP over WiFi: A real case. In *1st workshop on Wireless Network Measurements*, Riva Del Garda, Italy, Sep. 2005.

[35] Christine Fricker, Fabrice Guillemin, Philippe Robert, and Guilherme Thompson. Analysis of an offloading scheme for data centers in the framework of fog computing. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 1(4):16, 2016.

[36] Jingkun Gao, Joern Ploennigs, and Mario Berges. A data-driven meta-data inference framework for building automation systems. In *Proc. ACM Buildsys'15*, Seoul, South Korea, Nov. 2015.

[37] Cesar A Garc, Pedro Merino, et al. 3GPP standards to deliver LTE connectivity for IoT. In *Proc. IEEE IoTDI'16*, Berlin, Germany, Apr. 2016.

[38] Gopal S Gawande and KB Khanchandani. Efficient Design and FPGA Implementation of Digital Filter for Audio Application. In *Proc. IEEE ICCUBEA'15*, Pune, India, Feb. 2015.

[39] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. MCDNN: An approximation-based execution framework for deep stream processing under resource constraints. In *Proc. ACM MobiSys'16*, Singapore, Jun. 2016.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE CVPR'16*, Las Vegas, Nevada, Jul. 2016.

[41] Jianwei Huang and Lin Gao. Wireless network pricing. *Synthesis Lectures on Communication Networks*, 6(2):1–176, 2013.

[42] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proc. ACM mobisys'17*, Niagara Falls, NY, Jun. 2017.

[43] Jing Jiang and Yi Qian. Distributed communication architecture for smart grid applications. *IEEE Communications Magazine*, 54(12):60–67, 2016.

[44] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proc. ACM ASPLOS'17*, Xi'an, China, Apr. 2017.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[46] Amey Kulkarni and Tinoosh Mohsenin. Accelerating compressive sensing reconstruction OMP algorithm with CPU, GPU, FPGA and domain specific manycore. In *Proc. IEEE ISCAS'15*, Lisbon, Portugal, May 2015.

[47] Joseph HK Lai, Francis WH Yik, and Aggie KP Chan. Maintenance cost of chiller plants in hong kong. *Building Services Engineering Research and Technology*, 30(1):65–78, 2009.

[48] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In *Proc. IEEE IPSN'16*, Vienna, Austria, Apr. 2016.

[49] Steven Latre, Philip Leroux, Tanguy Coenen, Bart Braem, Pieter Ballon, and Piet Demeester. City of things: An integrated and multi-technology testbed for IoT smart city experiments. In *Proc. IEEE ISC2'16*, Trento, Italy, Sep. 2016.

[50] Dongheon Lee, Sheng Zhou, Xiaofeng Zhong, Zhisheng Niu, Xuan Zhou, and Honggang Zhang. Spatial modeling of the traffic density in cellular networks. *IEEE Wireless Communications*, 21(1):80–88, 2014.

[51] Jay Lee, Chao Jin, and Zongchang Liu. Predictive big data analytics and cyber physical systems for TES systems. In *Advances in Through-life Engineering Services*, pages 97–112. Springer, 2017.

[52] Bojie Li, Kun Tan, Layong Larry Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proc. ACM SIGCOMM'16*, Brazil, Aug. 2016.

[53] Chengzhe Li, Lai Yoong Yee, Hiroshi Maruyama, and Yoshiki Yamaguchi. FPGA-based volleyball player tracker. *ACM Transactions on SIGARCH Computer Architecture News*, 44(4):80–86, 2017.

[54] Xiaoyao Li, Xiuxiu Wang, Fangming Liu, and Hong Xu. DHL: Enabling Flexible Software Network Functions with FPGA Acceleration. Jul. 2018.

[55] David Linthicum. Make sense of edge computing vs. cloud computing. `https://www.infoworld.com/article/3197555/make-sense-of-edge-computing-vs-cloud-computing.html`. Accessed May, 2017.

[56] Wei Liu, Ryoichi Shinkuma, and Tatsuro Takahashi. Opportunistic resource sharing in mobile cloud computing: The single-copy case. In *Proc. IEEE APNOMS'14*, Hsinchu, Taiwan, Sep. 2014.

[57] Mehdi Mahdavikhah and Hamid Niazmand. Effects of plate finned heat exchanger parameters on the adsorption chiller performance. *Applied Thermal Engineering*, 50(1):939–949, 2013.

[58] Andrew Martin, Damir Jamsek, and K Agarawal. Fpga-based application acceleration: Case study with gzip compression/decompression streaming engine. *ICCAD Special Session C*, 7:2013, 2013.

[59] Abderrahmen Mtibaa, Afnan Fahim, Khaled A Harras, and Mostafa H Ammar. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 51–56. ACM, 2013.

[60] Ghasem Naddafzadeh-Shirazi, Lutz Lampe, and Gustav Vos. Coverage enhancement techniques for machine-to-machine communications over LTE. *IEEE Communications Magazine*, 53(7):192–200, 2015.

[61] Antonio L Maia Neto, Artur LF Souza, Italo Cunha, et al. Aot: Authentication and access control for the entire iot device life-cycle. In *Proc. ACM Senys'16*, CA, USA, Nov. 2016.

[62] Dusit Niyato, Xiao Lu, Ping Wang, Dong In Kim, and Zhu Han. Economics of Internet of Things: an information market approach. *IEEE Wireless Communications*, 23(4):136–145, 2016.

[63] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, et al. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In *Proc. IEEE FPL'16*, Lausanne, Switzerland, Aug. 2016.

[64] Alberto Oliveri, Luca Cassottana, Antonino Laudani, et al. Two FPGA-oriented high-speed irradiance virtual sensors for photovoltaic plants. *IEEE Transactions on Industrial Informatics*, 13(1):157–165, 2017.

[65] Jessica Oueis. *Joint communication and computation resources allocation for cloud-empowered future wireless networks*. PhD thesis, 2016.

[66] Maire ONeill et al. Insecurity by design: Todays iot device security problem. *Engineering*, 2(1):48–49, 2016.

[67] HyoJoo Park, TaeYong Kim, and SaJoong Kim. Network traffic analysis and modeling for games. In *Proc. Springer WINE'05*, Hong Kong, China, Dec. 2005.

[68] Hang Qi, Evan R Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *Proc. ICLR'17*, Toulon, France, Apr. 2016.

[69] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 26–35. ACM, 2016.

[70] Darijo Raca, Jason J Quinlan, Ahmed H Zahran, and Cormac J Sreenan. Beyond throughput: a 4G LTE dataset with channel and context metrics. In *Proc. ACM MMSys'18*, Amsterdam, The Netherlands, Jun. 2018.

[71] Abdelrahim Ramadan. District cooling designing for life power & cost saving. `https://www.slideshare.net/AbdoRamadan1/district-cooling-design-case-study`. Accessed Jul, 2015.

[72] Meike Ramon, Stephanie Caharel, and Bruno Rossion. The speed of recognition of personally familiar faces. *Perception*, 40(4):437–449, 2011.

[73] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[74] JS Roessler. LTE-advanced (3GPP rel. 12) technology introduction. Apr. 2015.

[75] Aimin Sang and San-qi Li. A predictability analysis of network traffic. *Computer networks*, 39(4):329–345, 2002.

[76] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards qos-aware fog service placement. In *Proc. IEEE ICFEC'17)*, Madrid, Spain, May 2017.

[77] Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *Proc. ACM STOC'96*, May. 1996.

[78] Thomas Stockhammer. Dynamic adaptive streaming over HTTP: standards and design principles. In *Proc. ACM MMSYS '11*, Santa Clara, CA, Feb. 2011.

[79] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proc. AAAI'17*, San Francisco, CA, Feb. 2017.

[80] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. IEEE CVPR'15*, Boston, MA, Jun. 2015.

[81] Savera Tanwir, Debanjana Nayak, and Harry Perros. Modeling 3D video traffic using a Markov modulated gamma process. In *Proc. IEEE ICNC'16*, Kauai, Hawaii, Feb 2016.

[82] Leonel Tedesco, Aline Mello, Leonardo Giacomet, Ney Calazans, and Fernando Moraes. Application driven traffic modeling for NoCs. In *Proc. ACM SBCCI'06*, New York, NY, Sep. 2006.

[83] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *Proc. IEEE ICDCS'17*, Atlanta, GA, Jun. 2017.

[84] Vutha Va, Takayuki Shimizu, Gaurav Bansal, Robert W Heath Jr, et al. Millimeter wave vehicular communications: A survey. *in Now Publishers Journal on Foundations and Trends in Networking*, 10(1):1–113, 2016.

[85] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Proc. NIPS'11*, Granada, Spain, Jan. 2011.

[86] Ehsan Variani, Xin Lei, Erik McDermott, Ignacio Lopez Moreno, and Javier Gonzalez-Dominguez. Deep neural networks for small footprint text-dependent speaker verification. In *Proc. IEEE ICASSP'14*, Florence, Italy, May 2014.

[87] Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, and Sotiris Ioannidis. {GASPP}: A gpu-accelerated stateful packet processing framework. In *Proc. USENIX ATC'14*, Philadelphia, PA, Jun. 2014.

[88] Anna Maria Vegni, Valeria Loscr, Alessandro Neri, and Marco Leo. A Bayesian packet sharing approach for noisy IoT scenarios. In *Proc. IEEE IoTDI'16*, Berlin, Germany, Apr. 2016.

[89] Jianyu Wang, Jianli Pan, and Flavio Esposito. Elastic urban video surveillance system using edge computing. In *Proceedings of the Workshop on Smart Internet of Things*, page 7. ACM, 2017.

[90] Kezhi Wang, Kun Yang, Xinhou Wang, and Chathura Sarathchandra Magurawalage. Cost-effective resource allocation in c-ran with mobile cloud. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[91] Shiqiang Wang, Rahul Urgaonkar, Ting He, Kevin Chan, Murtaza Zafer, and Kin K Leung. Dynamic service placement for mobile micro-clouds with predicted future costs. *IEEE Transactions on Parallel and Distributed Systems*, 28(4):1002–1016, 2016.

[92] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. Deepburning: automatic generation of fpga-based learning accelerators for the neural network family. In *Proc. ACM DAC'16*, Austin, TX, Jun. 2016.

[93] Thomas Watteyne, Kris Pister, Dominique Barthel, Mischa Dohler, and Isabelle Auge-Blum. Implementation of gradient routing in wireless sensor networks. In *Proc. IEEE GLOBECOM'09*, Dec. 2009.

[94] Business Wire. Finding success in the new iot ecosystem: Market to reach $3.04 trillion and 30 billion connected "things" in 2020, idc says. `https://www.businesswire.com/news/home/20141107005028/en/Finding-Success-IoTEcosystem-Market-Reach-3.04`. Accessed May, 2019.

[95] Harunori Yoshida and Sanjay Kumar. ARX and AFMM model-based on-line real-time data base diagnosis of sudden fault in AHU of VAV system. *Energy Conversion and Management*, 40(11):1191–1206, 1999.

[96] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. The Internet of Things has a gateway problem. In *Proc. ACM HotMobile'15*, Santa Fe, New Mexico, Feb. 2015.

[97] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *Proc. USENIX NSDI'17*, Boston, MA, Mar. 2017.

[98] Quan Zhang, Xiaohong Zhang, Qingyang Zhang, Weisong Shi, and Hong Zhong. Firework: Big data sharing and processing in collaborative edge environment. In

2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), pages 20–25. IEEE, 2016.

[99] Zimu Zheng, Dan Wang, Jian Pei, Yi Yuan, Cheng Fan, and Fu Xiao. Urban traffic prediction through the second use of inexpensive big data from buildings. In *Proc. ACM CIKM'16*, Indianapolis, IN, Oct. 2016.